Modular Arithmetic

XOR: addition modulo 2 in GF(2)
AND: multiplication modulo 2 in GF(2)
Let $a,r,m \in \mathbb{Z}$, $m>0$, r is remainder, m is modulus
$A \equiv r \bmod m$, if m divides $a-r$
Infinitely many r, for any a & m

Closure: For all possible input values from the set, the output values are also an element of the set.
i.e: $a \circ b = c \in S$, for $a,b,c \in S$

Associativity: The order of evaluation of operations doesn't change the result of the expression.
i.e $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a,b,c \in S$

Identity: There is an identity element $i \in S$ such that,
$i \circ a = a \circ i$ for all $a \in S$
i.e for '+' : i=0, for 'x' i=1

Invertibility:
For each element $a \in S$, the inverse $a^{-1} \in S$ also exist.
$a \circ a^{-1} = a^{-1} \circ a = i$, where $i \in S$, i is identity element wrt $\circ$
+ operator satisfy invertibility where $a^{-1} = -a$
x operator doesn't satisfy invertibility as there isn't $a^{-1} \in S$ for most a.

Group: $G = (S, \circ)$ is a set of elements, S, and an operation $\circ$ which combines 2 elements of S.
Properties: **closed**, **associative**, have **identity element** i, has **inverse element a^{-1}** for all $a \in S$.
If $G=(S,\circ)$ is commutative if $a \circ b = b \circ a$ for all $a,b \in S$
e.g group of 3x3 matrix and matrix multiply operator is not commutative.

Order of finite group:
$|G|$ is the number of elements in group G
i.e: for $G = (\mathbb{Z}_m, +)$, $|G| = |\mathbb{Z}_m| = m$

Order of element, a, in finite group:
ord(a) is the smallest +ve integer k s.t
$a^k = a \circ a \circ a \ldots a = i$
i.e order of 1 in $G=(\{0,1,2\},+)$ is 3 since $1+1+1 \equiv 0 \bmod 3$
order of element 1 in $G=(\mathbb{Z}_m,+)$ is m

Distributivity: for 2 different operators * and $\circ$
if $a*(b \circ c) = (a*b) \circ (a*c)$ for all $a,b,c \in S$
i.e x is distributive over +, + not distributive over x

Ring: Is a group $(S,\circ,*)$ with 2 operators s.t:
is closed, * is associative, there exist $i' \in S$ identity element, s.t $i'*a=a*i'$ for all $a \in S$, * is distributive over $\circ$

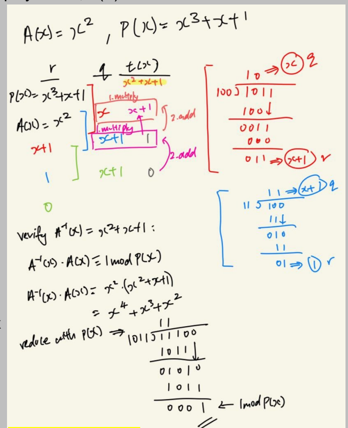
Integer Rings: $(\mathbb{Z}_m,+,\bullet)$, $\mathbb{Z}_m = \{0,1,\ldots m\}$ finite set.
$a+b \equiv c \bmod m$, $c \in \mathbb{Z}_m$ & $a \bullet b \equiv d \bmod m$, $d \in \mathbb{Z}_m$
Integer rings **closed**, + and x are **associative, distributive**.
Identity element 0 for +, for all **a** in ring there is **-a**, where $a+(-a) \equiv 0 \bmod m$; additive inverse always exists.
Identity element 1 for •, for all a in ring $a \bullet i \equiv a \bmod m$
multiplicative inverse a^{-1} exist for **some** elements.
if a^{-1} exists, $b/a \equiv b \bullet a^{-1} \bmod m$, $b \in \mathbb{Z}_m$. multiplicative inverse only exist if gcd(a,m)=1 i.e a & m are coprime.

Field: $F=(S,+,\bullet)$ has properties:
For all $a \in S$ form an additive group with operation + and identity element 0.
For all $a \in S$, except 0, form multiplicative group with • and identity element 1.
When the 2 group operations are mixed, operation • is distributive over +.
Field ∊ Rings, not all fields are rings, as stricter requirements for fields.
All non-zero element of field must have a^{-1} and multiplication operation must be commutative for field.

Galois Field: GF(q) Fields with a finite number of element a field of order m only exist if m is prime, i.e $m=p^n$ for n>0 and prime integer p (characteristic of finite field).
This means there are finite fields with x^n element as long as x is prime. There are no fields with 36 elements, since 36 is **not** a prime power.

Prime Field: GF(p) is a Galois field with prime order, i.e. if n=1 GF(p) = GF(q).
2 operations, integer **addition** modulo p and integer **multiplication** modulo p.
If p is prime integer, integer ring $\mathbb{Z}_p$ is GF(P) and is a prime field. All non-zero element in GF(p) **have inverse**, arithmetic in GF(p) is done modulo p.

Extension Field: $GF(p^n)$ where n>1, order is not prime
not all non-zero integers in $GF(p^n)$ have inverse, cannot perform integer addition and multiplication modulo p^n. Need to represent elements as polynomials of **degree n-1** for arithmetic.
+&-: **bitwise** XOR of coeffs (key addition in AES)
x&/: polynomial multiplication and reduction by fixed irreducible polynomial. AxB=C then ans=C/P(X).
inversion: extended Euclidean algorithm with the input polynomial, A(X) and irreducible polynomial, P(X).



Key establishment:
- Provide confidentiality and integrity
- DHKE & RSA establish keys over insecure, but authenticated channel (digital signature and certs), prevent Man in the Middle

2 main groups: key transport | key agreement
transport: 1 side gen & distribute secret key: RSA
agreement: both side gen a secret key tgt: DHKE

Euler's phi: number of integers in set $\mathbb{Z}_m$ that are relatively prime to $m = \phi(m)$, gcd(j,m)=1.
Large m: calc $\phi(m)$ **extremely slow**.
If we know the factorization of m, much faster way to calc $\phi(m)$. critical to RSA

Calculating $\phi(m)$ = $\prod_{j=1}^{n}(p_j^{e_j}-p_j^{e_j-1})$, p_j are distinct primes, e_j are +ve integers.
e.g if $m=240=2^4 \cdot 3 \cdot 5$, $\phi(240)=(2^4-2^3) \bullet (3^1-3^0) \bullet (5^1-5^0)=64$
Finite group $\mathbb{Z}^*_m$ consist of all integers from 1 to m-1 which gcd(j,m)=1, $\mathbb{Z}^*_m$ forms **commutative group** under multiplication modulo m, i =1, **$ord(\mathbb{Z}^*_m)$** = $\phi(m)$

Order of element x in $\mathbb{Z}^*_m$
Smallest k s.t $x^k \bmod m \equiv 1 \bmod m$

Cyclic Groups: a group that contains an element **α** that has maximum order, ord(**α**) = |G|.
α is **generator** if every element in the group, $a=\alpha^k$, some +ve integer k. i.e **α** can make every element
If p=m in $\mathbb{Z}^*_{m=p}$ P is prime integer, $(\mathbb{Z}^*_p,\bullet)$ is commutative finite cyclic group. i.e every prime group is cyclic and has ≥1 generator. $\phi(P) = p-1$

Discrete Logarithm Problem: given a commutative finite cyclic group of $\mathbb{Z}^*_p$ of order p-1, a generator **α**, and another element $\beta \in \mathbb{Z}^*_p$, the DLP is the trying to find an integer x, $1 \le x \le p-1$, s.t $\alpha^x \equiv \beta \bmod p$. i.e trying to find $x \equiv \log_\alpha \beta \bmod p$, x exist as **α** is generator, but hard for large p.

