# Lecture 9: Key Distribution

Stephen Huang





## Contents







## 1. Digital Signature

- Motivation: Message authentication does not protect the sender and receiver from each other
  - The receiver can forge a message and claim it is from the sender.
  - The sender can deny sending a message and claim that the receiver forged it.
- Non-repudiation: the sender cannot deny that it has sent a message.





## Digital Signature

- A digital signature is an electronic, encrypted stamp of authentication on digital information such as email messages or electronic documents.
- A signature confirms that the information originated from the signer and has not been altered.
- Digital signature
  ≈ message authentication + non-repudiation
  - provide integrity and authenticity protection as well as non-repudiation
  - similar to traditional signatures: the signee cannot deny signing a document
  - in many countries, digital signatures have legal significance

## Digital Signatures

A digital signature must meet two requirements and ideally would satisfy two more:

- **Unforgeable** (mandatory). No one other than the signer can produce the signature without the signer's private key.
- **Authentic** (mandatory). The receiver can determine that the signature came from the signer.
- **Not alterable** (desirable). No signer, receiver, or interceptor can modify the signature without the tampering being evident.
- **Not reusable** (desirable). The receiver will detect any attempt to reuse a previous signature.



## Digital Signature Schemes

Signature Algorithms:

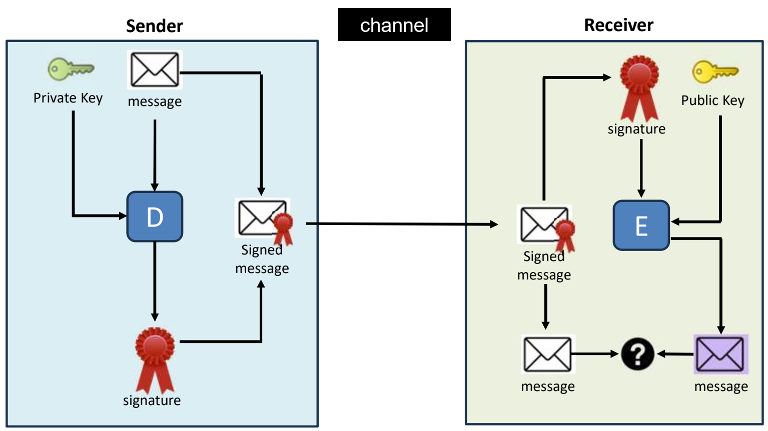
- Key generation G(): randomized algorithm, outputs key pair (PU, PR)
- Signature Sign(PR, M): takes private key PR and message M, outputs signature S
- Verification Verify(PU, M, S): takes public key PU, message M, and signature S, outputs accept/reject

Public-key encryption implementation:

- Key generation G(): randomized algorithm, outputs key pair (PU, PR)
- Decryption D(PR, C): takes private key PR and ciphertext C, outputs plaintext M
- Encryption E(PU, M): takes public key PU and plaintext M, outputs ciphertext C



