1. The advent of quantum computing poses significant challenges to the security of current cryptographic systems, particularly those based on public-key cryptography like RSA (Rivest-Shamir-Adleman) and ECC (Elliptic curve cryptography). This is primarily due to the power of quantum algorithms such as Shor's algorithms, which can efficiently solve mathmatical problems that are difficult for classical computers such as factoring large integers (critical for RSA) and solving discrete logarithms (essential for ECC). These algorithms threaten the foundational security assumptions of RSA & ECC, making them potentially insecure in a post-quantum world.

Implication on Quantum computing on cryptographic protocols:

Breakdown of RSA and ECC:

RSA: The security of RSA relies on the difficulty of factoring large numbers, a problem that is exponentially hard for classical computers. Shor's algorithm, however can factor large numbers in polynomial time, making RSA keys (typically 2048-bits or larger) vulnerable to quantum attacks.

ECC: The security of ECC is based on the difficulty of the elliptic curve discrete logarithm problem (ECDLP), which is also solvable efficiently by Shor's algorithm's in polynomial time. This would make the commonly used elliptic curve cryptography protocols insecure as well.

Impact on secure communication:

Much on modern digital security infrastructure, including SSL/TLS for secure web browsing, PGP for email encryption, and VPNs for secure connections rely on heavily on RSA and ECC. If there is systems are compromised, the confidentially and integrity of sensitive data could be at risk.

Additionally, digital signatures used in software distribution and financial transactions, which rely on RSA & ECC, would also be vulnerable to quantum attacks.

Quantum- Resillent systems:

To prepare for a quantum future, cryptographic systems must be evolve. This includes developing algorithms that can resist quantum cryptanalysis, particularly those that are based on mathmatical problems not solvable by shor's algorithms or similar quantum algorithm.

Post - quantum cryptographic Algorithms:

Given the threat posted by quantum computers, there has been significant research into developing cryptographic algorithms that are resistent to quantum attacks. These are collectively known as post quantum cryptography (PQC). A few key classes of algorithms that show promise in a post-quantum world include:

Lattice Based cryptography:

Algorithms: NTRU, Learning with Errors (LWE), Ring-LWE.

Quantum Resistance:

Lattice-babed problems are beleived to be hard even for quantum computers. The underlying problems (such as finding short vectors in a high dimentional lattice) have not been efficiently solved by quantum algorithms. These problems are conjectured to remain hard even with quantum computation.

Application: Lattice-based schemes can be used for public key encryption, digital signature and key exchange. They are already being considered for inclusion in standards like NIST's post quantum cryptography project.

Code-Based cryptography:

Algorithms: McEliece

Quantum Resistance: The McEliece cryptosystem is based on the hardness of decoding random linear codes. while this problems is also susceptible to classical algorithms (e.g. the information set decoding problem), no efficient quantum algorithms have been discovered for it, making it a strong candidate for post quantum security.

Applications: McEiece is considered for encryption and digital signatures, though it has the drawback of requring relatively large public keys compared to RSA or ECC.

Hash-Based cryptography:

Algorithms: Merkle signature scheme.

Quantum resistance: Hash-based cryptographic systems, such as Merkle's tree based signature schemes, rely on the security of cryptography hash functions - Quantum algorithms like Grover's algorithms offer a quadric speed up for searcing through hash functions but they do not provide an Exceptional advantage. Therefore, hash based cryptography remains resistant to quantum attacks as long as sufficiently large hash outputs are used.

Applications: These systems are primarily used for digital signatures though the signatures sizes may be larger than traditional RSA or ECC systems.

Multivariate polynomial cryptography:

Algorithms: Rainbow, unbalanced oil and vinegar.

Quantum Resistance: multivariate polynomial problems, such as solving systems of multivariate quadratic equations, are hard to solve both classically and quantumly. This makes them promising them for post quantum.

2. Creating a pseudo-Random Number generator (PRNG) can be an interesting task when combining multiple sources of entropy (randomness). In this case, we will use the current timestamp, the process ID and a modulus operation to constrain the output to a specefic range.

Here is a breakdown of how the algorithm will work:

Steps to design:

1. Current timestamp: The current time in seconds or milliseconds provides a good source of entropy since it is always changing.

2. Process ID: The process ID of the program adds more randomness since each more randomness since each running process on a system has a unique ID.

3. Combination of Timestamp and process ID: By combining both we can generate a number that is more unpredictible.

4. Modulus operation: To make sure the output fails within a desired range (e.g. between 0 and 100), we use the modulus operation.

