CSE 3231
Computer Networks

Cryptography
part 2

William Allen, PhD

Spring 2022

Key Distribution

- Key Distribution refers to methods for sharing an asymmetric key between two parties who wish to exchange data
- For *symmetric encryption* to work, the two parties to an exchange must share the same key, and that key must be protected from access by others
- Frequent key changes are desirable to limit the amount of data compromised if an attacker learns the key

Using Public Keys to Share Private Data

- If the sender encrypts a message using their *private key* to verify that they are the sender, it *does not hide the contents*
  - anyone can use their public key to read it
- To share keys, we need a method that can *hide* the key and *verify* who it came from
  - Public/Private keys can solve this problem and be used to *securely share* a symmetric encryption key between two people
    - remember, symmetric keys are relatively small

Key Exchange: Safely Delivering a Symmetric Key to another Person

- How can we safely deliver a symmetric encryption key to the intended recipient *R* and prove that it could only have come from sender *S*?
  - Assume that both S and R have public and private keys, but want to share a symmetric key *Kx* to exchange large amounts of data
  - They can use the following key exchange algorithm to share the symmetric key

Key Exchange: Safely Delivering a Symmetric Key to another Person

- Public key encryption provides a solution

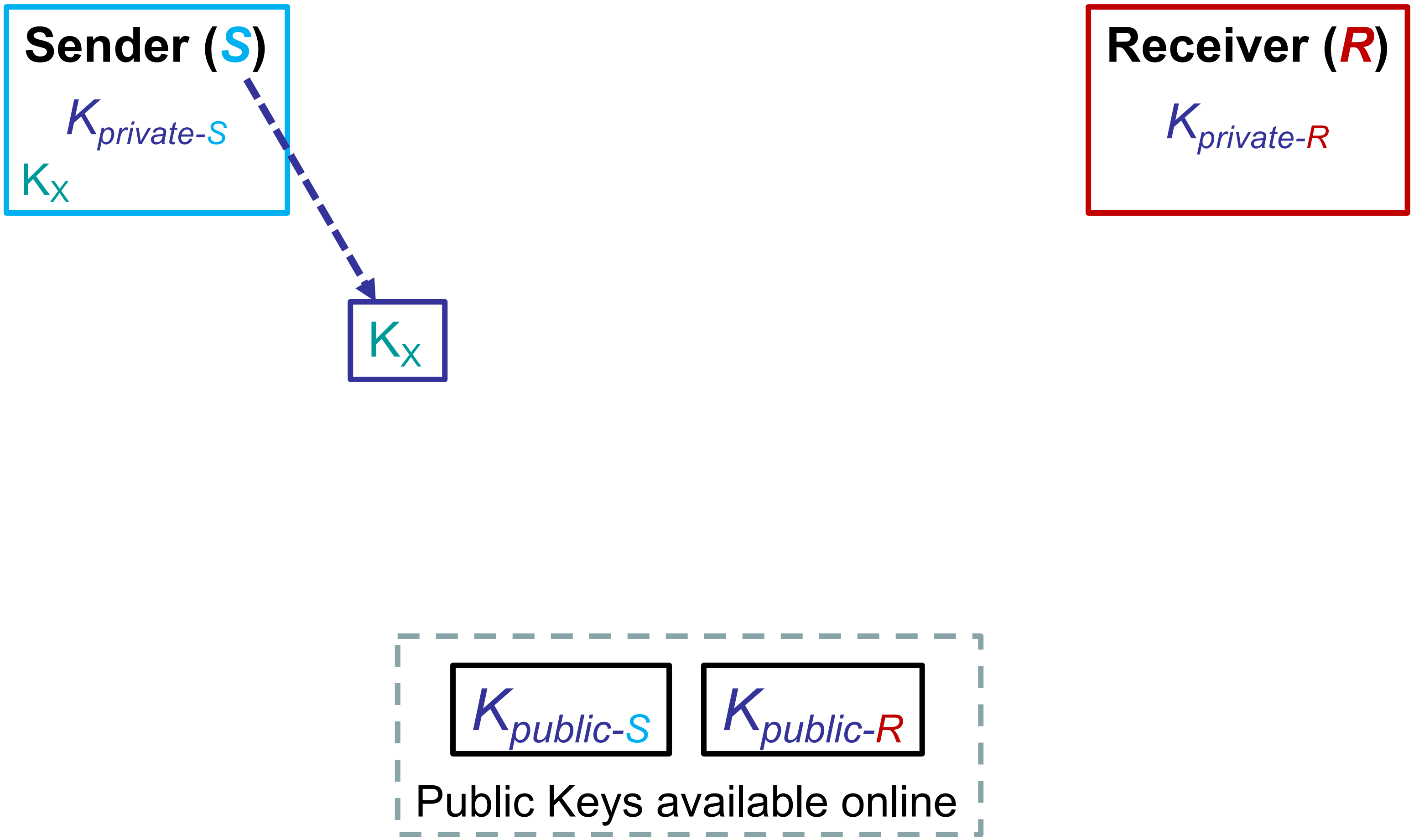
  sender=S, receiver=R, both have public keys

