European IT Certification Curriculum
Self-Learning Preparatory Materials

EITC/IS/ACC

Advanced Classical Cryptography

This document constitutes European IT Certification curriculum self-learning preparatory material for the EITC/IS/ACC Advanced Classical Cryptography programme.

This self-learning preparatory material covers requirements of the corresponding EITC certification programme examination. It is intended to facilitate certification programme's participant learning and preparation towards the EITC/IS/ACC Advanced Classical Cryptography programme examination. The knowledge contained within the material is sufficient to pass the corresponding EITC certification examination in regard to relevant curriculum parts. The document specifies the knowledge and skills that participants of the EITC/IS/ACC Advanced Classical Cryptography certification programme should have in order to attain the corresponding EITC certificate.

Disclaimer

This document has been automatically generated and published based on the most recent updates of the EITC/IS/ACC Advanced Classical Cryptography certification programme curriculum as published on its relevant webpage, accessible at:

https://eitca.org/certification/eitc-is-acc-advanced-classical-cryptography/

As such, despite every effort to make it complete and corresponding with the current EITC curriculum it may contain inaccuracies and incomplete sections, subject to ongoing updates and corrections directly on the EITC webpage. No warranty is given by EITCI as a publisher in regard to completeness of the information contained within the document and neither shall EITCI be responsible or liable for any errors, omissions, inaccuracies, losses or damages whatsoever arising by virtue of such information or any instructions or advice contained within this publication. Changes in the document may be made by EITCI at its own discretion and at any time without notice, to maintain relevance of the self-learning material with the most current EITC curriculum. The self-learning preparatory material is provided by EITCI free of charge and does not constitute the paid certification service, the costs of which cover examination, certification and verification procedures, as well as related infrastructures.

TABLE OF CONTENTS

Diffie-Hellman cryptosystem **4**
Diffie-Hellman Key Exchange and the Discrete Log Problem 4
Generalized Discrete Log Problem and the security of Diffie-Hellman 5
Encryption with Discrete Log Problem **6**
Elgamal Encryption Scheme 6
Elliptic Curve Cryptography **8**
Introduction to elliptic curves 8
Elliptic Curve Cryptography (ECC) 9
Digital Signatures **10**
Digital signatures and security services 10
Elgamal Digital Signature 11
Hash Functions **18**
Introduction to hash functions 18
SHA-1 hash function 24
Message Authentication Codes **25**
MAC (Message Authentication Codes) and HMAC 25
Key establishing **30**
Symmetric Key Establishment and Kerberos 30
Man-in-the-middle attack **35**
Man-in-the-middle attack, certificates and PKI 35

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: DIFFIE-HELLMAN CRYPTOSYSTEM
TOPIC: DIFFIE-HELLMAN KEY EXCHANGE AND THE DISCRETE LOG PROBLEM

This part of the material is currently undergoing an update and will be republished shortly.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: DIFFIE-HELLMAN CRYPTOSYSTEM
TOPIC: GENERALIZED DISCRETE LOG PROBLEM AND THE SECURITY OF DIFFIE-HELLMAN

This part of the material is currently undergoing an update and will be republished shortly.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: ENCRYPTION WITH DISCRETE LOG PROBLEM
TOPIC: ELGAMAL ENCRYPTION SCHEME

Encryption with Discrete Log Problem - Elgamal Encryption Scheme

The Elgamal encryption scheme is a cryptographic method based on the discrete logarithm problem. It provides a way to securely encrypt and decrypt messages using a public-private key pair. In this didactic material, we will explore the Elgamal encryption scheme and understand how it works.

The Elgamal encryption scheme is a type of public-key encryption, which means that it uses two different keys for encryption and decryption. The encryption key is made public and is known as the public key, while the decryption key is kept private and is known as the private key.

To understand the Elgamal encryption scheme, let's break it down into its key components:

1. Key Generation:
- Generate a large prime number, p.
- Select a primitive element, g, modulo p.
- Choose a random private key, a, such that $1 \leq a \leq p-2$.
- Compute the public key, A, as $A = g^a \bmod p$.

2. Encryption:
- Convert the message, M, into a numerical representation.
- Select a random secret key, k, such that $1 \leq k \leq p-2$.
- Compute the ciphertext as $C = (g^k \bmod p, A^k * M \bmod p)$.

3. Decryption:
- Compute the shared secret key, s, as $s = C1^a \bmod p$.
- Compute the plaintext message as $M = C2 * (s^{-1} \bmod p) \bmod p$.

The security of the Elgamal encryption scheme is based on the discrete logarithm problem, which is considered computationally difficult to solve. The discrete logarithm problem involves finding the exponent, a, in the equation $A = g^a \bmod p$, given A, g, and p. The security of the scheme relies on the assumption that it is difficult to compute the private key, a, from the public key, A.

By using the Elgamal encryption scheme, individuals can securely exchange encrypted messages without sharing their private keys. This makes it a valuable tool in ensuring the confidentiality and integrity of sensitive information.

The Elgamal encryption scheme is a powerful cryptographic method that utilizes the discrete logarithm problem to securely encrypt and decrypt messages. By generating a public-private key pair and performing encryption and decryption operations, individuals can communicate securely while protecting the confidentiality of their messages.

Encryption with Discrete Log Problem - Elgamal Encryption Scheme

The Elgamal encryption scheme is a public-key encryption algorithm based on the discrete logarithm problem. It was developed by Taher Elgamal in 1985 and is widely used for secure communication over the internet.

In the Elgamal encryption scheme, each user generates a pair of keys: a public key and a private key. The public key is shared with others, while the private key is kept secret. The security of the encryption scheme relies on the difficulty of solving the discrete logarithm problem.

The discrete logarithm problem is a mathematical problem that involves finding the exponent of a given number in a finite field. It is computationally difficult to solve, especially for large prime numbers. This makes the Elgamal encryption scheme secure against attacks by brute force or by solving the discrete logarithm problem.

To encrypt a message using the Elgamal encryption scheme, the sender first obtains the recipient's public key. The sender then randomly selects a number, called the ephemeral key, and computes the ciphertext by raising the recipient's public key to the power of the ephemeral key, modulo a large prime number. The sender also computes a shared secret by raising the recipient's public key to the power of their own private key.

The ciphertext and the shared secret are then used to encrypt the message using a symmetric encryption algorithm, such as AES. The encrypted message, along with the ephemeral key, is then sent to the recipient.

To decrypt the message, the recipient uses their private key to compute the shared secret. The shared secret is then used to decrypt the encrypted message using the same symmetric encryption algorithm. The decrypted message is then obtained.

The Elgamal encryption scheme provides confidentiality and integrity of the message. The confidentiality is ensured by the use of symmetric encryption, while the integrity is ensured by the use of the shared secret, which is unique to each message.

The Elgamal encryption scheme is a secure and widely used public-key encryption algorithm. It provides confidentiality and integrity of the message by utilizing the discrete logarithm problem. By understanding the concepts and techniques behind the Elgamal encryption scheme, one can better appreciate the importance of cryptography in ensuring secure communication.

Encryption with Discrete Log Problem - Elgamal Encryption Scheme

In the field of cybersecurity, encryption plays a crucial role in ensuring the confidentiality and integrity of sensitive information. One of the encryption schemes used in classical cryptography is the Elgamal Encryption Scheme, which is based on the discrete log problem.

The discrete log problem is a mathematical problem that involves finding the exponent to which a given number must be raised in order to obtain another given number. This problem is considered computationally difficult to solve, making it suitable for encryption purposes.

The Elgamal Encryption Scheme is a public-key encryption scheme that uses the discrete log problem as its foundation. It consists of two main steps: key generation and encryption.

During the key generation step, a user generates a public-private key pair. The public key is made available to anyone who wants to send encrypted messages to the user, while the private key is kept secret and used for decryption.

To encrypt a message using the Elgamal Encryption Scheme, the sender first converts the message into a numerical representation. Then, a random number called the ephemeral key is generated. Using the recipient's public key and the ephemeral key, the sender performs a series of mathematical operations to produce the ciphertext.

The ciphertext is then sent to the recipient, who can decrypt it using their private key. By using the private key and performing the inverse mathematical operations, the recipient can recover the original message.

The security of the Elgamal Encryption Scheme relies on the difficulty of solving the discrete log problem. If an attacker were able to solve this problem efficiently, they would be able to recover the private key and decrypt the ciphertext. However, no efficient algorithm for solving the discrete log problem on classical computers has been discovered so far.

The Elgamal Encryption Scheme is a public-key encryption scheme that utilizes the discrete log problem for secure communication. By leveraging the computational difficulty of solving the discrete log problem, the scheme provides a robust method for encrypting and decrypting messages.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: ELLIPTIC CURVE CRYPTOGRAPHY
TOPIC: INTRODUCTION TO ELLIPTIC CURVES

Good morning, and welcome to today's lesson on elliptic curve cryptography. In this session, we will introduce the concept of elliptic curves and explore their role in modern cryptography.

Elliptic curve cryptography (ECC) is a branch of public key cryptography that relies on the mathematics of elliptic curves. It offers a higher level of security compared to traditional cryptographic algorithms, such as RSA and Diffie-Hellman.

So, what exactly is an elliptic curve? An elliptic curve is a smooth curve defined by a mathematical equation of the form y^2 = x^3 + ax + b. This equation represents all the points (x, y) that satisfy it. The curve also has a special point called the "point at infinity" which acts as the identity element.

