SCHOOL OF CONTINUING AND DISTANCE EDUCATION

JAWAHARLAL NEHRU TECHNOLOGICALUNIVERSITY - HYDERABAD

Kukatpally, Hyderabad – 500 085, Telangana, India.

SIX MONTH ONLINE CERTIFICATE COURSES – 2023

CYBER SECURITY - ASSIGNMENT - 17

1Q) Explain Data Encryption Standard (DES) and Rivest-Shamir-Adleman (RSA) Algorithms.
Ans:

Certainly! Let's dive into the details of the Data Encryption Standard (DES) and the Rivest-Shamir-Adleman (RSA) algorithms.

Data Encryption Standard (DES)

Overview: DES is a symmetric key encryption algorithm that was developed in the 1970s by IBM and later adopted as a standard by the U.S. National Institute of Standards and Technology (NIST). It was widely used for securing sensitive data.

Key Features:

1. **Symmetric Key Encryption:** DES uses the same key for both encryption and decryption. This key must be kept secret between the communicating parties.
2. **Block Cipher:** DES operates on fixed-size blocks of data. Specifically, it encrypts data in 64-bit blocks.
3. **Key Length:** The key used in DES is 56 bits long, although the original key input is 64 bits (8 bits are used for parity checks).
4. **Feistel Structure:** DES uses a Feistel network, a structure that divides the data into two halves and processes them through multiple rounds of encryption.

Encryption Process:

1. **Initial Permutation (IP):** The 64-bit block of plaintext undergoes an initial permutation.
2. **16 Rounds of Feistel Function:** Each round consists of:

    1. Splitting the block into left (L) and right (R) halves.
    2. Applying a round function to the right half and combining it with the left half.
    3. Swapping the halves for the next round.

3. **Final Permutation (FP):** The block undergoes a final permutation that is the inverse of the initial permutation.

Security:

- DES was considered secure for many years, but with advancements in computing power, its 56-bit key length has become susceptible to brute-force attacks. It has largely been replaced by more secure algorithms such as AES (Advanced Encryption Standard).

Rivest-Shamir-Adleman (RSA)

Overview: RSA is an asymmetric key encryption algorithm invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. It is widely used for secure data transmission.

Key Features:

1. **Asymmetric Key Encryption:** RSA uses a pair of keys—a public key for encryption and a private key for decryption. The public key can be shared openly, while the private key must be kept secret.
2. **Based on Mathematical Problems:** RSA's security is based on the difficulty of factoring large prime numbers, specifically the product of two large primes.
3. **Variable Key Length:** RSA keys can vary in length, typically from 1024 to 4096 bits, with longer keys providing greater security.

Encryption Process:

Key Generation:

- Select two large prime numbers, ppp and qqq.
- Compute n=p×qn = p \times qn=p×q, which is used as part of both the public and private keys.
- Compute the totient ϕ(n)=(p−1)×(q−1)\phi(n) = (p-1) \times (q-1)ϕ(n)=(p−1)×(q−1).
- Choose an encryption exponent eee such that 1<e<ϕ(n)1 < e < \phi(n)1<e<ϕ(n) and gcd⁡(e,ϕ(n))=1\gcd(e, \phi(n)) = 1gcd(e,ϕ(n))=1.
- Determine the decryption exponent ddd as the modular inverse of eee modulo ϕ(n)\phi(n)ϕ(n).

Public Key: Consists of (e,n)(e, n)(e,n).

Private Key: Consists of (d,n)(d, n)(d,n).

Encryption:

- A message MMM is converted into an integer mmm such that 0≤m<n0 \le m < n0≤m<n.
- The ciphertext ccc is computed as c≡me(modn)c \equiv m^e \pmod{n}c≡me(modn).

Decryption:

- The original message mmm is recovered from the ciphertext ccc using the private key: m≡cd(modn)m \equiv c^d \pmod{n}m≡cd(modn).

Security:

- RSA's security depends on the computational difficulty of factoring the product of two large prime numbers. As of now, no efficient algorithm exists for factoring large semiprimes, making RSA secure with sufficiently large key sizes.

Comparison:

- **DES** is faster and suitable for encrypting large amounts of data but requires a shared secret key.
- **RSA** provides stronger security through public-key cryptography but is computationally intensive and slower, making it suitable for encrypting small amounts of data or for securely exchanging symmetric keys.

In summary, DES and RSA serve different purposes in the realm of cryptography, with DES being a symmetric algorithm used for fast encryption of data and RSA being an asymmetric algorithm used for secure key exchange and digital signatures.

