# Terminology and Principles:

* Cryptography: Science of making code
* Cryptanalysis: science of breaking codes
* Cryptology = Cryptography + Cryptanalysis
* Plaintext: Regular Text
* Ciphertext: Output of cryptosystem
* Kerckhoffs’ Principle: Cryptosystem known to attacker but key isn’t
* Cipher/Cryptosystem: Something used to encrypt plaintext into ciphertext
* Key: A parameter for a cryptosystem
* Symmetric systems: Systems in which Encrypt Key = Decrypt Key
* Confusion: relationship between plain and cypher text must be involved (i.e. each cyphertext bit must depend on many parts of key)
* Diffusion: Plaintext statistics should be spread through ciphertext (For same key, changing one bit of plaintext should change ½ of cyphertexts’ bits on average
* Asymmetric systems: Uses a key pair: One key for encryption, another for decryption
* Stream Cipher: Short key is stretched into long keystream and the used like one-time pad.
* Block Cipher: Each key determines different codebook for mapping blocks of plaintext into blocks of cipher text
  + Turn plain text into blocks of size accepted by cipher then pad any partial blocks
* Co-prime: When the greatest common denominator between the two is 1
* Salt: Random bits used in hashing
* SSO (Single sign on): Allows users to connect to different services provided they own account of one service
* Password Manager: Programs which manage many passwords via master password
* Cookie: Piece of data which maintains state across sessions when accessing associated website
* Authentication: The process of checking who one day they are
* Eulers Theorem: If gcd(x, n) = 1 then xΦ(n) = 1 (mod n), where Φ(n) is Euler’s totient function
* Collision: When Mallet finds that M’ != M s.t. h(M’)=h(M)
* One way: given a hash value v, hard to find x s.t. h(x) = v
* Weak collision resistance, given x and h(x), unfeasible to find x!=y s.t. h(y) = h(x)
* Strong collision resistance, unfeasible to find x and y with x!=y s.t. h(x) = h(y)
* Asymmetric password-authenticated key exchanges (asymmetric PAKEs): cryptographic protocol that allows two parties to establish a shared cryptographic key securely, even if only one party knows the password
* Oblivious PAKE (OPAQUE): Protocol based off an oblivious pseudorandom function (OPRF) in which one party does computation of a pseudorandom function (PRF) for another party
* Hardware authenticators: Physical devices which can generate and store keys
* Challenge: A randomly generated number by a server
* Identification: Who is the person
* Cooperative Subjects: Subjects who wishes to be authenticated
* Fraud: Someone is mis-authenticated as someone else
* Insult: Someone is not authenticated as themselves
* Multi-factor authentication (MFA): Combination of different authentication solutions
* Security Protocol: Rules followed in all security applications
* One-way authentication: When one person must prove identity to another
* Mutual Authentication: Both must prove identity to each other
* Session Key: A key that is created for one session only with the purpose of confidentiality or integrity during said session
* Nonce: Number only used once
* Key Distribution Centre(KDC): Server who shares a symmetric key with each user
* Perfect Forward Secrecy(PFS): This is achieved if Mallet is unable to decrypt recorded ciphertexts at a later time even if they get the key eventually.

# Notations & Formulae:

* P = plaintext block
* C = Ciphertext block
* K = Key
* E = Encrypt
* D = Decrypt
* Encrypt: C=E(P,K) = E(D(C,K),K)
* Decrypt: P=D(C,K) =D(E(P, K),K)
* F (.,.): Round Function
* ⊕: XOR
* GCD (n,m): Greatest common denominator between n and m
* {.} involves a public key
* [.] involves a private key
* N = Modulus
* S: Signature
* H(M): hash of message m
* Modulo Inverse of A mod C:
  + Find all values B (which is in the range 0 to C-1) such that A\*B Mod C gives us 1