In elliptic curve cryptography, we use a finite field of prime order to define the curve. This means that the x and y coordinates of points on the curve are integers modulo a prime number. The choice of this prime number is crucial for the security of the cryptographic scheme.

The security of elliptic curve cryptography lies in the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP). Given a point P on the curve and another point Q, it is computationally hard to find a scalar k such that Q = kP. This property forms the basis of ECC's security.

To illustrate this concept, let's consider an example. Suppose we have a curve defined by the equation y^2 = x^3 + 7 over a finite field of prime order 23. We can perform scalar multiplication on a point P by multiplying it with an integer k. For example, if P = (2, 3) and k = 4, then 4P = (20, 6).

The beauty of elliptic curve cryptography lies in its efficiency. ECC provides the same level of security as traditional cryptographic algorithms, but with much smaller key sizes. This makes it ideal for resource-constrained environments, such as mobile devices and embedded systems.

Elliptic curve cryptography is a powerful tool in modern cybersecurity. By leveraging the mathematical properties of elliptic curves, ECC offers strong security with smaller key sizes. In the next lesson, we will delve deeper into the mathematics behind elliptic curves and explore cryptographic operations on them.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: ELLIPTIC CURVE CRYPTOGRAPHY
TOPIC: ELLIPTIC CURVE CRYPTOGRAPHY (ECC)

This part of the material is currently undergoing an update and will be republished shortly.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: DIGITAL SIGNATURES
TOPIC: DIGITAL SIGNATURES AND SECURITY SERVICES

This part of the material is currently undergoing an update and will be republished shortly.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: DIGITAL SIGNATURES
TOPIC: ELGAMAL DIGITAL SIGNATURE

Welcome to the second week of the topic of digital signatures. In the previous week, we provided an introduction to digital signatures and discussed security services. Today's lecture is a continuation of last week's material. The main focus of today's lecture is an attack against RSA digital signatures and the Elgamal digital signature scheme.

Firstly, let's discuss the attack against RSA digital signatures. Unlike attacks such as factoring, which can be easily protected against by choosing large moduli, this attack is built into many digital signatures. It is known as the existential forgery attack against RSA digital signatures. The attack exploits a construction called "exists tensho" or "existential forgery". It is interesting to see the implications of this attack on the construction we discussed last week.

To understand the attack, let's revisit the protocol. The RSA digital signature scheme is similar to a regular RSA encryption scheme. Bob computes a public key consisting of the modulus N and the public exponent E. He keeps the private exponent secret. Bob openly distributes his public key over the channel. To sign a message X, Bob raises it to his private exponent and sends the message and signature over the channel. Upon receiving the message and signature, Alice verifies the signature by raising it to the private key power and checking if it matches the original message.

Now, let's explore what an attacker, named Oscar, can do in this protocol. Oscar's goal is to generate a message with a valid signature, without tampering with the original message. For example, Oscar may want to generate a fake message instructing a bank to transfer funds from one account to another. Oscar's attack is an existential forgery attack, where he generates a message and signature pair that appears valid.

To execute the attack, Oscar follows these steps:
1. Oscar chooses a signature S from the set of numbers modulo N.
2. Oscar computes X = S^E mod N, where E is the public exponent obtained from Bob's website.
3. Oscar sends X and S to Alice.

If Alice does not detect the attack, she will consider the message and signature pair valid. This allows Oscar to create a message with a valid signature, potentially causing harmful actions.

The attack against RSA digital signatures, known as the existential forgery attack, exploits the construction of the RSA digital signature scheme. By generating a message and signature pair, an attacker can create a seemingly valid message with a signature. This attack highlights the importance of carefully verifying digital signatures to prevent unauthorized actions.

Digital signatures are an essential component of modern cryptography, providing a means to verify the authenticity and integrity of digital messages. One widely used digital signature scheme is the Elgamal digital signature scheme. In this scheme, a sender, let's call her Alice, generates a digital signature for a message and sends it to the recipient, Bob. The recipient can then verify the signature to ensure that the message indeed came from Alice and has not been tampered with.

To understand how the Elgamal digital signature scheme works, let's break it down into its key steps. First, Alice generates a pair of keys: a private key and a corresponding public key. The private key is kept secret and is used for signing messages, while the public key is made available to anyone who wants to verify Alice's signatures.

To create a digital signature for a message, Alice follows these steps:

1. She computes a random number, let's call it "k".
2. She computes a value called "r" by raising a fixed generator "g" to the power of "k" modulo a large prime number "p".
3. She computes another value called "s" by taking the inverse of "k" modulo "p-1" and multiplying it with the difference between the message's hash value and the product of Alice's private key and "r" modulo "p-1".

4. The digital signature for the message is the pair of values (r, s).

Now, when Bob receives the digital signature along with the message, he can verify its authenticity by following these steps:

1. Bob computes a value called "v" by raising Alice's public key to the power of the message's hash value modulo "p".
2. He computes another value called "w" by raising "r" to the power of "s" modulo "p".
3. Finally, Bob checks if "v" is equal to "w". If they are equal, the signature is valid; otherwise, it is not.

It is important to note that the Elgamal digital signature scheme provides a strong level of security, as it relies on the computational hardness of certain mathematical problems. However, like any cryptographic scheme, it has its limitations. One limitation is that it is vulnerable to a specific attack known as the Z8X attack.

In the Z8X attack, an adversary, let's call him Oscar, can generate a valid signature for a message without knowing Alice's private key. This is achieved by carefully choosing the values of "r" and "s" in such a way that the verification process succeeds. Oscar exploits the fact that the message's hash value is raised to a fixed exponent, which cannot be directly controlled.

To mitigate this attack, additional countermeasures can be employed. One such countermeasure is to impose formatting rules on the message, which can be checked during the verification process. By imposing these rules, the likelihood of generating a valid signature for a malicious message is greatly reduced.

In practice, the Elgamal digital signature scheme is often used in conjunction with other cryptographic protocols and countermeasures to enhance its security. It is crucial to understand that the basic principles of Elgamal cryptography are important to grasp, but in real-world scenarios, modifications and additional precautions are necessary to ensure its effectiveness.

The Elgamal digital signature scheme provides a means for verifying the authenticity and integrity of digital messages. By following a series of steps, a sender can generate a digital signature, and a recipient can verify its validity. However, it is important to be aware of certain limitations and potential attacks, such as the Z8X attack, and to employ suitable countermeasures to enhance the security of the scheme.

In the study of advanced classical cryptography, one important topic is digital signatures. In this didactic material, we will focus on the Elgamal digital signature scheme.

To understand the Elgamal digital signature scheme, let's first discuss the concept of preventing certain attacks using specific X values. We can restrict the X values that are allowed, ensuring that only certain X values are permitted. This prevents potential attacks. For instance, we can use a formatting rule where the payload, denoting the actual message (e.g., an email or a PDF file), is limited to a certain length, such as 900 bits out of a total of 1024 bits. The remaining bits are used for padding, which is an arbitrary bit pattern. In this example, we choose to have 124 trailing ones as the padding. This formatting rule adds an extra layer of security but comes at the cost of not utilizing all the available bits.

Now, let's consider the problem that arises when an adversary, Oscar, computes random values for RX, which are 1024-bit values. We can analyze the probability of the least significant bit (LSB) being a 1. Intuitively, we might expect a 50% chance. However, when Oscar raises RX to the power of the public key (e), mod n, he obtains a random output. If the output is 1, he can choose a new RX and repeat the process. On average, it takes two trials to generate a 1. Extending this logic, to generate a specific bit pattern, such as 124 trailing ones, Oscar would need to generate an average of 2^124 different RX values. This number is astronomically large, similar to the estimated number of atoms on Earth.

It is worth mentioning that the padding scheme described here is a simplified example. In real-world scenarios, padding schemes are more complex. For further details, refer to the textbook.

Moving on to the second chapter, we will now explore the Elgamal digital signature scheme. In the previous chapters, we covered public key encryption and the RSA algorithm. Now, we will focus on the discrete logarithm-based Elgamal digital signature scheme.

In the setup phase of the Elgamal digital signature scheme, we need a cyclic group where the discrete logarithm problem resides. To achieve this, we select a large prime number, denoted as 'p'. Additionally, we choose a primitive element, 'alpha', which generates the entire cyclic group.

The Elgamal digital signature scheme involves two main steps: key generation and signature generation.

During the key generation step, the signer, let's call them Alice, selects a private key 'd' randomly from the set {1, 2, ..., p-2}. Alice then computes her public key 'y' as y = alpha^d mod p.

To generate a digital signature, Alice follows these steps:
1. Alice selects a random value 'k' from the set {1, 2, ..., p-2}.
2. Alice computes r = alpha^k mod p.
3. Alice computes the hash value of the message she wants to sign, denoted as 'H(m)'.
4. Alice computes s = (H(m) - d*r) * k^(-1) mod (p-1), where k^(-1) is the modular multiplicative inverse of k modulo (p-1).
5. The digital signature is the pair (r, s).

To verify the signature, the verifier, let's call them Bob, follows these steps:
1. Bob receives the message, the digital signature (r, s), and Alice's public key 'y'.
2. Bob computes the hash value of the message, denoted as 'H(m)'.
3. Bob computes w = s^(-1) mod (p-1), where s^(-1) is the modular multiplicative inverse of s modulo (p-1).
4. Bob computes u1 = H(m) * w mod (p-1) and u2 = r * w mod (p-1).
5. Bob computes v = (alpha^u1 * y^u2 mod p) mod p.
6. If v is equal to r, the signature is valid. Otherwise, it is invalid.

