

Advanced Data Security

Chapter 5

Public key encryption is more secure than symmetric encryption from cryptanalysis. When using public key encryption, key distribution is essential.

Asymmetric Keys: public + private key encryption- decryption / signature generation-verification /....

Public key (asymmetric) cryptographic algorithm: a cryptographic algorithm that uses two related keys (pub priv). Property of it: deriving the private key from the public one is infeasible.

PKI (pub key infrastructure)

Public key cryptography was made for the purpose of defeating the problem of the symmetric encryption (Key distribution and digital signature)

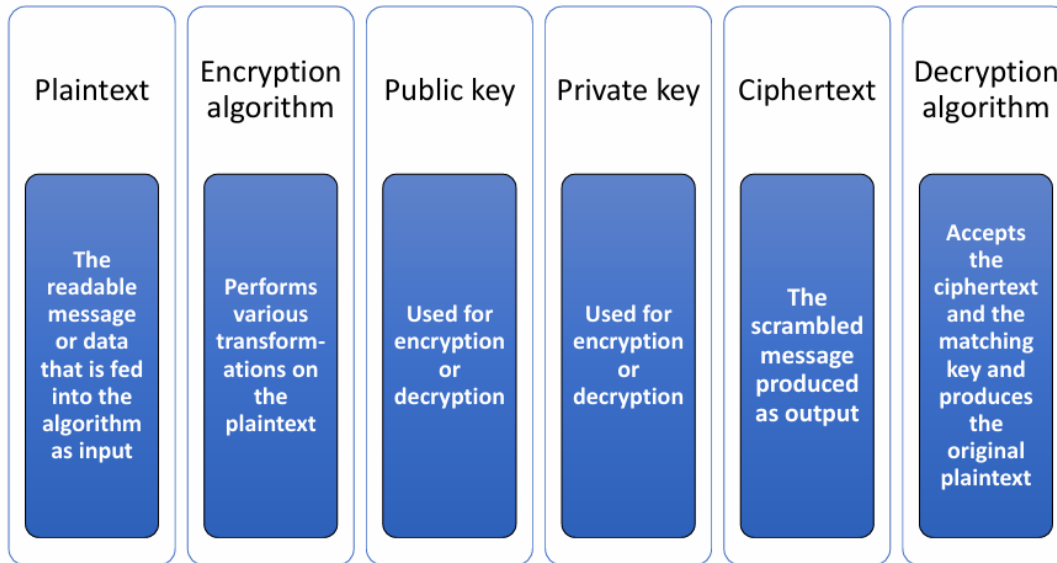
-Key distribution: Not to have to trust a key distribution center with our key to have a secure communication.

-Digital signature: To verify that a message comes intact from the claimed sender.

Conventional Encryption	Public-Key Encryption
<i>Needed to Work:</i> 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. <i>Needed for Security:</i> 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if the key is kept secret. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	<i>Needed to Work:</i> 1. One algorithm is used for encryption and a related algorithm for decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). <i>Needed for Security:</i> 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if one of the keys is kept secret. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

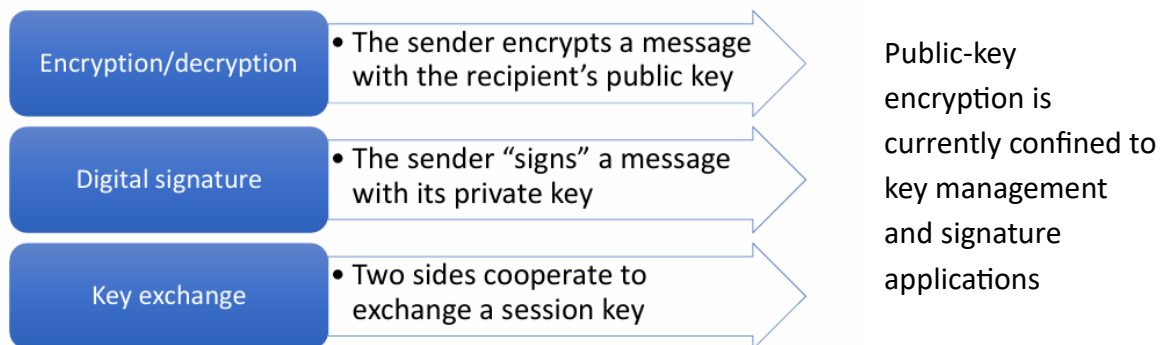
the use of public-key encryption to provide authentication and confidentiality

- A public-key encryption scheme has six ingredients:



It is possible to provide both the authentication function and confidentiality by a double use of the public-key scheme. Disadvantage of this approach is that the public-key algorithm(complex), must be exercised four times rather than two in each communication.

- Public-key cryptosystems can be classified into three categories:



- Some algorithms are suitable for all three applications, whereas others can be used only for one or two

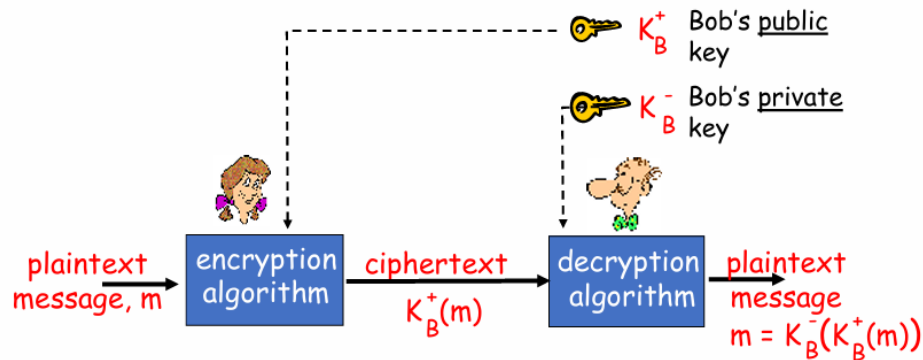
Public Key requirements:

- Computationally easy for a party to generate a pair of keys
- Comp easy for a sender to **generate the cyphertext** knowing the **public key** and the message to be encrypted
- Comp easy for a receiver B to **decrypt** the resulting cyphertext using the **private key** to recover the original message.
- Comp infeasible for an adversary to determine the private key knowing the public key. Or recover the original message knowing the public key and ciphertext
- The two keys can be applied in either order

Public Key Cryptanalysis:

- Public key encryption is vulnerable to a brute force attack. What we can do is use large keys. The key size must be small enough for practical encryption and decryption
- Another form of attack is to find some way to compute the private key given the public key.
- Finally, there is a probable-message attack. This attack can be thwarted by appending some random bits to simple messages

Public key cryptography



encrypting a message is equivalent to encrypting a number. To encrypt m , we encrypt the corresponding number, which gives a new number (the cyphertext).