1. S wants to send the symmetric key K_X, to R
2. S encrypts K_X using $E(K_{private\text{-}S}, K_X)$
3. Then, S encrypts that using R's public key

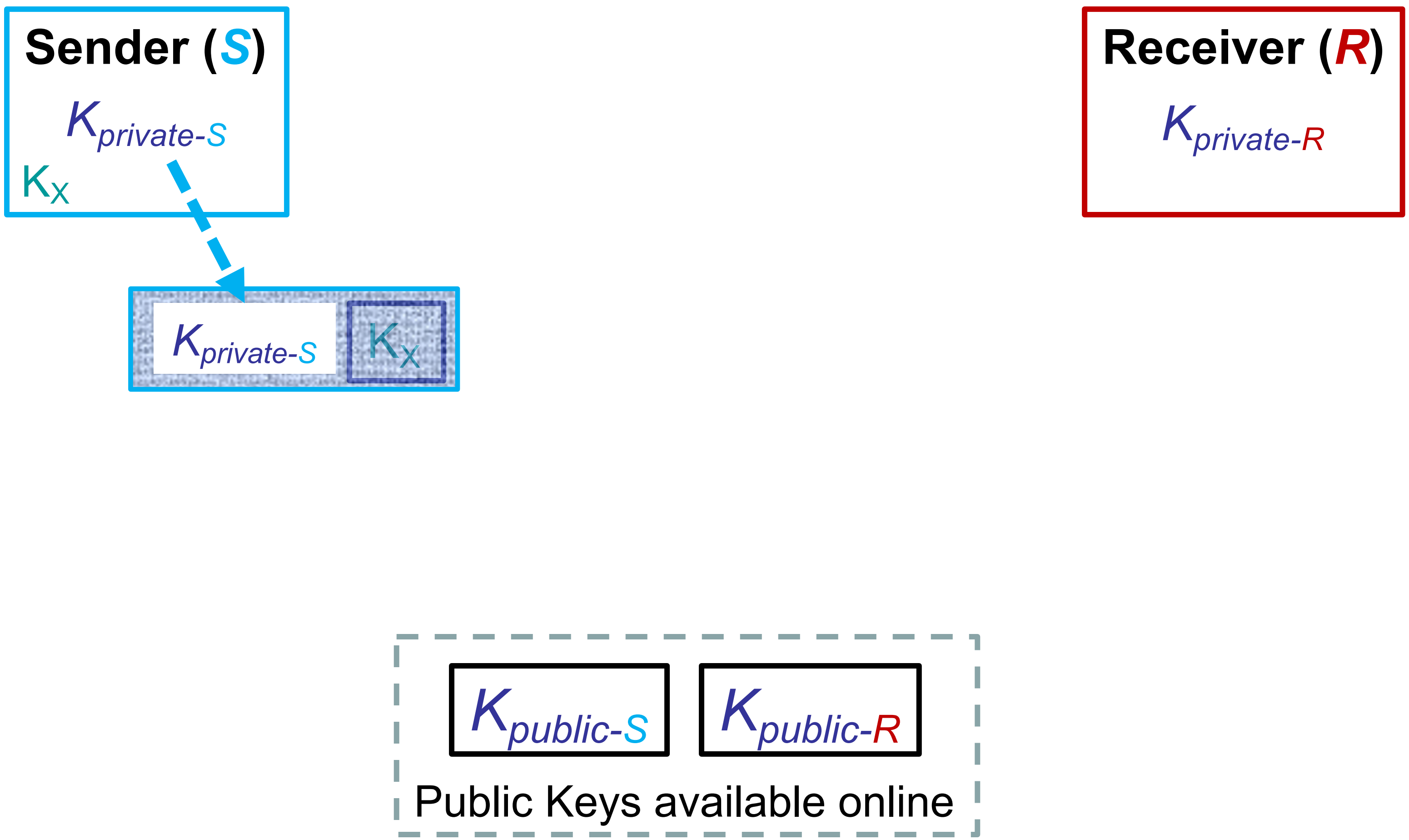
   $$E(K_{public\text{-}R}, E(K_{private\text{-}S}, K_X))$$