# Attacks:

* Brute Force Attack: Attempt all possible combinations till something works
* Related Key attack: An attack that exploits relationships between cryptographic keys to deduce information about the cipher. Occurs when keys aren’t completely independent. To prevent this, first 256 bytes of keystream is discarded
* Meet-in the middle attack: If an attacker knows a pair (P,C) they can then computes a table of E(P,K1) for every possible key K1 and does the same for K2 and D(C,K2). If a match between the two tables is found, they will get the keys. Only needs 257 tries as opposed to 2112
* Cube Root Attack: If M3 < N, then C = M3 mod N = M3, i.e. the modulo operation has no effect. So Mallet can simply take the usual cube root of C to find M
* Forward Search Attack: encrypting all possible plaintext messages in Knapsack Problem with the public key to see if a match is found
* Dictionary attack: Given a dictionary of common passwords, precompute a hashed dictionary. Compare hashed password to precomputed hashed dictionary to see if they get access to password, can reuse, no need to re-hash
* Replay Attack: This is when mallet intercepts and re-uses valid data that weas transmitted between two parties
* TCP Authentication Attack: Mallet sends multiple SYN packets to Bob until Mallet can predict bN and then Mallet is able to impersonate Alice and spoof Alice’s IP address in their IP packet.

# Ciphers

Caesar’s Cypher: Shift plain text by n positions (0 to 25). This is the key

* C = (P + n) mod 26
* P = (C – n) mod 26
* Brute search/Exhaustive Key analysis: try all possible 26 possible keys
* Special case of simple substitution

Simple substitution: Key can be any permutation of alphabet (i.e. not necessarily in alphabetical order)

* 26! possible keys
* Exploit statistics of letters (think of how e is the most common letter in words)
* Monoalphabetic: One permutation of alphabet

Vigenère Cipher: Polyalphabetic substitution

* Circular shift by the characters in key word.
* e.g. CAT indicates a shift on the first character by 2, the 2nd character by 0 and the 3rd by 19
* 26t keys where t is keyword length

Double Transposition: Encryption on permutations of plaintext

* Arrange into a m x n matrix, permute rows then permute columns then concatenate rows
* Key is matrix dimension plus column and row permutations

Vernam Cypher/one-time pad cipher:

* Encrypt: Cyphertext= P ⊕ K
* Decrypt: plaintext = C ⊕ K
* Letters in text are converted into binary (based on position in word)
* Key is randomly generated, same size as message, known only to sender and receiver and used only once

A5/1:

* 3 shift registers
  + X : 19 bits (X0🡪x18)
  + Y: 22 bits (Y0🡪Y21)
  + Z: 23 bits (Z0🡪Z22)
  + Key is a 64-bit string, used to initialize 3 registers
  + Bits used like 1-time pad
  + Each iteration:
    - Take majority of x8, y10 and z10 and depending on which is majority perform following:
      * If x8:
        + t= x13 ⊕ x16 ⊕ X17 ⊕ X18   
           xi= xi -1 for i 18🡪1, x0=t
      * If y10:
        + t= Y20 ⊕ Y21

yi= yi -1 for i 21🡪1, y0=t

* + - * If z10:
        + t= Z7 ⊕ Z20 ⊕ Z21 ⊕22   
           zi= zi -1 for i 22🡪1, z0=t
    - Keystream bit is X18 ⊕ Y21 ⊕ Z22 regardless of step or not

RC4:

* Self-modifying lookup table that contains permutation of values 0🡪255
* Key is used to initialise permutation in table
  + Each step:
    - Swaps elements I lookup table so new permutation is chosen
    - Selects keystream byte from table
  + Initialization:
    - S[i] contains permutation of 0🡪255
    - Key[i] contains N bytes of key for i🡪N-1 of byte length N which ranges from 1🡪256
    - Pseudocode: not needed, just know workings
      * For i in range (0,256):   
         s[i]=i   
         K[i]=key[i% N] this is mod n  
        j=0  
        For i in range (0,256):   
         J= (j+ S[i] +k[i]) % 256  
         Swap s[i] and s[j]  
        i=j=0
  + Keystream generation:
    - In each stop, two elements are swapped, and keystream byte is selected:
      * Swap elements
        + i = (i + 1) %256  
          j = (j + S[i]) %256  
          Swap s[i] and s[j]
      * Select keystream byte from table
        + Byte= S[ (S[i]+s[j]) % 256 ]