Creaing public/private key pair:

1. Choose two large prime numbers p and q
2. Compute $n = p \times q$
3. Compute $z = (p-1) \times (q-1)$
4. $e < n$ and e and z are relatively prime
5. d such as $e \times d - 1$ is exactly divisible by z
6. Public key: (e, n) private key (d, n)

Encryption/ decryption:

1. Given e, n and d, n as computed above:
2. To encrypt the message $M (<n)$: $C = M^e \text{ mod } n$
3. To decrypt the received bit pattern c : $M = C^d \text{ mod } n$

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\text{gcd}(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \text{ (mod } \phi(n))$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \text{ mod } n$

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \text{ mod } n$

Exercises:

RSA Algorithm: General Steps

Given:

- Two distinct primes: p, q
- Public exponent e (relatively prime to $\phi(n)$)
- Message M

We compute:

1. Compute modulus n

$$n = p \times q$$

2. Compute Euler's totient $\phi(n)$

$$\phi(n) = (p - 1)(q - 1)$$

3. Choose public exponent e

$$1 < e < \phi(n) \quad \text{and} \quad \gcd(e, \phi(n)) = 1$$

4. Compute private exponent d

$$d \equiv e^{-1} \pmod{\phi(n)}$$

(Find the modular inverse of e modulo $\phi(n)$)

5. Public key = (e, n)

Private key = (d, n)

Encryption

To encrypt message M :

$$C = M^e \pmod{n}$$

Decryption

To decrypt ciphertext C :

$$M = C^d \pmod{n}$$

Let's solve each case in detail:

(a) $p = 3, q = 11, e = 7, M = 5$

1. $n = 3 \times 11 = 33$
2. $\phi(n) = (3 - 1)(11 - 1) = 2 \times 10 = 20$
3. $e = 7, \gcd(7, 20) = 1$ 
4. Find $d \equiv 7^{-1} \pmod{20}$
 - Try $7 \cdot d \pmod{20} = 1$
 - $7 \times 3 = 21 \pmod{20} = 1$
 -  $d = 3$

Encryption:

$$C = 5^7 \pmod{33} = 78125 \pmod{33} = 14$$

Decryption:

$$M = 14^3 \mod 33 = 2744 \mod 33 = 5$$

✓ Final answer: **C = 14, M = 5**

(b) $p = 5, q = 11, e = 3, M = 9$

1. $n = 55$
2. $\phi(n) = (4)(10) = 40$
3. $e = 3, \gcd(3, 40) = 1$ ✓
4. Find $d \equiv 3^{-1} \mod 40$
 $\rightarrow$ Try $3 \times 27 = 81 \mod 40 = 1$ ✓
 $\rightarrow d = 27$

Encryption:

$$C = 9^3 \mod 55 = 729 \mod 55 = 14$$

Decryption:

$$M = 14^{27} \mod 55 = 9 \text{ (use modular exponentiation)}$$

✓ Final answer: **C = 14, M = 9**

Encryption:

$$C = 8^{17} \mod 77 = 27$$

Decryption:

$$M = 27^{53} \mod 77 = 8$$

✓ Final answer: **C = 27, M = 8**

(c) $p = 7, q = 11, e = 17, M = 8$

1. $n = 77$
2. $\phi(n) = 6 \times 10 = 60$
3. $\gcd(17, 60) = 1$ ✓
4. Find $d \equiv 17^{-1} \mod 60$
 $\rightarrow$ Extended Euclidean Algorithm gives $d = 53$

(d) $p = 11, q = 13, e = 11, M = 7$

1. $n = 143$
2. $\phi(n) = 10 \times 12 = 120$
3. $\gcd(11, 120) = 1$ ✓
4. Find $d \equiv 11^{-1} \mod 120$
 $\rightarrow 11 \times 11 = 121 \mod 120 = 1$
 $\rightarrow d = 11$

Encryption:

$$C = 7^{11} \mod 143 = 128$$

Decryption:

Decryption:

$$M = 128^{11} \mod 143 = 7$$

✓ Final answer: C = 128, M = 7

(e) $p = 17, q = 31, e = 7, M = 2$

1. $n = 527$

2. $\phi(n) = 16 \times 30 = 480$

3. $\gcd(7, 480) = 1$ ✓

4. Find $d \equiv 7^{-1} \mod 480$

→ Try multiples:

$$7 \times 343 = 2401 \mod 480 = 1$$

→ ✓ $d = 343$

Encryption:

$$C = 2^7 \mod 527 = 128$$

Decryption:

$$M = 128^{343} \mod 527 = 2$$

✓ Final answer: C = 128, M = 2

✓ Exercise 2:

In a public-key system using RSA, you intercept the ciphertext $C = 10$, public key: $e = 5, n = 35$. What is the plaintext M ?