## Digital Signatures Using Public-Key Encryption

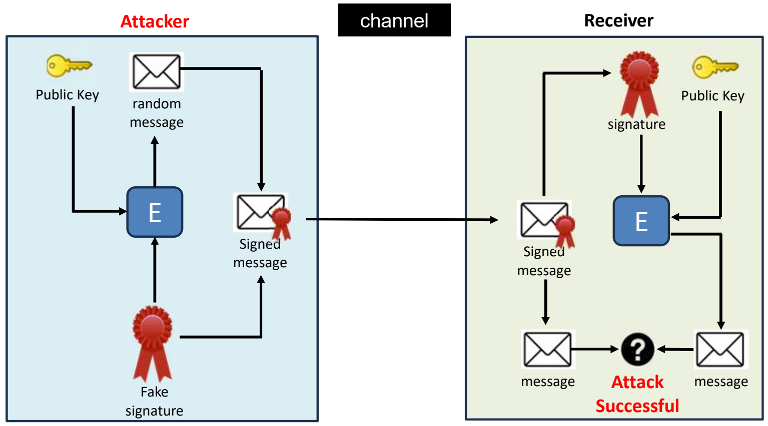





## Digital Signatures Using Public-Key Encryption

- Signing
  - The signer is sharing the public key with the receiver; the signer must use the private key.
  - The signer treats the message as a ciphertext and "decrypts" it using the private key.
  - The signature is the resulting "plaintext".
- Verification
  - The receiver treats the signature as plaintext and encrypts it using the public key (of the sender).
  - The receiver verifies if the resulting "ciphertext" is equal to the message.
- Reminder: encryption and decryption cancel each other, regardless of the order.

## Simple Forgery Attack







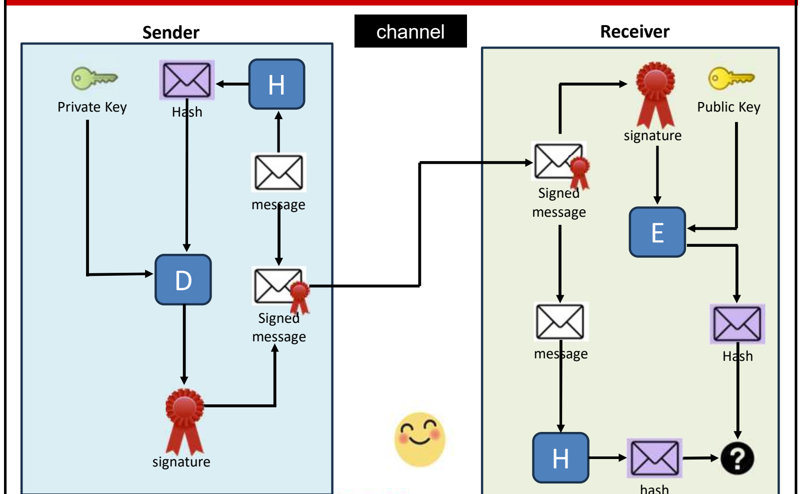

## Simple Forgery Attack

- An attacker can forge a signature for a random message
  - pick an arbitrary value X, and use it as a signature → signature for message E(PU, X) is X
  - The attacker uses the encryption, not decryption
  - The attacker uses the public key of the purported "sender," which is readily available.
- Probably not very useful. Can you come up with a good reason to do so?
- But we must ensure the sender has the private key, not the public one!





## Hash-then-Sign





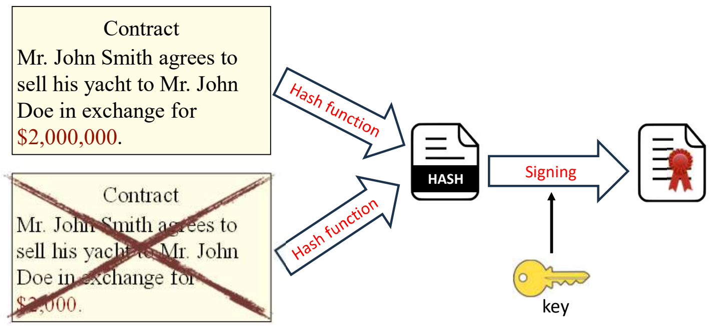



## Hash-then-Sign

- Advantages
  - **Compatibility**: most public-key encryption algorithms take fixed-length inputs.
  - **Efficiency**: The signature will be shorter (than the message) and faster to compute.
  - **Security**: prevents existential forgery. The attacker cannot compute a forged message for an arbitrary signature using only the public key.