Feistel Cipher:

* Plaintext/Cyphertext split into left and right halves:
  + Encrypt: For each round 1🡪n:  
    Li = Ri-1   
    Ri = Li-1 ⊕ F (Ri-1 ,Ki)  
    Ciphertext is Ln, Rn
  + Decrypt: For each round 1🡪n:  
    Ri-1 = Li   
    Li-1 = Ri ⊕ F (Ri-1, Ki) = Ri ⊕ F(Li ,Ki)  
    Plaintext is L0,R0
* Confusion and Diffusion are key here, more rounds, the higher the complexity, confusion and diffusion
* Subkeys are used to increase complexity of cipher

Data Encryption Standard (DES)

* Feistel Cipher with:
  + Operates on a 64-bit block length.
  + Uses a 56-bit key length.
  + Consists of 16 rounds.
  + Generates 48-bit subkeys for each round.
* Steps in DES Encryption:
  + Expansion and Permutation:
    - Expands a 32-bit input by repeating 16 random bits, resulting in 48 bits.
  + S-Boxes (Substitution Boxes):
    - DES uses 8 S-Boxes, each mapping 6-bit inputs to 4-bit outputs.
    - For each 6-bit input:
      * The first and last bits are combined to form a 2-bit row index.
      * The middle 4 bits form the column index.
      * The corresponding 4-bit output is determined via lookup tables.
    - S-Boxes introduce non-linearity, crucial for security.
  + P-Box (Permutation Box):
    - Permutes the 32-bit output of the S-Box stage according to a fixed index.
  + Key Schedule and Generation:
    - The initial 56-bit key undergoes a permutation and splits into two 28-bit halves (LK and RK).
    - For subkey generation:
      * In rounds 1, 2, 9, and 16, LK and RK are shifted left by 1 bit.
      * In other rounds, they are shifted left by 2 bits.
    - After shifting, a compression permutation reduces the 56-bit key to a 48-bit subkey.
    - This subkey is used in each round of encryption.
    - Initial and Final Permutations:
      * The plaintext undergoes an initial permutation (IP) before the first round.
      * After the final round, the left and right halves (LK and RK) are swapped.
      * An inverse initial permutation (IP⁻¹) produces the final ciphertext.
  + Security Considerations:
    - Non-linearity: S-Boxes are the primary source of non-linearity, enhancing security.
    - Exhaustive Key Search: The primary attack method involves brute-force due to the 56-bit key size.
  + Triple DES (3DES):
    - To enhance security, Triple DES (3DES) uses two different 56-bit keys, providing an effective key length of 112 bits.
      * C = E(D(E(P, K1), K2), K1)
      * P = D(E(D(C, K1), K2), K1)
      * We don’t do E(E(P,K1),K2) = C because meet-in the middle attack is possible

AES (Advances Encryption Standard)

* Replacement for DES
* Iterated block cipher and not a Feistel Cipher
* Permutation-Substitution Cipher
* 128 bits expressed as 4x4 matrix of bytes
* Key length of either 128,192 or 256 bits
* 10-14 rounds which is a concat of 4 functions belonging to 3 layers
  + ByteSub (non-linear layer)
    - Similar to S-Box; it’s a non-linear composition of 2 math operations or a sub operation
  + ShiftRow (linear mixing layer)
    - Cyclic shift of rows
    - For row 1, we shift the 1st element one, for row 2 we do it twice, and row 3 thrice.
  + MixColumn (linear mixing layer): Implemented like a lookup table
    - Matrix multiplication in which same matrix used for all columns
  + AddRoundKey (key addition layer)
    - XOR elements