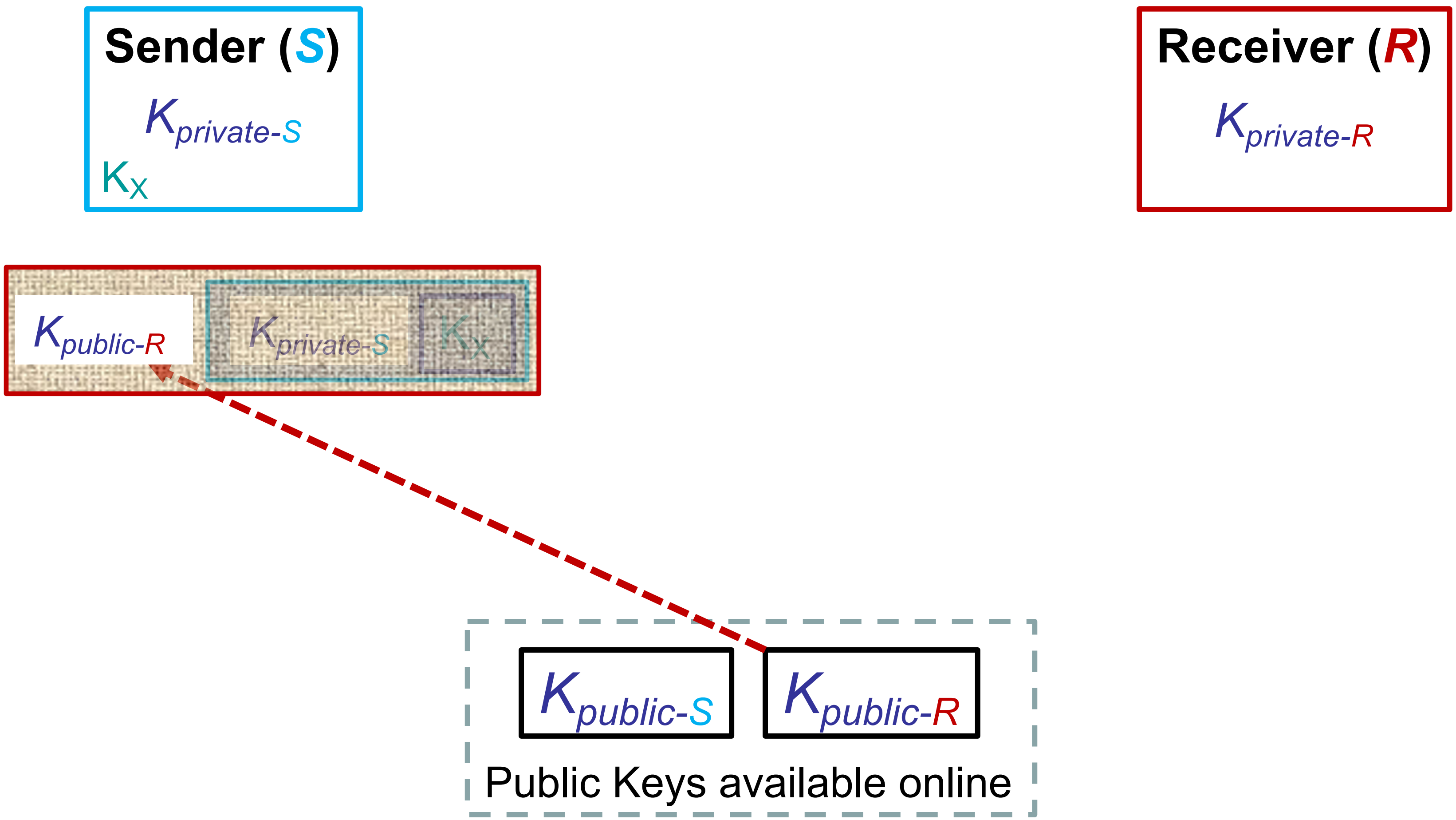
4. R uses his *private-R* to decrypt the outer layer, then uses *public-S* to decrypt the inner layer