One-way functions in cryptosystems:
1. y=f(x) is computationally easy
2. $x=f^{-1}(y)$ is computationally infeasible

Diff from 1-way in hash, where the inverse **doesn't** mathematically exist, e.g DLP, integer factorization.

DHKE: asymmetric (public-key) cryptoalgorithm modular exponentiation $\mathbb{Z}^*_p$, where p is large prime integer is a one-way function and commutative.
i.e $k=(a^x)^y \equiv (a^y)^x \bmod p$, **k is shared secret key**

setup phase:
1. choose large prime p
2. choose generator $\alpha \in \{2,3,\ldots,p-2\}$
3. publish p and **α** (for Alice and Bob to use)

key exchange: generate joint secret key k_{AB}
1. Alice pick private key, $k_{pr,A}=a \in \{2,3,\ldots,p-2\}$ and computes public key **$k_{pub,A}=A \equiv \alpha^a \bmod p$**, and send public key to Bob
2. Bob pick private key, $k_{pr,B}=b \in \{2,3,\ldots,p-2\}$ and compute public key **$k_{pub,B}=B \equiv \alpha^b \bmod p$**, and send public key to Alice
3. They both compute joint secret key k_{AB}, Alice computes $k_{AB}=(k_{pub,B})^a \bmod p$ and Bob computes $k_{AB}=(k_{pub,A})^b \bmod p$, $k_{AB} \equiv \alpha^{ab} \bmod p$.

Oscar knows **α**,p, k_A and k_b but cannot compute k_{AB}
k_{AB} used by Alice and Bob for symmetric cipher.

Attack DHKE:
1. solve the DLP by finding value a s.t $a \equiv \log_\alpha A \bmod p$
2. Then compute $k_{AB}=B^a \bmod p$
to defeat brute force search to find a, order of $\mathbb{Z}^*_p=2^{80}$
i.e p at least **80 bits long**
other methods: shanks's baby-step giant-step, pollard rho, pohlig-hellman, index-calculus (**req p > 1024 bit**)
asym need sig more bits than sym.

Algorithm Family	Cryptosystems	Security Level (bit) 80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

Square and multiply: works on $x^e \bmod m$, e.g $3^5 \bmod 11$
1. initialize result to x
2. scan bit of e from left to right (excl MSB)
    1. if bit scan is '0': sq curr result modm
    2. if bit scan is '1': sq curr result modm then multiply new result by x, modm
3. return final result

e.g $3^5 \bmod 11$
1. initialize result =3
2. $5_{10}=101_2$, 3 bit, hence 3-1=2 iterations
3. 1^{st} iter: bit ='0', $3^2 \bmod 11=9 \bmod 11$
4. 2^{nd} iter: bit ='1'
    1. $9^2 \bmod 11=81 \bmod 11=4 \bmod 11$
    2. $4 \bullet 3 \bmod 11=12 \bmod 11=1 \bmod 11$
5. $3^5 \bmod 11 \equiv 1 \bmod 11$

Asymmetric cryptography:
public key → encryption | private key → decryption
public key **not** used to encrypt plaintext, but encrypt shared **secret key** used in symmetric cipher,e.g AES.
RSA is used as **key transport** method.

Fermat's Little Theorem: for element in prime field
a is an integer, **p** is prime number, **$a^p \equiv a \bmod p$**
i.e $a \cdot a^{-2} \equiv 1 \bmod p$, a^{p-2} is multiplicative inverse of a

Euler's Theorem: for elements in integer ring
a & m integers, are relatively prime, **$a^{\phi(m)} \equiv 1 \bmod m$**

Egamal: make DHKE a cipher
1. Alice wants to send Bob a message $x \in \mathbb{Z}^*_p$
2. After Alice and Bob calculate **k_{AB}**, Alice can **encrypt** plaintext by **$y \equiv x \bullet k_{AB} \bmod p$**, k_{AB} is multiplicative mask
3. Bob can decrypt by **$x \equiv y \bullet k_{AB}^{-1} \bmod p$**, find k_{AB}^{-1} using Fermet's little theorem

RSA: uses integer factorization problem
Encryption & decryption performed in integer ring $\mathbb{Z}_n$ since $\mathbb{Z}_n$ only contains {0,1,...,n-1}, binary value of both plain x and ciphertext y less than n
Modular exponentiation plays important role.

Key Generation phase:
1. Choose p & q ,2 large prime numbers
2. Compute $n = p \bullet q$, n at least 1024 bit
3. Calculate $\phi(n) = (p-1) \bullet (q-1)$
4. Select public exponent **e**, s.t $\gcd(e,\phi(n))=1$, $1<e<\phi(n)-1$
5. Compute private key **d**, $d=e^{-1}(\bmod \phi(n))$

Encryption/decryption phase:
1. plaintext x and $k_{pub}=(n,e)$, encryption function is $y=e_{k_{pub}}(x) \equiv x^e \bmod n$, where $x,y \in \mathbb{Z}_n$
2. ciphertext y and $k_{priv}=d$, $x=d_{k_{pr}}(y) \equiv y^d \bmod n$

Integer Factorization problem:
Multiplying 2 large prime numbers is easy, but given the product of 2 large prime numbers, it's hard to factorize this product to obtain 2 prime numbers.

Choosing p & q: pick them randomly, then use fermat test or miller-rabin test to check primality
Computing k_{priv}: use Extend Euclidean Algorithm

Computationally intense: Encryption and decryption involve modular exponentiation, RSA 100-1000x slower than symmetric cipher like AES, hence RSA used to encrypt secret key.

Attack RSA: encryption is deterministic, specific pub key for particular plaintext, always mapped to a particular ciphertext. Oscar can transform ciphertext into another ciphertext that result in known transformation of original plaintext.
Protocol attack: exploit weakness of RSA (malleability), thwarted using padding
Mathematical attack: factorizing the modulo, n then calculate private key d, $d \bullet e \equiv 1 \bmod \phi(n)$, as he knows p,q and $\phi(n)=(p-1) \bullet (q-1)$, $d \equiv e^{-1} \bmod \phi(n)$.
factorizing n is hard if n> 1024 bit
Side Channel attack: exploit information about private key leaked through timing behavior or power consumption prevented using dummy operations of power masking

Digital Signatures:
Integrity: Attacker unable to modify the message transmitted without detection.
Authenticity: Message comes from legitimate source.
Non-repudiation: sender provided with proof of delivery, receiver provided proof of sender's identity, no part can deny sending/receiving the message

Message Authentication Certificate (MAC)
Provide integrity & authenticity, **not** non-repudiation
(Recap) MAC uses shared secret key and hash function, compute authentication tag m, send m and x, receiver recomputes m', check if m'==m.

