CISC 468: CRYPTOGRAPHY

LESSON 20: KEY ESTABLISHMENT USING SYMMETRIC-KEY TECHNIQUES

Furkan Alaca

READINGS

- Ch. 13.1 (Introduction: Key Establishment), Paar & Pelzl
- Ch. 13.2 (Key Establishment Using Symmetric-Key Techniques), Paar & Pelzl

SECURITY SERVICES, REVISITED

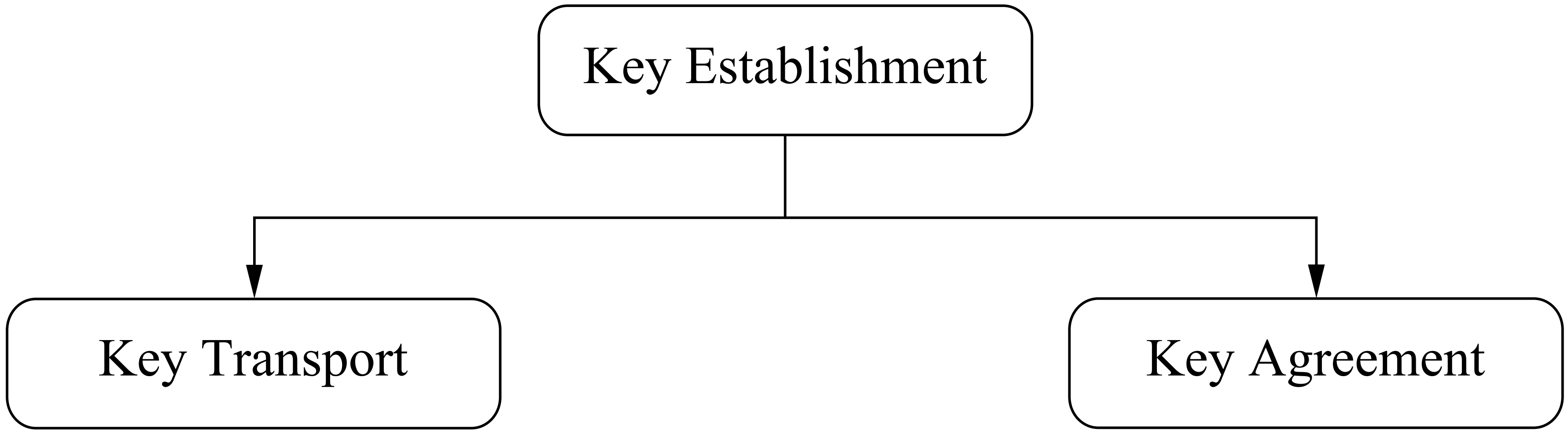
Using the cryptographic mechanisms we learned so far, we can achieve the folowing security services:

- Confidentiality: With symmetric algorithms
- Integrity: With MACs or digital signatures
- Message authentication: With MACs or digital signatures
- Non-repudiation: With digital signatures

KEY ESTABLISHMENT

- The symmetric algorithms we introduced assume that the secret keys are properly agreed upon between the two communicating parties
- We learned that the Diffie-Hellman Key Exchange, a public-key algorithm, can be used for this purpose
- We will see that it is also possible to perform key agreement using only symmetric-key algorithms