# Cryptographic Hash Function

- One-way: prevents existential forgery with public-key encryption
- Collision-resistant






# RSA Signatures

- Very widely used with SHA-256 (and other versions of SHA)
  - example: SSL/TLS
- Standard: PKCS #1 by RSA Laboratories, republished as RFC 3447
  - RSASSA-PKCS1-v1_5: older standard
  - RSASSA-PSS
    - PSS (Probabilistic Signature Scheme): adds randomized padding (called salt) to the message
    - provably secure (given that RSA is secure)



# Digital Signature Algorithm (DSA)

- Digital Signature Standard:
  - FIPS (Federal Information Processing Standard) 186
  - introduced in 1993, updated multiple times
  - latest version includes RSA, DSA, and elliptic-curve signatures
- Digital Signature Algorithm
  - proposed by NIST in 1991
  - designed for signature, cannot be used for encryption
  - efficient variant of the ElGamal signature scheme (much smaller signatures, modular arithmetic operations with lower moduli)
- Elliptic Curve Digital Signature Algorithm (ECDSA)
  - based on elliptic curve cryptography
  - shorter keys and increased efficiency



# Digital Signatures Conclusion

- Digital signature
  ≈ message authentication + non-repudiation
  - provides integrity and authenticity protection as well as non-repudiation
- Based on asymmetric-key cryptography: much slower than message authentication
- Algorithms
  - RSA
  - DSA
  - ECDSA

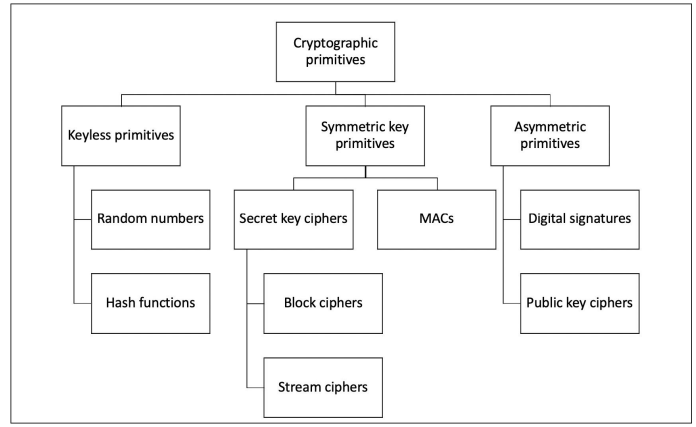
# 2. Summary of Cryptographic Primitives

- Types of Cryptographic Primitives

| | Symmetric-key | Asymmetric-key | |
|---|---|---|---|
| Confidentiality | Block ciphers<br>Stream ciphers | Asymmetric-key encryption | |
| Integrity | Message authentication | Digital signatures | Hash functions |



# Cryptographic Primitives




[Cryptographic primitives | Mastering Blockchain - Third Edition (packtpub.com)]



# Lessons Learned

1. Obscurity is not security
   - example: A5/1 cipher (GSM) was designed in secret but was eventually broken
2. The security of practical cryptographic primitives is not proven
   - symmetric primitives are built on design principles, and asymmetric primitives are built on mathematical problems that are believed to be hard
3. Nonetheless, widely used cryptographic primitives are rarely broken
   - cryptographic primitives are much more trustworthy than software, users, etc.



# Lessons Learned

4. However, even secure primitives may be used, implemented, or combined in insecure ways
   - example: earlier versions of the SSL/TLS protocol had some weaknesses and very vulnerable implementations
5. Security is a process, not a product
   - key lengths and algorithms must be upgraded from time to time

## 3. Key Distribution

*How can parties exchange or agree on a secret key?*

- Review: Cryptographic primitives are well-established, low-level cryptographic algorithms that are frequently used to build cryptographic protocols for computer security systems. These routines include but are not limited to,
  - one-way hash functions and
  - encryption functions.





## Review: Encryption Systems

- Symmetric (also called "Secret Key") algorithms use one key, which works for both encryption and decryption. Usually, the decryption algorithm is closely related to the encryption one, essentially running the encryption in reverse.
- The symmetric systems provide a two-way channel to their users.
- Symmetry is a major advantage of this type of encryption.





