727 Crypto Management

Lecture 11

GIOVANNI RUSSELLO

G.RUSSELLO@AUCKLAND.AC.NZ

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_B||N_1

KDC -> A: E(K_A, (K_S||ID_A||ID_B||N_1)||E(K_B,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_B||N_1 <- Assuming that an attacker changes here the id

KDC -> A: E(K_A, (K_S||ID_A||ID_B||N_1)||E(K_B,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_E||N_1 <- Assuming that an attacker changes here the id

KDC -> A: E(K_A, (K_S||ID_A||ID_B||N_1)||E(K_B,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_E||N_1 <- Assuming that an attacker changes here the id

KDC -> A: E(K_A, (K_S||ID_A||ID_E||N_1)||E(K_E,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_E||N_1

KDC -> A: E(K_A, (K_S||ID_A||ID_E||N_1)||E(K_E,(ID_A||K_S))) <- There is no way for the attacker to prevent this

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_E||N_1

KDC -> A: E(K_A, (K_S||ID_A||ID_E||N_1)||E(K_E,(ID_A||K_S))) <- When A receives the message, it will noticed that something is wrong and drop the message

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2)

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_B||N_1

KDC -> A: E(K_A, (K_S||ID_A||ID_B||N_1)||E(K_B,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, N_2) <- Here is the main issue: an attacker could reply this message to B

Establishing a Session Key via a KDC

KDC, A, B where A is the initiator; K_A and K_B master keys

A -> KDC: ID_A||ID_B||N_1

KDC -> A: E(K_A, (K_S||ID_A||ID_B||N_1)||E(K_B,(ID_A||K_S)))

A -> B: E(K_B,(ID_A || K_S))

B -> A: E(K_S, N_2)

A -> B: E(K_S, F(N_2)) <- Add a transformation function (+1) that can only be performed at A

Hierarchical Key Control

Relying for the key distribution on a single KDC might be a bad idea

- Single point of failure and a bottleneck for the network

Better is to have multiple KDCs to manage sub-networks

- This could be a LAN or a single building
- A local KDC will serve the needs of local entities

For communication across different LANs/Domains then?

- The KDCs of each domain establish a session key through a global KDC

This forms a hierarchical structure of KDCs that can be extended over to three or more layers

- This minimize the effort to distributed master keys only at a local level
- A faulty/compromised KDC limits the damage only to the entities with the local domain

Session Key Lifetime

The fresher the session keys the more secure the system is

However it is also a burden to establish a session key

A security manager has to take this into account when deciding the lifetime of a session key

In a connection-oriented protocol, it is reasonable to establish a new key every time a new connection is started

- For long lived connections, it is recommended to periodically change the session key even if the connection is still open

In a connectionless protocol, it is more difficult to establish when a session key is to be shared

- Use a session key for a given period of time or for a given number of transactions

Two More Considerations

Establishing a session key can be done transparently to the user

- Extra services are needed that communicate directly with the KDC

Not all the data is the same, so not all keys should be used for encrypting data

- There are cases in which the use of a key must be controlled or limited for specific operations

Transparent Key Distribution

It is possible to implement end-to-end encryption using a variant of the KDC scheme