KEY ESTABLISHMENT

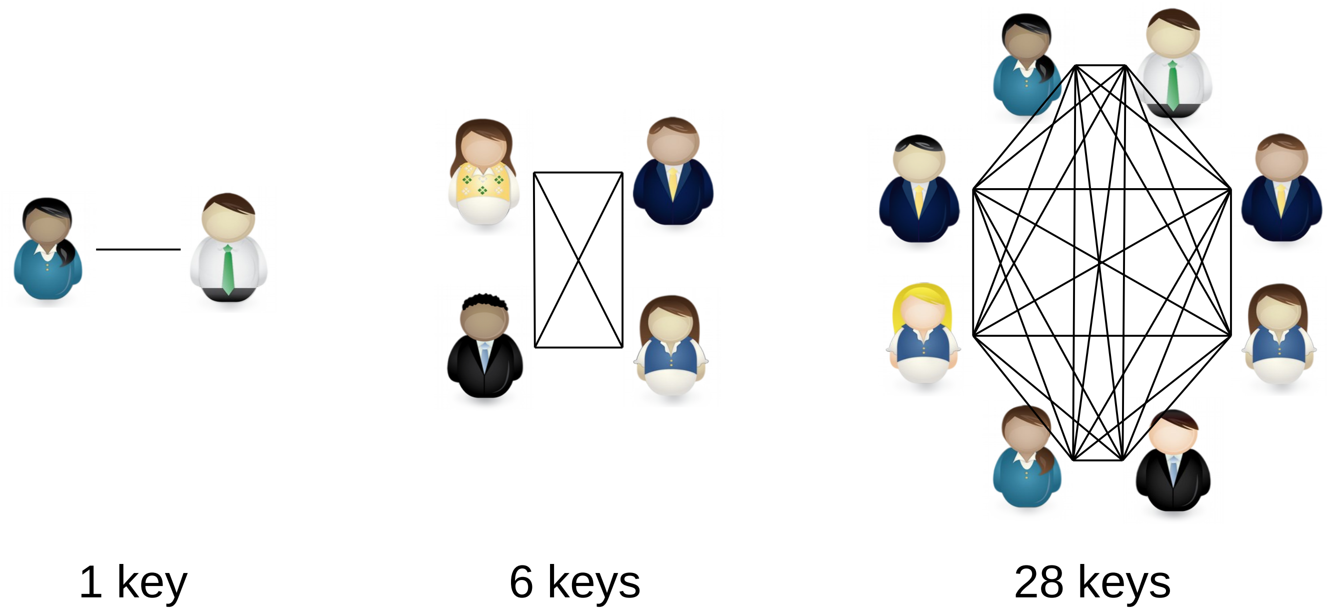


One party generates and distributes a secret key

Parties jointly generate a secret key

- DHKE is a *key agreement* protocol
- We will see how to use symmetric cryptography to perform *key transport*

KEY PREDISTRIBUTION



- Predistribution requires $\frac{n(n-1)}{2}$ keys to be distributed
- Adding new users requires sending new keys to all other users
- Impractical, unless there are a small number of users that do not change frequently

KEY FRESHNESS

- In many systems, it is desirable to limit the validity period of cryptographic keys
- Such keys are called *session keys* or *ephemeral keys*
- This limits the data exposed if the key is compromised
- This also limits the ciphertext generated using the same key, making cryptanalysis more difficult

LONG-TERM VS. SHORT-TERM KEYS

- Symmetric ciphers can be used to establish session keys
- This is achievable in practice if a long-term secret key can be pre-installed out-of-band, e.g.,
  - A system administrator may manually install a key on a device before connecting it to the network
  - The device manufacturer may install a key at the factory
- The long-term key can then be used to securely establish a new session key for each connection

KEY ESTABLISHMENT WITH A KEY DISTRIBUTION CENTER

- A *Key Distribution Center* (KDC) can be used to perform key transport using symmetric cryptography
- Each user U must establish a *Key Encryption Key* (KEK) k_U with the KDC prior to joining the network
- If Alice requests a secure session with Bob, the KDC can:
  1. Generate a session key k_{ses}
  2. Send $e_{k_A}(k_{ses})$ to Alice
  3. Send $e_{k_B}(k_{ses})$ to Bob

Only n keys are required for n users.

KEY ESTABLISHMENT WITH A KEY DISTRIBUTION CENTER

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A, ID_B)}$			
		generate random k_{ses} $y_A = e_{k_A}(k_{ses})$ $y_B = e_{k_B}(k_{ses})$		
	$\xleftarrow{y_A}$		$\xrightarrow{y_B}$	
$k_{ses} = e^{-1}_{k_A}(y_A)$				$k_{ses} = e^{-1}_{k_B}(y_B)$
$y = e_{k_{ses}}(x)$		$\xrightarrow{y}$		$x = e^{-1}_{k_{ses}}(y)$

- The KEKs k_A, k_B are long-term keys
- k_{ses} is a short-term key that ideally changes for every communication session

KEY ESTABLISHMENT WITH A KEY DISTRIBUTION CENTER

The following variant of the protocol saves one communication session (i.e., the KDC does not need to communicate with Bob):

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A,ID_B)}$			
		generate random k_{ses} $y_A = e_{k_A}(k_{ses})$ $y_B = e_{k_B}(k_{ses})$		
	$\xleftarrow{y_A,y_B}$			
$k_{ses} = e^{-1}_{k_A}(y_A)$ $y = e_{k_{ses}}(x)$				
		$\xrightarrow{y,y_B}$		
				$k_{ses} = e^{-1}_{k_B}(y_B)$ $x = e^{-1}_{k_{ses}}(y)$

REPLAY ATTACKS

- The previous protocol is susceptible to *replay attacks*
- An attacker can eavesdrop $e_{k_B}(k_{ses})$ and subsequent encrypted messages sent from Alice to Bob, and then *replay* the messages to Bob at a later date
  - e.g., the attacker can duplicate a transaction that was only meant to be performed once
- Alice and Bob also do not know if the session key is fresh
  - An old key is more likely to have been leaked or compromised

IMPERSONATION ATTACKS

The previous protocol allows a legitimate (but malicious) user Oscar to impersonate Bob via an *active attack*:

Alice		**Oscar**		**KDC**	**Bob**
KEK: k_A		KEK: k_O		KEK: k_A, k_B, k_O	KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A, ID_B)}$				
		substitute			
			$\xrightarrow{\text{RQST}(ID_A, ID_O)}$		
				random k_{ses} $y_A = e_{k_A}(k_{ses})$ $y_O = e_{k_O}(k_{ses})$	
			$\xleftarrow{y_A, y_O}$		
$k_{ses} = e^{-1}_{k_A}(y_A)$ $y = e_{k_{ses}}(x)$					
	$\xrightarrow{y, y_O}$				
		intercept $k_{ses} = e^{-1}_{k_O}(y_O)$ $x = e^{-1}_{k_{ses}}(y)$			

Alice believes that she has received y_A, y_B, but she has actually received y_A, y_O

KERBEROS

- Kerberos is a widely-used protocol that uses symmetric-key cryptography to provide:
  - Mutual entity authentication between two parties A and B that wish to communicate
  - Key establishment between A and B
  - With the help of a trusted KDC
- Usually, A is a user and B is a host that is providing a service (e.g., a printer or network file server)

KERBEROS: SECURITY GOALS

- Messages are protected against replay attacks by embedding a timestamp T_S into the messages
- The KDC specifies a lifetime T for each session key, to guarantee key freshness
- The above require all hosts to have synchronized clocks
- A challenge-response protocol sequence is used to provide entity authentication

KEY ESTABLISHMENT WITH KERBEROS (SIMPLIFIED)

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
generate nonce r_A				
	$\xrightarrow{\text{RQST}(ID_A, ID_B, r_A)}$			
		generate random k_{ses} generate lifetime T $y_A = e_{k_A}(k_{ses}, r_A, T, ID_B)$ $y_B = e_{k_B}(k_{ses}, ID_A, T)$		
	$\xleftarrow{y_A, y_B}$			
$k_{ses}, r_A, T, ID_B = e^{-1}_{k_A}(y_A)$ verify $r_A = r_A$ verify ID_B verify lifetime T generate time stamp T_S $y_{AB} = e_{k_{ses}}(ID_A, T_S)$				
	$\xrightarrow{y_{AB}, y_B}$			
				$k_{ses}, ID_A, T = e^{-1}_{k_B}(y_B)$ $ID_A, T_S = e^{-1}_{k_{ses}}(y_{AB})$ verify $ID_A = ID_A$ verify lifetime T verify time stamp T_S
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$			$x = e^{-1}_{k_{ses}}(y)$

KERBEROS: CHALLENGE-RESPONSE SEQUENCE

- In the beginning, when Alice requests the session key from the KDC, it includes a random nonce r_A in the request
- This can be considered a *challenge*, to which the KDC must *respond* by including the nonce r_A in the same message with the new session key k_{ses} and Bob's identity ID_B, and encrypting them together with the shared long-term key k_A
- This assuress Alice that the response legitimately corresponds to her request to initiate a session with Bob

KERBEROS: AUTHENTICATION AND REPLAY PROTECTION

- Bob decrypts y_B to obtain k_{ses} and Alice's identity ID_A
  - The lifetime parameter T guarantees key freshness
- Then, Bob authenticates Alice by decrypting y_{AB} with k_{ses}, and:
  - Checking that the identity in the message matches ID_A
  - Checking Alice's timestamp T_S to ensure the message is not replayed
  - Thus, Bob ensures that a secure session has been established with Alice
- Optionally, for *mutual authentication*, Bob can encrypt T_S with k_{ses} and send it back to Alice

KERBEROS: PASSWORD-DERIVED KEYS

- Users' long-term keys can be password-derived, e.g., using PBKDF2
- However, this enables password-guessing attacks:
  - When Alice requests a session key from the KDC to use with Bob, the KDC responds with y_A and y_B
  - Alice can perform an offline password-guessing attack on y_B
- So, is strongly encouraged that **randomly-generated keys** should be used by any entities that can be requested from the KDC as a destination host for establishing a secure session
  - In practice, users request sessions with servers such as printers, file servers, etc. and **not** with other users

GENERAL PROBLEMS WITH KDC-BASED KEY DISTRIBUTION

- Communication requirements: The KDC needs to be contacted to initiate a secure session between any two parties
  - If the KDC is down, all Kerberos-protected services are down
  - This can be acceptable for corporate networks, but not at an Internet scale
- No *perfect forward secrecy*: Compromising long-term keys allows an attacker to obtain past session keys
  - If an eavesdropper records $y_A = e_{k_A}(k_{ses})$ and later learns k_A, they can decrypt y_A to obtain k_{ses}, which exposes all past and future data encrypted with k_{ses}