The Elgamal digital signature scheme provides a way for Alice to sign messages using her private key and for Bob to verify the authenticity of the signatures using Alice's public key.

In the context of classical cryptography, one important concept is the discrete logarithm problem. This problem involves finding the exponent in a given setting, where we have a public key (X) and a private key (Y). The difficulty lies in computing this logarithm, making it a challenging task. In the case of digital signatures using Elgamal, the private key is denoted as "D" and the public key as "alpha". The setup phase involves generating the private key by subtracting "D" from a designated element "G". The public key consists of a triple, including the actual public key and certain parameters.

Elgamal digital signatures differ from Elgamal encryption in that they involve two public keys. One is the long-term public key denoted as "beta", while the other is unique to each message. The latter is referred to as a temporary private key and is used in conjunction with a temporary public key. The signing process becomes more complex, as it requires an ephemeral key denoted as "K_sub_e" specific to each message. This ephemeral key must satisfy the condition that its greatest common divisor with "P-1" is equal to one.

To compute the signature, the parameter "E" is calculated as "alpha" raised to the power of "K_e" modulo "P". The second part of the signature, denoted as "Z", is obtained by multiplying the difference between "S" and "D" with "R" and the inverse of "K". Unlike previous signatures, which consisted of a single value, Elgamal digital signatures involve multiple 24-bit values.

To verify the signature, the recipient (Alice) computes an auxiliary parameter "T" using the public key "beta" raised to the power of eight multiplied by "R" raised to the power of eight modulo "P". Alice then checks if "T" is congruent to "alpha" raised to the power of "X" modulo "P". If the congruence holds, the signature is deemed valid; otherwise, it is considered invalid.

Elgamal digital signatures in classical cryptography involve the use of discrete logarithms and multiple public keys. The signing process requires ephemeral keys specific to each message, and the resulting signature consists of multiple 24-bit values. Verification involves computing an auxiliary parameter and checking for congruence with the original public key.

Digital signatures play a crucial role in ensuring the authenticity and integrity of digital documents. One widely used digital signature scheme is the Elgamal digital signature scheme. In this scheme, the signer generates a pair of keys: a private key and a corresponding public key. The private key is kept secret, while the public key is

shared with others.

To create a digital signature, the signer follows a specific process. First, the signer computes a random value, denoted as "r." Then, the signer calculates a value called "R," which is equal to the public key raised to the power of "r." Next, the signer computes a value called "e," which is derived from the message being signed. Finally, the signer calculates the signature value "s" using the formula s = (e - x*r) * (r^-1) mod (p-1), where "x" is the signer's private key, "p" is a prime number, and "r^-1" is the modular inverse of "r" modulo (p-1).

To verify the digital signature, the verifier performs the following steps. First, the verifier computes a value called "v," which is equal to the public key raised to the power of "s" multiplied by "R" raised to the power of "e." Then, the verifier checks whether "v" is equal to "R" raised to the power of "x" modulo "p." If the equality holds, the signature is considered valid; otherwise, it is considered invalid.

The proof of correctness for the Elgamal digital signature scheme involves substituting values and applying mathematical principles. By substituting the values used in the signature generation process and applying modular arithmetic properties, it can be shown that the verification equation holds true. This provides assurance that the signature verification process is correct when the signature is constructed using the prescribed method.

It is worth noting that the bit length of the signature parameters "R" and "s" in the Elgamal digital signature scheme is twice the bit length of the signed message, which may not be ideal in terms of efficiency. However, this is a trade-off that is acceptable considering the security provided by the scheme.

The Elgamal digital signature scheme is a widely used cryptographic technique for providing digital signatures. By following a specific process and applying mathematical principles, the scheme ensures the authenticity and integrity of digital documents. Understanding the proof of correctness for this scheme is essential for ensuring the proper implementation and verification of digital signatures.

Digital signatures are an important aspect of cybersecurity, as they provide a means to verify the authenticity and integrity of digital messages. One commonly used digital signature algorithm is the Elgamal digital signature.

To understand the Elgamal digital signature, let's first look at the concept of a digital signature. A digital signature is a mathematical scheme that verifies the authenticity of a digital message. It involves the use of cryptographic techniques to generate a unique signature for each message.

The Elgamal digital signature algorithm is based on the Elgamal encryption scheme, which is a public-key encryption algorithm. In the Elgamal digital signature, the signer generates a pair of keys: a private key and a public key. The private key is kept secret and used to generate the digital signature, while the public key is shared with others to verify the signature.

The process of generating an Elgamal digital signature involves several steps. First, the signer selects a large prime number, typically with a length of 2048 bits, as the modulus for the encryption scheme. This prime number is used to define the size of the keys and the length of the signature.

Next, the signer generates a random number, known as the ephemeral key, for each message to be signed. The ephemeral key is used in the signature generation process and must be unique for each message. Reusing the ephemeral key for multiple messages can lead to vulnerabilities and compromises the security of the digital signature.

The signature generation process involves raising the ephemeral key to a power modulo the prime number. This computation, known as exponentiation, is computationally intensive and requires significant computational resources. Additionally, the signer must perform other computations, such as modular multiplications and inversions, to generate the final signature.

It is important to note that the length of the signature is directly proportional to the length of the prime number used in the encryption scheme. Longer prime numbers result in longer signatures. While longer signatures may be acceptable for certain applications, they can pose challenges for devices with limited computational capabilities or bandwidth constraints.

The Elgamal digital signature algorithm is widely used and serves as the basis for other popular digital signature algorithms, such as the Digital Signature Algorithm (DSA). DSA is commonly used in various applications, including secure communication protocols and digital certificates.

The Elgamal digital signature algorithm is a widely used cryptographic technique for verifying the authenticity and integrity of digital messages. It involves the generation of unique signatures using a private key and the verification of these signatures using a corresponding public key. However, it is crucial to ensure the uniqueness of the ephemeral key for each message to maintain the security of the digital signature.

In classical cryptography, digital signatures play a crucial role in ensuring the authenticity and integrity of digital messages. One widely used digital signature scheme is the Elgamal digital signature scheme. In this scheme, a private key is used to generate a signature, and a corresponding public key is used to verify the signature.

To understand the Elgamal digital signature scheme, let's consider an example. Suppose we have two parties, Alice and Bob. Bob wants to send a message to Alice and wants to ensure that the message is not tampered with during transmission. To achieve this, Bob uses the Elgamal digital signature scheme.

First, Bob generates a pair of keys - a private key (D) and a public key (P, alpha, beta). The public key is shared with Alice, while the private key is kept secret. The private key D is an integer, and the public key consists of three integers - P, alpha, and beta.

To sign a message, Bob follows a series of steps. He selects a random integer k and computes two values - r and s. The value r is computed as alpha raised to the power of k modulo P. The value s is computed as (k^(-1) * (hash(message) - D * r)) modulo (P-1), where hash(message) is the hash value of the message.

Bob then sends both r and s along with the message to Alice. Upon receiving the message, Alice can verify the signature by performing the following calculations. She computes two values - u and v. The value u is computed as (beta^r * r^s) modulo P. The value v is computed as alpha^(hash(message)) modulo P.

If u is equal to v, then the signature is valid, indicating that the message has not been tampered with during transmission.

Now, let's consider the security of the Elgamal digital signature scheme. It is important to note that the security of this scheme relies on the secrecy of the private key D. If an attacker, let's call him Oscar, can somehow obtain the private key D, he can forge valid signatures and impersonate Bob.

To illustrate this, let's assume that Oscar has intercepted a message signed by Bob. Oscar knows the public key (P, alpha, beta) and the signature (r, s). Oscar's goal is to compute the private key D.

Oscar can compute the private key D by using the equation D = (hash(message) - s * r^(-1)) * k modulo (P-1). This equation can be derived from the calculations performed by Bob during the signature generation.

Once Oscar has the private key D, he can generate valid signatures for any message, impersonating Bob. This highlights the importance of keeping the private key secret and not reusing the ephemeral key k.

To prevent such attacks, it is crucial to generate a new ephemeral key k for each signature. Additionally, the Elgamal digital signature scheme should not reuse the same ephemeral key k for different messages.

The Elgamal digital signature scheme is a widely used classical cryptography scheme for ensuring the authenticity and integrity of digital messages. However, it is important to generate a new ephemeral key for each signature and not reuse the same key. This prevents attacks that can lead to the compromise of the private key and the ability to forge valid signatures.

In classical cryptography, digital signatures play a crucial role in ensuring the authenticity and integrity of digital messages. One widely used digital signature scheme is the Elgamal digital signature scheme. In this scheme, a signer generates a pair of keys: a private key and a corresponding public key. The private key is kept secret, while the public key is made available to anyone who wants to verify the signatures.

To create a digital signature using the Elgamal scheme, the signer follows these steps:

1. Compute R and s: The signer selects a random value R and computes s such that a specific check equation holds. This equation ensures that the signature is valid and can be verified by anyone using the signer's public key.

2. Compute the message: Once the signature is computed, the signer computes the message value X, which is equal to s times a parameter I modulo P-1. This message, along with R and s, is sent to the recipient.