General Knapsack

* Knapsack problem/subset sum problem: Given a set of N positive integer weights (W0🡪Wn-1 ) and a target sum S, find coefficients ai ∈ {0, 1} such that S = a0 W0 + a1 W1 + … an-1 Wn-1
* General Knapsack (GK) is hard to solve as it is NP (nondeterministic polynomial) complete
* A super increasing knapsack (SIK) is easy to solve despite a large number of weights
  + Each weight is greater than the sum of previous weights
  + Given SIK value S and starting with j = Number of weights – 1 and ending j < 0:
    - if Wj<=S:  
       then aj = 1 and S 🡨 S - WjElse aj=0 and j 🡨 j-1
  + Given the above 2, we can use them to create a public key cryptosystem by creating an SIK then converting it to a GK via parameters
    - N > Sum of elements in SIK
    - Multiplier m an integer that is coprime with n
    - SIK🡪GK:
      * Multiply all SIK values by m then mod N
    - Public Key: GK and N
    - Private key: SIK and mod inverse
      * To find the inverse of a \*b mod c is to find the value of b (range from 0 to c-1) in which a \* b mod c = 1
    - To encrypt an 8-bit plain text, multiply each bit by the corresponding GK weight
      * To decrypt we multiply the GK value by the modular inverse then mod N
    - Prone to forward attack search if number of weights is small
* Modes of operations used for Block Ciphers:
  + Electronic Codebook(ECB):
    - Encrypt each block independently
    - Seen like a single codebook cipher
    - Assume 64-bit blocks and 8-bit ASCII
    - Mallet can cut and paste cipher text in transits, can switch order
    - Integrity issues and if two plaintext blocks are similar, giving Mallet information to attack
    - If using an image, structure will leak in ciphertext due to repeated blocks
  + Cipher Block Chaining:
    - Blocks are chained together and as such, aren’t encrypted independently
    - Uses an initialisation vector (IV), a block-sized bit string shared that is shared between the two people, and the following equation to encrypt/decrypt
      * Ci = E (C i-1 ⊕ Pi, K)
      * Pi = C i-1 ⊕ D(Ci , K), where C 0 is IV
    - IV should not be reused because if it is, the same first cyphertext will be produced if the first plaintext block is repeated
    - IV can be randomly chosen and sent as the first ciphertext block
    - Identical plaintext blocks have different cipher text blocks
    - If transmission errors occur, one cipher block error creates two block errors in decryption
      * i.e. If C1 happens to corrupted to but rest is correct then P 1 C0 ⊕ D( , P2 but P 3 C2 ⊕ D(
  + Counter (CTR)
    - Makes block cipher act like stream cipher
      * Used when random access to encrypt data is required like read or write
    - Encryption: Ci = Pi ⊕ E(IV+I,K)
    - Decryption: Pi = Ci ⊕ E(IV+I,K)
* Message Authentication Code (MAC) is sent with ciphertext for integrity
  + To produce this, we can use CBC-MAC which is done by computing the CBC encryption of the plaintext but save the final ciphertext block, the MAC
  + MAC is CN−1 = E(CN−2 ⊕ PN−1, K) = MAC, we then send the P0 🡪 PN-1 and the Mac in which the receiver computes the MAC and checks if it’s the same as the same
  + Receiver and sender must share K and IV which is fixed
  + If error occurs due to man in the middle, the MAC would be different thus the receiver would know the message is compromised
  + Thus, we usually combine encryption and integrity with CBC as if authentication fails we can just discard the data
  + Encrypt and produce MAC with 2 different keys
  + Twice as much work

RSA

* Modulus(denoted as N) = pq where p and q are large prime numbers
* ed= 1 mod(p-1)(q-1) where e is co-prime to (p-1)(q-1)
* Private key: d
* Public Key: (N,e)
* Plaintext here is usually denoted M and is an integer between 1 and N-1
* E = Me mod N; D = Cd mod N
* Repeated squaring:
  + Turn the exponent into binary
  + Build a list from the bits with the first element having one bit then adding one in each element
  + Convert the list into decimal list of exponents
  + Calculate each individual exponent in terms of previous ones
* Security Concerns:
  + If e is small, prone to cube root attack
  + If Mallet knows N=pq, can find private key d