♦ Step 1: Factor $n = 35$

$$p = 5, \quad q = 7 \Rightarrow n = 5 \times 7 = 35$$

♦ Step 2: Compute $\phi(n)$

$$\phi(n) = (p - 1)(q - 1) = 4 \times 6 = 24$$

♦ Step 3: Compute private key $d \equiv e^{-1} \mod \phi(n)$

We want:

$$5 \cdot d \equiv 1 \mod 24$$

Try values:

- $5 \times 5 = 25 \mod 24 = 1$

✓ So, $d = 5$

♦ Step 4: Decrypt

$$M = C^d \mod n = 10^5 \mod 35$$

Compute:

- $10^2 = 100 \mod 35 = 100 - 2 \times 35 = 30$
- $10^4 = 30^2 = 900 \mod 35 = 900 - 25 \times 35 = 25$
- $10^5 = 10 \cdot 25 = 250 \mod 35 = 250 - 7 \times 35 = 250 - 245 = 5$

✓ Plaintext $M = 5$

✓ Exercise 3:

RSA public key: $e = 31, n = 3599$.
Find the private key d

♦ Step 1: Factor $n = 3599$

We try to factor it:

- $\sqrt{3599} \approx 60$
Try small primes near 60:
- $59 \times 61 = 3599$ ✓

So:

- $p = 59, \quad q = 61$

♦ Step 2: Compute $\phi(n)$

$$\phi(n) = (p - 1)(q - 1) = 58 \times 60 = 3480$$

♦ Step 3: Find $d \equiv e^{-1} \mod 3480$

We need to find the modular inverse of $31 \mod 3480$
Using the **Extended Euclidean Algorithm**:

We solve:

$$31 \cdot d \equiv 1 \mod 3480$$

Apply EEA:

- $3480 = 112 \cdot 31 + 8$
- $31 = 3 \cdot 8 + 7$
- $8 = 1 \cdot 7 + 1$
- $7 = 7 \cdot 1 + 0$

Now back-substitute:

- $1 = 8 - 1 \cdot 7$
- $= 8 - 1 \cdot (31 - 3 \cdot 8) = 4 \cdot 8 - 1 \cdot 31$
- $= 4(3480 - 112 \cdot 31) - 1 \cdot 31 = 4 \cdot 3480 - 449 \cdot 31$

So:

$$1 = 4 \cdot 3480 - 449 \cdot 31 \Rightarrow -449 \cdot 31 \equiv 1 \mod 3480$$

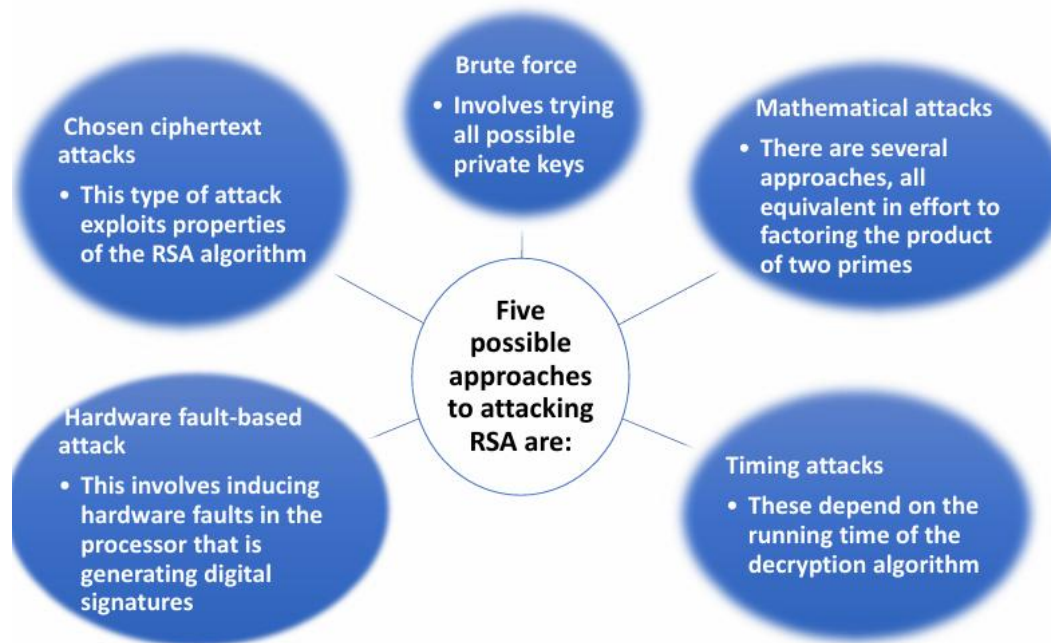
Thus:

$$d \equiv -449 \mod 3480 = 3031$$

✓ Private key: $d = 3031$

With a very small public key, RSA becomes vulnerable to a simple attack.

The Security of RSA



- **Timing attacks:** Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages • Are applicable not just to RSA but to other public-key cryptography systems
Countermeasure: Constant exponentiation time, Random Delay, Blinding
- **Fault-Based Attack:** An attack on a processor that is generating RSA digital signatures • Induces faults in the signature computation by reducing the power to the processor • The faults cause the software to produce invalid signatures which can then be analyzed by the attacker to recover the private key
- **Chosen Ciphertext Attack (CCA)** • The adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key • Thus the adversary could select a plaintext, encrypt it with the target's public key, and then be able to get the plaintext back by having it decrypted with the private key



Diffie-Hellman Key Exchange Summary:

It is a method for **two users to securely exchange a shared secret key** over an insecure channel. They use a **common public prime q** and a **primitive root a** .

Formula Overview:

- **Public parameters (agreed on by both):**
 - A prime number q
 - A primitive root a of q
- **Each user picks a private key:**
 - A picks private X_A
 - B picks private X_B
- **Each user computes their public key:**
 - $Y_A = a^{X_A} \mod q$
 - $Y_B = a^{X_B} \mod q$
- **Each user computes the shared secret:**
 - A computes $K = Y_B^{X_A} \mod q$
 - B computes $K = Y_A^{X_B} \mod q$

Both get the **same shared key K** , even though they never sent their private key!

YA: public key of A XA: private key of A



Given:

- $q = 71$
- $a = 7$
- User A: $X_A = 5$
- User B: $X_B = 12$

♦ a. Compute A's public key Y_A

$$Y_A = a^{X_A} \mod q = 7^5 \mod 71$$

$$7^5 = 16807, \quad 16807 \mod 71 = \boxed{45}$$

✅ A's public key is 45

♦ b. Compute B's public key Y_B

$$Y_B = a^{X_B} \mod q = 7^{12} \mod 71$$

Use exponentiation by squaring:

- $7^2 = 49$
- $7^4 = 49^2 = 2401 \mod 71 = 2401 \mod 71 = 49$
- $7^8 = (7^4)^2 = 49^2 = 2401 \mod 71 = 49$
- $7^{12} = 7^8 \cdot 7^4 = 49 \cdot 49 = 2401 \mod 71 = \boxed{49}$

✓ B's public key is 49

♦ c. Compute the shared secret key K

Both users should compute the **same** key independently.