3. Verification: The recipient, who has access to the signer's public key, performs the verification process to ensure the authenticity of the message. The recipient computes a value called T, which is equal to beta raised to the power of R times R raised to the power of s modulo P. Beta is a parameter obtained from the signer's public key. The recipient then checks if T is equal to alpha raised to the power of X modulo P, where alpha is another parameter obtained from the public key.

If T is equal to alpha raised to the power of X modulo P, the recipient concludes that the signature is valid. Otherwise, the signature is considered invalid.

The Elgamal digital signature scheme provides a way to ensure the integrity and authenticity of digital messages. However, it is important to note that this scheme is susceptible to attacks, such as existential forgery, where an attacker can create a valid-looking signature without knowing the private key.

The Elgamal digital signature scheme is a widely used cryptographic scheme that allows for the creation and verification of digital signatures. It provides a way to ensure the authenticity and integrity of digital messages. However, it is important to be aware of the potential vulnerabilities and attacks that can compromise the security of this scheme.

Digital signatures play a crucial role in ensuring the authenticity and integrity of digital messages. One widely used digital signature scheme is the Elgamal digital signature.

The Elgamal digital signature scheme is based on the Diffie-Hellman key exchange protocol and is named after its creator, Taher Elgamal. It provides a way for the signer to generate a digital signature using their private key, which can then be verified by anyone using the corresponding public key.

In the Elgamal digital signature scheme, the signer first generates a pair of keys: a private key and a public key. The private key is kept secret and is used for signing messages, while the public key is made available to anyone who wants to verify the signatures.

To generate a digital signature for a message, the signer randomly selects a secret value, typically denoted as "k". Using this secret value, the signer computes two values: "r" and "s". The value "r" is calculated as the modular exponentiation of a generator value raised to the power of "k". The value "s" is calculated as the modular multiplication of the inverse of the signer's private key multiplied by the sum of the message's hash value and the product of the secret value "k" and the signer's private key.

The resulting pair of values ("r" and "s") forms the digital signature for the message. The signer then sends the message along with the digital signature to the recipient.

To verify the digital signature, the recipient uses the signer's public key to compute two values: "w" and "v". The value "w" is calculated as the modular exponentiation of a generator value raised to the power of the message's hash value. The value "v" is calculated as the modular multiplication of the modular exponentiation of the public key raised to the power of "r" and the modular exponentiation of the generator value raised to the power of "s".

If the calculated value of "v" matches the value of "w", then the digital signature is considered valid. Otherwise, it is considered invalid.

The Elgamal digital signature scheme provides a way for the signer to generate valid signatures for any message, without the need to control the message itself. This property makes the scheme resistant to forgery, as the signer cannot generate a valid signature for a different message without knowing the secret value "k".

The Elgamal digital signature scheme is a powerful cryptographic technique that allows for the generation and verification of digital signatures. It provides a way to ensure the authenticity and integrity of digital messages, making it an essential tool in modern cybersecurity.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: HASH FUNCTIONS
TOPIC: INTRODUCTION TO HASH FUNCTIONS

Hash Functions - Introduction to hash functions

Hash functions are auxiliary functions used in cryptography. They translate data into a fixed-size output, known as a hash value or hash code. Hash functions are not used for encryption, but rather in conjunction with other cryptographic mechanisms. They have various applications, including signatures, message authentication codes, key derivation, and random number generation.

One important application of hash functions is in digital signatures. In a digital signature protocol, Alice and Bob exchange public keys. Alice uses a signature function to sign her message X using her private key. She then sends the message and the signature over the channel to Bob. Bob can verify the authenticity of the message by using Alice's public key and the signature.

However, there is a limitation when using hash functions in digital signatures. If a hash function like RSA is used, the length of the message that can be signed is restricted to the size of the hash function's output, typically 256 bytes. This poses a problem when dealing with longer messages, such as PDF files.

To overcome this limitation, an ad hoc approach is often used. The message is divided into blocks, and each block is individually signed. However, this approach is insecure and impractical for several reasons.

One practical problem is that important attachments at the end of a message may be left unsigned if the message is divided into blocks. An attacker can exploit this by dropping or interrupting the transmission of certain blocks, resulting in an incomplete or manipulated signature.

Another issue is the possibility of reordering or exchanging blocks or messages. This can further compromise the integrity and authenticity of the signature.

Hash functions are crucial in real-world cryptographic implementations. They have various applications, including digital signatures. However, the limitations of hash functions in signing longer messages require careful consideration and alternative approaches.

Hash Functions - Introduction to hash functions

In classical cryptography, hash functions play a crucial role in ensuring data integrity, non-repudiation, and security. Unlike block-level encryption, where the message is treated as a whole, hash functions process individual messages. However, this approach has its drawbacks.

Firstly, from a security perspective, treating messages individually is not ideal. It leaves room for attacks similar to those against electronic codebook modes. While the signature may still work on a block level, it lacks the same level of security when applied to individual messages.

Secondly, from a practical standpoint, using hash functions for long messages can be problematic. For instance, if a message is one megabyte in size, and the signature can only handle 156 bytes at a time, it would require one million hours of exponentiation to generate a signature. This would make the process extremely slow and inefficient.

To address these issues, the solution lies in compressing the message before signing it. This is where hash functions come into play. By applying a hash function, such as H, to the message, we can compress it into a shorter form. This compressed output can then be easily signed, reducing the computational burden.

To illustrate this concept, consider a message X, which is a 256-byte PDF file. By feeding this message into the hash function H, we obtain a shorter output, denoted as Z. The signature operation is then performed on Z, rather than the entire message. This significantly reduces the computational complexity, as the signature operation is only performed once on the shorter output.

This approach forms the basis of the basic protocol for digital signatures with hash functions. In this protocol, instead of directly signing the message X, we compute the hash output Z using the hash function H. The hash output Z is then signed using the private key. The message and the signature are sent over as the inputs for verification. The verification process involves using the public key to verify the signature, along with the hash output Z.

One important aspect to note is that the message itself is not directly involved in the verification process. Instead, the verification function requires the signature and the hash output Z. To obtain the hash output Z, the verifier recomputes the hash function using the received message.

Hash functions are essential in classical cryptography for ensuring data integrity and security. By compressing messages before signing them, hash functions simplify the signature process and improve efficiency. Understanding the basic protocol for digital signatures with hash functions is fundamental for anyone interested in cybersecurity.

Hash Functions - Introduction to hash functions

Hash functions are an essential part of classical cryptography. They are used to transform input data into a fixed-size output, called a hash value or message digest. In this didactic material, we will explore the basics of hash functions and their requirements.

The motivation behind using hash functions is to address the limitation of signing long messages. Hash functions allow us to create a fingerprint or summary of a message, regardless of its length. This fingerprint, also known as the message digest, serves as a validation or verification process for the message.

The first requirement for hash functions is to support arbitrary input lengths. This means that the hash function should work with any data length, whether it's a short email or a large file. We want to avoid constraints on the length of the input data.

On the output side, we need fixed and short output lengths. This is because traditional signing algorithms work best with shorter outputs. We want to ensure that the hash function generates a fixed-size output, which can be easily signed and verified.

Another important requirement for hash functions is efficiency. Computationally, hash functions should be fast and efficient. We don't want to wait for a long time when processing large amounts of data. Speed is crucial, especially when dealing with software applications.

In addition to these requirements, there are two more requirements related to security. The first one is called preimage resistance. It means that given a hash value, it should be computationally infeasible to find the original input that produced that hash value. This ensures that the hash function is secure against reverse engineering and finding the original data from its hash value.

The second requirement is called collision resistance. It means that it should be extremely difficult to find two different inputs that produce the same hash value. This property ensures that it is highly unlikely for two different messages to have the same hash value, which would compromise the integrity of the hash function.

Hash functions are essential tools in classical cryptography. They provide a way to create a fixed-size summary or fingerprint of input data. Hash functions should support arbitrary input lengths, have fixed and short output lengths, be efficient in computation, and have preimage and collision resistance properties for security.

Hash Functions - Introduction to Hash Functions

Hash functions are an essential component of classical cryptography. They provide a way to transform data of arbitrary size into a fixed-size output, known as the hash value or hash code. In this didactic material, we will explore the concepts of preimage resistance, second preimage resistance, and collision resistance in hash functions.

Preimage resistance, also known as one-wayness, refers to the property that it should be computationally infeasible to determine the original input data from its hash output. In other words, given a hash value, it should

be impossible to compute the original input. This property is crucial for various applications, such as key derivation and digital signatures.

Second preimage resistance, on the other hand, ensures that it is difficult to find a different input that produces the same hash value as a given input. This property is important in scenarios where an attacker intercepts a message and attempts to modify it without being detected. For example, if a message instructs a bank to transfer 10 euros, an attacker should not be able to change the amount to 10000 euros without the change being detected.

Collision resistance is the property that two different inputs should not produce the same hash value. In other words, it should be difficult to find two inputs that collide, meaning they produce identical hash values. This property is crucial in preventing malicious actors from creating fraudulent data that has the same hash value as legitimate data.

To understand the importance of collision resistance, let's consider a scenario where an attacker, Oscar, intercepts a message from Bob to a bank, requesting a transfer of 10 euros. Oscar wants to change the message to request a transfer of 10000 euros without being detected. If Oscar can find two different inputs, X1 and X2, that produce the same hash value, he can replace the original message with X2, which requests the larger transfer amount. If the hash values of X1 and X2 are the same, the bank's verification process will not detect the fraudulent change.