Algorithm in python:

```
import os
import time
def custom_prng (min_value, max_value):
# step 1: Get current timestamp (in millisecond for better precision)
  time stamp = int (time. time) () * 1000 # convert to milliseconds
# step 2: Get process ID
    pid = os. getpid ()
# step 3: combine timestamp and PId to generate a raw random number
        raw_random = time stamp + pid
```

# step 4: Apply a modulus operation to constrain the number within the range [min_value, max-value]
random_number = raw_random % (max_value - min_value + 1) + min-value
return random number
# Example usage:
random_number = custom_Prng (0, 100) # Get a random number between 0 e 100
Print (random number)

Explanation:

1. Timestamp: we use the time.time() function to get the current time in seconds. Multiplying it by 1000 gives us milliseconds for finer precision.
2. Process ID: We use os.getpid() to fetch the process id, which is unique to each running instance of your program.
3. Combine Timestamp and PID: We simply add the timestamp and pid together. The idea is that the combination of both values will give us a number that varies unpredictably over time and between processes.
4. modulus operation: We use the modulus operator (%) to ensure that the Final output is within the desired range (From min_value to max_value). The modulus operation will give us a reminder when divided by (max_value - min_value) which is then shifted to the range [min_value, max_value] by adding min-value.

Example:

If you run the code, you will get a pseudo random number, For example in;
random_number = 43.
This value will be different everytime you run the program, as both the timestamp and the process ID change with each execution.

3. Traditional ciphers:

1. caesar cipher

- key: A small number (1-25)
- speed: very Fast.
- security: weak. It is very easy to crack because there are only 25 possible keys and you can easily try all of them.

2. Vigenere cipher:

- key: A word or phrase (can be long)
- speed: Fast, but a bit more complex than Caesar.
- security: Better than caesar, but still weak if the key is short. If the key is long and random, it can be stronger.

3. Playfair cipher:

- Key: A 5x5 letter grid.
- speed: Fast, but a bit more complicated to use than the others.
- security: Moderate. It is harder to crack than caesar or vigenere, but still not secure by modern standard.

Modern ciphers:

1. DES (Data Encryption standard)

- key: 56 bits (a small key todays standard)
- speed: Fast for old computers but slow for modern ones.
- security: weak. It can be cracked easily using modern computers because the key is too short.

Key points:

- The combination of time and pid ensures that the randomness isn't entirely predictable.
- The modulus operation confines the result to a particular range.
- This approach uses two simple sources of entropy: system time and the process ID. making it simple but effective for small tasks where high quality randomness isn't a strict requirement.

This basic PRNG might not be cryptographically secure or suitable for all applications, but it is a fun and simple way to generate pseudo-random numbers!

2. AES (Advanced Encryption standard)

- Key: 128, 192, or 256 bits (much longer and stronger)
- Speed: Very Fast and easy.
- weakness: The key is too short now, so it is easy to break.

Simple comparison:

- Key length: Traditional ciphers have very short keys, so they are easily to crack. Modern ciphers (like AES) have much longer keys, making them hard to break.
- Speed: Caeser cipher very Fast, vigenere cipher moderate, Playfair cipher moderate, Des Fast, AES Fast.
- Security: Caesar cipher very weak, vigenere cipher moderate, Playfair cipher Better but weak, DES weak, AES very strong.

In simple Terms:

Old ciphers (ceaser, vigenere, playfair): They are basic and easy to break.

Modern ciphers (DES, AES): Des is old and insecure - AES is the real deal - strong, fast and widely used for securing sensitive data today.

4. Step 1: What is the action of S4S-4S4 onthe set of 2-element subsets?

The group S4S-4S4 consists of all the ways to rearrange the numbers {1,2,3,4}\{1, 2,3,4\}. {1, 2, 3,4}. A 2-element subset is just a pair of distinct numbers from this set. For example {1,2}\{1,2\}{1,2} is a 2-element subset.

We define the action of S4S_4S4 on the 2-element subsets as follows:

If σ∈S4\sigma\in S_4σ∈S4 is a permutation and {i, j}\{i, j\}{i,j} is a 2-element subset then:

σ.{i, j}={σ(i), σ(j)}\sigma\cdot\{i,j\} =\{\sigma(i),\sigma(j)\} σ·{i,j}
= {σ(i), σ(j)}

This means that σ\sigma moves iii and jjj to new positions, but it keeps the set of two elements.

Step 2: Is the action well defined?

Yes, the Action is well defined because:

A permutation will always result in a valid 2-element subset (Just a rearranged pair numbers)

The group action rules hold: applying two permutations in sequence is the same as applying their composition, and applying the identity permutation leaves the set unchanged

Step 3: Size of orbit of $\{1,2\}\setminus\{1,2\}\}\{1,2\}$

The orbit of $\{1,2\}\setminus\{1,2\}\}\{1,2\}$ is the set of all 2-element subsets that can be obtained from $\{1,2\}\setminus\{1,2\}\}, \{1,2\}$ by applying any permutation from $S4S-4S4$

Since any 2-element subset can be reached by some permutation of $\{1,2\}\setminus\{1,2\}\}. \{1,2\}$ is the set of all 2-element subsets of $\{1,2,3,4\}\setminus\{1,2,3,4\}\}, \{1,2,3,4\}$ which are

$\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{3,4\}\setminus\{1,2\}\}, \setminus\{1,3\}\}, \setminus\{1,4\}\}, \setminus\{2,3\}\}, \setminus\{2,4\}, \setminus\{3,4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}$.