RSA digital Signature protocol:
1. Select RSA parameters, p, q
2. Calculate $n = p \bullet q$, $\phi(n) = (p-1) \bullet (q-1)$
3. Calculate k_{piv}, k_{pub} as per RSA, send k_{pub} to receiver.
4. Sender → sign plaintext x, $s \equiv x^d \bmod n$, send (x, s)
5. Receiver → use k_{pub} to encrypt s, $x' \equiv s^e \bmod n$, check $x' \equiv x \bmod n$

RSA digital Signature protocol:
1. Select RSA parameters, p, q, calculate $n = p \bullet q$, $\phi(n) = (p-1) \bullet (q-1)$
2. Calculate k_{piv}, k_{pub} as per RSA, send k_{pub} to receiver.
3. Sender → sign plaintext x, $S \equiv x^d \bmod n$, send x and S to receiver
4. Receiver → use k_{pub} to encrypt s, $x' \equiv s^e \bmod n$, check $x' \equiv x \bmod n = x$

Proof of correctness: $x' \equiv s^e \equiv (x^d)^e \equiv x \bmod n$
n usually 1024 to 3072 bits. use sq and mul for mod exponentiation.
use short k_{pub} to make signature verification fast, as x is signed once, but verified many times.

Attacking RSA signature:
Mathematical attack: factorizing mod n, defeat by make n>1024 bits
Existential forgery attack:
1. Attacker knows modulus (n) and pub exponent e in k_{pub}
2. Attacker choose some signature s, computes message $x \equiv s^e \bmod n$
3. Attacker sends this fake x and s to receiver
4. Receiver verify signature is valid

Content of x **can't** be controlled, prevent attack by only allowing certain valid formats for x. e.g pad 64 trailing '1' bits.
Prevented by **hashing** the message, then **signing the hash** instead of the message. e.g RSA-PSS.

MITM DHKE:
1. Oscar intercepts $k_{pub,A}$ & $k_{pub,B}$ replace them with his $k_{pub,O}$
2. Alice computes k_{AO}, Bob computes k_{BO}, A & B unaware they are sharing a secret key with O, instead of each other
3. O can act as relay between A & B, if O receive cipher text from A, he can decrypt it using k_{AO}, modify, then re-encrypt with k_{BO}, then send to B.

Authenticated Channel vs Secure Channel:
Authenticated → Both party know the other party is legitimate
Secure → in addition to authenticated, provides **confidentiality,** message **integrity**.
Asymmetric Cryptosystem **don't** need **secure** channel, but **need authenticated** channel for **key distribution**.

Certificate: create auth channel using digital signature, stop MITM
Certificate for $k_{pub,A}$: $Cert_A=[(k_{pub,A},ID_A),sig_{k\,priv}(k_{pub,A},ID_A)]$
ID_A is some identifying info of Alice, e.g ip, signed with k_{priv} as it is mutually trusted **3rd party** that signs cert, not Alice.
Certificate tie identity of user to their own k_{pub}
Certification Authorities (CA):
Provides 3rd party private key to sign certificate, also gens the cert.
Ensures Alice and Bob **don't need** each other's public key, they don't have to send their pub keys over unauthenticated channel.

Certificate Generation:
With user-provided keys:
A computes her own pub-priv key pair, then request CA to sign her $k_{pub,A}$, A send her pub key to CA over auth channel.
With CA-generated keys:
A request CA to gen pub-priv key pair, sign the pub key. this request is sent over authenticated channel.
CA sends priv key to A over secure and authenticated channel.

DHKE with certificates:
1. A & B choose **α** and p, generate their own pub-priv keys
2. A & B ask CA to sign their own pub keys, sign using RSA
3. A & B send their certs to each other (**not** their pub key)
4. A & B verify the certs are authentic using $k_{pub,CA}$
5. They extract their respective public keys from certs, use these pub keys for joint secret key.

To verify the certs, A & B require pub key of CA, transmitted through authenticated channel (need **only once** during setup)
Transfer of Trust: instead of A & B having to trust each other's pub key, they only need to trust pub key of CA. When CA signs their pub keys, they know they can trust those pub keys too.

Public Key Infrastructure (PKI):
Not all users will have pub key of all CA installed in their OS, need PKI for CAs to certify each other, allow A to request for a cert of a CA she doesn't have, this cert will be signed by CA A alr has.

CA chaining:
1. A only has pub key of CA1, B only has pub key of CA2.
2. A need B's pub key for DHKE
3. B send A his cert with pub key signed by CA2
4. A cannot verify the cert as she doesnt have pub key of CA2
5. A req CA2 to send her CA2 pub key
6. CA2 send A its pub key, signed by CA1
7. A verify CA2 cert, use its pub key to verify B's pub key.

X.509 certificates:

Serial number	
Certificate Algorithm -Algorithm -Parameters	the cryptoalgorithm used to sign the cert, e.g RSA
Issuer	
Period of validity -Not before date -Not after date	not certified indefinitely, to limit amount of dmg if priv key is compromised
Subject	
Subject's Public Key: -Algorithm -Parameters -Public Key	this is the pub key protected by the cert. algorithm used in pub key maybe not same as the one used in cert itself.
Signature	

Side Channel Attacks
Any attack based on **extra** information that can be gathered based on the fundamental way a cryptoalgorithm is implemented.
e.g timing information, power consumption, electromagnetic leak, sound.
Skimming attack: magnetic stripe of ATM cards
Magnetic stripe store card details, **copy** the details.
Prevented using embedded microchip performs secure transactions using 3DES & RSA.
Attack smart key systems: **Relay attack**
Attacker can **clone** the smart key, then use antenna relays to extend the range of the key from 5m to >100m, increase likelihood of locating victim's car.
Preventions: distance bounding protocol
when smart key and vehicle are exchanging message, record timing of the message, calculate distance between between key and car.
drawback: timing precise (ns scale), clock of car and key need to be synced
Other side channel attacks on wireless:
Jamming (DOS), radio fingerprinting, Attacks on wifi based localization (RSSI).
Other side channel attacks:
Power monitoring attack, cache-based spectre attack, timing attack, electromagnetic attack, audio side-channel, optical side-channel.
Power monitoring attack:
Attacker monitors power consumption of the smart card using digital oscilloscope, when card is doing some algorithm, can observe the private key based on high/low power to identify each bit.
Prevention: power masking / execute dummy computations. the computation does nothing but looks like some operation in the oscilloscope.
Cached-based attack (Spectre):
Single-cycle CPU→all inst take 1 clock to complete.
Issues with single cycle CPU:
all inst take as long as slowest inst, slow clock speed.
adding instr tat req more logic gates, slower clock spd
all parts of dataparth (regfile, ALU) idle while waiting for instructions to complete.
Pipelined datapath: Split datapath into many smaller stage, each stage do a simple task.
- Reduces logic gates in each stage, faster clock
- Use register to store state of inst btwn stages
- overlap the processing of several instr at 1 time, better utilization of gates.