## Review: Encryption Systems

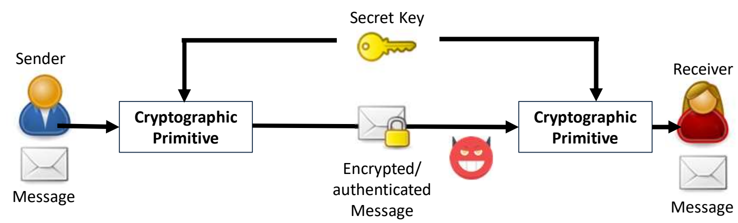
- Asymmetric or public-key systems typically have matched pairs of keys.
- The keys (private key and public key) are produced together, or one is derived mathematically from the other. Thus, a process computes both keys as a set.
- Asymmetric systems excel at key management.





## Key Distribution

- Symmetric-key cryptography
  - much more efficient than asymmetric-key cryptography




- However, to use symmetric-key cryptography
  - communication parties must share the same key
  - unauthorized parties must not know the key

## Key Freshness

- Secret keys may become insecure when used for a long time
  - More ciphertexts encrypted using the same key
    → It is easier for the attacker to recover the key.
- Examples:
  - Most stream ciphers produce pseudorandom sequences that repeat eventually.
  - Block ciphers with 64-bit blocks in CBC mode are likely to output the same block after ~34 GB of data, revealing XOR of corresponding plaintext blocks.



## Key Freshness

- Key freshness requirement: renew (i.e., change) secret key frequently
  - Example: SSH protocol usually requires a new key after 1 hour or 232 packets (rekeying).
- Problem:
  - Secret keys have to be renewed frequently.
  - Setting up a secret key is a complex operation.



## Secret-Key Hierarchy

**Session key**

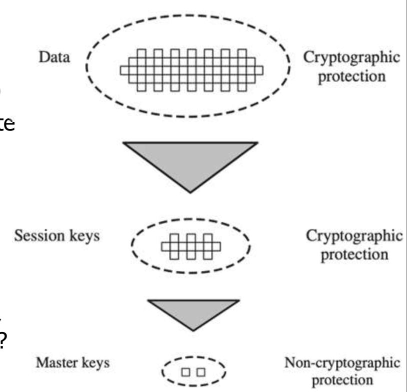
- renewed frequently (e.g., one key for each logical connection)
- used to encrypt and authenticate data

**Master key**

- renewed infrequently
- used to distribute session keys

**Questions:**

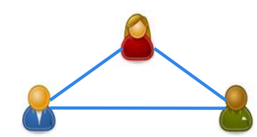
- What are the master keys (e.g., symmetric or asymmetric keys)?
- Who has the master keys?
- How to obtain a session key from a master key?



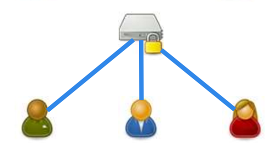



## Secret-Key Distribution Approaches

- Decentralized
  - each pair of communication parties shares a secret master key

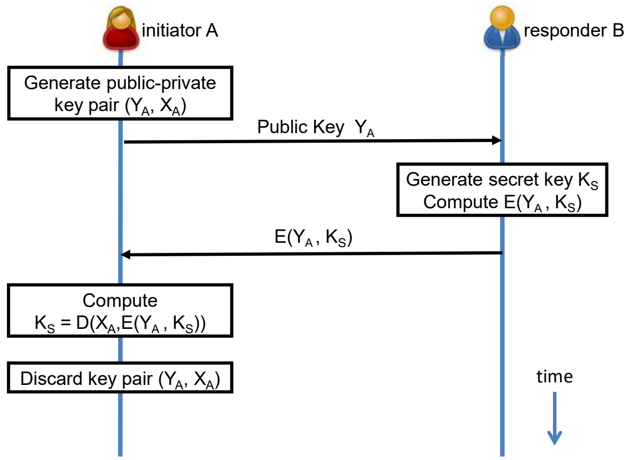


- Key Distribution Center (KDC)
  - KDC shares a secret master key with each of the communication parties