User A computes:

$$K = Y_B^{X_A} \mod q = 49^5 \mod 71$$

- $49^2 = 2401$
- $49^4 = (49^2)^2 = 2401^2 = \dots \mod 71$

But an easier way is just:

$$49^5 = 49 \cdot 49 \cdot 49 \cdot 49 \cdot 49 = 49^5 = ?$$

Let's compute:

- $49^2 = 2401$
- $2401 \mod 71 = 2401 - 33 \times 71 = 2401 - 2343 = 58$
- Now $49^4 = 58^2 = 3364 \mod 71 = 3364 - 47 \times 71 = 3364 - 3337 = 27$
- $49^5 = 27 \times 49 = 1323 \mod 71 = 1323 - 18 \times 71 = 1323 - 1278 = 45$

✓ Shared key computed by A: 45

User B computes:

$$K = Y_A^{X_B} \mod q = 45^{12} \mod 71$$

We expect the same result. To check:

- You can repeat exponentiation by squaring, or trust the math because the algorithm guarantees:

$$a^{X_A X_B} \mod q = a^{X_B X_A} \mod q \Rightarrow K = 7^{5 \times 12} \mod 71 = 7^{60} \mod 71 = \boxed{45}$$

✓ Shared secret key is 45

Chapter 6

Cryptographic Hash Functions, MACs, and Digital Signatures:

- **Cryptographic Hash Functions:** These functions take an input (message) and produce a **fixed-size** output called a hash value or message digest. This hash value is unique for each message, and even a small change in the message will result in a significantly different hash value. This property is essential for data integrity, ensuring that a message has not been tampered with. Message authentication assures that data received are exactly as sent. The sender computes a hash value and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value. If there is a mismatch, the receiver knows that the message (or the hash value) has been altered.

A hash function H accepts a **variable-length block of data** M as input and produces a **fixed-size** hash value

- $h = H(M)$
- Principal object is **data integrity**

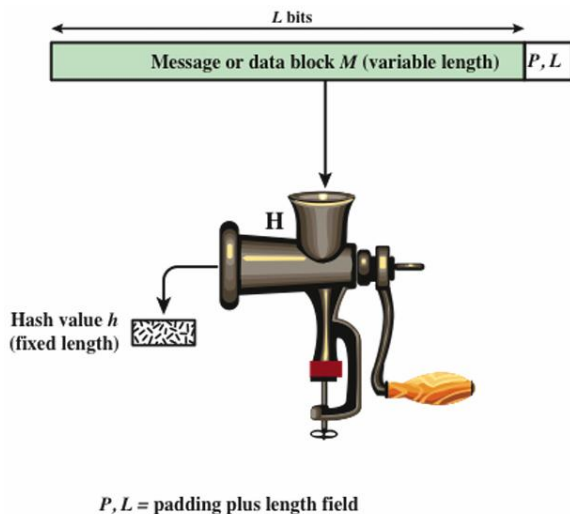


Figure 11.1 Cryptographic Hash Function; $h = H(M)$

Requirements and Security

Before proceeding, we need to define two terms.

Preimage

- x is the preimage of h for a hash value $h = H(x)$
- Is a data block whose hash function, using the function H , is h
- Because H is a many-to-one mapping, for any given hash value h , there will in general be multiple preimages

Collision

- Occurs if we have $x \neq y$ and $H(x) = H(y)$
- Because we are using hash functions for data integrity, collisions are clearly undesirable

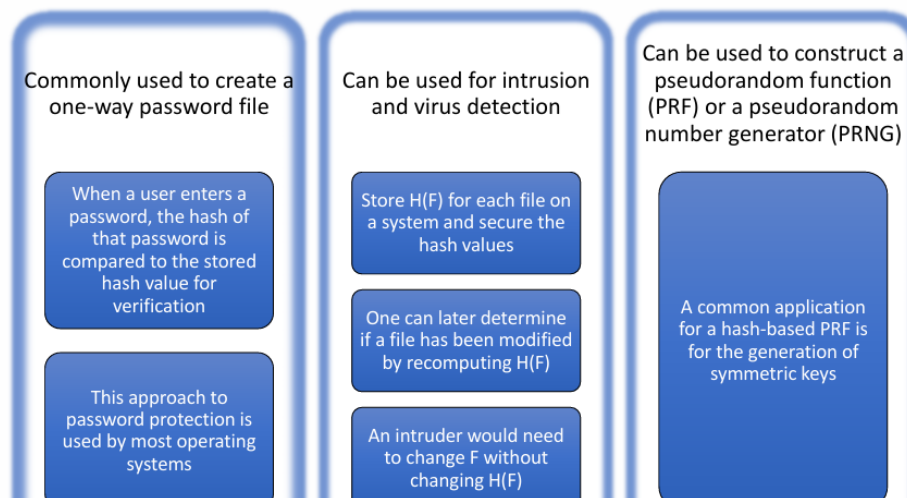


Preimage resistant (one-way property)	For any given hash value h , it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness

There are two categories of attacks on hash functions: brute-force attacks (only depends on bit length, Method is to pick values at random and try each one until a collision occurs) and cryptanalysis (An attack based on weaknesses in a particular cryptographic algorithm Seek to exploit some property of the algorithm to perform some attack)

- **Message Authentication Codes (MACs):** MACs are used to ensure both data integrity and authenticity. They are generated using a secret key shared by the sender and receiver. The sender calculates the MAC of the message using the key and sends both the message and the MAC. The receiver recalculates the MAC using the same key and verifies if it matches the received MAC. If they match, the message is authentic and has not been altered.
- **Digital Signatures:** Digital signatures provide authenticity, integrity, and non-repudiation. They are created using asymmetric cryptography (public and private key pairs). The **sender signs the message (or its hash) using their private key**, and the **receiver verifies the signature using the sender's public key**. This ensures that the message originated from the claimed sender and has not been modified. Non-repudiation means the sender cannot deny having sent the message.

Other Hash Function Uses



Digital Signature Standard (DSS):

One of the documents mentions the Digital Signature Standard (DSS). DSS is a U.S. government standard for digital signatures. It specifies algorithms for generating and verifying digital signatures. While the document mentions DSS, it doesn't provide the full details of the algorithms or how to implement them.