Solutions to problem with pipelining:
- **Stall** pipeline until result of BEQ is known (slow)
- Do **speculative execution**, guess outcome of BEQ, fetch the instr to execute speculatively.
- After outcome of BEQ is known:
  - if correct, CPU proceeds
  - if wrong, CPU squash all wrong instructions, rewind reg state, fetch correct instruction, then continue

Speculative execution:
Require CPU to guess the outcome of a branch instruction using **branch prediction**
Branch predictor approach:
1. Always take/ always not take branch
2. Branch predictor circuit

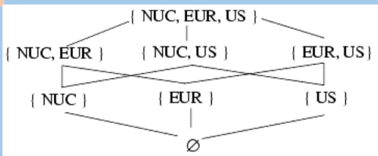
Simple Branch predictor:
2 bit saturating counter- moore FSM with 4 states
input: "branchOutcome" | output: "branchPredict"
input connected to CPU control unit, if actual outcome of BEQ is branch is taken, branchOutcome set to 1, else set to 0
output connected to CPU control unit, if branchPredict set to 1, Control unit fetches instruction at branch target, if branchPredict 0, CU fetch next inst PC+4
Spectre Attack:
1. Prime the branch predictor of the CPU by running code snippet few times with input "x" to "train" predictor to predict the "if" will result to "true".
2. Now **intentionally** input x > array1_size
3. As array1_size not in cache, **cache miss**, CPU need time to retrieve this from mem.
4. Since branch predictor primed to return "true", it **runs the code anyway**, and read value at array1[x], returning value **k**, smwhr in cache
5. since k is in cache, it **returns faster** than array1_size from memory.
6. CPU now **speculatively** request for integer located at array2[k*4096]
7. Since no elements in array2 are cached, only block with array2[k*4096] brought into cache
8. Now, outcome of BEQ determined, CPU realized it was wrong
9. CPU squash all wrong instruction and revert register states, but **cache state is unchanged**
10. Attacker still **doesn't know** value of k
11. Now he sequentially requests data for array2[i*4096], for i =0,1,2... and measure time taken for each mem read. when i=k, **time** taken for the mem read is **much shorter** as k is in cache, now **Attacker knows k.**

CIA triad:
Confidentiality: attacker unable to decipher any secret data being transmitted between legitimate parties.
Integrity: attacker unable to modify data being transmitted between legitimate parties without being detected
Availability: services provided by some party are resilient against interruptions caused by attackers
Security policy: **statement** of what is & is not allowed
A type of system requirement, or refinement of more abstract properties.
Security mechanism: **method, tool or procedure** to enforce a security policy. 3 main classes:
Prevention, Detection, Recovery
Security model: model that represents particular set of security policies.
Access Control: used alone or in combination
Discretionary Access control (DAC): individual user sets access control mechanism to allow or deny access to an object, blog owner decide who can r&w.
Mandatory Access control (MAC): system mechanism controls access to object and the individual cannot alter that access, police can intercept suspect phone

Type of security policy:
Military: primarily provide confidentiality, secret
Commercial: primarily provide integrity, tamper proof
Confidentiality: dealing only with confidentiality
Integrity: dealing only with integrity
Security policy language: express security policies (policy constraints) in precise way
High-level: constraints expressed in abstract manner, **ignores** implementation issues
ISO27000 Definition of confidentiality: The property, that information is not made available or disclosed to unauthorized individuals, entities or processes.
X is set of all entities, I is some information. I is confidential wrt X if no x∈X can obtain info from I.
Program (low-level policy language):
Program: a function with multiple inputs and 1 output
R is set of outputs that are non-erroneous
E is set of outputs that indicate errors
eg. mechanism ask for user & pw. input: user, pw
R (non-erronous output): Success/Fial
if user is in illegal format, output error
Every legal input to m produce either same result as p or an error message.
Confidentiality policy: prevent unauthorized disclosure of information.
c(i,j,k) = (i,j) is a policy that indicates i and j can be disclosed but k is confidential
Bell-LaPadula model:
specifies confidentiality policies
military style classification
significant influence in computer security
Step 1: security level (highest to lowest)
Top Secret, Secret, Confidential, Unclassified
Subject (s) has security clearance L(s)
Object (o) has security classification L(o)

Security Level	**Subject** (security clearance)	**Object** (security classification)
Top Secret (TS)	Tamara, Thomas	Personnel Files
Secret (S)	Sally, Samuel	Email Files
Confidential (C)	Claire, Clarence	Activity Logs
Unclassified (U)	Ursula, Ulaley	Telephone Lists

Bell-LaPadula model – reading information:
Information flows up, not down, "no reads up"
Subject s can read object o if **L(o) ≤ L(s).**
Bell-LaPadula model – writing information:
"Writes up" allowed, "Writes down" not allowed
Subject s can write object o if **L(S) ≤ L(o).**
This ensures no information of higher sensitivity is maliciously/ accidentally leaked to lower levels
Step 2: extension to categories
Security level: (clearance, category set)
e.g (Top Secret, {NUC,EUR,US})



(L,C) **dom** (L',C') if and only if L'≤ L **and** C' ⊆ C
Reading information: "no reads up"
Subject s can read object o if L(s) **dom** L(o)
Write information: "no write down"
Subject s can write object o if L(o) **dom** L(s)
Integrity policy: prevents unauthed mod of data influenced by commercial requirements:
1. Users use existing production programs and db
2. Programmers will dev and test programs on non-production system
3. Special process to install program from dev system to production system, controlled and audited
4. auditor and manager have access to both system state and logs that are generated.