It is important to note that the hash function operates on the hash output, not the original message itself. This means that if an attacker can manipulate the hash value, they can potentially bypass the verification process. Therefore, it is crucial to have hash functions that possess second preimage resistance and collision resistance to prevent such attacks.

Hash functions play a vital role in classical cryptography by transforming data into fixed-size hash values. Preimage resistance ensures that it is computationally infeasible to determine the original input from its hash output. Second preimage resistance prevents finding a different input with the same hash value. Collision resistance ensures that it is difficult to find two inputs that produce the same hash value. These properties are essential for maintaining the integrity and security of cryptographic systems.

Hash Functions - Introduction to hash functions

In classical cryptography, hash functions play a crucial role in ensuring data integrity and security. A hash function is a mathematical function that takes an input (or message) and produces a fixed-size output called a hash value or digest. This hash value is unique to the input data, meaning that even a small change in the input will result in a completely different hash value.

The purpose of a hash function is to provide a way to verify the integrity of data. By comparing the hash values of two sets of data, we can quickly determine if they are identical or not. If the hash values match, we can be confident that the data has not been tampered with. Hash functions are widely used in various applications, including digital signatures, password storage, and data integrity checks.

However, it is important to note that hash functions are not foolproof. In some cases, it is possible to find two different inputs that produce the same hash value. This is known as a collision. Collisions are undesirable because they can be exploited by malicious actors to create fake data or alter the integrity of the original data.

One example of a collision attack is the "Article X" scenario. In this scenario, Bob is tricked by Oscar into signing a document with a specific hash value (X1). However, Oscar intercepts the document and replaces it with a different one that has the same hash value (X1). When Alice, who knows Bob's public key, verifies the document, she unknowingly accepts the fake document as genuine.

To prevent collision attacks, hash functions need to be designed in a way that makes it extremely difficult to find two inputs that produce the same hash value. The strength of a hash function lies in its resistance to collision attacks. A stronger hash function will have a lower probability of collisions.

It is worth noting that collision attacks are more challenging to prevent than other types of attacks, such as second preimage attacks. In a collision attack, the attacker aims to find any two inputs that produce the same

hash value, while in a second preimage attack, the attacker aims to find a specific input that produces the same hash value as a given input. Collision attacks are generally more powerful and can cause significant security issues.

The topic of hash functions is an active and evolving field in cybersecurity. The development of new hash functions is an ongoing process, with the aim of creating stronger and more secure algorithms. The cybersecurity community is actively working on creating new hash functions that are resistant to collision attacks.

Hash functions are fundamental tools in classical cryptography that ensure data integrity and security. However, they are not immune to collision attacks, which can compromise the integrity of data. The development of stronger hash functions is an ongoing process in the cybersecurity community to address this issue.

A hash function is a fundamental concept in classical cryptography. It is a mathematical function that takes an input and produces a fixed-size output, known as the hash value or hash code. In this didactic material, we will explore the concept of hash functions and their significance in cybersecurity.

Hash functions are designed to be one-way functions, meaning that it is computationally infeasible to reverse-engineer the input from the output. This property makes them useful for various applications, such as data integrity verification, password storage, and digital signatures.

One important aspect of hash functions is the possibility of collisions. A collision occurs when two different inputs produce the same hash value. It is important to note that collisions are inevitable due to the nature of hash functions. This is because the input space, which represents all possible inputs, is much larger than the output space, which represents all possible hash values.

To illustrate this concept, let's consider an analogy. Imagine a drawer with a finite number of compartments and a larger number of socks. If there are more socks than compartments, it is guaranteed that at least one compartment will contain multiple socks. This analogy represents the concept of collisions in hash functions.

There are two terms commonly used to describe this phenomenon. The first is the "pigeonhole principle," which states that if there are more pigeons than available pigeonholes, there must be at least one pigeonhole with multiple pigeons. The second term is the "birthday paradox," which refers to the counterintuitive fact that in a group of just 23 people, there is a 50% chance that two people share the same birthday.

Given that collisions are unavoidable, the focus shifts to making collisions difficult to find. This is where the strength of a hash function lies. A secure hash function should make it computationally impractical to find two inputs that produce the same hash value.

Attackers can attempt to find collisions by manipulating the input in various ways. For example, they can add or modify characters in the message while maintaining its semantic meaning. They can also take advantage of unused bits in character encodings or introduce invisible characters to generate multiple variations of the same message.

However, it is important to note that finding collisions in a secure hash function is a complex task that requires significant computational resources. The complexity increases exponentially with the size of the hash output.

Hash functions play a crucial role in cybersecurity by providing a means to verify data integrity and secure sensitive information. While collisions are inevitable, the challenge lies in making collisions difficult to find. Secure hash functions are designed to withstand attacks and ensure the integrity and authenticity of data.

A hash function is a fundamental concept in classical cryptography that plays a crucial role in ensuring data integrity and security. In this context, a hash function takes an input, which can be of any length, and produces a fixed-size output called a hash value or digest. The primary purpose of a hash function is to convert data into a unique representation that is computationally infeasible to reverse-engineer or recreate the original input.

One important property of hash functions is that they should be deterministic, meaning that the same input will always produce the same output. Additionally, even a small change in the input should result in a significantly

different output. This property is known as the avalanche effect and is essential for ensuring the integrity of the data.

In the context of classical cryptography, hash functions are extensively used for various purposes, including data integrity checks, password storage, and digital signatures. They are designed to be computationally efficient, making them suitable for real-time applications.

When analyzing the security of hash functions, one crucial aspect to consider is the likelihood of collisions. A collision occurs when two different inputs produce the same hash value. In the context of hash functions, finding a collision is considered a significant security vulnerability, as it allows an attacker to manipulate data without detection.

To understand the likelihood of collisions, let's consider an analogy known as the birthday paradox. Imagine you are hosting a party and want to know how many people you need to invite to have at least two individuals with the same birthday. Surprisingly, the answer is much lower than expected. With only 23 people, there is a 50% chance of a collision.

This concept applies to hash functions as well. If we have T input values and one output, the likelihood of a collision can be calculated using the formula P = 1 - (365/365) * (364/365) * ... * ((365 - T + 1)/365). For example, with T = 23, the probability of a collision is approximately 50%.

In the context of hash functions, the output space refers to the number of possible hash values that can be generated. Unlike the 365 possible birthdays in the birthday paradox analogy, hash functions typically have a much larger output space. This ensures that the likelihood of a collision is significantly reduced, making it computationally infeasible to find two different inputs that produce the same hash value.

It is important to note that modern hash functions, such as SHA-256 (Secure Hash Algorithm 256-bit), have significantly larger output spaces and are designed to be resistant to collision attacks. They are extensively used in various cryptographic applications, including secure communication protocols and digital signatures.

Hash functions are an essential component of classical cryptography, providing data integrity and security. They convert data into fixed-size hash values, ensuring the uniqueness of the representation. Understanding the likelihood of collisions is crucial in evaluating the security of hash functions, and modern algorithms are designed to provide a high level of resistance against collision attacks.

A hash function is an essential component of classical cryptography that plays a crucial role in ensuring data integrity and security. In this lesson, we will introduce hash functions and discuss their significance in cybersecurity.

A hash function takes an input, known as a message, and produces a fixed-size output, known as a hash value or digest. The output is typically a string of characters that is unique to the input message. Hash functions are designed to be quick and efficient, allowing for fast computation of the hash value.

One important property of hash functions is that they are deterministic, meaning that for a given input, the output will always be the same. This property enables the verification of data integrity, as any change in the input message will result in a different hash value.

Hash functions are widely used in various applications, including password storage, digital signatures, and data integrity checks. They provide a way to securely store passwords by hashing them and comparing the hash values instead of storing the actual passwords. This protects user passwords in case of a data breach.

Another crucial property of hash functions is their resistance to collisions. A collision occurs when two different input messages produce the same hash value. A good hash function should minimize the probability of collisions, making it computationally infeasible to find two different messages with the same hash value.

The formula mentioned in the transcript is a key aspect of understanding the collision resistance of hash functions. The formula describes the relationship between the number of inputs (T) and the probability of at least one collision. By plugging in the appropriate values, we can determine the required output length to achieve a desired level of security.

For example, if we have 80 output bits and want a 50/50 chance of a collision, the formula helps us calculate the necessary output length. It reveals that in order to achieve 80-bit security, we need an output length of 160 bits.

Understanding the collision resistance of hash functions is crucial in ensuring the security of cryptographic systems. It highlights the importance of choosing appropriate output lengths to prevent the possibility of collisions and maintain data integrity.

Hash functions are fundamental tools in classical cryptography that provide data integrity and security. They enable the efficient computation of unique hash values for input messages and play a vital role in various cybersecurity applications. Understanding the collision resistance of hash functions is essential in designing secure cryptographic systems.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: HASH FUNCTIONS
TOPIC: SHA-1 HASH FUNCTION

This part of the material is currently undergoing an update and will be republished shortly.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: MESSAGE AUTHENTICATION CODES
TOPIC: MAC (MESSAGE AUTHENTICATION CODES) AND HMAC

Welcome to this lecture on message authentication codes (MAC) and HMAC. In this lecture, we will explore the concept of MAC, which can be thought of as digital signatures implemented using symmetric cryptography. We will also discuss HMAC, which is a hash-based MAC.