So the size of the orbit is 6.

Step 4: Size the stabilizer of $\{1,2\}\setminus\{1,2\}\}\{1,2\}$

The stabilizer of $\{1,2\}\setminus\{1,2\}\}\{1,2\}$ consists of all permutations in $S4S-4S4$ that leaves $\{1,2\}\setminus\{1,2\}\}\{1,2\}$ unchanged. This means the only allowed changes are:

Swapping 111 and 222, or

Leaving 111 and 222 in place

The other numbers 333 and 444 can be swapped independently

So the stabilizer Includes:

The identity permutation eee.

The permutation $(1,2)(1\setminus 2)(1,2)$ (which swaps 111 and 222).

The permutation $(3,4)(3\setminus 4)(3\ 4)$ (which swaps 333 and 444)

The permutation $(1\ 2)(3\ 4)(1\setminus 2)(3\setminus 4)(1\ 2)(3\ 4)$ (which swaps both 111 and 222) and 333 and 444)

Thus the size of the stabilizer is 4

Final Answer:

The size of the orbit of $\{1,2\}\setminus\{1,2\}\}\{1,2\}$ is 6

The size of the stabilizer of $\{1,2\},\setminus\{1,2\}\}\{1,2\}$ is 4

5
① To show that $GF(2)^2$ forms a group under multiplication, we need to check the four group properties for the set of non-zero elements of $GF(2)^2$:

1. Closure: For all $a, b \in GF(2)^2$, a, b must also be in $GF(2)^2$

2. Associativity: Multiplication must be associative. Since we're using the field $GF(2)^2$, which is a finite field, the multiplication is naturally associative.

3. Identity element: There must be an element $e \in GF(22)$ e \in GF(2^2) $e \in GF(22)$ such that for any element $a \in GF(22)$ a\in GF(2^2) $a \in GF(22)$, $a.e = a$a\cdot e = a$a.e = a$

Inverses: Every element $a \in GF(22)$ a \in GF(2^2) $a \in GF(22)$ must have an inverse $a-1 \in GF(22)$ a^{-1} \in GF(2^2) a-1GF(22) such that $a.a-1 = e$a\cdot a^{-1} = ea.a-1=e

Step by step verification:

1. The set of elements of $GF(22)$ GF(2^2) $GF(22)$:
We are given that $GF(22)$ GF(2^2) $GF(22)$ is constructed using the irreducible polynomial $x2+x+1$x^2+x+1$x2+x+1$ over $GF(2)$ $GF(2)$ $GF(2)$. The elements of $GF(22)$ GF(2^2) $GF(22)$ are $\{0, 1, a, a+1\}$ \{0, 1, \alpha \alpha+1\} $\{0, 1, a, a+1\}$, where a\alpha a is a root of the polynomial $x2+x+1$ x^2+x+1 $x2+x+1$
This means that:
$a2 = a+1$ (since $a2+a+1=0$ in $GF(2)$). \alpha^2=\alpha+1\quad \text{(since} \alpha^2 +\alpha+1=0\text{in}
$GF(2)$). $a_2 = a+1$ (since $a_2+a+1=0$ in $GF(2)$)
Thus, the elements of $GF(22)$ GF(2^2) $GF(22)$ are $0, 1\ a, a+1 0, 1$

\alpha + 1 0, 1, a, a+1

2. Closure:

We need to check that multiplying any two non-Zero elements of GF(22) GF(2^2) GF(22) results in another non-Zero element in GF(22) GF(2^2) GF(22)

1·1 = 11 \cdot 1 = 11·1 = 1

1·a = a1 \cdot \alpha = \alpha 1·a = a

1·(a+1) = a+11 \cdot (\alpha+1) = \alpha + 11.(a+1) = a+1

a·a = a2 = a+1 \alpha \cdot \alpha = \alpha ^ 2 = \alpha + 1 a·a = a2 = a+1

a·(a+1) = a^2+a = a+1+a = 1 \alpha \cdot (\alpha + 1) = \alpha ^ 2 + \alpha = \alpha + 1 + \alpha = 1a·(a+1) = a2+a = a+1+a = 1 (since a+a=0 \alpha + \alpha = 0 a+a = 0 in GF(2) GF(2) GF(2)

(a+1)·(a+1) = a2+2a+1 = a+1+1 = a (\alpha + 1) \cdot (\alpha + 1) = \alpha ^ 2 + 2 \alpha + 1 = \alpha + 1 + 1 = \alpha (a+1)·(a+1) = a+2+2a+1 = a+1+1 = a

Thus multiplying any two non-Zero elements results in another non-zero element of GF(22) GF(22) GF(22). So closure is satisfied.

3. Identity element:

The identity element is the element eee such that a·e = aa \cdot e = aa·e = a for all a ∈ GF(22) a \in GF(2^2) a ∈ GF(22). From the multiplication table above, we see that multiplying by 111 leaves any element unchanged.