Only S could have sent K_X, only R can read it

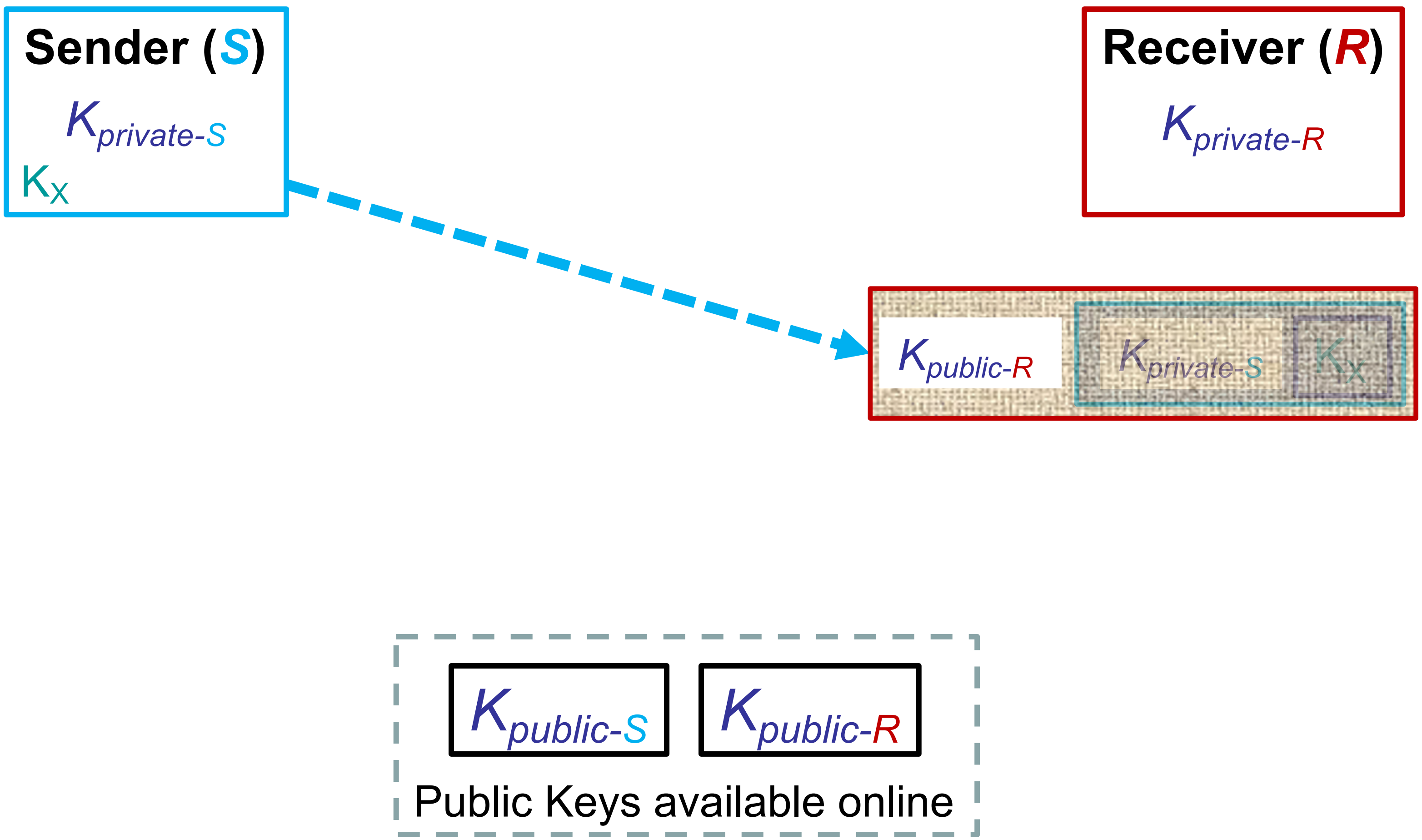


Sender (S)
$K_{private\text{-}S}$
K_X
K_X
Receiver (R)
$K_{private\text{-}R}$
$K_{public\text{-}S}$
$K_{public\text{-}R}$
Public Keys available online