Before diving into the details, let's briefly recall the motivation behind digital signatures. In the real world, documents are often signed to ensure their authenticity and integrity. In the digital world, we aim to achieve the same level of assurance. Digital signatures provide a means to authenticate a message, ensuring that it comes from the right sender. This process is known as message authentication.

Now, let's move on to MACs. A message authentication code (MAC) is a cryptographic checksum that can be used to authenticate a message. It is similar to a digital signature but is implemented using symmetric cryptography. MACs are also known as cryptographic checksums, which is a more descriptive term.

The main goal of a MAC is to ensure the integrity and authenticity of a message. To achieve this, we use a symmetric key that is shared between the sender (Alice) and the receiver (Bob). The sender computes the MAC of the message using the shared key, and the receiver verifies the MAC using the same key. If the MAC verification is successful, it indicates that the message has not been tampered with and comes from the expected sender.

To build a MAC, we can utilize hash functions. Hash-based MAC (HMAC) is a widely used approach to implement MACs. HMAC combines a hash function with a secret key to generate the MAC. By using a hash function, we can ensure the integrity of the message, and by using a secret key, we can verify the authenticity of the message.

HMAC offers a practical and efficient solution for message authentication. It allows us to achieve similar results as digital signatures while leveraging the speed and efficiency of symmetric cryptography. By using MACs, we can authenticate messages in various scenarios where symmetric block ciphers and hash functions are sufficient.

MACs provide a means to authenticate messages using symmetric cryptography. They ensure the integrity and authenticity of the message by utilizing a shared secret key. HMAC, a hash-based MAC, is a commonly used approach to implement MACs.

In the study of cryptography, an important aspect is message authentication, which ensures the integrity and origin of a message. In this context, we will discuss the concept of Message Authentication Codes (MAC) and HMAC.

A Message Authentication Code (MAC) is a cryptographic checksum computed over a message using a symmetric key. It provides a way to verify the authenticity and integrity of a message. The MAC algorithm takes the message as input and produces a fixed-length output, called the MAC value. This MAC value is then appended to the original message.

The process of computing a MAC involves using a symmetric key, which is shared between the sender and the receiver. The sender computes the MAC value by applying the MAC algorithm to the message using the shared key. The receiver, on the other hand, recomputes the MAC value using the same algorithm and the shared key. The receiver then compares the computed MAC value with the one received from the sender to verify the authenticity and integrity of the message.

One important property of MAC is that it can accept messages of arbitrary lengths. Unlike digital signatures, which have limitations on the length of the message, MAC allows for the processing of messages of varying lengths without the need for additional steps such as hashing. This makes MAC more practical and efficient.

Another desirable property of MAC is that the length of the MAC value remains fixed regardless of the length of the input message. This means that whether the input is a short message or a large file, the MAC value will always have the same length. This property is useful in ensuring a consistent and efficient cryptographic

checksum for any message, independent of its length.

In terms of security services, MAC provides message authentication, which means that if a message claims to be from a specific sender, the receiver can verify if it is indeed from that sender and has not been tampered with. By comparing the computed MAC value with the received MAC value, the receiver can determine the authenticity of the message.

It is important to note that MAC does not provide non-repudiation, which means that it does not prevent the sender from denying their involvement in the message. However, MAC is a useful tool in ensuring the integrity and origin of a message within a secure communication channel.

Message Authentication Codes (MAC) are cryptographic checksums computed over messages using a shared symmetric key. MAC provides a way to verify the authenticity and integrity of a message. It allows for the processing of messages of arbitrary lengths and ensures a fixed-length cryptographic checksum. MAC provides message authentication, allowing the receiver to verify the origin of a message. However, MAC does not provide non-repudiation.

In the study of advanced classical cryptography, one important topic is Message Authentication Codes (MAC) and HMAC (Hash-based Message Authentication Code). A MAC is a cryptographic technique used to verify the integrity and authenticity of a message. It ensures that the message has not been tampered with during transmission and that it originates from a trusted source.

To understand the concept of MAC, we need to consider the role of a secret key. For a MAC to be valid, both the sender and receiver must possess the same secret key. This key is used to compute the MAC value for the message. If the MAC value is correct, it indicates that the message was computed by someone who knows the secret key.

It is important to note that the security of MAC protocols relies on the assumption that key distribution works effectively. If an unauthorized party gains access to the secret key, the security of the MAC is compromised. Therefore, a secure channel for key distribution is crucial.

The purpose of a MAC is to provide a security service called message authentication. This service ensures that the message is indeed from the claimed sender, in this case, Bob. It also guarantees the integrity of the message, meaning that any tampering or manipulation during transmission will be detected by the receiver, Alice.

Let's consider a practical example of a financial transaction. Suppose the message is a request to transfer $10 to Oscar's account. However, there is a malicious actor, Oscar, who wants to alter the transaction to transfer $10,000 instead. Can Oscar successfully replace the original message with his altered version?

The answer is no. The MAC verification process will fail because the altered message will produce a different MAC value. This failure ensures the integrity of the security service. Even if Oscar attempts to manipulate the message by flipping a single bit, the resulting MAC value will be completely different.

In addition to MAC, another crucial security service is provided by digital signatures. A digital signature allows the recipient of a message to verify the authenticity and integrity of the message. It prevents non-repudiation, which is the denial of generating a particular message.

Consider the scenario where Bob orders a car from Alice, the car dealer. Bob fills out a web form with the order details and attaches his signature using a MAC. Later, Alice claims that she never received the order. In this case, Bob needs to prove to a judge or a registrar that he indeed generated the message. However, due to the symmetric setup of the MAC, it is not possible to prove who generated the message. Therefore, non-repudiation is not achieved in this scenario.

To implement MACs, one approach is to use hash functions. A hash function, denoted as H, takes an input and produces a fixed-size output called a hash value. The basic idea is to bind together the key (K) and the message (X) using a hash function. The output of the hash function, denoted as M, becomes the MAC value.

By using a hash function, we can scramble the key and message together, creating a function that is difficult to

attack. Hash functions have desirable properties that make them suitable for MACs. They are resistant to pre-image attacks, meaning it is computationally infeasible to find the original input given the hash value.

Message Authentication Codes (MAC) and HMAC play a crucial role in ensuring the integrity and authenticity of messages in cybersecurity. They rely on the use of secret keys and hash functions to bind the key and message together, creating a MAC value that can be verified by the receiver. However, it is important to note that MACs do not provide protection against dishonest parties trying to cheat each other.

In the field of cybersecurity, message authentication codes (MAC) play a crucial role in ensuring the integrity and authenticity of transmitted messages. MAC provides a way to verify that a message has not been tampered with during transmission and that it originated from a trusted source.

There are two common approaches to constructing MACs: the secret prefix and secret suffix methods. In the secret prefix method, the key is concatenated with the message, and then the resulting string is hashed. In the secret suffix method, the message is concatenated with the key before hashing. While these methods may seem intuitive, they both have weaknesses that can be exploited.

One weakness of the secret prefix method is that an attacker can generate their own message by appending their chosen value to the original message. This allows them to manipulate the resulting MAC. Similarly, in the secret suffix method, an attacker can prepend their chosen value to the original message, altering the MAC.

To better understand these weaknesses, let's delve into the details of how MACs are constructed. Typically, the message is divided into blocks, and each block is hashed individually. For example, if we use the SHA-1 hash function, the input width for each block is 512 bits. In the secret prefix method, the key is hashed first, followed by each block of the message. In the secret suffix method, the message blocks are hashed first, followed by the key.

Most hash functions use the Merkle-Damgard construction, where a compression function is applied iteratively to process the blocks. This construction also incorporates an initial vector (IV) to enhance security.

Now, let's consider a scenario where Alice and Bob are communicating using MACs. Bob computes the MAC by concatenating the key with each block of the message and hashing the result. However, an attacker named Oscar interferes with the transmission and inserts his own message block, denoted as Xn+1. This allows Oscar to manipulate the MAC and potentially deceive Alice.

It is important to be aware of these weaknesses when designing and implementing MACs. By understanding the vulnerabilities, we can take appropriate measures to mitigate the risks associated with message authentication.

A message authentication code (MAC) is a cryptographic technique used to verify the integrity and authenticity of a message. It is a form of classical cryptography that provides a way to ensure that a message has not been tampered with during transmission.

The MAC is computed using a secret key and the message itself. The process involves hashing the message and the key together to produce a unique output, which is the MAC. This MAC is then appended to the message and sent along with it.

To compute the MAC, the sender first feeds the key into the algorithm. Then, the message is iteratively processed, with each block being hashed along with the previous blocks. At the end of the iteration, the final output is the MAC.

However, there is a vulnerability in this process. An attacker, named Oscar, can intercept the message and append his own malicious blocks to it. To do this, he computes his own MAC for the modified message. He can then send this modified message, along with the fake MAC, to the receiver.

To verify the authenticity of the message, the receiver, named Ellis, recomputes the MAC using the same process as the sender. If the recomputed MAC matches the one received with the message, Ellis considers the message to be valid.

This vulnerability can be mitigated by using a technique called padding with length information. By including the

length of the message in the hashing process, the attacker's attempt to append malicious blocks can be detected. However, not all hash functions include this padding with length information, so it is important to use hash functions that provide this protection.

Another approach to prevent this vulnerability is to use a secret suffix instead of a secret prefix. In this case, the message is hashed first, and then the key is included in the hashing process. This prevents an attacker from appending malicious blocks at the end of the message.