RSA Scheme Exercise:

The other document includes an exercise on the RSA scheme for digital signatures. This exercise provides specific values for p , q , and d in an RSA scheme and asks you to:

1. Calculate the public key e .
2. Sign and verify messages $M1 = 100$ and $M2 = 50$.
3. Show that if $M = M1 \times M2 = 5000$, then $S = S1 \times S2$, where $S1$ and $S2$ are the signatures of $M1$ and $M2$, and S is the signature of M .

General Form of RSA Digital Signature:

In the RSA scheme for digital signatures:

- You generate a public and private key pair. The public key (e, n) is known to everyone, while the private key (d, n) is kept secret. n is the product of two large prime numbers, p and q .
- To sign a message M , you compute the signature $S = M^d \bmod n$.
- To verify a signature S for a message M , you check if $M = S^e \bmod n$.

Solving the Exercise:

To solve the exercise, you would need to perform the calculations based on the provided values of p , q , and d .

1. **Calculate n :** $n = p * q$
2. **Calculate Euler's totient function $\phi(n)$:** $\phi(n) = (p-1) * (q-1)$
3. **Calculate e :** Find a value e such that $(e * d) \bmod \phi(n) = 1$.
4. **Sign $M1$:** $S1 = M1^d \bmod n$
5. **Verify $S1$:** Check if $M1 = S1^e \bmod n$
6. **Sign $M2$:** $S2 = M2^d \bmod n$
7. **Verify $S2$:** Check if $M2 = S2^e \bmod n$
8. **Calculate M :** $M = M1 * M2$
9. **Calculate $S1 * S2$:** $(S1 * S2) \bmod n$
10. **Verify if $S = S1 * S2$:** Check if $M^d \bmod n = (S1 * S2) \bmod n$

The exercise provides the following values for an RSA scheme:

- $p = 809$
- $q = 751$
- $d = 23$

We need to perform the following steps:

1. Calculate the public key e :

- First, calculate n : $n = p * q = 809 * 751 = 607559$
- Next, calculate Euler's totient function $\phi(n)$: $\phi(n) = (p-1) * (q-1) = (809-1) * (751-1) = 808 * 750 = 606000$
- Now, find e such that $(e * d) \bmod \phi(n) = 1$. This means $(e * 23) \bmod 606000 = 1$. We need to find the modular multiplicative inverse of 23 modulo 606000. Using the extended Euclidean algorithm, we can find that $e = 395651$.

2. Sign and verify $M1 = 100$:

- Sign $M1$: $S1 = M1^d \bmod n = 100^{23} \bmod 607559$. Calculating this large exponentiation requires modular exponentiation. Using a modular exponentiation calculator or software, we find $S1 = 585241$.
- Verify $S1$: Check if $M1 = S1^e \bmod n$. $100 = 585241^{395651} \bmod 607559$. Again, using modular exponentiation, we find that $585241^{395651} \bmod 607559 = 100$. The signature is verified.

3. Sign and verify $M2 = 50$:

- Sign $M2$: $S2 = M2^d \bmod n = 50^{23} \bmod 607559$. Using modular exponentiation, we find $S2 = 207662$.
- Verify $S2$: Check if $M2 = S2^e \bmod n$. $50 = 207662^{395651} \bmod 607559$. Using modular exponentiation, we find that $207662^{395651} \bmod 607559 = 50$. The signature is verified.

4. Show if $M = M1 * M2 = 5000$, then $S = S1 * S2$:

- $M = M1 * M2 = 100 * 50 = 5000$
- $S1 * S2 = 585241 * 207662 \bmod 607559$. Calculating this product modulo n gives $121424961502 \bmod 607559$. Using modular arithmetic, we find $(S1 * S2) \bmod n = 572070$.

- Now, let's calculate the signature of M directly: $S = M^d \bmod n = 5000^{23} \bmod 607559$. Using modular exponentiation, we find $S = 572070$.
- Since $(S_1 * S_2) \bmod n = 572070$ and $S = 572070$, we have shown that $S = S_1 \times S_2$ in this case.

Relevant Information from 'Advanced Data Security' regarding public key cryptography and the RSA algorithm:

The document "AdvDataSec-Ch5 - PKC and RSA.pdf" provides information on public key cryptography and the RSA algorithm. It explains:

- **Principles of Public-Key Cryptosystems:** The concept evolved to address key distribution and digital signatures. It involves a pair of keys: a public key for encryption/verification and a private key for decryption/signing.
- **Applications for Public-Key Cryptosystems:** These include encryption/decryption, digital signatures, and key exchange.
- **Public-Key Requirements:** It lists several conditions that these algorithms must fulfill, such as the computational ease of key generation, encryption, and decryption, and the computational infeasibility of determining the private key from the public key or recovering the original message from ciphertext without the private key.
- **Public-Key Cryptanalysis:** It discusses various attacks on public-key schemes, including brute-force attacks, mathematical attacks (like factoring n), timing attacks, hardware fault-based attacks, and chosen ciphertext attacks.
- **RSA Scheme:** It describes RSA as a widely used general-purpose approach to public-key encryption, where plaintext and ciphertext are integers between 0 and $n-1$. It outlines the steps for creating a public/private key pair (choosing prime numbers p and q , computing n and z , choosing e and d) and for encryption and decryption.
- **RSA Properties:** It highlights the property that applying the public key followed by the private key, or vice versa, results in the original message.
- **Security of RSA:** It discusses the difficulty of factoring large numbers as the basis for RSA's security.
- **Efficient Operation:** It mentions techniques for speeding up RSA operations, such as choosing specific values for e and using the Chinese Remainder Theorem for decryption.

Chapter 7

Understanding Network Security:

- **Confidentiality:** Ensures only the sender and intended receiver understand the message contents through encryption-decryption.
- **Authentication:** Confirms the identities of the sender and receiver.
- **Message Integrity:** Ensures that messages are not altered during transmission.
- **Access and Availability:** Services must remain accessible to users.

Security Protocols: the importance of true cryptosystems and the increasing need for security measures as threats to the Internet grew. Securing Internet Communication with:

Key Protocols:

- **SSL (Secure Socket Layer):** Uses public key encryption for securing Internet channels. Provides • confidentiality • integrity • authentication
- **S-HTTP (Secure Hypertext Transfer Protocol):** Extends HTTP for encrypting messages between client and server.