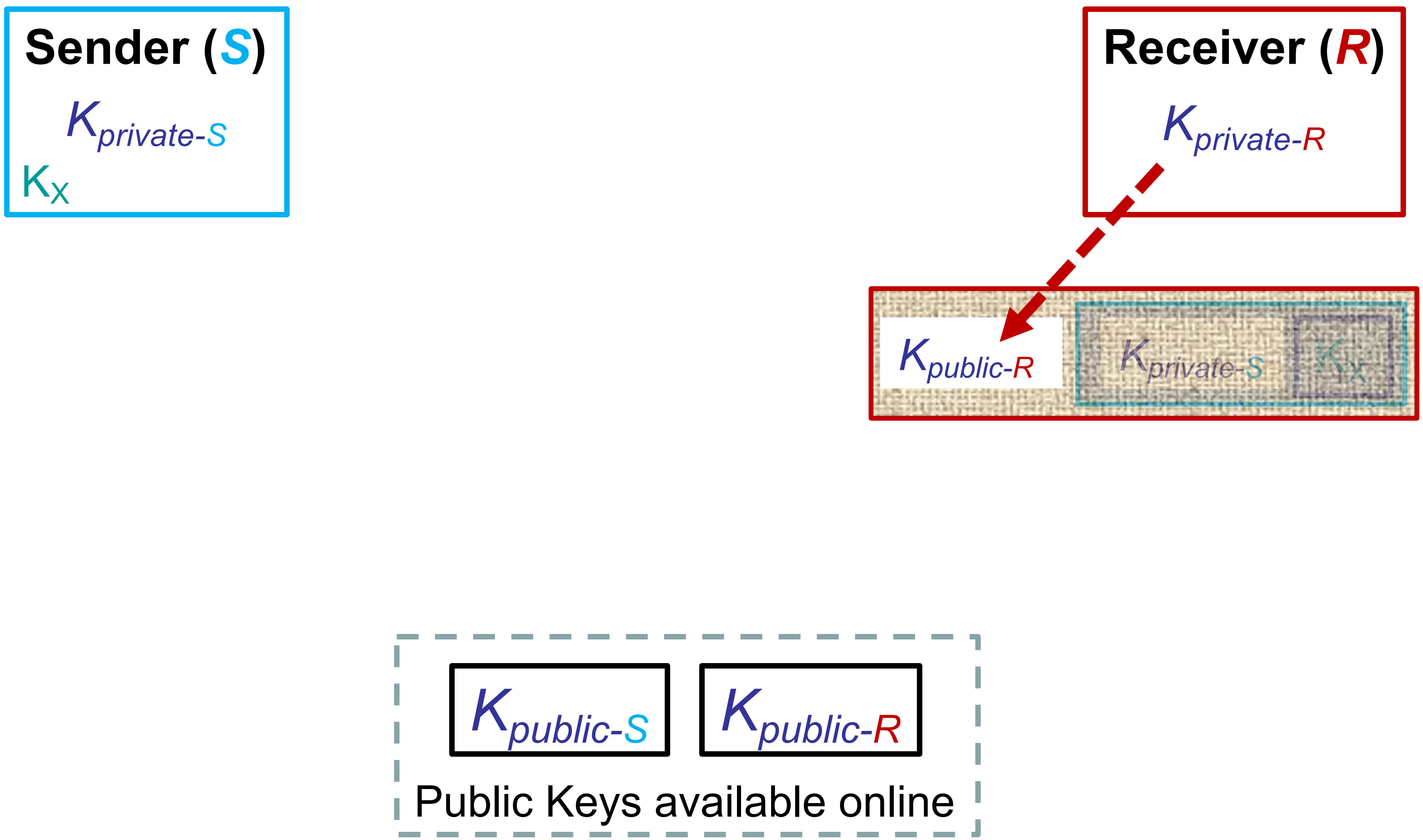

Sender (S)
$K_{private\text{-}S}$
K_X
$K_{private\text{-}S}$
K_X
Receiver (R)
$K_{private\text{-}R}$
$K_{public\text{-}S}$
$K_{public\text{-}R}$
Public Keys available online