It is worth noting that if an attacker can find collisions, where two different messages produce the same hash output, the security of the MAC is compromised. This highlights the importance of using hash functions that are resistant to collisions.

Message authentication codes (MACs) are an important tool in ensuring the integrity and authenticity of messages. However, they can be vulnerable to attacks if not implemented properly. By using techniques such as padding with length information and secret suffixes, the security of MACs can be enhanced.

Message Authentication Codes (MACs) are cryptographic techniques used to ensure the integrity and authenticity of messages. In this context, we will discuss the problem that arises when the same value is appended to both the message and the key.

When the message and the key are concatenated, the output of the MAC becomes the same in both cases. This means that the MAC of X concatenated with K is equal to H concatenated with X in that index. This creates a vulnerability where an attacker, Oscar, can intercept the message and replace X with X' without changing the MAC value.

The question then arises whether this vulnerability poses a significant threat. To answer this, we need to compare the effort required for collision finding, which is necessary for the attack, with the effort required for brute force.

Collision finding is the process of finding two different inputs that produce the same output. In this case, collision finding for the MAC would require less effort than brute force, which involves trying all possible keys. However, the difficulty of collision finding depends on the specific situation and the hash function being used.

Taking the example of the popular hash function SHA-1, with a 128-bit key space, the attack complexity is 2^128. This means that an attacker would need to try 2^128 keys to successfully perform a brute force attack.

On the other hand, collision finding for SHA-1 has a complexity of 2^80, thanks to the birthday paradox. This is because the birthday paradox reduces the number of steps required to find a collision. However, it is important to note that the birthday paradox only applies to weak collision resistance, and not to finding a full collision.

Therefore, using a hash function with an output length that is not long enough may make the MAC vulnerable to the birthday paradox. This compromises the cryptographic strength of the MAC and allows an attacker to gain an advantage.

In practice, it is crucial to choose a hash function with a sufficient output length to ensure the security of the MAC. Additionally, other techniques, such as HMAC (Hash-based Message Authentication Code), can be used to enhance the security of MACs. HMAC combines the properties of a hash function and a secret key to provide stronger authentication and integrity guarantees.

The vulnerability that arises when the same value is appended to both the message and the key in a MAC can be exploited by an attacker. The feasibility of the attack depends on the effort required for collision finding compared to brute force. It is essential to choose a secure hash function and employ additional techniques, such as HMAC, to ensure the security of MACs.

Message Authentication Codes (MAC) and HMAC are important concepts in the field of cybersecurity. MAC is a type of cryptographic algorithm used to verify the integrity and authenticity of a message. It ensures that the message has not been tampered with during transmission. HMAC, on the other hand, is a specific construction of MAC that provides additional security features.

The concept of MAC was proposed in the mid-1970s and has since been widely used in various applications, such as SSL and TLS protocols. These protocols are commonly used to establish secure connections between web browsers and servers. The presence of a small lock icon in the browser indicates a secure connection.

The idea behind MAC is to use two nested hash functions, namely the inner hash and the outer hash. This construction helps prevent certain vulnerabilities, such as collision attacks. In the HMAC construction, the keys undergo preprocessing before being used in the hash functions.

In the outer hash, the key is XORed with a fixed value called the "outer pad." The key is also expanded to match the length of the hash function's input. Similarly, in the inner hash, the key goes through a preprocessing step and is XORed with a different fixed value called the "inner pad." The message is then appended to the inner hash.

To illustrate this concept, let's consider an example. Suppose we have a 128-bit key and a 512-bit hash function. In this case, the key would be padded with zeros and appended to the outer pad. The inner pad would be a different fixed bit pattern, and the key would undergo the same preprocessing steps. The message would then be appended to the inner hash.

It is important to note that the specific bit patterns used for the pads are defined in the standard and not arbitrarily chosen. The pads ensure that the input lengths of the hash functions match the desired length.

MAC and HMAC are cryptographic techniques used to ensure the integrity and authenticity of messages. They involve the use of nested hash functions and preprocessing of keys. These techniques are widely used in various applications to provide secure communication.

In classical cryptography, message authentication codes (MAC) play a crucial role in ensuring the integrity and authenticity of transmitted messages. In this context, HMAC (Hash-based Message Authentication Code) is a widely used algorithm that combines a cryptographic hash function with a secret key to generate a message authentication code.

To better understand the concept of MAC and HMAC, let's examine a block diagram. Please refer to Figure 12.2 on page 324 (or 325) of your textbook. The diagram illustrates the process of generating a MAC using an inner pad, an expanded key, and the message itself. The inner pad is combined with the expanded key, and then hashed together with the message. This initial hashing step may take some time, depending on the length of the message.

It is important to note that although two hashes are mentioned, only one hash is required for a long message. The outer hash, which consists of only two input blocks, has minimal computational overhead. Therefore, the additional work involved in generating the outer hash is negligible compared to the main hashing of the long message.

To enhance security, an initialization vector (IV) is used in conjunction with the HMAC algorithm. The IV adds an extra layer of randomness and uniqueness to the process, further strengthening the integrity of the MAC.

MAC and HMAC are cryptographic techniques used to verify the authenticity and integrity of transmitted messages. The HMAC algorithm combines a secret key with a hash function to generate a message authentication code. By using an inner pad, an expanded key, and the message itself, the MAC is calculated. The outer hash, consisting of only two input blocks, adds minimal computational overhead. The use of an initialization vector enhances the security of the HMAC process.

Thank you for your attention today. If you have any questions, please feel free to ask.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: KEY ESTABLISHING
TOPIC: SYMMETRIC KEY ESTABLISHMENT AND KERBEROS

Today, we will be addressing a fundamental problem in the field of cryptography - key establishment. In the past semesters, we have covered topics such as algorithms, digital signatures, hash functions, and other cryptographic protocols. However, we have largely ignored the issue of key distribution, which is an essential prerequisite for secure communication.

In the typical setup, we use block ciphers or stream ciphers for encryption and decryption. These ciphers provide strong security and fast performance. However, before we can use these ciphers, we need to distribute the key. This is where key distribution protocols come into play.

Over the next three weeks, we will focus on key distribution protocols. We will start by introducing symmetric key distribution, which involves the use of a shared secret key. This is the first time we will be discussing a proper protocol in detail.

The first topic for today is the introduction to symmetric key distribution. We will also discuss key distribution center (KDC) protocols, which are used to securely distribute keys. By the end of this session, everyone will have a clear understanding of these concepts.

Now, let's take a look at the classification of key establishment protocols. There are two main approaches - key transport and key agreement. In key transport, one party, either Alice or Bob, generates the key, and it is then transported to the other party. In key agreement, both parties are involved in generating the key.

Key transport protocols are considered more secure because it is harder for a third party to manipulate the protocol. However, both approaches require solutions that are secure against all possible attacks.

It is important to note that certain block ciphers, such as DES, have weak keys. These weak keys can be exploited by attackers. If both parties are involved in key generation, it becomes more difficult for an attacker to manipulate the protocol.

One example of a key agreement protocol that we have discussed in detail is the Diffie-Hellman protocol. This protocol is based on public key cryptography and allows two parties to establish a shared secret key.

Key establishment is a crucial aspect of cryptography. By understanding the different types of protocols and their security implications, we can ensure the secure distribution of keys for encrypted communication.

Symmetric Key Establishment and Kerberos

In the context of classical cryptography, one of the key challenges is establishing secure keys between users. In this didactic material, we will explore symmetric key establishment and introduce the concept of Kerberos.

Symmetric key establishment involves the distribution of pairwise secret keys between users. To illustrate this, let's consider a small network with four users: Alice, Bob, Chris, and Dorothy. The goal is to enable secure communication between any two users while preventing others from eavesdropping.

In the naive approach, known as N square key distribution, every user pair is assigned a unique key. For example, Alice needs to share keys with Bob, Chris, and Dorothy. Similarly, Bob needs keys for Alice, Chris, and Dorothy. Chris and Dorothy also require keys for communication with the other users.

To establish these keys, a secure channel is needed. This could involve a system administrator physically visiting each user and uploading the necessary keys. However, it is important to note that a secure channel is always required when dealing with symmetric ciphers.

Let's analyze the number of keys in the system. For N users, there are N squared possible key pairs. However, each key pair has a counterpart, resulting in N times (N-1) key pairs. Therefore, the total number of keys is (N times (N-1))/2.

While this may not seem problematic for a small number of users, it becomes significant when dealing with larger organizations. For example, a company with 750 employees would require 280,875 key pairs. This highlights the scalability issues of the N square key distribution method.

To address these challenges, the concept of Kerberos was introduced. Kerberos is a network authentication protocol that provides a secure way to establish and manage keys. It uses a trusted third party, known as the Key Distribution Center (KDC), to facilitate key exchange between users.

In Kerberos, each user has a unique secret key known only to them and the KDC. When two users want to communicate, they request a session key from the KDC, which is then used for secure communication. This eliminates the need for pairwise key distribution and reduces the number of keys required in the system.

Symmetric key establishment is crucial for secure communication in classical cryptography. The N square key distribution method, while simple, can lead to scalability issues as the number of users increases. Kerberos provides an alternative approach by using a trusted third party to manage key exchange and reduce the number of keys required.