Integrity Model:
Set S of subjects, O of object, I of integrity levels
relation ≤ ⊆ I x I hold when 2nd integrity level **dom** 1st
min: I x I → I gives lesser of the 2 integrity levels
i: S U O → I returns integrity level of subject or object
r ⊆ S x O defines ability of subject to read object
w ⊆ S x O defines ability of subject to write object
x ⊆ S x S defines ability of subject to invoke (or **execute**) another subject.
Integrity level:
Higher integrity level → more confidence that a program will execute correctly, data is accurate/ reliable
Integrity level vs Security level:
Security level: limit flow of information
Integrity level: restrict modification of information
Use trustworthiness to measure integrity level:
High trust, Medium trust, Low trust
Low-Water-Mark policy:
1. if s ∈ S reads o ∈ O, then i'(s) = min(i(s),i(o)), where i'(s) is the subject's integrity level after read, prevent less trusted object from contaminating subject
2. s ∈ S can write to o ∈ O if and only if i(o)≤i(s), prevent subject from writing to more trusted object
3. s_1∈S can execute s_2∈S if and only if i(s_2)≤i(s_1) prevent less trusted invoker from controlling execution of the invoked subject, corrupt the invoked subject even though its more trustworthy
Problem with Low-Water-Mark:
s integrity lowered when s reads from low-integrity o
As time pass, no s will be able to access high integrity lvl
This policy prevents indirect modification.
Ring policy: keep subject's integrity level static
1. Any s ∈ S can read any o ∈ O
2. s ∈ S can write to o ∈ O if and only if i(o)≤i(s)
3. s_1 ∈ S can execute s_2 if and only if i(s_2)≤i(s_1)
Eliminates indirect modification problem.
Strict integrity model (Biba model): opp to BLP
1. s ∈ S can read o ∈ O if and only if i(s)≤i(o), no reads down
2. s ∈ S can write to o ∈ O if and only if i(o)≤i(s), no write up
3. s_1 ∈ S can execute s_2 if and only if i(s_2)≤i(s_1)
BLP for commercial purpose:
2 security clearances: (higher to lower)
- AM (audit mnger): system audit, management functions
- SL (system Low): any process can read at this level
5 security categories:
- D (development): production program in dev not in use
- PC (Production Code): production processes and prog
- PD (Production Data): data covered by integrity policy
- SD (System Development): system programs in dev but not yet in use
- T (Software Tools): progs in production system not related to protected data
BLP principles:
1. Ordinary users can execute & read production code but cannot alter it
2. Ordinary users can alter and read production data
3. Sys managers require access to all logs buy cannot change the security level of objects
4. Sys controllers require ability to install code, these users will be given downgrade capabilities (ability to move object from dev to production)
5. Logs can only be append to (not overwritten), logs must dom any subject that is writing to them

Subject	Security level
ordinary users	(SL, {PC,PD})
Application dev	(SL, {D,T})
System programmer	(SL,{SD,T})
System manager & auditor	(AM, {D,PC,PD,SD,T})
System controller	(SL, {D,PC,PD,SD,T}) + downgrade priviledge

BLP object & security level:

Object	Security level
Dev Code/ Test data	(SL, {D,T})
Production code	(SL, {PC})
Production Data	(SL,{PC,PD})
Software tools	(SL,{T})
System programs	(SL, ∅)
System prog under mod	(SL, {SD,T})
System & app logs	(AM, {appropriate category})

Problem with BLP:
- Too inflexible, sys manager cannot run prog for repairing inconsistent or erroneous prod db, sys man have AM clearance, but production data has SL security lvl, sys man cannot write production data
- complicated security categories, 5 is too many

Lipner's model: combine BLP and Biba
1. implement BLP component first to control reading
2. implement Biba component to control writing
3. simplify security categories in BLP

Simplified BLP:
Reduce categories to 3:
- SP (production): production code and data (combine previous 'PC' and 'PD' categories)
- SD (development): same as previous 'D' category
- SSD (System development): same as previous 'SD'

Retain 2 security clearance level: AM and SL
Adding integrity level to BLP security model:
3 integrity classification (highest to lowest):
- ISP (system prog): for sys programs
- IO (operational): for production programs & dev software
- ISL (System Low): users get this classification on login

2 integrity categories:
- ID (development): development entities
- IP (production): production entities

No reads up (security) + **No writes up** (integrity) enforced
Simplified BLP subject and level:

Subject	**Security Level (simplified)** <controls reading>	**Integrity Level** <controls writing>
Ordinary users	(SL, {SP})	(ISL, {IP})
Application developers	(SL, {SD})	(ISL, {ID})
System programmers	(SL, {SSD})	(ISL, {ID})
System managers and auditors	(**AM**, {SP, SD, SSD})	(**ISL**, {IP, ID})
System controllers	(**SL**, {SP, SD, SSD}) and downgrade privilege	(**ISP**, {IP, ID})

Simplified BLP Object and level:

Object	**Security Level (simplified)** <controls reading>	**Integrity Level** <controls writing>
Development code / test data	(SL, {SD})	(ISL, {ID})
Production code	(SL, {SP})	(**IO**, {IP})
Production data	(SL, {SP})	(ISL, {IP})
Software tools	(SL, Ø)	(**IO**, {ID})
System programs	(SL, Ø)	(**ISP**, {IP, ID})
System programs under modification	(SL, {SSD})	(ISL, {ID})
System and application logs	(**AM**, {appropriate categories})	(ISL, Ø)

	Development Code / Test Data	Production Code	Production Data	Software Tools	System Programs	System Programs in modification	System and App Logs
Ordinary User	-	R	R, W	R	R	-	W
App Developer	R, W	-	-	R	R	W	W
System Programmer	W	-	-	R	R	R, W	W
System Manager	R, W	R	R, W	R	R	R, W	R, W
System Controller	R, W	R, W	R, W	R, W	R, W	R, W	W

Step 1. check for clearance lvl:
read: AM > SL, check for write: ISP>IO>ISL
Step 2. check for category:
check if object categories are subset of subject categories

Information flow models:

access control constrains the rights of the user but can't constraint the flow of information about the system.
Information flow models and information flow policies abstract the essence of security policies.
Information flow policy: Defines how info moves through system.
Prevent info from flowing to user who's **not authed** (confidentiality)
info to process that are **not more trustworthy** than the data (integrity)
Basic idea: system is secure if groups of subjects cannot interfere with one another.
System is **not** secure if it can **enter** an unauthorized state from authorized state.