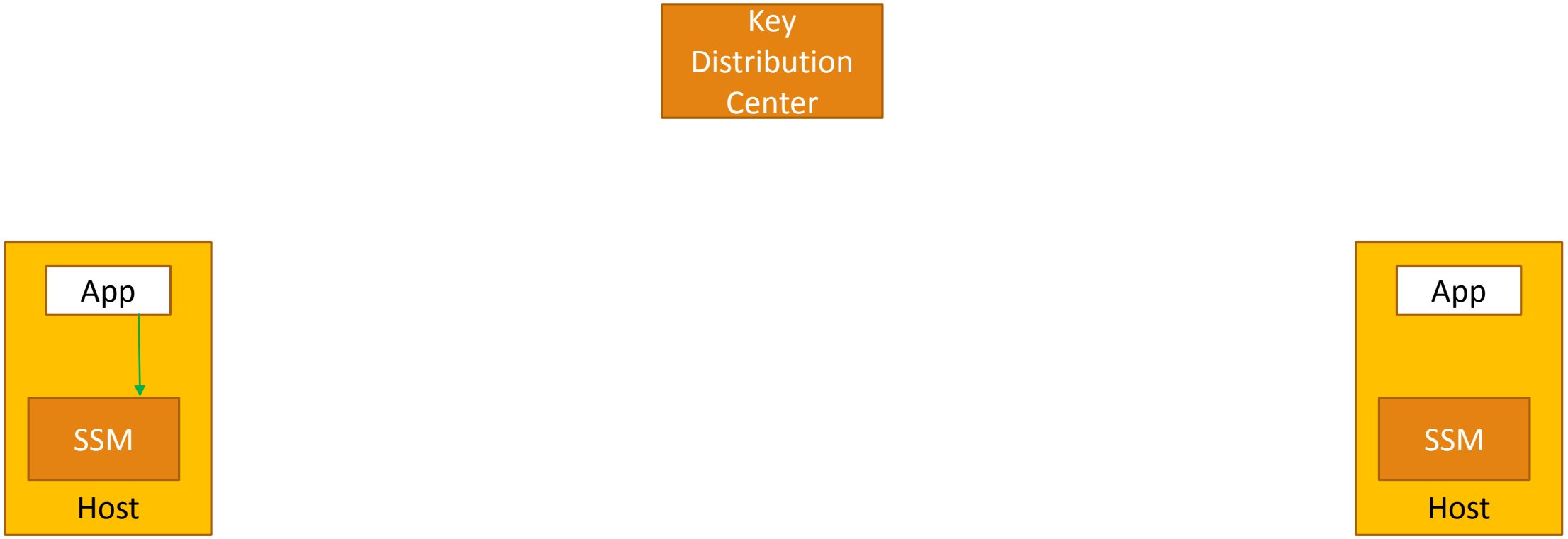
The main idea is to provide security at the transport layer transparently to the user/application

Assuming that the applications use a connection-oriented protocol such as TCP

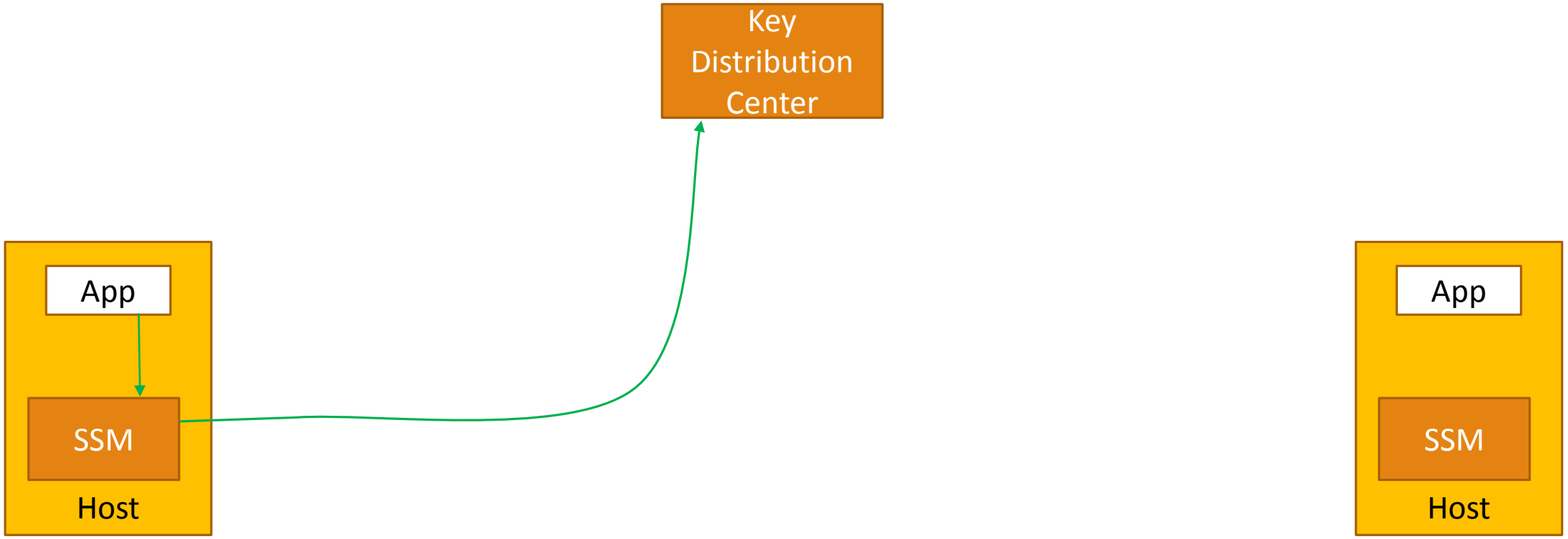
The scheme requires a dedicated module: the *Session Security Module (SSM)*