1·1 = 11 \cdot 1 = 11·1 = 1

1·a = a1 \cdot \alpha = \alpha 1·a = a

1·(a+1)=a+11\cdot(\alpha+1)=\alpha+11·(a+1)=a+1

Therefore, 111 is the identity element.

4. Inverse:

Each element must be an inverse such that a·a−1=1a\cdot a^{1}=1a·a−1=1. From the multiplication table:

1·1=11\cdot 1=11·1=1, so the inverse of 111 is 111.

a·(a+1)=1\alpha\cdot(\alpha+1)=1a·(a+1). so the inverse of a\alpha a is a+1\alpha+1a+1

(a+1)·a=1(\alpha+1)\cdot\alpha=1(a+1)·a=1. so the inverse of a+1\alpha+1a+1 is a\alpha.

Thus every element has an inverse in GF(22) GF(2^2) GF(22)

Conclusion:

Since all four group properties ((closure, associatively, identity element, and inverses) are satisfied, we have shown that the non-zero elements of GF(22) GF(2^2) GF(22) under multiplication from a group.

To verify if the set of all non zero elements of $GF(2^2)$ is cyclic, we need to check if there is a element (called a generator) whose powers generate all non zero elements of the field.

step 1: understand the elements of $GF(2)^2$:

The finite field $GF(2^2)$ is constructed using the irreducible polynomial x^2+x+1 over $GF(2)$, the finite field with two elements: 0 and 1

- Elements of $GF(2^2)$ can be represented as polynomials of degree less than 2, with coefficients in $GF(2)$. so the elements of $GF(2)^2$ are:
- 0 (the additive identity)
- 1 (the multiplicative identity)
- α, where α is a root of the irreducible polynomial x^2+x+1
- $\alpha+1$

Thus, the non-zero elements of $GF(2)^2$ are $1, \alpha, \alpha+1$

step 2: check if these elements form a cyclic group

A set is cyclic if there is an element (called a generator) such that the powers of this element (under multiplication) produce all other elements in the set.

Let us try α as the potential generator:

we will compute successive powers of α and see if we can generate all non zero elements. of $GF(2)^2$

- $\alpha^1 = \alpha$ (which is one of the non zero elements)
- $\alpha^2 = \alpha \cdot \alpha$. Since α is a root of $x^2+x+1=0$, We have $\alpha^2+\alpha+1=0$, or $\alpha^2 = \alpha+1$. Thus $\alpha^2 = \alpha+1$ (which is the second non Zero element)
- $\alpha^3 = \alpha^2 \cdot \alpha = (\alpha+1)\cdot\alpha = \alpha^2+\alpha = (\alpha+1)+\alpha = 1$ (which is the multiplicative identity.

Thus, the powers of α generate the elements $\alpha, \alpha+1, 1$.

Step 3: conclusion

Since the powers of α generate all the non Zero elements $1, \alpha, \alpha+1$ the set of non Zero elements of $GF(2)^2$ is cyclic and α is a generator of this cyclic group.

6. scalar Matrices

A scalar matrix in $GL(2,\mathbb{R})$ is a matrix of the form:

$$A = \lambda I = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$$

where $\lambda \in \mathbb{R}$ (non zero real numbers) and I is the identity matrix.

1 subgroup

The set S of scalar matrices is a subgroup of $GL(2,\mathbb{R})$ because it contains the identity, is closed under multiplication and contain inverses.

2. Normal subgroup

- To show S is normal, we check that for any $A \in GL(2,\mathbb{R})$ and any scalar matrix λI, the conjugate $A(\lambda I)A^{-1} = \lambda I$ is still a scalar matrix. Thus, S is normal.

3. Factor group:

The Factor group $GL(2,\mathbb{R})/S$ represents matrices in $GL(2,\mathbb{R})$, without considering multiples.

- This Factor group is isomorphic to $PGL(2,\mathbb{R})$, the projective general linear group, which considers matrices up to scalar multiples.

summary:

The set of scalar matrices S is a normal subgroup of $GL(\mathbb{R},2)$ and the corresponding factor group is isomorphic to $PGL(2,\mathbb{R})$ which represents matrices modulo scalar matrices.

7. The Diffle-Hellman key exchange allows two parties to securely share a secret key over an in secure channel by using modular arithmatic and the Discrete Logarithm problem (DLP), which is hard to reverse.

However, its vulnerable to man-in-the-middle attacks where an attacker intercepts and replaces public keys. To prevent this, authentication is needed.

If the prime modulus is too small the discrete logarithm problem becomes easier to solve, weakening security and making the key exchange susceptible to attacks.

8. To prove that the intersection of any two subgroups of G is also a subgroup. we need to check that the intersection $H_1 \cap H_2$ satisfies the subgroup criteria:

1. Identity Element: The identity element of G is in both H_1 and H_2 (since both are subgroups), so it must be in their intersection.

2. Closure: If $a, b \in H_1 \cap H_2$, then a, b are in both H_1 and H_2. since both are subgroups, ab must be also be in both H_1 and H_2 hence $ab \in H_1 \cap H_2$

3. Inverses: If $a \in H_1 \cap H_2$, then a has an inverse in both H_1 and H_2, so a^{-1} is in both and hence $a^{-1} \in H_1 \cap H_2$

Thus, $H_1 \cap H_2$ satifies all the conditions to be a subgroup.