- Public-key cryptography
  - one communication party needs to have the public key of the other

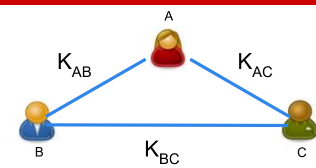
## Secret Key Distribution using Public-Key








## Decentralized Key Distribution




N communication parties
→ N · (N-1) / 2 pairs

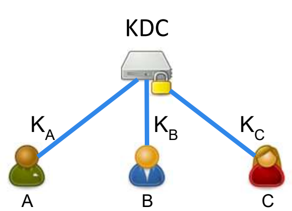
- Each pair of communication parties has to share a secret master key
- The master key needs to be set up for each pair manually
  - Any pair can then exchange or agree on session keys easily
- May work for securing small, local networks
  - Example: physically delivering the key for each pair
- However, it does not scale well
  - Especially difficult in a wide-area distributed system





## Key Distribution Center (KDC)

- Key Distribution Center (KDC)
  - acts as a **trusted third party**:
    all communication parties trust the KDC
  - each party X shares a secret master key $K_X$ with the KDC




N communication parties
→ only N master keys





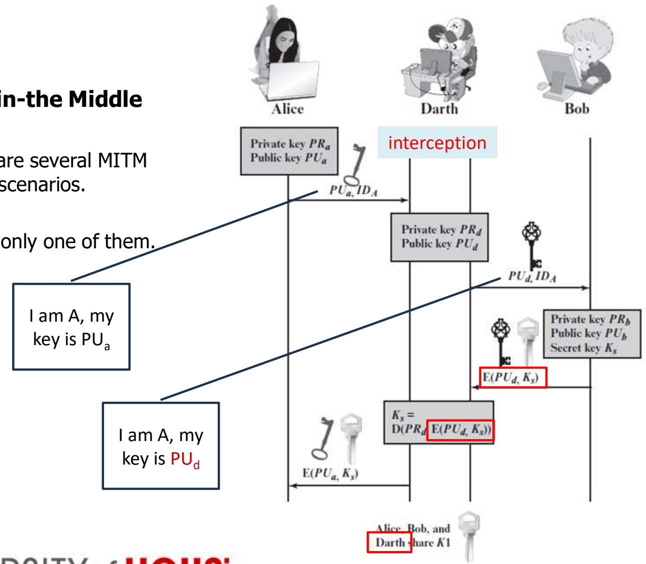
## 4. Key-Distribution Protocols

- How to obtain a session key from master keys?
- Man-in-the-Middle Attacks
- Notation reminder:
  - E(K, X) Symmetric encryption of plaintext X using secret key K.
  - x||y or x|y, x concatenated with y, or simply (x, y)

**Man-in-the Middle**

There are several MITM attack scenarios.

This is only one of them.

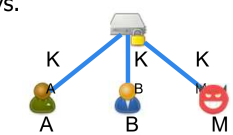







# Assumptions and Adversary Model

- Cryptographic primitives are secure.
- Each master key is known only by the KDC and the corresponding communication party.
- Every non-malicious participant follows the protocol.
- Adversary
  - may be a legitimate protocol participant (i.e., insider),
  - has complete control over the communication channels,
  - may have old, compromised session keys.