Diffie-Hellman(DH) Key exchange:

* Key exchange algorithm
* v= gk mod p
  + The problem consists in getting the exponent k given v, g and p
* How it works:
  + Let p be prime and g (generator) be a number between 1 an p-1 s.t for any x, there is n such that x = gn mod p. g and p are both public
  + Alice and Bob each select a value a and b respectively s.t a and b are between 1 and p-2
  + A send ga to B and B sends gb to A
  + They both then compute gab mod p and is then used as a symmetric key
* Mallet can pretend to be the sender and receiver in a Man in the Middle attack (MiM)
  + Can encrypt DH with symmetric or public key to prevent this or sign the DH values with a private key

PKI

* Something needed to securely use Public key Cryptography
* Has 3 Trust Models:
  + Monopoly model:
    - Single trusted organisation is the CA for known universe
    - Who to trust?
  + Oligarchy Models:
    - Multiple trusted Cas
    - User decides who to trust
    - Chaining: CA can get certificate for its public key from another CA and do the same with its public key and again with another CA
      * Root certificate leading to same safety as self-signed certificates
  + Anarchy Model:
    - Everyone is a CA and users decide who to trust
    - More signatures a public key has, more trustable

# Signature

* Order matters:
  + {[M]Alice}Bob
    - Issue: after decrypting with his private key, Bob can forward  
      [M]Alice to a third party (Charlie) as follows:  
      {[M]Alice}Charlie which leads Charlie to think that he is the destination of Alice’s message, but the original destination was Bob
  + [{M}Bob]Alice
    - Issue: MiM (man in the middle). The MiM can be somebody that Bob trusts, such as Charlie, who obtains {M}Bob using Alice’s public key and then sends [{M}Bob]Charlie to Bob
* Digital Certificate: Essentially acts like a way to verify someone is who they are using their public key
* A CA (Certificate Authority) is a trusted source that can be used in signatures
* S= [M]CA (sign the message CA private key) which I the same as Md mod N where N is modulus and d is private key
* Alice’s Certificate would be (M,A)
* To verify signature: M= {S}CA Check using CA’s public key
  + Used to verify integrity of public and identity of owner
* S = [h(M)]Alice when Signatures are combined with hashes

# Hashes

* Acts as a smaller fingerprint of M
* Are deterministic
* If Mallet can calculate M’ such that h(M’)=h(M), Mallet can replace (M,s) with (M’,S) and Bob wont detect the tampering as h(M’) = h(M) = {S}Alice and thinks M’ was signed by Alice
* Features of a Hash:
  + Compression: the size of the output should be small
  + Efficiency: h(x) should be easy to compute for any x
  + One-way
  + Weak collision resistance
  + Strong collision resistance
* If h(x) is n bits long, Pr(h(x)) = 1/2n
* Hash values are unrelated for different message
* Different hash functions take an arbitrary length input and produces a fixed length output. Always same sized output despite of input size, a tiny change in input completely changes output.
* Merkle-Damgard Scheme:
  + Message is divided into b-bit blocks M1🡪Mt, may need padding
  + Then processed block by block, using intermediate n-bit changing variables hk for k = 1🡪t+1
  + Procedure:
    - h1 = IV (initial value)
    - hk+1 = g(hk , Mk) 1<=k<=t where g is a compression function
    - h(M) = ht+1
  + If g is collision resistant, so is the iterated hash
  + Compression functions used here can be built using block ciphers
  + Schemes for compression function are based on encryption function E( . , .)
    - Matyas-Meyer-Oseas: hk+1  = Mk ⊕ E( Mk , f(hk))
    - Davies-Meyer: hk+1 = hk ⊕ E( hk , f(hk))
    - Miyaguchi-Preneel: hk+1 = hk ⊕ E( Mk , f(hk))
    - F acts as a mapping to the key size if needed
* HMAC:
  + Alternative to CBC-MAC
  + Symmetric key hash
  + Prone to length extension attacks
  + Hash functions are public
  + We need a key for the same reason as CBC-MAC: if we just send M and M =h(M), mallet can replace M and create a new MAC
  + Before applying hash function, we can concat the key and message or the other way around
  + If we follow Merkle scheme, we compute in blocks
    - M=(M1,M2) then h(m) = g(g(IV,M1),M2) = g(h(M1),M2)
    - This is an issue because:
      * If HMAC is h(K,M). Mallet can produce M’ s.t. M’=(M1,M2,X) leading to h(K,M’) = g( h(K,M),X). This means that Mallet can get the HMAC of M’ with needing the key
      * If HMAC is h(M,K). Mallet can produce M’ s.t. h(M’) = h(M). If M is a multiple of the block size then h(M,K) = g(h(M),K) = g(h(M’),K) = h(M’K) leading to mallet finding a keyed collision as they found a collision without a key
  + To compute it securely:
    - Let B be message block length used in hash
    - ipad = byte 0x36 which is repeated B times
    - opad = byte0x5c repeated B times
    - HMAC(M,K) = h(K ⊕ opad, h(K ⊕ ipad, M))