Securing Email Communications: Describes how Alice sends a message to Bob using symmetric and asymmetric encryption to ensure confidentiality and integrity (3 cases:)

Case 1: Alice wants to send a confidential email to Bob

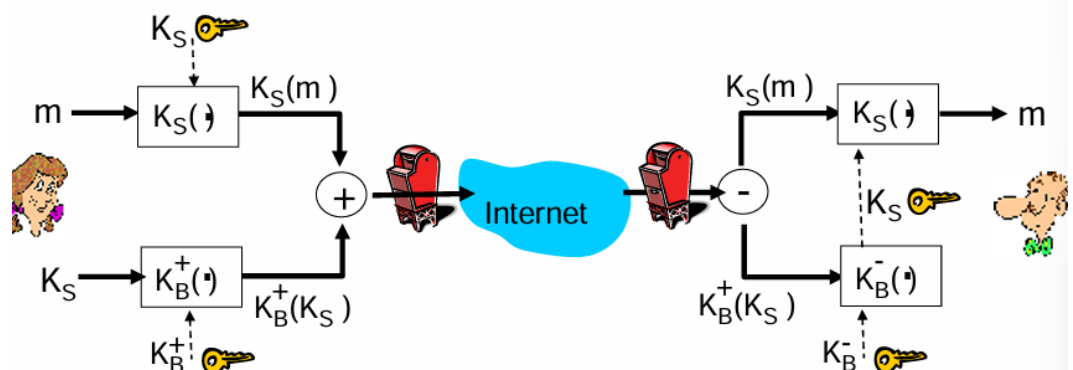
- **Alice generates a symmetric private key (K_s):** This key is used for encrypting her message for efficiency.
- Alice encrypts her message (m) using this key: $K_s(m)$.
- She additionally encrypts the symmetric key (K_s) using Bob's public key: $K_B(K_s)$.

Transmission:

Alice sends both the encrypted message $K_s(m)$ and the encrypted key $K_B(K_s)$ over the Internet.

Decryption by Bob:

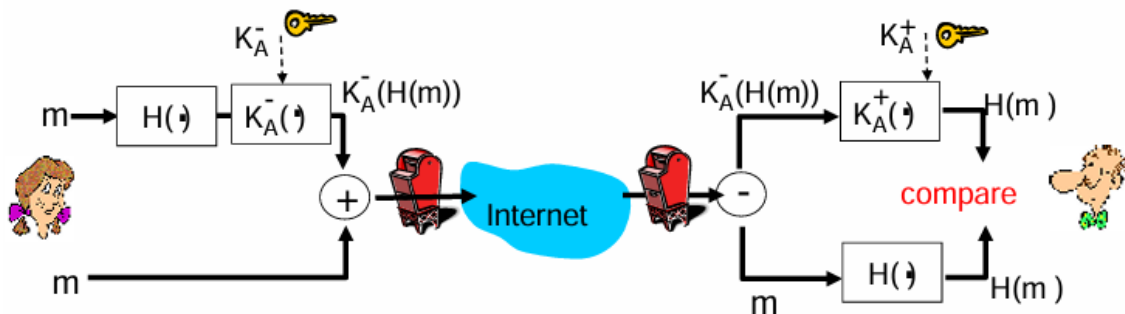
- Bob uses his private key to decrypt $K_B(K_s)$ to recover K_s .



- He then uses K_s to decrypt $K_s(m)$ and recover the original message (m).

Case 2: Sender Authentication and Message Integrity:

- Alice wants to ensure that the message has not been altered and comes from her.
- She digitally signs the message, creating a hash $H(m)$ and encrypting this hash with her private key K_A : $K_A^-(H(m))$.
- Alice sends both the original message m (in clear) and the digital signature over the Internet.



2. Validation:

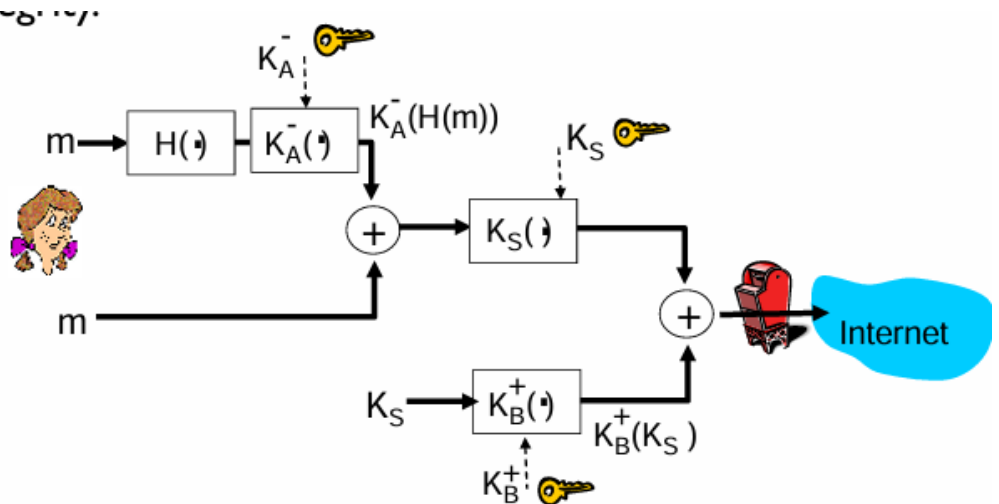
- The receiving party can compare the hash from the received message $H(m)$ with the decrypted hash from Alice's digital signature to ensure integrity and authenticity.

Case 3: Multiple Keys for Added Security:

Alice combines the earlier measures to provide secrecy, sender authentication, and message integrity simultaneously.

She uses her private key (K_A), Bob's public key (K_B), and the newly created symmetric key (K_s) to accomplish this.

The message goes through various hashing and signing steps involving these keys, ensuring a robust security layer for email transmission.