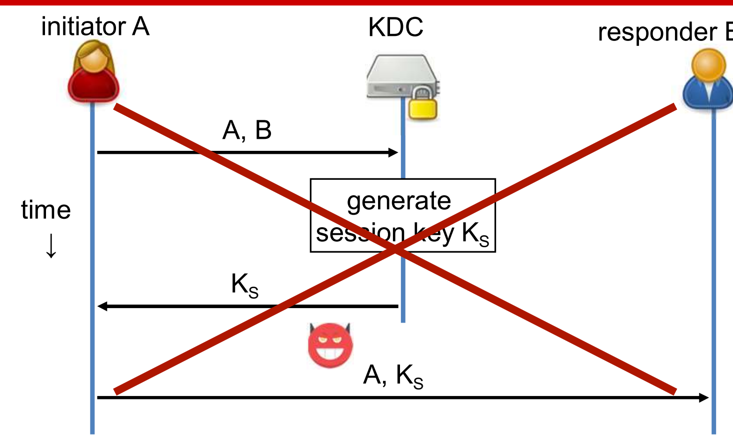

# Key Distribution Objectives

- Effectiveness: both parties should learn the session key.
- Implicit key authentication: no other parties (except for the trusted third party) should know the session key.
- Key freshness: both parties should be able to verify that the key was freshly generated.
- (Key confirmation: both parties should be able to verify that the other party also has the key).



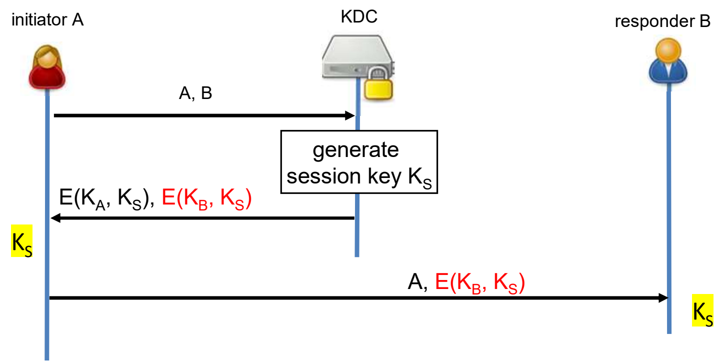


# Basic Key Transport




- Attacker can eavesdrop on the session key $K_S$.

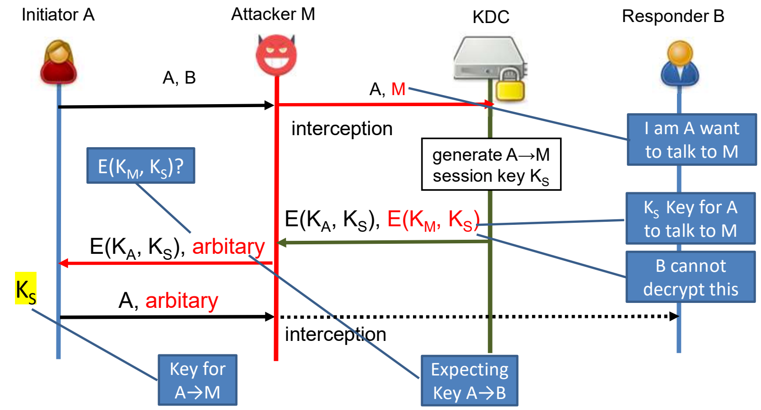
## Basic Key Transport with Encryption




- The attacker cannot eavesdrop on the session key $K_S$.
- However, a man-in-the-middle attacker can impersonate B.



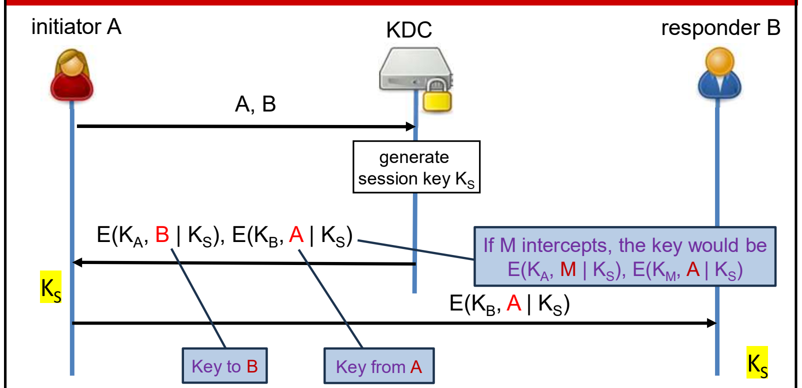
## Man-in-the-Middle Attack




- A thinks it shares a secret key with B, but it actually shares a key with attacker M.