Example:

Let $G = Z_6$ (the integers modulo 6) and consider the subgroups $H_1 = \{0, 2, 4\}$ and $H_2 = \{0, 3\}$

The intersection is $H_1 \cap H_2 = \{0\}$, which is a subgroup of Z_6.

9. Commutativity: Z_n is commutative because both addition and multiplication are commutative in Z_n.

Zero Divisors: Z_n has zero divisors if n is composite (i.e. not prime) as there exist non-zero elements a and b such $a \cdot b \equiv 0$ (mood n)

Field: Z_n is a field if and only if n is prime because every non-zero element has a multiplicative inverse where n is prime.

10. Vulnerabilities of DES:

DES (Data Encryption Standard) is considered insecure for modern use due to its relatively small key size of 56 bits. This key length is vulnerable to brute-force attacks where all possible keys are tried until the correct one is Found. With the advancement of computational power brute forcing a 56 bit key is now feasible within a short time.

Brute-Force Attacks and key length:

Brute Force attacks involve trying every possible key. For DES with 56 bit keys, there are 2^{56} possible combinations. As computing power increased this became impractical to resist with DES being broken in a matter of days using modern hardware.

AES and Improvements:

The development of AES addressed DES Weaknesses by.

1. Increased key size: AES supports key lengths of 128, 192 or 256 bits making it vastly more resistant to brute force attacks.
2. stronger cryptographic Design: AES was designed to withstand modern cryptanalytic techique, such as differential and linear cryptanalysis which DES was vulnerable to.

Thus AES offers better security and efficacy compared to DES making it the standard for modern Encryption.

10. DES is insecure due to its short 56 bit key making it vulnerable to brute force attacks with modern computing power attackers can easily break it. AES improves security by using longer key sizes (128, 192 and 256 bits) and offers better resistance to cryptanalytic attacks addressing the weaknesses of DES.

11

i) The Feistel structure of DES helps resist differential cryptanalysis by creating a complex relationship between the input and output bits. It divides the 64 bit data into two halves and each round involves non-linear transformations (like the S-boxes) and mixing operations (like XOR), making it difficult to track how input differences affect the output. The left and right halves are processed seperately and swapped which further complicates the attack. However, DES's relatively short 56 bit key and 16 rounds make it vulnerable to brute-force attacks despite these defenses.

ii) AES is more resistant to differential cryptanalysis compared to DES because of its more complex design and larger key size. Here is how its operation help:

1. SubBytes: AES uses a non-linear S-box for substitution which introduces strong non-linearly, making it hard to predict how differences propagate.

2. shift Rows: This operation shifts rows of the state matrix creating diffusion and mixing the bits which spreads the differences across the entire block.

3. Mix columns: This step further increases diffusion by mixing the columns of the state, ensuring that small input differences affect many output bits.

4. AddRoundkey: AES XORs the key with the data further complicating the analysis by making the key influence non-linearly across rounds.

Additionally, AES uses 128-bit blocks and key sizes of 128, 192 or 256 bits, making it far more resistant to differential cryptanalysis than DES which uses a 56 bit key and 64 bit blocks. The combination of more rounds (10, 12, or 14) and stronger key schedules also enhances it security.

(12) Modular Inverse using the Extended Euclidean Algorithm:

To Find the modular inverse of a modulo n (i.e. x such that $a \cdot x \equiv 1$)

1. Euclidean Algorithm: compute the gcd of a and n. If gcd = 1, they are coprime and an inverse exists.

2. Extended Euclidean Algorithms: use it express gcd as a linear combination of a and n, i.e. $1 = ax + ny$, where x is the modular inverse of a modulo n.

3. If x is negative, add n to get a positive inverse.

Use in RSA Key Generation:

In RSA, after selecting primes p and q, and computing n and $\phi(n)$ the private key exponent d is the modular inverse of e modulo $\phi(n)$ where e is the public key exponent. This enables efficient decryption.

Importance of Efficacy:

The extended Eucidean Algorithm which is crucial for handling large numbers in RSA and ensuring fast, secure cryptographic operations.

(i) In ECB (Electronic codebook) mode, each block of plaintext is independently encrypted using the same key. This means that identical plaintext blocks will produce identical ciphertext blocks.

Proof of Insecurity for Highly Redundant Data:

- Consider plaintext data that contains redundancy, such as repeated blocks. In ECB mode, each repeated block is encrypted block is encrypted in the same way, resulting in identical ciphertext blocks.
- Mathmatically for a block cipher E with block size n, and plaintext message $P = P_1 P_2 \ldots P_m$, where P_i represents the i-th plaintext block. the ciphertext C is obtained by:
$C_i = E(K, P_i)$ for each block P_i where K is the secret key and E is the encryption fuction.
- If two blocks P_i and P_j are identical (i.e. $P_i = P_j$), then their corresponding ciphertext blocks will also be identical (i.e. $C_i = C_j$)
- Insecurity: This predictable pattern allows an attacker to detect identical plaintext blocks from their ciphertext equivalents, revealing information about the structure or redundancy in the data. For highly redundant data, this exposure comprises confidentiality.