# Passwords

* Possible number of passwords: Given x possible characters and a password is of length n, we have xn possible passwords
* To attack an account: Either target a specific account, target any account on a single system or any account in any system
* If storing password (no encryption or hashing) in a file, if system is compromised, they’ll be exposed. If we encrypt the file, the decryption key must be accessible for verifying passwords which lead to the issue that if Mallet can access the password file, access to key
  + Thus, store a hashed password rather than password
* Salts help make these more difficult
  + Compute hashed password s.t. y = (s, password)
  + Not Secret
* Password cracking:
  + Assumptions:
    - Password has n characters with x choices per character, meaning xn possible passwords
    - File has 2h hashed passwords
    - Dictionary has 2d passwords
  + One Specific Password:
    - Avg Work (No Dictionary): xn/2
    - Avg Work (Salt && With Dictionary): Pr(In Dictionary) \* 2d /2 + Pr(Not in Dictionary) \* (xn - 2d)/2
    - Avg Work (No Salt && With Dictionary): Pr(Not in Dictionary) \* (xn - 2d)/2 as no work involved in trying dictionary. There is a one-time work to compute dictionary hashes
  + Attack any password:
    - Avg Work (No Dictionary && Salt): (xn/2) as Mallet can’t recycle computed hashes
    - Avg Work (No Dictionary && No Salt): (xn/2) / 2h
    - Avg Work (Salt && With Dictionary): 2d / Pr(In Dictionary)
    - Avg Work (No Salt && With Dictionary): (2d/2) / 2h
* Asymmetric PAKE is prone to Man in the Middle
* Oblivious PAKE (OPAQUE):
  + PRF is like MAC which uses a uniformly random output
  + How it works:
    - Alice blinds input with a random value and sends it to Bob
    - Bob then runs the PRF on the blinded input using his secret key
    - Alice unblinds the input to get output
  + Output only depends on input
  + Can use exponentiation modulo and a large prime p
    - Alice’s input is an element x ∈ {1, ... , p − 1}
    - They then generate a random binding factor r ∈ {1, . . . , p − 2}
    - Blinds input xb= xr (mod p) and is sent to Bob
    - Bob computes blinded output s.t. yb = (xb)k mod p (k is secret key)
    - Alice then unblinds the result: y = (yb)inverse r = ( (xb)k )inverse r  = ( xkr)inverse r = xk (mod p)
  + Can be used to register to a server
    - Alice generates key pair and sends public key to server who stores it
    - Then uses OPRF with her password to get a symmetric key which is used to encrypt their key pair, later sent to server
    - When logging in, they obtain keypair from server and then uses OPRF with password to get a symmetric key to decrypt key pair then signs a challenge

# Authentication

* Can use keys to avoid passwords (Either symmetric or asymmetric)
* Symmetric Key Authentication:
  + Avoid using a symmetric key directly because if compromised, attacker has unlimited access
  + Indirect scheme:  
    A couple of different names