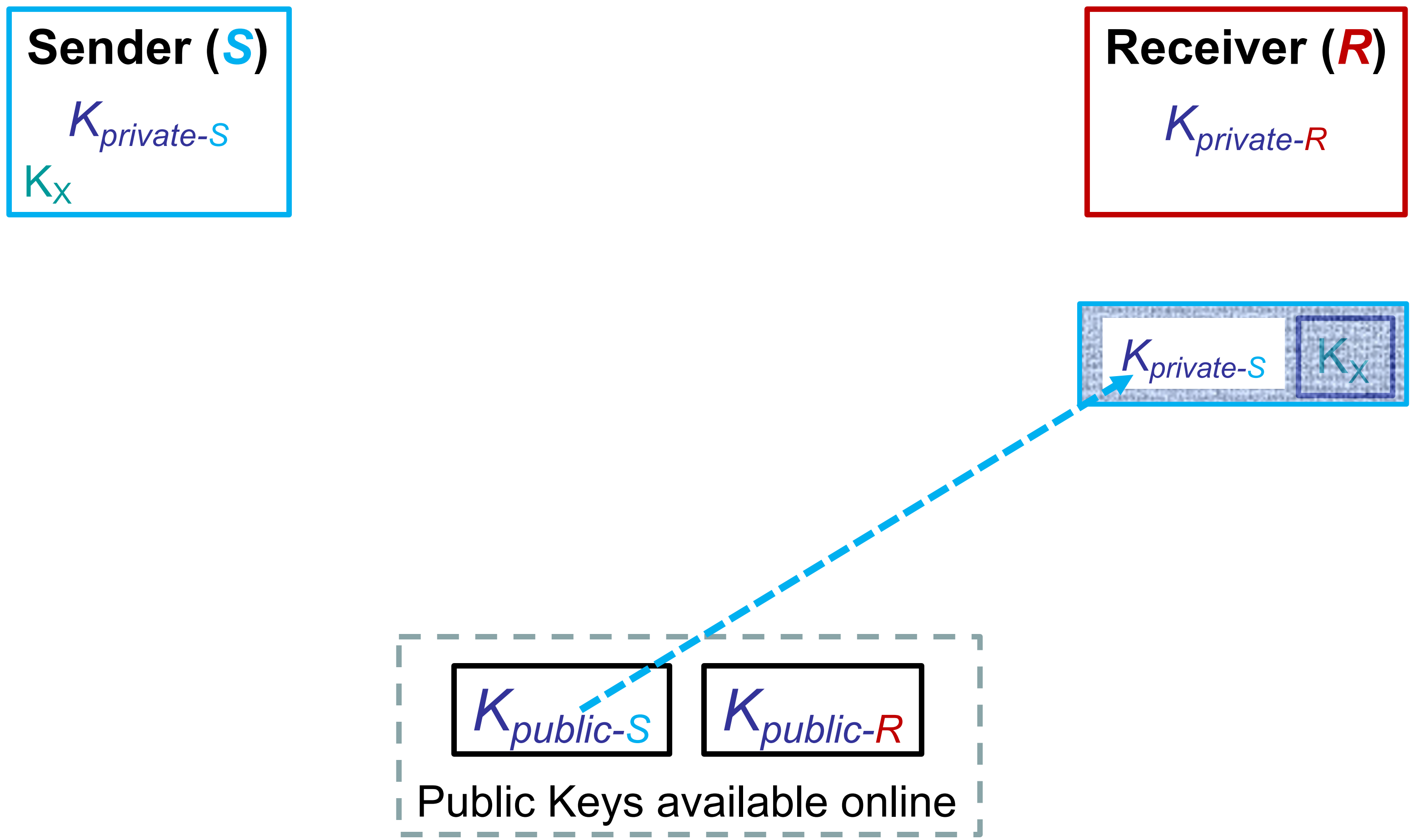


Sender (S)
$K_{private\text{-}S}$
K_X
Receiver (R)
$K_{private\text{-}R}$
$K_{public\text{-}R}$
$K_{private\text{-}S}$
K_X
$K_{public\text{-}S}$
$K_{public\text{-}R}$
Public Keys available online