Thus, ECB mode fails to hide patterns in the data making it insecure for encryption highly redundant information.

ii

CBC mode Recurrence Relations:

- Encryption:

$C_1 = E_K(P_1 \oplus IV)$

$C_i = E_K(P_i \oplus C_{i-1})$ for $i \geq 2$

- Decryption:

$P_1 = D_K(C_1) \oplus IV$

$P_i = D_K(C_i) \oplus C_{i-1}$ for $i \geq 2$

Error propagation:

- In Encryption: A Error in ciphertext affects only the current block.
- In Decryption: An error in a ciphertext block effects:
  - The current plaintext block.
  - The next plaintext block.

However, the error does not affect previous blocks, so error propagation is limited to the current and next blocks.

14. LFSRs are vulnerable to known plaintext attacks because their output is linear and predictable. If an attacker knowns part of the plaintext and ciphertext, they can reverse engineer the LFSR's state and predict the key stream.

Mitigation: Introducing a non-linear function (e.g. combining multiple LFSRs with XOR) makes the output unpredictable and harder to analyze reducing the vulnerability to such attacks.

15

(i) Shannon's defination of perfect secracy is stated mathmatically as:

For a cryptographic system to have perfect secrecy, the probability of a plaintext M given a ciphertext C must be the same as the probability of the plaintext M without knowledge of the ciphertext.

mathmatically:

$P(M=m \mid C=c) = P(M=m)$ for all $m \in M$, $c \in O$

This means that knowing the ciphertext C gives no additional information about the plaintext M. Therefore the ciphertext reveals no information about the plaintext.

15

ii

The one time pad achieves perfect secrecy because:

Encryption: $C = P \oplus KC = P \oplus K$ (Plaintext XOR key)

Decryption: $P = C \oplus KP = C \oplus K$ (ciphertext XOR key)

with a uniformly random key of length at least as long as the plaintext ($|K| \geq |M|$ $|K| \geq |M|$) each ciphertext is equally likely to correspond to any plaintext ensuring that the ciphertext gives no information about the plaintext. Thus the system achieves perfect secrecy.

15

iii

perfect secrecy in cryptography requires that the ciphertext provides no information about the plaintext regardlessly of the computational power available to an attacker. To achieve perfect secrecy the key must be as long as the message and every possible key must be equally likely for any given ciphertext.

For large scale communication systems perfect secrecy is impractical because,

1. Key management: The need for a key as long as the message makes key generation, distribution and storage infeasible for large messages or frequent communication.

2. Key Exhaustion: The number of keys required grows exponentially with message size, making it computationally unmanageable.

3. Efficiency: The encryption and decryption processes are significantly slower compared to modern encryption methods reducing the overall systems efficiency.

Thus the practically of secure communication in large scale systems relies on approximations to perfect secrecy such as probabilistic encryption schemes like AES.

16. A linear Congruential Generator (LCG) is a simple algorithm used to generate a sequence of pseudo random numbers. It is defined by the recurrence relation:

$$x_{n+1} = (ax_n + c) \bmod m$$

where:

- x_n is the current value in the sequence (with x_0 as the initial seed or starting value)
- a is the multiplier (a constant)
- c is the increment (another constant)
- m is the modulus (determines the range of values the generator can produce)
- The LCG generates the next value x_{n+1} based on the previous value x_n, and the process is repeated to create a series of pseudo-random numbers. The choice of parameters (a, c, m) strongly influences the quality and period of the sequence.

17 An LCG (Linear Congruential Generator) is a method used to generate pseudo-random numbers. The general Formula for generating the sequence is:

$x_{n+1} = (a \cdot x_n + c) \bmod m$

Where

- x_n is the current pseudo random number.
- a is the multiplier.
- c is the increment
- m is the modulus.
- x_0 is the initial seed.
- To compute the first 5 numbers of the sequence, we start with a given seed $x_0 = 7$ and apply the formula iteratively to get $x_1, x_2 \dots$ you just need to plug in the specefic values for a, c and m to compute the numbers.

For example, lets take specefic values for $a = 5, c = 1, m = 16$

Step by step:

1. $x_0 = 7$
2. $x_1 = (5 \cdot 7 + 1) \bmod 16 = (35 + 1) \bmod 16 = 36 \bmod 16 = 4$
3. $x_2 = (5 \cdot 4 + 1) \bmod 16 = (20 + 1) \bmod 16 = 21 \bmod 16 = 5$
4. $x_3 = (5 \cdot 5 + 1) \bmod 16 = (25 + 1) \bmod 16 = 26 \bmod 16 = 10$
5. $x_4 = (5 \cdot 10 + 1) \bmod 16 = (50 + 1) \bmod 16 = 51 \bmod 16 = 3$

So, the First 5 numbers of the sequence are: 7, 4, 5, 10, 3

This is the process to generate pseudo random numbers with LCG.