    Description automatically generated with medium confidence
    - Alice first gets password generator device which stores the symmetric key K and can compute hashes.
    - When Alices attempts to log in, a challenge R is sent to her by the Server
    - Alice then generates a password h(K,R) to login (only accesses key indirectly)
    - Server then verifies Alice’s password by hashing (K,R)
  + Once could replace the challenge by additional data which changes every authentication attempt (think OTP)
  + HOTP: HMAC of a counter using a symmetric key. Sever-client synchronization is an issue
  + TOTP( time-based OTP): Variation of HOTP which uses current time. Prone to lack of non-repudiation, OTP can be stolen
* Asymmetric key Authentication:
  + User is either asked to authenticate during (Secure Shell) or after handshake(FIDO2)
  + Must sign a random challenge using their private key while server verifies with users public key
  + FIDO2
    - Fast Identity Online 2
    - Only works with hardware authenticators as it targets phishing attacks
    - Two specifications
      * Client to Authenticator Protocol (CTAP)
      * Web Authentication Protocol (WebAuthn)
  + Identification is one to many whereas authentication is one to one
  + Biometrics have two phases:
    - Enrolment in which the biometric info is added to a db
      * Slow and must be precise
    - Recognition in which we use biometric detection to identify the person
      * Must be accurate and quick and simple
  + For biometrics: we can either decrease fraud or insult at the cost of the other
  + Equal Error Rates is when fraud rate is the same as insult rate and the lowe this is the better
  + For fingerprint:
    - Enrolment is via image capture, image enhancement or automatic identification of points
    - Recognition is by extracting points and comparing it to info stored in a db
  + For Iris:
    - Average Hamming Distance =No of non-matching bits at same position / total number of bits in iris code.
      * Perfect match is 0 and worst match is 0.5

# Protocols

* Ideal protocols must be:
  + Robust: Work under attacks and environment changes
  + Efficient: Minimize computational requirements, delays, bandwidth
  + Ease of implementation, use and flexibility
* Network authentication is prone to:
  + Replay attacks
  + An attacker can observe protocol messages
* A screenshot of a computer screen

  Description automatically generated
  + This is bad as Bob must know Alices password to authenticate. Plus, it is also prone to replay attacks and MiTM
  + Can adjust the above to make it one line (include name and password) or use a hashed password but still prone to replay attack and Alice doesn’t authenticate Bob
* Challenge-response are used to prevent replay attacks
  + Bob sends a non-reusable challenge (Nonce is uses); Alice gives correct response and Bob verifies it.
* Nonces are used here to make sure the response are fresh
* Generic Challenge-response using nonces:  
  A close-up of a sign

  Description automatically generated
  + Can use a hash function which takes in the Nonce and password but using a key is safer
* Symmetric One-way authentication:  
  A close up of a white background

  Description automatically generated
* Symmetric mutual authentication:  
  A close up of a text

  Description automatically generated
  + However this is prone to reflection A diagram of a mathematical equation

    Description automatically generated with medium confidence
  + Can complete Bobs Step 2 Challenege in Step 5
  + To prevent this, we can change it to the following:  
    A group of black text

    Description automatically generated
    - We can even make RA and RB  odd and even respectively or encrypt RB in the second message
* One way authentication with asymmetric keys: 
  + A white background with black text

    Description automatically generated
    - Note Alice should use two different keys for encryption and signing as Mallet can then force her to decrypt/sign anything.
* Mutual Authentication with a Session Key and asymmetric keys: A screenshot of a computer code

  Description automatically generated  
  which can be combined to   
  A group of text boxes

  Description automatically generated with medium confidence  
  or  
  A group of black text

  Description automatically generated
* A KDC generates fresh symmetric session key pairs for users in system
* Generic Session Key With KDC
* A black text with black lines

  Description automatically generated with medium confidence
* Variant includes A close up of a text