N.B: The secure email process described differs from RSA in that it combines both symmetric and asymmetric encryption rather than solely relying on asymmetric encryption like RSA:

RSA Encryption:

- *RSA is a public-key cryptosystem used primarily for secure data transmission.*
- *It uses asymmetric keys: a public key for encryption and a private key for decryption.*
- *RSA is computationally intensive, which can make it slower for encrypting large amounts of data.*

Secure E-mail Process:

- *This process employs **symmetric encryption** (using a shared secret key, K_s) for the actual message encryption, which is much faster and more efficient than RSA for bulk data.*
- *It also uses **asymmetric encryption** (Bob's public key, K_B) to encrypt the symmetric key, ensuring that only Bob can decrypt it with his private key.*
- *This hybrid approach provides the efficiency of symmetric encryption for the message while leveraging the security of asymmetric encryption to share the symmetric key.*

Standards for Email Security: work to secure e-mail operations.

- **S/MIME:** Secure Multipurpose Internet Mail Extensions. Adds encryption and digital signatures to emails.
- **Privacy Enhanced Mail (PEM):** uses 3DES symmetric key encryption
- **PGP (Pretty Good Privacy):** A hybrid cryptosystem that combines best cryptographic practices for securing email and file storage encryption. uses RSA for symmetric key exchange and for digital signatures. Offers services like digital signatures, message encryption, and key management.

Web Transaction Security:

SET (Secure Electronic Transactions): Developed to protect electronic and card payment transactions. Uses DES to encrypt credit card information transfers and RSA for key exchange.

Securing Wireless Networks: WEP (Wired Equivalent Privacy) and WPA (Wi-Fi Protected Access- created to resolve issues with WEP)

IPSec (Internet Protocol Security): An open-source protocol framework that secures communications across IP networks. Protects data integrity, user confidentiality, and authenticity at the IP packet level. Combines cryptosystems like Diffie-Hellman key exchange and public-key cryptography.

Securing TCP/IP with IPsec and PGP

IPsec Components: Uses protocols like Encapsulating Security Payload (ESP) for data encryption. Ensures communication security by adding information to IP packets.

Chapter 8

Introduction to Software Security

- **Goal:** Protect software from being exploited through coding flaws.
- **Key Threat: Buffer Overflow** – one of the most common and dangerous vulnerabilities.
- **Security Principle: "Never trust user input."** Always validate and sanitize it.

Understanding Buffer Overflows:

What they are: Buffer overflows happen when a program tries to write more data into a buffer (a temporary storage area) than it can hold. This extra data can overwrite adjacent memory, including critical control information like the return address on the stack.

How attackers exploit them: Attackers can intentionally cause a buffer overflow to overwrite the return address with the address of malicious code they've injected. When the function attempts to return, it instead jumps to and executes the attacker's code.

Impact: This can allow an attacker to execute arbitrary code with the privileges of the compromised program, potentially leading to full system compromise.

buffer overflows are a common and persistent vulnerability, often due to not trusting or properly validating user inputs.

Example: A vulnerable password-checking program is used as an example, where inputting a password longer than the allocated buffer size can overwrite the return address.

```
#include <stdio.h>
#include <strings.h>

int main(int argc, char *argv[]) {
    int allow_login = 0;
    char pwdstr[12];
    char targetpwd[12] = "MyPwd123";
    gets(pwdstr);
    if (strncmp(pwdstr, targetpwd, 12) == 0)
        allow_login = 1;

    if (allow_login == 0)
        printf("Login request rejected");
    else
        printf("Login request allowed");
}
```

We type a correct password (MyPwd123) of less than 12 characters: The login request is allowed. Now let us type "BadPassWd" when we are asked to provide the password: The login request is rejected. If we type a really long string, we will overflow into the return address space. We can carefully overflow the return address so it contains the value of an address where we put some code we want executed.

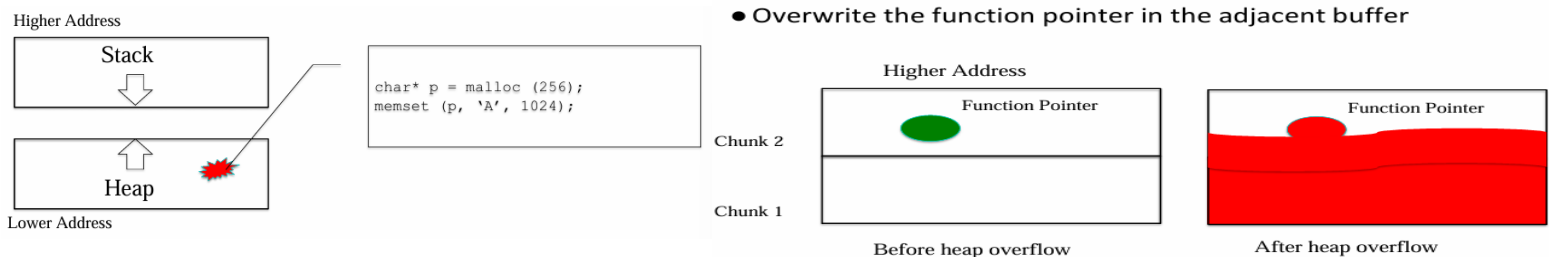
Stack Memory Layout (Key to Understanding Exploits)

Stack grows downward. Contains: Local variables, Function parameters, Return address

If attacker overflows a local buffer, they can **overwrite the return address** → redirect execution.

Types of Buffer Overflows:

- **Stack Buffer Overflows:** Occur in the stack memory, which is used for function calls, local variables, parameters, and control information (like return addresses).
- **Code Injection (Shellcode):** Injects small, malicious code (like `/bin/sh`) into the buffer. Overwrites return address to jump to this code. Executes with the same privileges as the program (often root/system).
- **Heap Overflows:** Occur in the heap data area. Used to overwrite metadata, function pointers, or VTables. The `malloc()` and `free()` functions are often involved.



- **Return-to-libc:** The return address is overwritten to point to a standard library function instead of injected code. attacker uses existing **library functions** (like `system()`).
- **OpenSSL Heartbleed Vulnerability:** A specific example where more data could be read from a buffer than intended (beyond buffer bounds) , potentially exposing sensitive information like passwords or private key.

