Cyber Laws and Security Management

Assignment-17

N Ravinder Reddy

Roll No: 2406CYS106

Unit II

Fundamentals of Algorithms in Cybersecurity Assignment Questions

Q1. Explain Data Encryption Standard (DES) and Rivest-Shamir-Adleman (RSA) Algorithms.

Ans:

The Data Encryption Standard (DES) is a symmetric-key algorithm for the encryption of digital data. Developed in the early 1970s by IBM and later adopted by the U.S. National Institute of Standards and Technology (NIST) in 1977, DES became the federal standard for data encryption.

Key Features of DES

1. Symmetric Key Encryption: DES uses the same key for both encryption and decryption. This means that both the sender and the receiver must have access to the same secret key.
2. Block Cipher: DES is a block cipher, which means it processes data in fixed-size blocks. Specifically, DES operates on 64-bit blocks of data.
3. Key Length: DES uses a 56-bit key, which is actually derived from an initial 64-bit key by removing every 8th bit (used for parity checking).
4. Feistel Structure: DES employs a Feistel network structure, which involves multiple rounds of processing (16 rounds in the case of DES). Each round uses a different 48-bit subkey derived from the original key.

The DES Algorithm Process

The DES algorithm consists of several steps:

1. Initial Permutation (IP): The 64-bit plaintext block is permuted using a predefined permutation table.

2. Rounds of Encryption: The permuted block is then split into two 32-bit halves. Each half is processed through 16 rounds of the Feistel function. Each round consists of:
- Expansion: Expands the 32-bit half block to 48 bits using an expansion permutation.
- Key Mixing: XORs the expanded block with a 48-bit round key.
- Substitution: Uses S-boxes (substitution boxes) to substitute the 48-bit result with a 32-bit block.
- Permutation: Applies a predefined permutation to the 32-bit block.
- Combination: The result is XORed with the other 32-bit half.

3. Final Permutation (FP): After the 16 rounds, the two halves are concatenated and a final permutation (inverse of the initial permutation) is applied to produce the ciphertext.

Security Concerns and Successors

While DES was considered secure for some time, advances in computing power and cryptanalysis techniques eventually made it vulnerable to brute-force attacks. The main security issues include:

- Short Key Length: The 56-bit key length is relatively short by modern standards, making it susceptible to brute-force attacks.
- Known Cryptographic Attacks: Several theoretical attacks, such as differential cryptanalysis, have been developed against DES.

Due to these concerns, DES has been largely replaced by the Advanced Encryption Standard (AES) and Triple DES (3DES), which involves applying DES encryption three times with different keys, effectively increasing the key length and security.

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests some data.

2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Advantages:

- Security: RSA algorithm is considered to be very secure and is widely used for secure data transmission.
- Public-key cryptography: RSA algorithm is a public-key cryptography algorithm, which means that it uses two different keys for encryption and decryption. The public key is used to encrypt the data, while the private key is used to decrypt the data.
- Key exchange: RSA algorithm can be used for secure key exchange, which means that two parties can exchange a secret key without actually sending the key over the network.
- Digital signatures: RSA algorithm can be used for digital signatures, which means that a sender can sign a message using their private key, and the receiver can verify the signature using the sender's public key.
- Speed: The RSA technique is suited for usage in real-time applications since it is quite quick and effective.
- Widely used: Online banking, e-commerce, and secure communications are just a few fields and applications where the RSA algorithm is extensively developed.

Disadvantages:

- Slow processing speed: RSA algorithm is slower than other encryption algorithms, especially when dealing with large amounts of data.
- Large key size: RSA algorithm requires large key sizes to be secure, which means that it requires more computational resources and storage space.
- Vulnerability to side-channel attacks: RSA algorithm is vulnerable to side-channel attacks, which means an attacker can use information leaked through side channels such as power consumption, electromagnetic radiation, and timing analysis to extract the private key.
- Limited use in some applications: RSA algorithm is not suitable for some applications, such as those that require constant encryption and decryption of large amounts of data, due to its slow processing speed.
- Complexity: The RSA algorithm is a sophisticated mathematical technique that some individuals may find challenging to comprehend and use.
- Key Management: The secure administration of the private key is necessary for the RSA algorithm, although in some cases this can be difficult.
- Vulnerability to Quantum Computing: Quantum computers have the ability to attack the RSA algorithm, potentially decrypting the data.

Q2. Explain Diffie-Hellman Key Exchange Algorithm With an Example

Ans:

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.

- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Step-by-Step explanation is as follows:

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated =	Key generated =
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key =	Generated Secret Key =
Algebraically, it can be shown that	
Users now have a symmetric secret key to encrypt	

Example:

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x =(9^4 mod 23) = (6561 mod 23) = 6
Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys
Alice: ka = y^a mod p = 65536 mod 23 = 9
Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

Q3. Explain Digital Signature Algorithm (DSA) With an Example.

Ans:

A Digital Signature is a verification method made by the recipient to ensure the message was sent from the authenticated identity. When a customer signs a check, the bank must verify that he issued that specific check. In this case, a signature on a document acts as a sign of authentication and verifies that the document is authentic.

Suppose we have:

- Alice is the entity that sends a message or initiates communication.
- Bob represents the recipient or receiver of the message.
- Eve represents an eavesdropper or adversary who may attempt to intercept or tamper with the communication.

In Public Key cryptography (also known as Asymmetric cryptography), the communication process is as follows:

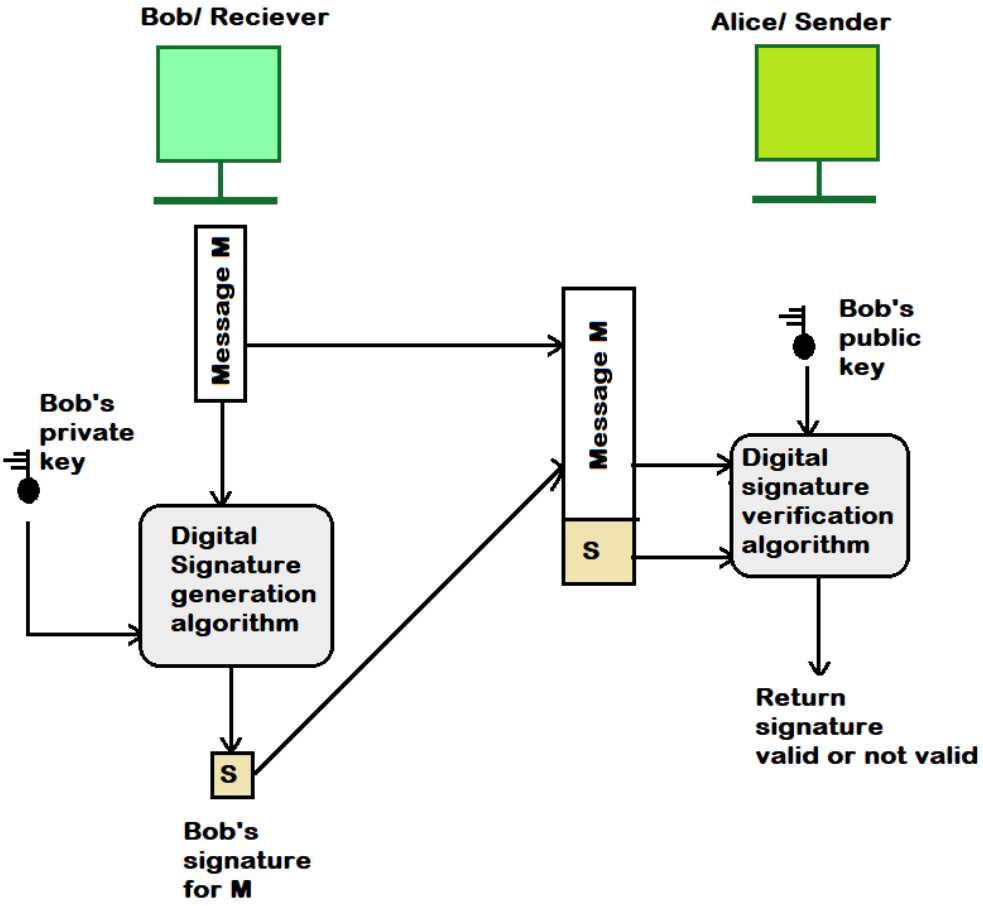
- Alice encrypts the message using Bob's public key.
- The encrypted message reaches Bob.
- Bob decrypts the message sent by Alice using his private key.

Now, suppose when Alice sends a message to Bob, then Bob will check if the sender is authentic; to ensure that it was Alice who sent the message, not Eve. For this, Bob can ask Alice to sign the message electronically. So we can say that an electronic signature can prove that Alice is authentic and is

the one sending the message. We called this type of signature a digital signature.