18. A ring is a set with two operations (addition and multiplication) satisfying specefic properties like closure, associativity and distributy.

- commutative Ring: Z (integers with addition and multiplication)
- Non-commutative Ring: $n \times n$ matrices (for $n > 1$)
- A finite field is a commutative ring where every non zero element has a multiplicative inverse important for constructing fields like Fp.

In RSA encryption, rings (especially modular arithmatic) are used for key generation and encryption / decryption, working with numbers modulo n.

19. key Generation:

- $n = 55$, $\phi(n) = 40$, $e = 3$, $d = 27$
- Public key: $(e, n) = (3, 55)$
- Private key: $(d, n) = (27, 55)$

- Encryption:

$M = 2$, ciphertext $c = m^e$ mod $n = 2^3$ mod $55 = 8$

- Decryption:

$c = 8$, original message $M = c^d$ mod $n = 8^{27}$ mod $55 = 2$

Thus, the decrypted message is 2.

20. To sigh the hash of a massage $H(m) = 3$ using the private key d, we use the RSA signature algorithm. The process involves:

1. Signing

· The signature s is computed as:

$$s = H(m)^d \bmod P$$

Since $H(m) = 3$, the signature would be:

$$s = 3^d \bmod P$$

2. Varification:

The signature is verified using the public key (e, n) by checking if

$$s^e \bmod P = H(m)$$

If this holds true, the signature is valid.

Integrity and Authenticity:

Integrity: The signature ensure the message has not been altered. If the message is modified, the hash $H(m)$ will change and the verification will fail

Authenticity: The signature proves that the message was signed by the holder of the private key d, ensuring that the sender is who they claim to be.

IT: 23621

20

ⓘ To generate the ECDSA signature for a message (H) M = 8 using the elliptic curve $y^2 = x^3 + 7x + 10$ mod 37, base point G = (2,5) G = (2,5), G = (2,5) and private key d = 9d = 9d = 9. Follow these steps:

Public key calculation:

Compute Q = dG = 9GQ = dG = 9GQ = dG = 9G using elliptic curve scalar multiplication.

Signature Generation:

Given k = 3k = 3k = 3 (random nonce):

Compute the point kGkGkG:

Let rrr be the x-coordinate of kGkGkG modulo 37

Calculate s = k−1 (H(M) + r·d) mod 19s = k^{-1}
(H(M) + r\cdot d) \mod 19s = k−1 (H(M) + r·d) mod 19, where k−1k^{-1}
k−1 is the modular inverse of kkk modulo 19.

This generate the signature (r,s) (r,s) (r,s) ii

To generate the ECDSA signature for a message with the given parameters:

1. public key $Q = dG$: multiply the base point $G = (2, 5)$ by the private key $d = 9$ using elliptic curve scalar multiplication to get the public key Q

2. signature Generation:

- choose a random integer k.
- compute $P = kG$, where $P = (x_P, y_P)$
- compute $r = x_P \bmod n$
- compute $s = k^{-1}(H(m) + r \cdot d)$ mod n, where $H(m)$ is the hash of the message.

The result is the signature pair (r, s). This process ensures that only the holder of the private key can generate a valid signature.

iii) To generate the public key $Q = dG$ and and verify the signature follow these steps:

1. Elliptic curve set up:

The elliptic curve is given by: $y^2 = x^3 + 7x + 10 \pmod{37}$ with the base point $G = (2,5)$ order $n = 19$ and private key $d = 9$

2. Calculate the public key $Q = dG$

To compute the public key $Q = 9G$, use the scalar multiplication of the base point $G = (2,5)$ with $d = 9$. Scalar multiplication is done through successive doubling and adding operations on the elliptic curve.

Scalar multiplication steps:

We perform $9G$ through successive doubling and addition of points:

- $G = (2,5)$
- $2G$..
- $4G$
- $8G$
- Then $9G = 8G + G$

The result of $9G$ is the public key Q.

3- Signature varification

The signature verification using (r,s) involves checking is the following equation holds:

$s \cdot G \equiv r \cdot Q + s \cdot H(m) \pmod{37}$

Where:

· $H(m)$ is the hash of the message m,

· r and s are the Components of the signature.

· Q is the public key.

For the verification: we substitute the values of r, s, Q and $H(m)$ perform elliptic curve additions and check if both sides are congruent modulo 37.

Conclusion:

After performing scalar multiplication and verifying the signature with the above equation we can confirm if the signature is valid.

IT: 23621

21

(i) A secure hash Function such as those in the SHA (e.g. SHA-256) should have the Following key properties:

1. Pre-image Resistance: Given a hash value h, it should be computationally infeasible to find any input x such that Hash $(x) = h$.