Defenses Against Buffer Overflow Attacks:

- **Secure Programming Practices:**

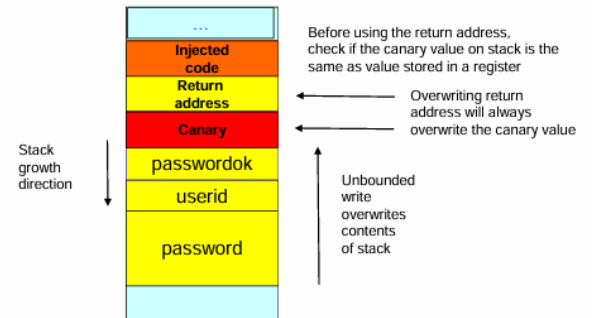
- **Code defensively:** Assume all input is potentially malicious and validate it thoroughly
- **Language Choice:** Use "safe" languages like Java, C++, or Python that perform automatic bounds checking and memory management, making buffer overflows less likely. However, these can sometimes have performance drawbacks.
- **Safer Functions:** When using "unsafe" languages (like C), use library functions that perform bounds checking.

- **Analysis Tools:** Employ static and dynamic analysis tools to scan code for potential vulnerabilities and unsafe functions.

●Countermeasure – Stack Protection

- **Stack Canaries:** A random "canary" value is placed on the stack before the return address. If a buffer overflow attempts to overwrite the return address, the canary value will also be changed. The system checks the canary before using the return address; if it's altered, an overflow is detected, and the program can be terminated.

n Canary for tamper detection



n No code execution on stack

- **Address Space Layout Randomization (ASLR):** This technique randomizes the memory locations of the stack, heap, and libraries. This makes it significantly harder for an attacker to know the exact address to jump to for their malicious code or for a return-to-libc attack.
- **Non-Executable Stack (NX Bit / DEP):** Marks the stack region of memory as non-executable. Even if an attacker manages to inject code onto the stack, the hardware prevents it from being run. This is often used in conjunction with ASLR.
- **Principle of Least Privilege:** Programs should run with the minimum privileges necessary to perform their tasks. This limits the damage an attacker can do if they successfully exploit a vulnerability.

Chapter 9

What is Penetration Testing?

Penetration testing (pentesting) is ethical hacking done to find security weaknesses in your IT environment. It shows you where your systems are vulnerable before malicious hackers do. Because every business has something to lose- and data is one of the most valuable assets that a hacker can gain. Whether through an application, device, or network

Phases of Penetration Testing

1. Pre-Engagement Interactions or Information

Gathering: This initial phase involves outlining logistics, expectations, legal implications, and the client's objectives. It's crucial to understand the organization's risks, culture, and determine the best testing strategy (white box, black box, or gray box).

2. Reconnaissance or Open-Source Intelligence (OSINT) Gathering:

The goal is to gather as much intelligence as possible about the organization and potential targets. This includes identifying key personnel, business relationships, and publicly available data like IP addresses, hardware, and software used. Techniques include search engine queries, domain name searches, social engineering, and internet/internal footprinting (e.g., ping sweeps, port scanning).

3. Threat Modeling and Vulnerability Identification (Discovery and Scanning):

Testers identify targets and map attack vectors using information from the reconnaissance phase. This involves categorizing business assets (employee data, customer data, technical data) and threats (internal and external). Vulnerability scanners are used to discover and inventory security risks, followed by validating if these vulnerabilities are exploitable.

4. Exploitation:

Pentesters attempt to exploit the identified vulnerabilities in the network, applications, and data. The objective is to see how far they can penetrate the environment, identify high-value targets, and avoid detection. Common tactics include web application attacks, network attacks, memory-based attacks, zero-day exploits, Wi-Fi attacks, physical attacks, and social engineering.

Phases of Penetration Testing



5. **Post-Exploitation, Risk Analysis and Recommendations:** This phase focuses on documenting the methods used to access valuable information. The tester should determine the value of compromised systems and captured data. It's important to obtain a report with clear recommendations for fixing security holes. Cleanup activities are also crucial, including removing any tools or accounts created during the test and reconfiguring systems to their original state.
6. **Reporting:** The penetration testing company provides written recommendations and reviews the findings with the client. The report should detail how entry points were discovered and how to remediate security issues, often including an overall security risk score and a roadmap for addressing vulnerabilities.

Types of Penetration Testing

1. **Network Penetration Testing:** Focuses on identifying vulnerabilities through port scans, network sniffing, traffic analysis, password cracking, buffer overflows, and spoofing server traffic.
2. **Web Application Penetration Testing:** Examines vulnerabilities such as application logic flaws, session management issues, cookie manipulation, brute-force attacks, poor server configuration, file upload/download vulnerabilities, and injection attacks (like SQL or HTML).
3. **Mobile Application Penetration Testing:** Ensures sensitive data protection on mobile devices, safe handling of credentials, secure data transmission, user authentication, and integration with third-party services, focusing on consent management and prevention of unauthorized access.
4. **Wireless Penetration Testing:** Identifies risks from rogue access points, vendor-supplied defaults, eavesdropping vulnerabilities, insufficient monitoring, slow data rates, misconfigurations, and attacks such as DoS and man-in-the-middle.
5. **API (REST or SOAP) Penetration Testing:** Assesses for excessive data exposure, lack of rate limiting, security misconfiguration injection, improper asset management, and insufficient logging.
6. **Continuous Penetration Testing:** Advocates for ongoing testing in response to changing environments, ensuring a proactive approach to security rather than periodic assessments.

How to Choose a Good Vendor

1. **Determine Project Scope:** Define the framework for the engagement.
2. **Identify Needed Specializations:** Different industries require different configurations and strategies.
3. **Evaluate Skills:** Ensure the vendor possesses necessary skills for your specific systems.
4. **Check Credentials:** Look for adherence to industry standards (PTES and certifications like OSSTM, OWASP).
5. **Seek Reputation and References:** Assess the vendor's credibility through previous client feedback.
6. **Request Sample Reports:** Analyze report clarity and quality of provided insights.
7. **Understand Data Security Practices:** Know how the vendor will handle data during testing.
8. **Confirm Liability Insurance:** Ensure the vendor has coverage to address potential issues during testing.
9. **Define Rules of Engagement:** Clearly outline what is included in the testing agreement.
10. **Set Clear Expectations:** Agree on the level and frequency of services expected.