What is Digital Signature?

Digital Signature is a verification method. Digital signatures do not provide confidential communication. If you want to achieve confidentiality, both the message and the signature must be encrypted using either a secret key or a public key cryptosystem. This additional layer of security can be incorporated into a basic digital signature scheme.



Q4. Explain the Following Types of One-time Password (OTP) Algorithms with Examples:

a. Time-based OTP (TOTP)

Time-based One-Time Passwords (TOTPs) are temporary passcodes used to fortify the user authentication processes. They represent a dynamic and time-

sensitive approach to verifying identity—a cornerstone of robust security strategies.

So how do they work? TOTPs are generated by algorithms that synchronize between the authentication server and your device—typically a smartphone. This synchronization hinges on two key components: a shared secret key unique to each user's account and the current time.

TOTPs are a type of two-factor authentication (2FA) that adds significant depth to security defenses. 2FA and multifactor authentication (MFA) insist not just on something you know (like your regular password) but also on something you have. In most cases, that's your smartphone, which is synchronized to receive unique, continuously refreshing passcodes, making it extremely difficult to hack into an account.

What sets TOTP apart is its temporary nature. These passwords, typically ranging between five and eight digits, have an expiration period after which they're no longer valid—often just 30 seconds to a minute. Once expired, a new password is created based on the ongoing ticking of time combined with an original secret key.

By leveraging what is essentially akin to using disposable keys for every login attempt—keys only available through devices typically kept close at hand—we create a moving target that is much more challenging for malicious actors to hit than static passwords alone.

b. HMAC-based OTP (HOTP)

HMAC-based One-Time Password (HOTP) is a standardized algorithm that generates one-time passwords based on the Hash-based Message Authentication Code (HMAC). It is commonly used for two-factor authentication (2FA) to provide an additional layer of security beyond just a username and password. Here's an overview of how HOTP works:

How HOTP Works

1. Key Components: - Secret Key (K): A shared secret key between the server and the client. - Counter (C): A counter value that is incremented with each OTP generation.

2. HMAC Generation: - The HOTP value is generated by taking an HMAC hash of the counter value using the shared secret key. The hash function used is typically SHA-1, but can also be SHA-256 or SHA-512.

3. Truncation: - The HMAC output is large, so it is truncated to produce a shorter, more manageable OTP. This is usually done by extracting a subset of the HMAC output bits.

4. Dynamic Truncation: - The OTP is further shortened to a desired length, often 6-8 digits. This is done using a dynamic truncation method, which involves selecting a specific part of the HMAC result.

Algorithm Steps

1. HMAC Calculation: - Compute the HMAC hash of the counter value using the secret key: \[HMAC(K, C) \] - If SHA-1 is used, this results in a 20-byte (160-bit) output.

2. Dynamic Truncation: - Use the 4 least significant bits of the last byte of the HMAC result as an offset. - Extract a 4-byte string starting at the offset.

3. Truncate and Modulo Operation:

- Convert the 4-byte string to an integer. - Take the modulo of this integer with \(10^d\), where \(d\) is the desired number of digits for the OTP (usually 6 or 8).

4. Generate OTP:

- The result is a numeric code that is the OTP.

Security Features –

Time-Independent

Unlike Time-based OTP (TOTP), HOTP does not rely on the current time. Instead, it uses an incrementing counter, making it suitable for systems where the exact time synchronization between client and server is impossible. –

Replay Protection**: Each OTP can only be used once, preventing replay attacks.

Usage HOTP is often used in hardware tokens, software applications, and various security systems to provide an additional layer of authentication. It is standardized by the Internet Engineering Task Force (IETF) in RFC 4226.

Example Here's a simplified example of generating a HOTP value:

1. Inputs: - Secret Key (K): "12345678901234567890" - Counter (C): 1

2. HMAC Calculation**: - Using SHA-1, compute the HMAC of the counter value.

3. Dynamic Truncation:

-Assume the HMAC result is: `0x1f8698690e02ca16618550ef7f19da8e945b555a`

- Offset (last nibble of the result): `0xa` (10 in decimal).

- Extract 4 bytes starting at offset: `0x50ef7f19`.

4. Truncate and Modulo:

- Convert `0x50ef7f19` to decimal: `1357872921`.

- Modulo \(10^6\) (for a 6-digit OTP): `1357872921 % 1000000 = 872921`. 5. **Generated OTP:

- The OTP is `872921`. This OTP can now be used for authentication.