The SSM establishes a secure connection implemented at the protocol layer and manages the session keys on behalf of the host

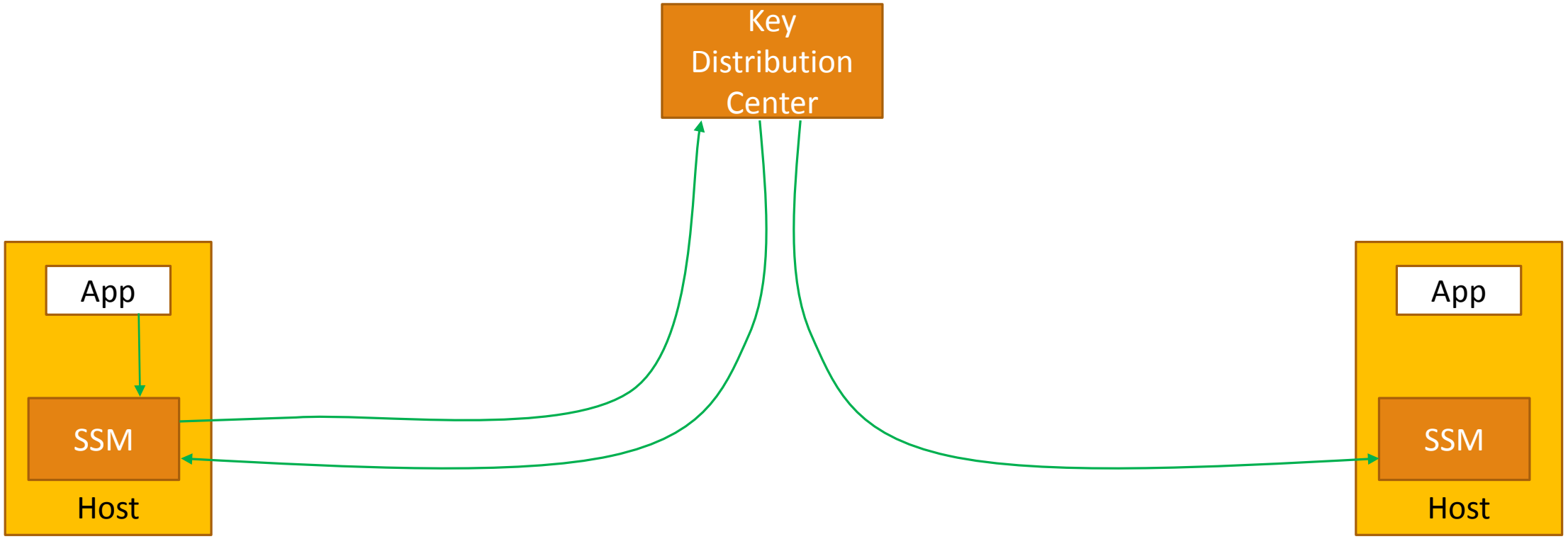
App requests the starting of a session with a remote host