## With Encryption and Identifiers

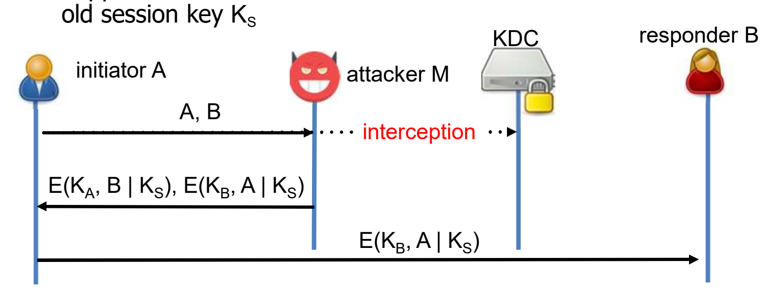



- The attacker cannot impersonate protocol participants.
- However, a man-in-the-middle attacker may replay old session keys.



## Replaying Old Session Key

- Suppose that the attacker has observed the distribution of an old session key $K_S$



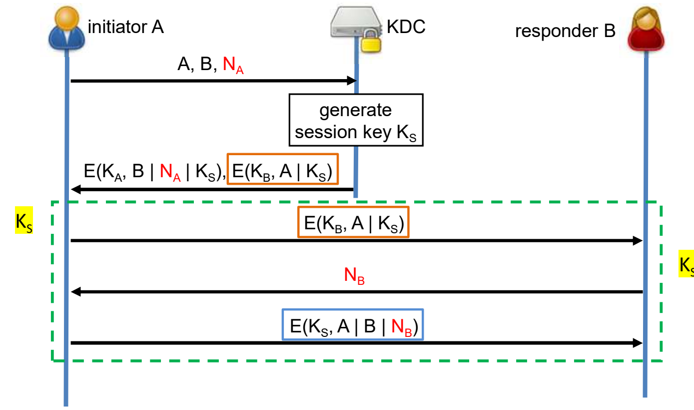

- Key freshness is not guaranteed by the protocol
  - neither A nor B can tell if the session key $K_S$ was generated recently
- The attacker can force A and B to use the old compromised key indefinitely.

# Nonce

- Nonce, short for "Number used once".
- A nonce is an arbitrary number that can be used once in a cryptographic communication. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks.
- Challenge/response: Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires B's subsequent message (response) to contain the correct nonce value.
- Nonce is not a timestamp (but it can be). The only assumption is that it has not been used in any earlier interchange, with high probability.



# with Identifiers and Nonces

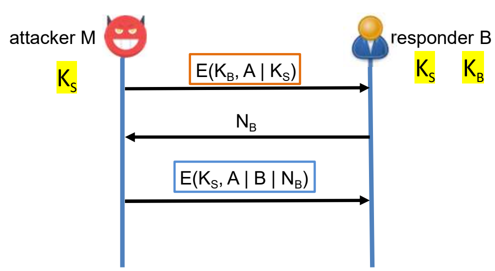



- Replaying an old, compromised session key is possible.



# Replaying an Old Session Key

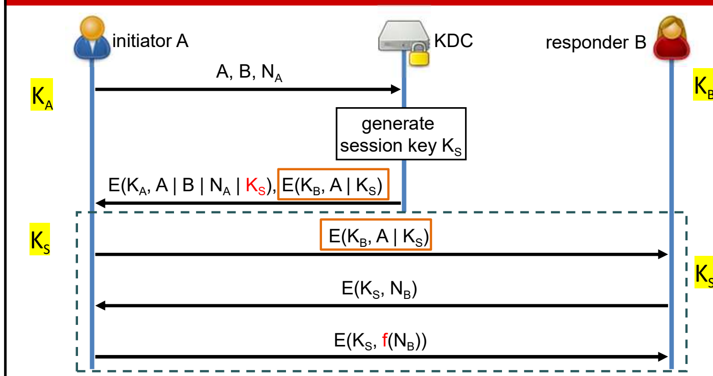
- Suppose that the attacker has compromised an old session key $K_S$