Information flow model:
consist of: Subjects: $S=\{s_i\}$, States: $\sum = \{\sigma_i\}$, Outputs: $O=\{o_i\}$,
Commands: $Z = \{z_i\}$, State transition commands $C = S \times Z$
inputs select the commands to execute, or inputs themselves encoded in set of state transitions.
State transition function T: $C \times \sum \rightarrow \sum$
Executing state transition command c when in state $\sigma_0 \rightarrow$ state σ_1
Let C* be set of possible sequences of state transition commands in C, then function $T^*: C^* \times \sum \rightarrow \sum$ and $c^* = c_0,...,c_n$
$\mathbf{T^*(c^*,\sigma_i) = T(c_n,T(c_{n-1},...,T(c_0,\sigma_i)...))=\sigma_{i+n+1}}$
Output function P: $C \times \sum \rightarrow O$
Output of the FSM when execute state transition command c in state σ
Let O* be set of possible sequences of outputs after executing some sequence $c^* \in C^*$, then $P^*: C^* \times \sum \rightarrow O^*$ and $c^* = c_0,...,c_n$
$\mathbf{P^*(c^*,\sigma_i)=o_{i+1},o_{i+2},...,o_{i+n},o_{i+n+1}}$
Projection: $proj(s,c^*,\sigma_i)$ is the sequence of outputs in $P^*(c^*,\sigma_i)$ that subject s is **authorized** to see, i.e proj function removes outputs that s is not supposed to see.
Purge: $\pi_{i,j}(c^*)$: subseq c* with all elements (s,z), $s \in i$ and $z \in j$ deleted
deleted means element is removed from observable sequence.
Non-interference: set of output Lucy observe corresponds to the set of commands that she can observe. $proj(s,c^*,\sigma_i) = proj(s,\pi_{G,A}(c^*),\sigma_i)$
System is secure wrt subjects in G' & commands in A, if any only if **every subject in G** is non-interfering with **all subjects in G'.**
G' cannot deduce info about subject in G, but G can deduce info about subject in G'.
Example: $\sigma_0=(0,1)$, c* = (Holly, xor0), (Lucy, xor1), (Holly, xor1),
$P^*(c^*,\sigma_0)$ = 01,10,01
let G ={Holly}, G' = {Lucy}, A=Z={xor0,xor1}
Purge Holly: $\pi_{Holly,A}(c^*)$=(Lucy,xor1), $proj(Lucy,c^*,\sigma_0)$=1,0,1
$proj(Lucy,\pi_{Holly,A}(c^*),\sigma_0)=proj(Lucy,(Lucy,xor1),\sigma_0)=0$
$proj(Lucy,c^*,\sigma_0) \neq proj(Lucy,\pi_{Holly,A}(c^*),\sigma_0)$
{xor0,xor1}, {Holly} :| {Lucy} is false, Holly **interfering** with Lucy as commands to change H bit changes the L bit also.
Even though Lucy **should not be aware** of Holly's existence, Lucy can tell that there is **some other user**, bases solely on the observation of her own output sequence.

Probability concepts:

X is a discrete random variable, has some probability of taking one of the values $x_1,x_2,...,x_n$.
$\sum^n_{i=1}P(X=x_i)=1$, sum of probability of all outcome =1
$E(X) = \sum^n_{i=1}P(X=x_i)\cdot x_i$,
Information of a particular outcome: $I_X(x_i)$ or $I(X=x_i)$
Measure amount of **info received in bits**, when outcome of X is x
$I(X=x_i) = \log_2(1 / (P(X=x_i)))$
Amount of info received by learning the outcome of X is x_i is inversely proportional to the probability of x_i occurring.
Example: if P(X=1) =0.5, P(X=0) =0.5, $I(X=1)=\log_2(1/0.5)=\log_2 2=1$
Gain 1 bit of information by learning the outcome of X is 1
Entropy of discrete random variable: H(X)
Measures the uncertainty of X, in bits.
H(X) is the expectation of information of X
$H(X) = \sum^n_{i=1}P(X=x_i)\cdot I(X=x_i) = \sum^n_{i=1}P(X=x_i)\cdot\log_2(1/P(X=x_i))$
$H(X) = -\sum^n_{i=1}P(X=x_i)\cdot\log_2 P(X=x_i)$
Example: $H(X) = -(0.5)\cdot(\log_2 0.5)-0.5\cdot\log_2 0.5=1$
Entropy of variable is inversely proportional to its predictability, lower entropy → more predictable
H(X) = 0, completely predictable
H(X) = ∞, completely unpredictable
if H(X') < H(X), X' more predictable than X
we know more about X' than X

Conditional Entropy:

Conditional entropy of X given $Y=y_i$, outcome of Y is known
$H(X \mid Y=y_i) = -\sum^n_{i=1}P(X=x_i \mid Y=y_i)\cdot\log_2 P(X=x_i \mid Y=y_i)$
Conditional entropy of X given Y, outcome of Y is unknown
$H(X \mid Y) = -\sum^m_{i=1}P(Y=y_i)\cdot\sum^n_{i=1}P(X=x_i \mid Y=y_i)\cdot\log_2 P(X=x_i \mid Y=y_i)$

Entropy and information flow:

Information flows X → Y if execution of sequence of commands c* cause information initially in X to affect information in Y.
c* are set of commands to change from state a to state b
if $H(X_a \mid Y_b) < H(X_a \mid Y_a)$, flow of info form X to Y.
$H(X_a \mid Y_b)$ more predictable now as more info in Y_b that came from X
if Y_b doesn't exist in state a
if $H(X_a \mid Y_b) < H(X_a)$, flow of info from X to Y
important note: $\log_2 0=0$
Baye's theorem: $P(Y \mid X) = P(X \mid Y)\cdot P(Y)/P(X)$
Implicit flow of information: flows of information from X to Y **without** explicit assignment of **y=f(x)**, e.g if x ==1 then y =0, else y=1
Notation for security class:
X: security class of X, defined in Bell-LaPadula based system
X ≤ Y: info allowed to flow from element in security class of X to ~ in Y
Compiler-based mechanism:
Detects and **block** unauthorized flow of info in a program during compilation, wrt given security policy. Analysis conducted by mechanism is not **precise**, but **secure**.
Not precise: path of info flow marked unauthorized, but should be authorized, i.e it is false positive
Secure: No unauthorized path of info flow will remain undetected
Certified: Set of statements is certified with respect to an information flow security policy if the information flow **within that set** of statements **does not violate the security policy**.
Example: if x ==1 then y := m, else y := n.
There is info flow from X and m to y, info flow from X and n to y.
If the security policy states: X≤Y and M≤Y and N≤Y, certified
Array:
... := m[i]; # nformation flowing out
i and M[i] affect the var assigned, security class of array is max(I,M[i])
m[i] := ... #information flowing in
only variable m[i] is affected, security class for the array is m[i].
Assignment statement:
x := y+z
Info flow from Y & Z to X, certified if security policy states max(Y,Z) ≤ X
G: statement y:= $f(x_1,...,x_n)$, security policy must state $\mathbf{max(X_1,...,X_n) \leq Y}$
Compound statement:
statement 1: x := y+z;
statement 1 certified if sec policy states max(Y,Z)≤X
statement 2: m := n*o-x;
statement 2 certified if security policy states max(N,O,X)≤M
entire code to be certified, sec policy is max(Y,Z)≤X and max(N,O,X)≤M
G: <s_1>;...<s_n>; certify each statement with security policy
Conditional statement:
if x + y < z then m := n, else p := n * o – x;
security policy: N≤M and max(N,O,X)≤P
info about x,y,z revealed in conditional step, since they are part of the condition, security policy also must have max(X,Y,Z) ≤ min(M,P)
G: if $f(x_1,...,x_n)$ then <s_1> else <s_2>, certify <s_1> & <s_2> and $max(X_1,...,X_n) \leq min(Y|Y$ is target of assignment in <s_1>, <s_2>)
Iterative statement:
while $f(x_1,...,x_n)$ do
    <s_1>
    end
G: check loop terminates eventually, certify <s_1>,
$max(X_1,...,X_n) \leq min(Y \mid Y$ is target of assignment in <s_1>)
Infinite loops:
Compiler-based mech cant detect if loop will terminate at compile time.
Use execution-based mech (dynamic), check info flow at run time
Execution based mechanism: dynamic in nature
Stops any info flow that violate security policy.
Before y := $f(x_1,...,x_n)$; executed, execution-based mech verify $max(X_1,...,X_n) \leq Y$. Block if verification fails.
Easily checks for **explicit** flow of information, **hard** to check implicit flow