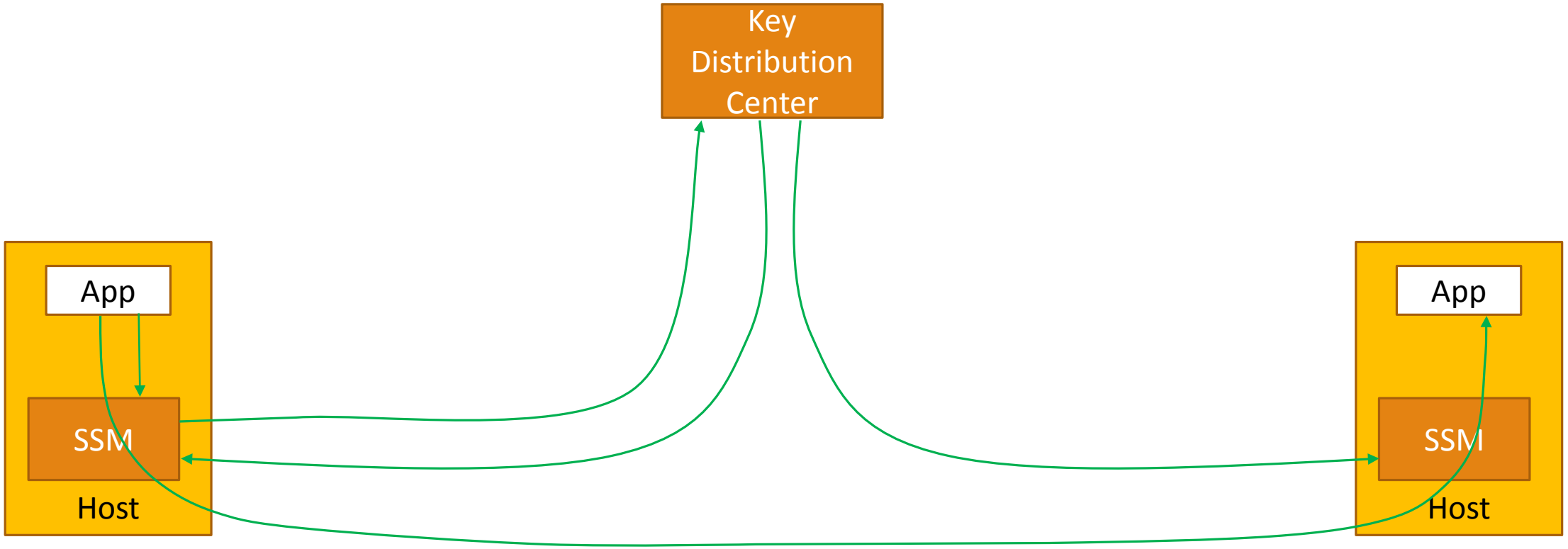
The SSM sends request to KDC



KDC creates and distributes new Session Keys to both parties



App establishes connection with remote host



Controlling Key Usage

Sometimes it is desirable to be able to use different keys for different data

Define different types of session keys for different data that needs to be sent

- Data encrypting keys for general communication across the net
- Pin-encrypting keys for transferring PINs for e-fund transfer or point-of-sale applications
- File-encrypting keys for encrypting files

Control Tags

A simple way is to use tags associated with keys to control the way keys are used

Tags can be embedded in the keys:

E.g., DES could use extra 8 bits (normally used for parity check):

- One bit for checking if the key is a master or a session key
- One bit indicates whether the key can be used for encryption only
- One bit indicates whether the key can be used for decryption only

Control Tags – Pros and Cons

Pro:

- The tag is embedded in the key

- The tag is encrypted along with the key

Cons:

- Limited to only 8 bits

- The key needs to be decrypted before we can check the tag

Control Vector

The control vector (CV) contains a set of fields to specify use and restriction of a session key