2Q) Explain Diffie-Hellman Key Exchange Algorithm With an Example
Ans:

Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman Key Exchange algorithm is a method used to securely exchange cryptographic keys over a public channel. It allows two parties to generate a shared secret key, which can then be used for encrypting subsequent communications, without ever having to transmit the secret key itself.

Key Concepts:

- **Prime Number (p):** A large prime number used as a base.
- **Primitive Root (g):** A number whose powers modulo ppp generate all the numbers from 1 to p−1p-1p−1.

Steps in the Algorithm:

Public Agreement:

- Both parties agree on a large prime number ppp and a primitive root ggg. These values are public and can be known by anyone.

Private Keys:

- Each party selects a private key: aaa for Alice and bbb for Bob. These private keys are kept secret.

Public Keys:

- Alice computes her public key as A=gamod pA = g^a \mod pA=gamodp.
- Bob computes his public key as B=gbmod pB = g^b \mod pB=gbmodp.
- Alice and Bob exchange their public keys AAA and BBB over the public channel.

Shared Secret Calculation:

- Alice computes the shared secret key as s=Bamod ps = B^a \mod ps=Bamodp.
- Bob computes the shared secret key as s=Abmod ps = A^b \mod ps=Abmodp.
- Due to the properties of modular arithmetic, both computations result in the same shared secret key sss.

Example:

Public Agreement:

- Choose a large prime number p=23p = 23p=23.
- Choose a primitive root g=5g = 5g=5.

Private Keys:

- Alice selects a private key a=6a = 6a=6.
- Bob selects a private key b=15b = 15b=15.

Public Keys:

Alice computes her public key:

- A=gamod p=56mod 23=15625mod 23=8A = g^a \mod p = 5^6 \mod 23 = 15625 \mod 23 = 8A=gamodp=56mod23=15625mod23=8

Bob computes his public key:

- B=gbmod p=515mod 23=30517578125mod 23=19B = g^b \mod p = 5^{15} \mod 23 = 30517578125 \mod 23 = 19B=gbmodp=515mod23=30517578125mod23=19

Alice and Bob exchange public keys. Alice receives B=19B = 19B=19 from Bob, and Bob receives A=8A = 8A=8 from Alice.

Shared Secret Calculation:

- Alice computes the shared secret key: s=Bamod p=196mod 23=47045881mod 23=2s = B^a \mod p = 19^6

\mod 23 = 47045881 \mod 23 =
2s=Bamodp=196mod23=47045881mod23=2

- Bob computes the shared secret key:
  s=Abmod p=815mod 23=35184372088832mod 23=2s = A^b \mod p = 8^{15} \mod 23 = 35184372088832 \mod 23 = 2s=Abmodp=815mod23=35184372088832mod23=2

Both Alice and Bob have computed the same shared secret key s=2s = 2s=2. This key can now be used to encrypt further communications.

Security Considerations:

The security of the Diffie-Hellman algorithm relies on the difficulty of the Discrete Logarithm Problem. Given gamod pg^a \mod pgamodp and gbmod pg^b \mod pgbmodp, it is computationally infeasible to determine aaa or bbb without knowing the private keys.

To enhance security, larger prime numbers ppp and corresponding primitive roots ggg should be used, typically of 2048 bits or more.

The Diffie-Hellman key exchange allows secure key exchange over an insecure channel, forming the foundation for many encryption protocols such as TLS (Transport Layer Security).

3Q) Explain Digital Signature Algorithm (DSA) With an Example.
Ans:

Digital Signature Algorithm (DSA)

The Digital Signature Algorithm (DSA) is a Federal Information Processing Standard for digital signatures. It was proposed by the National Institute of Standards and Technology (NIST) in 1991 for use in their Digital Signature Standard (DSS). DSA is used to verify the authenticity and integrity of a message, software, or digital document.

Key Concepts:

1. **Prime Modulus (p):** A large prime number.
2. **Subprime (q):** A prime divisor of p−1p-1p−1.
3. **Generator (g):** A number less than ppp that generates a subgroup of order qqq in the multiplicative group of integers modulo ppp.
4. **Private Key (x):** A random integer chosen by the signer.
5. **Public Key (y):** Computed from the private key.

Steps in the DSA:

Parameter Generation:

1. Choose a large prime ppp.

2. Choose a prime divisor qqq of p−1p-1p−1.
3. Choose a generator ggg such that g=h(p−1)/qmod pg = h^{(p-1)/q} \mod pg=h(p−1)/qmodp for some hhh.

Key Generation:

1. Select a private key xxx such that 0<x<q0 < x < q0<x<q.
2. Compute the public key yyy as y=gxmod py = g^x \mod py=gxmodp.