  Description automatically generated
* Timestamps are usually in ms and can be used as a nonce and while it reduces the number of messages, time ironically becomes a critical parameter here as we need to consider clock skew and synchronisation. (Uses sign and encrypt)
* To achieve PFS, we cant use a shared key to encrypt messages, use a new session key and forget it after each use
* We can use Diffie-Hellman for key exchange then drop a and b (Session key = gab mod p) . This is called ephemeral Diffie-Hellman. This gives us  
  A group of black text

  Description automatically generated
* TCP  
  A math equations with black text

  Description automatically generated with medium confidence
  + SYN(Synchronisation): Request to start the TCP/IP connection using initial sequence a
  + SYN-ACK: Acknowledgement of SYN and establishes actual initial sequence number b
  + ACK: Acknowledges SYN-ACK
  + A and B are meant to be random(although most times are pseudorandom)
* SSH
  + Cryptographic protocol which creates a secure tunnel for accessing a remote machine over an insecure network
  + Client server protocol
  + 4 Parts:
    - Transport Layer Protocol (SSH-TRANS)
    - Authentication Protocol (SSH-AUTH)
    - Connection Protocol (SSH-CONN)
    - File Transfer Protocol (SSH-SFTP)
  + SSH Authentication steps:
    - Alice sends an identifier and a nonce (RA) to Bob
    - Bob then sends his nonce to her
    - Alice then sends her public key (ga mod p) to Bob
    - Bob then sends his public key (gb mod p,) his certificate and his signature. The signature is made of a hash includes Alice and Bobs identifiers, Both nonces, the session key and both of their public keys
    - Alice then sends an encryption of her certificate and signature using the session key. Her signature includes her certificate and includes Alice and Bobs identifiers, Both nonces, the session key and both of their public keys
    - Then a protected session occurs
  + Man in the middle attacks fail here as Alice realises, she would sign a different value due to the changed keys hence detecting tampering.
    - H = h(Alice,Bob,CP,CS, RA, RB, gc mod p, gb mod p, gbt mod p) is what Bob signs
    - Alice would sign
    - H′ = h(Alice,Bob,CP,CS, RA, RB, ga mod p, gd mod p, gat mod p)
  + Can opt to not use certificates using TOFU(Trust on first use) authentication:
    - Alice (Client) firstly furnishes Bob (Server) with public key directly, then the handshake will occur similar to above but instead, Bob send public key to Alice and Alice then receives a warning thus she needs to verify a hash of Bobs public key via another secure channel and then after being satisfied, she trusts it
* SSL (Secure Socket Layer)/TLS(Transport Layer Security)
  + SSL: Found on top of transport layer and implemented within the application layer
  + SSL main motivation is web security
  + Basic Motivation behind SSL:
    - Alice sends a message to Bob
    - Bob sends his certificate to Alice
    - Alice then sends Bob’s public Key
    - Secure line is open
  + Alice can only verify Bob if he can decrypt something she previously sent using K
  + Simplified SSL(Here one way authentication occurs):
    - Alice sends a ClientHello (i.e. A message and a her nonce)
    - Bob then sends the ServerHello which include his nonce and certificate
    - Alice then sends a pre-master Secret encrypted by Bob’s public key and a hash of previous messages in the session, a string Constant CLNT and the session Key
    - Bob then sends the hash of previous messages in the session, a string Constant SRVR and the session Key
  + Hashes are sent to verify all previous messages have been received correctly and check if any tampering has occurred
  + 6 Parameters derived from session key one for client and other for server:
    - 2 Encryption keys
    - 2 IVs
    - 2 Integrity keys
  + Mutual authentication in TLS/SSL can occur
  + MiTM can’t occur as Alice (or browser usually) usually will realize the public key in the Certificate is changed
  + For a previously established session:
    - Alice sends Bob session ID and her nonce
    - Bob then sends a session ID, his nonce and a hash which includes the previous messages, SRVR and session K
    - Alice then sends a has of the previous messages, CLNT constant and the session key
    - Main takeaway here is that since Alice and Bob share a session key K, Bob doesn’t need to send his certificate and Alice’s public key operation is skipped