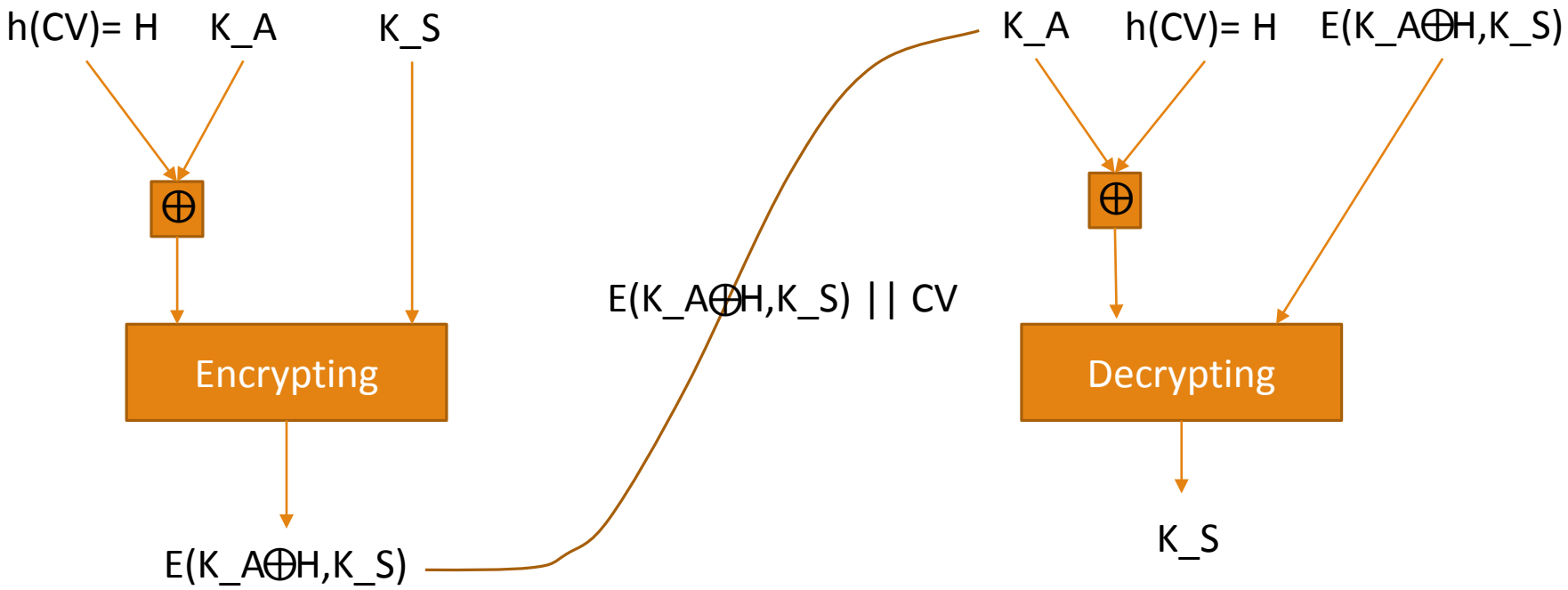
A CV is cryptographically linked to a session key

To bind a CV to a session key:

- Hash the CV to a hash value of the same length of the key
- The hash value is XORed with the master key
- This combined value is used to encrypt the session key

The CV (in plaintext) is delivered together with the encrypted session key by the KDC

Binding the CV to the Session Key



Control Vector – Pros

The CV is in cleartext so it can be examined without decrypting the session key

The CV has a flexible structure that can accommodate any need

Decentralised Key Control

It is also possible to avoid the use of KDCs

- Main reason is to avoid trusting an extra entity

A, B, K_m is a master key already established between the two

A -> B: ID_A || N_1

B -> A: E(K_m, (K_s||ID_A||ID_B||f(N_1)||N_2))

A -> B: E(K_s, f(N_2))

Key Distribution with Asymmetric Encryption

Public-key cryptosystems are not used for direct encryption of large amount of data

They are used mainly for relatively small size blocks, like secret keys

Merkle proposed a very simple protocol for key exchange based on public keys

A generates (PU_A, PR_A), B generates (PU_B, PR_B), K_s

A -> B: PU_A || ID_A

B: generates K_s

B -> A: E(PU_A, K_s)

A & B discard (PU_A, PR_A), (PU_B, PR_B)

After connection is closed discard K_s

Simple but…

Q: There is a very serious flaw in this protocol. Give an example of an attack.

Key Distribution with Confidentiality and Authentication

Assume that A and B already have securely exchanged PU_A, and PU_B

A -> B: E(PU_B, (N_1||ID_A))

B -> A: E(PU_A, (N_1||N_2))

A -> B: E(PU_B, N_2)

A -> B: E(PU_B, E(PR_A, K_s))

B uses PU_A to retrieve K_s