- Key freshness is still not guaranteed by the protocol

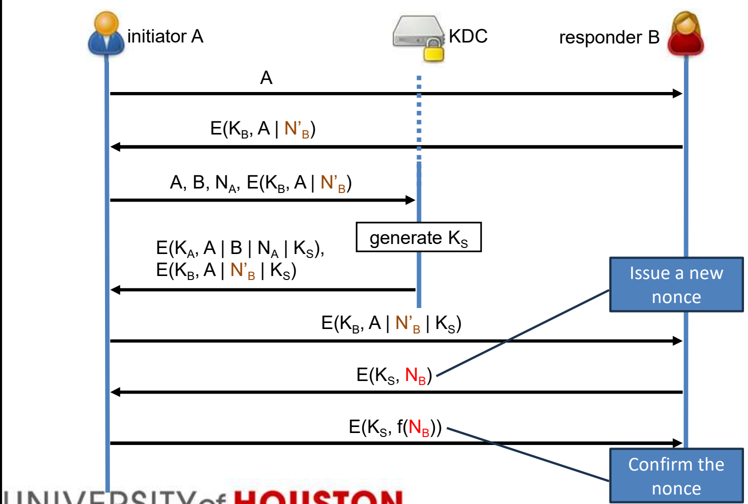


# Needham-Schroeder Symmetric-Key Protocol



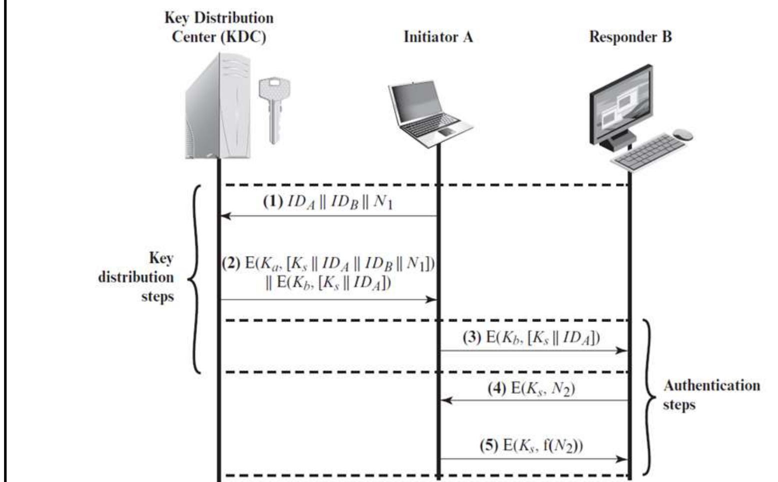

- f: some mathematical function (e.g., subtracting one)
- Replaying an old, compromised session key is still possible

## Extended Needham-Schroeder Protocol





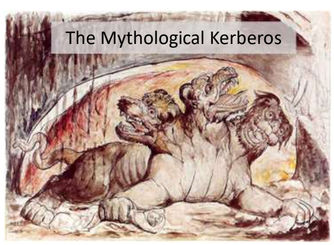

## Another View






## Kerberos Network Authentication Protocol

- Allows nodes to communicate over a non-secure network and to prove their identities to each other.
- Similar to the extended Needham-Schroeder protocol, but uses timestamps instead of nonces.
  - in addition to timestamps, messages may also contain lifetimes → can limit usage time
- Windows 2000 and later versions use Kerberos as the default authentication for clients that want to join a Windows domain.






## Next Topic

- Key (Asymmetric) Distribution
- Public-key (Symmetric) Distribution
- WiFi Security