2. Second pre-image Resistance: It should be infeasible to find two distinct inputs x_1 and x_2 such that hash $(x_1) =$ Hash (x_2)

3. Collision Resistance: It should be infeasible to find any two distinct inputs x_1 and x_2 such that Hash $(x_1) =$ Hash (x_2)

4. Deterministic: The same input always produces the same output hash.

5. Fast to compute: Hashing an input should be computationally efficient.

6. Small changes in input produce Large changes in output: Even a small change in the input should result in a significantly different hash output.

These properties ensure the security and integrity of hash Function in cryptographic systems.

IT: 23621

ii) The length of the output hash, such as 256 bits in SHA-256, directly impacts the security by increasing the difficulty of brute force attacks. A longer hash provides a larger number of possible outputs (2^{256} for SHA-256), making it computationally infeasible to find collisions (two different inputs producing a same hash) or preimages (finding a input given a hash. This enhances the algorithms resistance to attacks, ensuring data integrity and security.

iii

SHA Functions, like SHA-256 are widely used in the world wide applications For data integrity and security. In digital signature they hash the message and the hash is then signed with the private key, allowing verification of the messages authenticity and integrity. In block-chain systems, SHA 256 is used to hash transaction data and create blocks, ensuring immutability and linking blocks securely. The output hash serves as a unique identifier for each block, preventing tampering and maintaining the integrity of entire block chain.

Galois Field (finite fields) are mathmatical structures with a finate numbers of elements, where operations like addition, multiplication and their inverses are defined.

GF(P) refers to a field with a prime number of elements where operations are modulo p.

GF(2^n) is a binary field with 2^n elements commonly used in cryptography where operations are modulo a polynomial of degree n.

In eliptic curve cryptography (ECC), the underlying field (typically GF(P) or GF(2^n)) is used for the arithmetic of eliptic curves, providing efficient and secure key generation and signing algorithms.

In AES, GF(2^8) is used for operations in the key schedule and during the byte substitution (S-Box) step, ensuring strong diffusion and confusion.

Field Arithmatic is crucial in these systems as it ensures that operations are invertible preserving the integrity and security of the cryptographic algorithms.

23 IT: 23621

(i) The shortest vector problem (SVP) in lattice based cryptography involves finding the shortest non-zero vector in a lattice, given a set of basis vectors. It is considered a hard problem, meaning that no efficient algorithm is known to solve it in polynomial time, even with quantum computers. The difficulty of solving SVP forms the foundation for the security of lattice based cryptographic schemes, as attackers must overcome this computationally infeasible challenge to break the encryption. The resistance to quantum attacks makes lattice based cryptography a strong candidate for post-quantum security.

ii

Lattice based cryptography is based on problems like the shortest vector problem (SVP) and learning with errors (LWE) which are believed to be hard even for quantum computers. In contrast traditional cryptographic schemes like RSA and ECC (Elliptic curve cryptography) rely on the difficulty of factoring large integers or solving the discrete logarithm problem both of which are efficiently solveable by Shor's algorithms on a quantum computer. As a result while RSA and ECC are vulnerable to quantum attacks, lattice based schemes are resistant to them making lattice based cryptography a promising alternative for post quantum security.

IT: 23621

25

(i)

In an LWE-based signature scheme;

1. key Generation: Generate a private key (secret vector) and public key (derived using LWE)

2. signing: use the private key to sign the message by solving an LWE based problem with noise.

3. Verification: The verifier checks if the signature matches the message and public key using LWE relations.

security relies on the hardness of the LWE problem.

ii

To sign a message M using an LWE-based scheme with public key Pk, and private key sk, Follow these steps:

1. key Generation:

· Generate private key sk (a secret vector)

· Compute the public key Pk using sk and the LWE problem (public key is typically derived from noisy linear equations)

2. signing:

Take the message M and create a signature by solving an LWE problem using the private key sk. This typically involves adding noise to the solution and forming a noisy vector that satisfies the LWE relation.

iii

Quantum cryptography uses quantum mechanics to secure communication like with quantum key Distribution (QKD) ensuring security against quantum attacks. Lattice based cryptography on the other hand relies on hard lattice problems (like SVP and LWE) for security designed to resist quantum attacks while remaining a classical approach.

24

The key stream $k = (k_1, k_2, k_3, \dots)$ is generated by a Linear Feedback shift register (LFSR) using the recurrence relation:

$k_1 = c_1 k_t - 1 \oplus c_2 k_t - 2 \oplus \dots \oplus c_m k_1 - m$

where $c_1 c_2 \dots c_m$ are coefficients in GF(2) (Galois Field) and $\oplus$ denotes XOR (binary addition)

The recurrence relation means that each bit of the keystream is generated as a linear combination (XOR operation) of the previous m bits, determined by the coefficients

$c_1, c_2 \dots c_m$ The sequence is periodic and the length of the period depends on the length of the register m and the choice of coefficients.

3. verification:

- The verifier uses the public key pk, and the signature satisfies the LWE relation ensuring the message was signed correctly.

Role of LWE: The LWE problem enures security because its computationally hard to solve, even for quantum attackers. if an attackers tries to forget a signature, they must solve the hard LWE Problem which is believed to be infeasible.