Signature Generation:

1. Choose a random integer kkk such that 0<k<q0 < k < q0<k<q.
2. Compute rrr as r=(gkmod p)mod qr = (g^k \mod p) \mod qr=(gkmodp)modq.
3. Compute sss as s=(k−1(H(m)+xr))mod qs = (k^{-1}(H(m) + xr)) \mod qs=(k−1(H(m)+xr))modq, where H(m)H(m)H(m) is the hash of the message.

Signature Verification:

1. Compute the hash of the received message H(m)H(m)H(m).
2. Compute www as w=s−1mod qw = s^{-1} \mod qw=s−1modq.
3. Compute u1u1u1 as u1=(H(m)⋅ w)mod qu1 = (H(m) \cdot w) \mod qu1=(H(m)⋅ w)modq.
4. Compute u2u2u2 as u2=(r⋅ w)mod qu2 = (r \cdot w) \mod qu2=(r⋅ w)modq.
5. Compute vvv as v=((gu1⋅ yu2)mod p)mod qv = ((g^{u1} \cdot y^{u2}) \mod p) \mod qv=((gu1⋅ yu2)modp)modq.
6. The signature is valid if and only if v=rv = rv=r.

Example:

Parameter Generation:

1. Let's choose p=23p = 23p=23, q=11q = 11q=11, and g=2g = 2g=2 (for simplicity, although in practice, ppp and qqq are much larger).

Key Generation

1. Alice chooses a private key x=6x = 6x=6.
2. Alice computes her public key:
   y=gxmod p=26mod 23=64mod 23=18y = g^x \mod p = 2^6 \mod 23 = 64 \mod 23 = 18y=gxmodp=26mod23=64mod23=18

Signature Generation:

1. Alice wants to sign a message mmm with hash H(m)=9H(m) = 9H(m)=9.
2. She selects a random k=3k = 3k=3.

3. Compute rrr: r=(gkmod p)mod q=(23mod 23)mod 11=8mod 11=8r = (g^k \mod p) \mod q = (2^3 \mod 23) \mod 11 = 8 \mod 11 = 8r=(gkmodp)modq=(23mod23)mod11=8mod11=8
4. Compute sss: s=(k−1(H(m)+xr))mod qs = (k^{-1}(H(m) + xr)) \mod qs=(k−1(H(m)+xr))modq

    1. Compute k−1mod qk^{-1} \mod qk−1modq: k=3k = 3k=3, so k−1=4k^{-1} = 4k−1=4 (since 3⋅4≡1mod 113 \cdot 4 \equiv 1 \mod 113⋅4≡1mod11).

    s=(4⋅(9+6⋅8))mod 11=(4⋅(9+48))mod 11=(4⋅57)mod 11=228mod 11=8s = (4 \cdot (9 + 6 \cdot 8)) \mod 11 = (4 \cdot (9 + 48)) \mod 11 = (4 \cdot 57) \mod 11 = 228 \mod 11 = 8s=(4⋅(9+6⋅8))mod11=(4⋅(9+48))mod11=(4⋅57)mod11=228mod11=8

5. The signature for the message is (r,s)=(8,8)(r, s) = (8, 8)(r,s)=(8,8).

Signature Verification:

1. Bob receives the message mmm and the signature (r,s)=(8,8)(r, s) = (8, 8)(r,s)=(8,8).
2. Compute www: w=s−1mod q=8−1mod 11=7(since 8⋅7≡1mod 11)w = s^{-1} \mod q = 8^{-1} \mod 11 = 7 \quad (\text{since } 8 \cdot 7 \equiv 1 \mod 11)w=s−1modq=8−1mod11=7(since 8⋅7≡1mod11)
3. Compute u1u1u1: u1=(H(m)⋅w)mod q=(9⋅7)mod 11=63mod 11=8u1 = (H(m) \cdot w) \mod q = (9 \cdot 7) \mod 11 = 63 \mod 11 = 8u1=(H(m)⋅w)modq=(9⋅7)mod11=63mod11=8
4. Compute u2u2u2: u2=(r⋅w)mod q=(8⋅7)mod 11=56mod 11=1u2 = (r \cdot w) \mod q = (8 \cdot 7) \mod 11 = 56 \mod 11 = 1u2=(r⋅w)modq=(8⋅7)mod11=56mod11=1
5. Compute vvv: v=((gu1⋅yu2)mod p)mod q=((28⋅181)mod 23)mod 11v = ((g^{u1} \cdot y^{u2}) \mod p) \mod q = ((2^8 \cdot 18^1) \mod 23) \mod 11v=((gu1⋅yu2)modp)modq=((28⋅181)mod23)mod11

    1. Compute 28mod 23=256mod 23=32^8 \mod 23 = 256 \mod 23 = 328mod23=256mod23=3
    2. Compute (3⋅18)mod 23=54mod 23=8(3 \cdot 18) \mod 23 = 54 \mod 23 = 8(3⋅18)mod23=54mod23=8
    3. Thus, v=8mod 11=8v = 8 \mod 11 = 8v=8mod11=8

Since v=rv = rv=r, the signature is valid.