1. (Certificate authority) Which of the following is **true**, with regards to a certificate authority (CA)?
   a. The CA **always** generates a public and private key pair on behalf of the user
   b. The CA signs the user's **private** key, using its own **public** key and a digital signature protocol
   c. The CA generates the user's **private** key, using its own **public** key and a key establishment protocol
   d. The CA signs the user's **public** key, using its own **private** key and a digital signature protocol

2. (Digital signatures) Which key is used to **sign** the plaintext message in a digital signature scheme?
   decrypt
   a. The sender's **public** key
   b. The sender's **private** key
   c. The receiver's **public** key
   d. The receiver's **private** key

3. (Symmetric ciphers) Which of the following is **not** a **symmetric** key algorithm?
   a. AES
   b. RSA
   c. OTP
   d. DES

4. (Modular arithmetic) Which one of the following statements is **true**?
   a. $\mathbb{Z}^*_{10}$ is not a group
   b. The order of $\mathbb{Z}^*_{13}$ is 13
   c. $\mathbb{Z}^*_{7}$ is not a cyclic group
   d. $\mathbb{Z}^*_{19}$ contains an element that is a generator
   every prime group has generator
   prime

(Simplified Bell-LaPadula model) Use the information and tables below for **questions 5 and 6**.

- Security clearances: **SL** (lower), **AM** (higher)
- Integrity clearances: **ISL** (lowest), **IO** (middle), **ISP** (highest)
- **SP, SD** and **SSD** are security categories
- **IP** and **ID** are integrity categories

Subject	Security Level (simplified)	Integrity Level
Ordinary users	(SL, {SP})	(ISL, {IP})
Application developers	(SL, {SD})	(ISL, {ID})
System managers	(AM, {SP, SD, SSD})	(ISL, {IP, ID})

Read | write

Object	Security Level (simplified)	Integrity Level
Development code / test data	(SL, {SD})	(ISL, {ID})
Production code	(SL, {SP})	(IO, {IP})
Production data	(SL, {SP})	(ISL, {IP})
System programs	(SL, Ø)	(ISP, {IP, ID})
System programs under modification	(SL, {SSD})	(ISL, {ID})

5. (Simplified Bell-LaPadula model) Which of the following statements is **true** for **system managers**?
   a. They have both read and write access to system programs → R only
   b. They have only write access to production data → R & W
   c. They have only read access to system programs → R only
   d. They have only read access to development code → R & W

6. (Simplified Bell-LaPadula model) Which of the following statements is **true** for **production code**?
   a. They can be written by system managers
   b. They can be written by ordinary users
   c. They can be read by system managers
   d. They can be read by application developers

Compute 7^{25} *mod* 52 using the square and multiply algorithm. Show your work.

$x = 7$

$25_{10} = 11001_2 \rightarrow$ 4 iters

1^{st} iter $\rightarrow$ bit = '1'

$7^2 \bmod 52 \times x$

$= 49 \bmod 52 \times 7$

$= 31 \bmod 52$

2^{nd} iter $\rightarrow$ bit = '0'

$31^2 \bmod 52$

$= 961 \bmod 52$

$= 25 \bmod 52$

3^{rd} iter $\rightarrow$ bit '0'

$25^2 \bmod 52$

$= 625 \bmod 52$

$= 1 \bmod 52$

4^{th} iter $\rightarrow$ bit '1'

$1^2 \bmod 52 \times 7$

$= 7 \bmod 52$

$7^{25} \bmod 52 \equiv 7 \bmod 52$

Find the order of the following elements, given their respective groups.

find smallest 5^k st $5^k \bmod 8 \equiv 1 \bmod 8$

i. Element: 5
Group: $\mathbb{Z}^*_8$

$5^1 \bmod 8 = 5$

$5^2 \bmod 8 = 25 \bmod 8 = 1 \bmod 8$

order of 5 in $\mathbb{Z}^*_8 = 2$.

Element: 9
Group: $\mathbb{Z}^*_{11}$

$9^1 \bmod 11 = 9 \bmod 11$

$9^2 \bmod 11 = 81 \bmod 11$

$= \boxed{4 \bmod 11}$

$9^3 \bmod 11 = \boxed{4 \bmod 11} \times 9$

$= 36 \bmod 11 = \boxed{3 \bmod 11}$

$9^4 \bmod 11 = \boxed{3 \bmod 11} \times 9$

$= 27 \bmod 11 = \boxed{5 \bmod 11}$

$9^5 \bmod 11 = \boxed{5 \bmod 11} \times 9$

$= 45 \bmod 11 = 1 \bmod 11$

(Information flow and entropy) Suppose we have the following code segment:

```
x = w - z
k = z + y[i]
```

From our lectures, we know that with respect to the **first line** of the code segment, there is information flow **from** the variables **w and z to** the variable **x**.

a) Let *I*, *K*, *W*, *X*, *Y[I]* and *Z* be **random variables** representing the variables **i**, **k**, **w**, **x**, **y[i]** and **z** respectively in the code segment above.
Also, let *I*, *K*, *W*, *X*, *Y[I]* and *Z* represent the **security classes** of *I*, *K*, *W*, *X*, *Y[I]* and *Z* respectively.

Write an expression (in terms of *I*, *K*, *W*, *X*, *Y[I]* and *Z*) that must be stated in the **security policy** for a **compiler-based mechanism**, in order for the above code segment to be **certified**.

$\max(W, Z) \le X$, $\max(Z, I, Y[I]) \le K$

or $W \le X$, $Z \le X$, $Z \le K$, $I \le K$, $Y[I] \le K$.

In the subsequent parts of this question, we will focus only on the **first line** of the above code segment. We will show that there is information flow from the variable **w** to the variable **x**.

For the rest of this question, let *X*, *W* and *Z* be **discrete random variables** representing the variables **x**, **w** and **z** respectively in the code segment above. Assume that **state *a*** represents the state **before** the above code segment is executed, while **state *b*** represents the state **after** the above code segment is executed.

Note the following important points:

- W_a is distributed between the set of **integer** values {2, 3}, with the following probabilities:

$P(W_a = 2)$	$P(W_a = 3)$
$\frac{2}{3}$	$\frac{1}{3}$

$\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{3}$

- Z_a is distributed **equally** between the set of **integer** values {0, 1, 2}
- *X* does **not** exist in state *a*. This means that X_a does **not** exist.

b) Based on the above code segment, *X* can be one of four integer values in state *b*: 0, 1, 2 or 3. Calculate the probabilities of X_b and fill in the table below. Leave your answers as fractions or as numbers rounded to 3 decimal places. Show your work.