In classical cryptography, the establishment of keys is a crucial aspect of ensuring secure communication. One method of key establishment is through symmetric key establishment, which involves the use of a trusted authority known as the Key Distribution Center (KDC). The KDC acts as a central entity that shares a single key, referred to as "ka," with every user in the network.

Symmetric key establishment using the KDC involves a straightforward protocol. Let's consider an example with three participants: Alice, Bob, and the KDC. Alice and Bob are known entities, while the KDC serves as the trusted authority.

To establish a secure communication channel between Alice and Bob, the following steps are taken:

1. Alice initiates the process by sending a request to the KDC, requesting a session key for communication with Bob.
2. Upon receiving the request, the KDC generates a session key, which is a randomly generated symmetric key specifically for the communication between Alice and Bob. Let's call this key "ks."
3. The KDC encrypts the session key "ks" using Alice's key "ka" and sends the encrypted session key to Alice.
4. Alice receives the encrypted session key and decrypts it using her key "ka," obtaining the session key "ks."
5. Alice now has the session key "ks" and can use it to encrypt her messages to Bob.
6. Alice sends a message to Bob, including the encrypted session key "ks."
7. Bob receives the message and decrypts the session key "ks" using his own key.
8. Bob now has the session key "ks" and can use it to decrypt Alice's messages.

This protocol ensures that only Alice and Bob possess the session key "ks," which is required to decrypt the encrypted messages. The KDC acts as a trusted intermediary, facilitating the secure exchange of keys between Alice and Bob.

This approach to key establishment using a trusted authority like the KDC offers several advantages. It eliminates the need for manual key installation and distribution, which can be time-consuming and costly. Additionally, it simplifies the process of adding new users to the network, as the KDC can generate and distribute the necessary keys.

However, it's important to note that this method may not be suitable for networks with frequent changes in users or dynamic network structures. In such cases, more advanced approaches may be required.

In the next part of this lecture, we will explore a practical approach to key distribution using symmetric ciphers like AES or Triple DES. This approach allows for key establishment without relying on the Diffie-Hellman key exchange. We will delve into KDC protocols, which involve the use of a trusted authority for distributing keys.

In the context of symmetric key establishment and Kerberos, the key distribution center (KDC) plays a crucial role in securely sharing keys between users. Each user, such as Alice and Bob, is assigned a unique key. The key establishment process involves the KDC sharing the key with Alice and Bob.

Unlike other scenarios where multiple keys may be assigned to users, in this case, there is only one key per user. However, it is important to note that this key needs to be established only once. This means that whether there are two parties or 750 users in the network, there is still only one key per user.

To facilitate communication between Alice and Bob, a secure channel is required. One way to achieve this is by encrypting the message with the shared key and sending it to the KDC. The KDC would then decrypt the message and re-encrypt it with the recipient's key before sending it to the recipient. However, this approach has several drawbacks.

One major problem is that all traffic would be routed through the KDC, causing a communication bottleneck. To overcome this limitation, a different approach is used in practice. This approach involves a new concept where Alice initiates communication with Bob without Bob's prior knowledge.

Alice sends a request to the KDC containing her ID and Bob's ID. The KDC then generates a random session key, also known as the recorded session key. Here, an exciting and revolutionary concept is introduced. The KDC encrypts the session key using a shared key with Alice, and also encrypts it using Bob's key. Both encrypted keys are then sent to Alice.

At this point, Alice can decrypt and recover the session key from the encrypted key intended for her. However, she cannot do anything with the encrypted key intended for Bob. The session key allows Alice to securely communicate with Bob. She can now encrypt the message using the session key and send it to Bob.

Upon receiving the encrypted message, Bob needs the session key to decrypt it. Since he does not have the session key, he cannot proceed with decryption. However, Alice can forward the encrypted key intended for Bob to him. Bob can then decrypt the encrypted key using his shared key with the KDC, allowing him to recover the session key.

With the session key in hand, Bob can now decrypt the message sent by Alice and proceed with further actions.

This approach ensures secure communication between Alice and Bob without all traffic being routed through the KDC, improving efficiency and scalability.

In the field of cybersecurity, one important aspect is the establishment of keys for secure communication. In this context, symmetric key establishment and the use of the Kerberos protocol are key topics to understand.

Symmetric key establishment involves the generation and distribution of a shared secret key between two parties, such as Alice and Bob, who want to communicate securely. The goal is to establish a key that is known only to them and can be used for encryption and decryption.

The Kerberos protocol is a widely used authentication protocol that provides secure key establishment in a network environment. It involves a trusted third party, called the Key Distribution Center (KDC), which helps in establishing and distributing the secret keys.

To understand the process, let's consider a scenario where Alice wants to send an email to Bob. Alice initiates the process by sending a request to the KDC, stating her intention to communicate with Bob. The KDC then generates a session key, which is a shared secret key between Alice and Bob. This session key is encrypted with Bob's secret key and sent back to Alice.

Once Alice receives the encrypted session key, she forwards it to Bob. Bob, using his secret key, decrypts the session key and both Alice and Bob now have a shared secret key that they can use for secure communication.

This approach has several advantages. Firstly, it reduces the number of communications required for key establishment. In the traditional approach, each user would need to establish a separate key with every other user, resulting in a quadratic complexity. With the Kerberos protocol, the complexity becomes linear, making it more efficient, especially in scenarios with a large number of users.

Additionally, the Kerberos protocol simplifies the process of adding new users to the network. If a new user, say Chris, enters the network, the KDC only needs to add a key for Chris in its database. This is much simpler

compared to the traditional approach, where updating all existing users would be required.

It is important to note that the security of the system relies on keeping the secret keys secure. However, the session key generated by the KDC can be made public without compromising security. This is a key concept in cryptography, where certain keys need to be kept secret while others can be made public.

Symmetric key establishment and the use of the Kerberos protocol provide an efficient and secure way to establish shared secret keys for secure communication in network environments. The Kerberos protocol reduces the complexity of key establishment and simplifies the process of adding new users to the network.

In the field of cybersecurity, one important aspect is the establishment of secure communication channels. One method used for this purpose is symmetric key establishment, which involves the use of a shared secret key between two parties. In this didactic material, we will discuss the concept of symmetric key establishment and a specific protocol called Kerberos.

Symmetric key establishment is a process where a secure channel is established between two parties using a shared secret key. This key is used to encrypt and decrypt the messages exchanged between the parties, ensuring confidentiality and integrity of the communication. The key needs to be securely distributed to the parties involved, and this is where the challenge lies.

Kerberos is a widely used protocol for symmetric key establishment. It provides a centralized authentication server called the Key Distribution Center (KDC) that securely distributes the secret keys to the users. The KDC acts as a trusted third party, facilitating the establishment of secure channels between users.

The advantage of using Kerberos is that it allows for the easy addition of new users. When a new user is added, the only requirement is a secure channel between the user and the KDC during initialization. This simplifies the process of adding new users and reduces the complexity of the system.

However, there are some weaknesses in the Kerberos protocol. One major weakness is that the KDC acts as a single point of failure. If an attacker manages to compromise the KDC, they can gain access to all the secret keys and decrypt past communication. This is known as a lack of perfect forward secrecy.

Perfect forward secrecy refers to the property where the compromise of a long-term secret key does not compromise the confidentiality of past communication. In the case of Kerberos, if the KDC key is compromised, all past communication can be decrypted. This poses a significant security risk.

Symmetric key establishment is an important aspect of cybersecurity, and Kerberos is a widely used protocol for achieving this. However, the Kerberos protocol has weaknesses, such as the lack of perfect forward secrecy, which can compromise the confidentiality of past communication if the KDC key is compromised.

This didactic material focuses on the topic of symmetric key establishment and Kerberos in the field of advanced classical cryptography. Symmetric key establishment is a crucial aspect of cybersecurity, ensuring secure communication between entities. Kerberos is a widely used commercial system based on this approach.

Symmetric key establishment involves the exchange of secret keys between communicating parties. One important concept to consider is perfect forward secrecy (PFS), which guarantees that even if one key is compromised, past and future communications remain secure. Public key-based protocols may or may not provide PFS.

To ensure the security of the key distribution center (KDC) database, where all the keys are stored, it is essential to implement strong security measures. The KDC serves as the foundation for Kerberos, a popular commercial system used in real-world scenarios. It is important to note that Kerberos can be further enhanced with additional features such as timestamps.

In addition to the mentioned concepts, there are potential weaknesses and attacks to be aware of. Two notable attacks are replay attacks and key confirmation attacks. Replay attacks involve the malicious retransmission of previously captured messages, while key confirmation attacks exploit vulnerabilities in the key confirmation process.

While the lecture briefly touched upon these topics, it is important to explore them in more detail in future courses or resources. The textbook provides a simplified version of the Cow Burrows protocol, which is based on the principles discussed. This protocol can be further enhanced by incorporating timestamps and addressing the weaknesses mentioned.

Symmetric key establishment and Kerberos play a vital role in ensuring secure communication in the field of cybersecurity. Understanding the concepts of perfect forward secrecy, the role of the KDC, and the potential attacks is crucial for implementing robust security measures.

EITC/IS/ACC ADVANCED CLASSICAL CRYPTOGRAPHY DIDACTIC MATERIALS
LESSON: MAN-IN-THE-MIDDLE ATTACK
TOPIC: MAN-IN-THE-MIDDLE ATTACK, CERTIFICATES AND PKI

This part of the material is currently undergoing an update and will be republished shortly.