Summary:

DSA provides a way to authenticate the sender and verify the integrity of a message. The example demonstrates the core steps: parameter and key generation, signature

creation, and signature verification. In practice, much larger values of ppp and qqq are used for enhanced security.

4Q) Explain the Following Types of One-time Password (OTP) Algorithms with Examples:
a. Time-based OTP (TOTP)
b. HMAC-based OTP (HOTP)
Ans:

One-Time Password (OTP) Algorithms

One-Time Passwords (OTPs) are dynamic, unique codes generated for authentication purposes, typically valid for a short period or a single session. Two common types of OTP algorithms are Time-based OTP (TOTP) and HMAC-based OTP (HOTP).

a. Time-based OTP (TOTP)

Overview: TOTP is an extension of HOTP that generates a password based on the current time. It ensures that the generated OTP is valid only for a short time window, adding an extra layer of security.

Key Components:

1. **Shared Secret:** A base32-encoded secret key shared between the server and the client.
2. **Time Interval:** The time step (typically 30 seconds) which determines how long each OTP is valid.
3. **Hash Function:** Usually, HMAC-SHA-1, although others can be used.

Algorithm Steps:

1. **Current Time:** Retrieve the current Unix time (number of seconds since January 1, 1970).
2. **Time Counter:** Divide the current time by the time step to get a time counter value.
3. **Generate OTP:** Use the HOTP algorithm with the time counter value as the moving factor.

Example:

- **Shared Secret:** JBSWY3DPEHPK3PXP
- **Time Step:** 30 seconds
- **Current Unix Time:** 1628678400 (for example)

Steps:

1. **Time Counter:** Tc=⌊ 162867840030⌋ =54289280T_c = \left\lfloor \frac{1628678400}{30} \right\rfloor = 54289280Tc =⌊ 301628678400 ⌋ =54289280
2. **HOTP with Time Counter:**

- Convert the time counter to an 8-byte array.
- Use HMAC-SHA-1 with the shared secret and the time counter.
- Apply the dynamic truncation to get a 6-8 digit OTP.

If we get a hash value (for simplicity, in hex):
0x31c4b21f0b7ab1d5fa471d1db7e1847e7b16b7ec

The truncated value will be converted to a 6-digit OTP, for example: 654321.

b. HMAC-based OTP (HOTP)

Overview: HOTP generates a password based on a counter value, incremented with each new OTP request. This counter ensures that each OTP is unique.

Key Components:

1. **Shared Secret:** A base32-encoded secret key shared between the server and the client.
2. **Counter:** A counter value incremented with each new OTP request.
3. **Hash Function:** Usually, HMAC-SHA-1.

Algorithm Steps:

1. **Counter Value:** Retrieve or increment the counter value.
2. **Generate OTP:** Apply the HMAC-SHA-1 hash function with the secret key and the counter value.
3. **Dynamic Truncation:** Extract a portion of the HMAC result to generate a 6-8 digit OTP.

Example:

- **Shared Secret:** JBSWY3DPEHPK3PXP
- **Counter Value:** 1

Steps:

Convert Counter:

- Convert the counter value to an 8-byte array: 0x0000000000000001.

HMAC-SHA-1:

- Use HMAC-SHA-1 with the shared secret and the counter value.
- Assume the hash output (in hex) is: 0x4fd1c5e60022ba9d0f710cf76dc0e1e8e9b6a9b7.

Dynamic Truncation:

- Extract a 4-byte dynamic binary code from the hash:

- Use the last nibble of the hash as an offset (7 in this case).
- Extract 4 bytes starting at the offset: 0x22ba9d0f.

Convert to Integer:

- Convert 0x22ba9d0f to an integer: 582716303.

Generate OTP:

- Calculate the OTP by taking the integer modulo 10^6 (for a 6-digit OTP): 582716303 % 1000000 = 716303.

The OTP generated is 716303.

Summary:

TOTP:

- Based on the current time.
- Changes at fixed intervals (e.g., every 30 seconds).
- Adds a temporal factor, enhancing security.

HOTP:

- Based on a counter value.
- Changes with each new request.
- Ensures uniqueness based on sequential usage.

Both algorithms are widely used for two-factor authentication, with TOTP being more common due to its time-based validity, making it ideal for real-time authentication scenarios.