$x = w - z$

$w = 2 : \frac{2}{3}$ $z = 0 : \frac{1}{3}$

$w = 3 : \frac{1}{3}$ $z = 1 : \frac{1}{3}$

$z = 2 : \frac{1}{3}$

2-2

2-1 $\frac{2}{9}$ 2-0 : $\frac{2}{9}$ 3-0 : $\frac{1}{9}$

3-2 $\frac{1}{9}$ 3-1 : $\frac{1}{9}$

$P(X_b = 0)$	$P(X_b = 1)$	$P(X_b = 2)$	$P(X_b = 3)$
$\frac{2}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{9}$

$P(w=2) = \frac{2}{3}$ $P(w=3) = \frac{1}{3}$

c) Calculate the value of $H(W_a)$, i.e. the entropy of W_a. Leave your answer as a number rounded to 3 decimal places. Show your work

$H(W_a) = -\sum_{i=1}^{2} P(W_a = w_i) \cdot \log_2 P(W_a = w_i)$

$= -\left[P(w=2) \cdot \log_2 P(w=2) + P(w=3) \cdot \log_2 P(w=3)\right]$

$= -\left[\frac{2}{3} \cdot \log_2\left(\frac{2}{3}\right) + \frac{1}{3} \log\left(\frac{1}{3}\right)\right]$

$= 0.918$

$w = 2 \Rightarrow \frac{2}{3}$ $x = 0 \Rightarrow \frac{2}{9}$ $z = 0 : \frac{1}{3}$

$w = 3 \Rightarrow \frac{1}{3}$ $x = 1 \Rightarrow \frac{1}{3}$ $z = 1 : \frac{1}{3}$

$x = 2 \Rightarrow \frac{1}{3}$ $z = 2 : \frac{1}{3}$

$x = 3 \Rightarrow \frac{1}{9}$

d) Calculate the various **conditional** probabilities of W_a with respect to X_b and fill in the table below. Leave your answers as fractions or as numbers rounded to 3 decimal places. Show your work.

Hint: use Baye's Theorem.

$$P(W_a = 2 \mid X_b = 0) = \frac{P(X_b = 0 \mid W_a = 2) \cdot P(W_a = 2)}{P(X_b = 0)} = \frac{\frac{1}{3} \cdot \frac{2}{3}}{\frac{2}{9}} = 1$$

$$P(W_a = 3 \mid X_b = 0) = \frac{P(X_b = 0 \mid W_a = 3) \cdot P(W_a = 3)}{P(X_b = 0)} = \frac{0 \cdot \frac{1}{3}}{\frac{2}{9}} = 0$$

$$P(W_a = 2 \mid X_b = 1) = \frac{P(X_b = 1 \mid W_a = 2) \cdot P(W_a = 2)}{P(X_b = 1)} = \frac{\frac{1}{3} \cdot \frac{2}{3}}{\frac{1}{3}} = \frac{2}{3}$$

$$P(W_a = 3 \mid X_b = 1) = \frac{P(X_b = 1 \mid W_a = 3) \cdot P(W_a = 3)}{P(X_b = 1)} = \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{3}} = \frac{1}{3}$$

$$P(W_a = 2 \mid X_b = 2) = \frac{P(X_b = 2 \mid W_a = 2) \cdot P(W_a = 2)}{P(X_b = 2)} = \frac{\frac{1}{3} \cdot \frac{2}{3}}{\frac{1}{3}} = \frac{2}{3}$$

$$P(W_a = 3 \mid X_b = 2) = \frac{P(X_b = 2 \mid W_a = 3) \cdot P(W_a = 3)}{P(X_b = 2)} = \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{3}} = \frac{1}{3}$$

$$P(W_a = 2 \mid X_b = 3) = \frac{P(X_b = 3 \mid W_a = 2) \cdot P(W_a = 2)}{P(X_b = 3)} = \frac{0 \cdot \frac{2}{3}}{\frac{1}{9}} = 0$$

$$P(W_a = 3 \mid X_b = 3) = \frac{P(X_b = 3 \mid W_a = 3) \cdot P(W_a = 3)}{P(X_b = 3)} = \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{9}} = 1$$

$P(W_a = 2 \mid X_b = 0)$	$P(W_a = 2 \mid X_b = 1)$	$P(W_a = 2 \mid X_b = 2)$	$P(W_a = 2 \mid X_b = 3)$
1	$\frac{2}{3}$	$\frac{2}{3}$	0
$P(W_a = 3 \mid X_b = 0)$	$P(W_a = 3 \mid X_b = 1)$	$P(W_a = 3 \mid X_b = 2)$	$P(W_a = 3 \mid X_b = 3)$
0	$\frac{1}{3}$	$\frac{1}{3}$	1

Calculate the value of $H(W_a \mid X_b)$, i.e. the **conditional** entropy of W_a given X_b. Leave your answer as a number rounded to 3 decimal places. **Verify** that information flows from *W* to *X*, by comparing the value of $H(W_a \mid X_b)$ with the value of $H(W_a)$ you obtained in part c). Show your work.

$$H(Y_a \mid X_b) = -\sum_{j=1}^{4} P(X_b = x_j) \cdot \sum_{i=1}^{2} P(W_a = w_i \mid X_b = x_j) \cdot \log_2 P(W_a = w_i \mid X_b = x_j)$$

Thus, $H(W_a \mid X_b) =$

$-P(X_b = 0) \cdot [P(W_a = 2 \mid X_b = 0) \cdot \log_2 P(W_a = 2 \mid X_b = 0) + P(Y = W_a = 3 \mid X_b = 0) \cdot \log_2 P(W_a = 3 \mid X_b = 0)]$

$-P(X_b = 1) \cdot [P(W_a = 2 \mid X_b = 1) \cdot \log_2 P(W_a = 2 \mid X_b = 1) + P(Y = W_a = 3 \mid X_b = 1) \cdot \log_2 P(W_a = 3 \mid X_b = 1)]$

$-P(X_b = 2) \cdot [P(W_a = 2 \mid X_b = 2) \cdot \log_2 P(W_a = 2 \mid X_b = 2) + P(Y = W_a = 3 \mid X_b = 2) \cdot \log_2 P(W_a = 3 \mid X_b = 2)]$

$-P(X_b = 3) \cdot [P(W_a = 2 \mid X_b = 3) \cdot \log_2 P(W_a = 2 \mid X_b = 3) + P(Y = W_a = 3 \mid X_b = 3) \cdot \log_2 P(W_a = 3 \mid X_b = 3)]$

$$= -\frac{2}{9}[1 \cdot \log_2 1 + 0 \cdot \log_2 0] - \frac{1}{3}\left[\frac{2}{3} \cdot \log_2 \frac{2}{3} + \frac{1}{3} \cdot \log_2 \frac{1}{3}\right] - \frac{1}{3}\left[\frac{2}{3} \cdot \log_2 \frac{2}{3} + \frac{1}{3} \cdot \log_2 \frac{1}{3}\right] - \frac{1}{9}[0 \cdot \log_2 0 + 1 \cdot \log_2 1]$$

$$= 0.612$$

Since $H(W_a) = 0.918$, we have $H(W_a \mid X_b) < H(W_a)$ and so information has flowed from W to X.