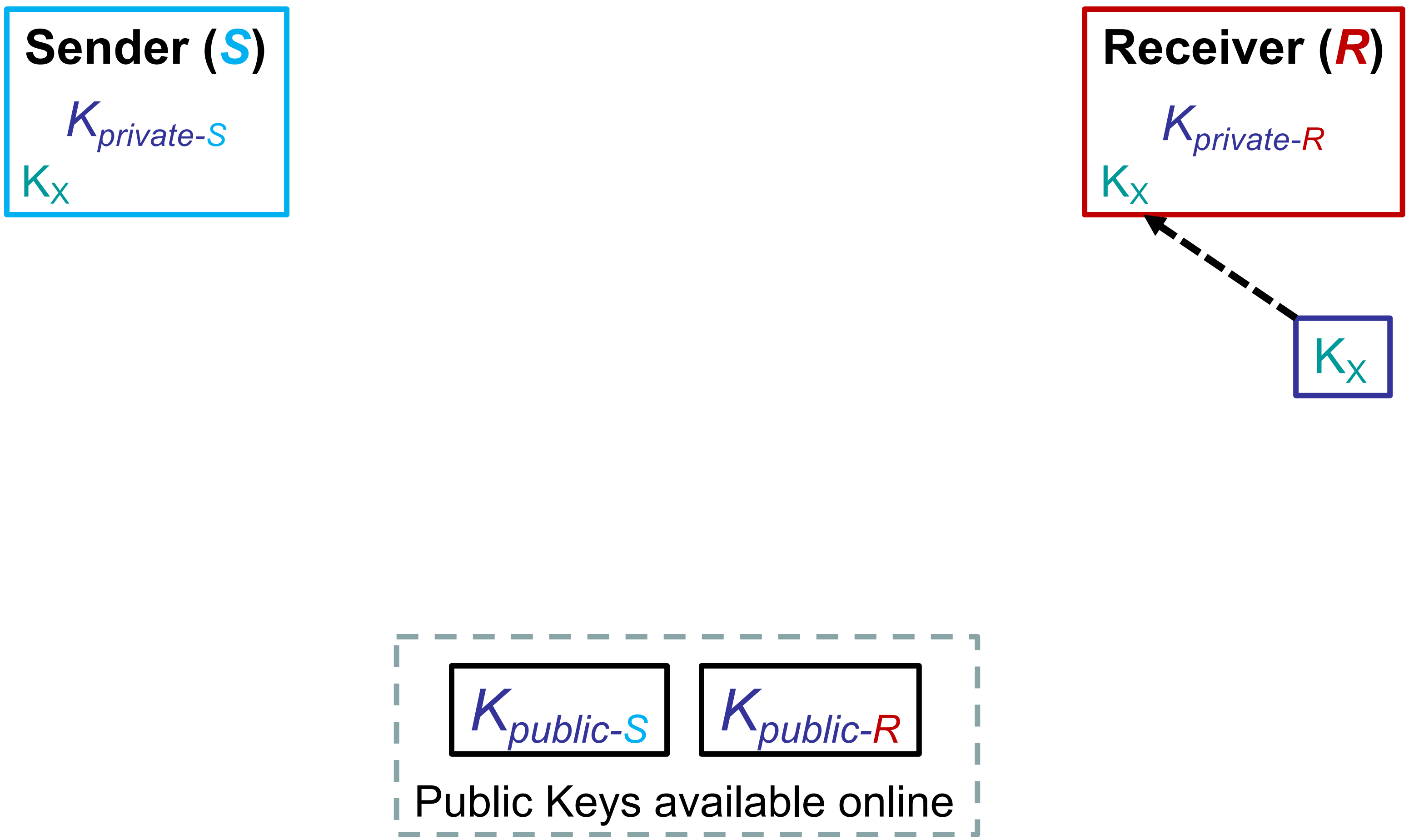


Sender (S)
$K_{private\text{-}S}$
K_X
Receiver (R)
$K_{private\text{-}R}$
$K_{public\text{-}R}$
$K_{private\text{-}S}$
K_X
$K_{public\text{-}S}$
$K_{public\text{-}R}$
Public Keys available online


Sender (S)
$K_{private-S}$
K_X
Receiver (R)
$K_{private-R}$
$K_{public-R}$
$K_{private-S}$
K_X
$K_{public-S}$
$K_{public-R}$
Public Keys available online



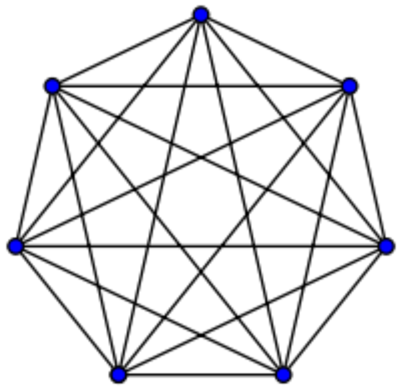
Sender (S)
$K_{private-S}$
K_X
Receiver (R)
$K_{private-R}$
$K_{private-S}$
K_X
$K_{public-S}$
$K_{public-R}$
Public Keys available online



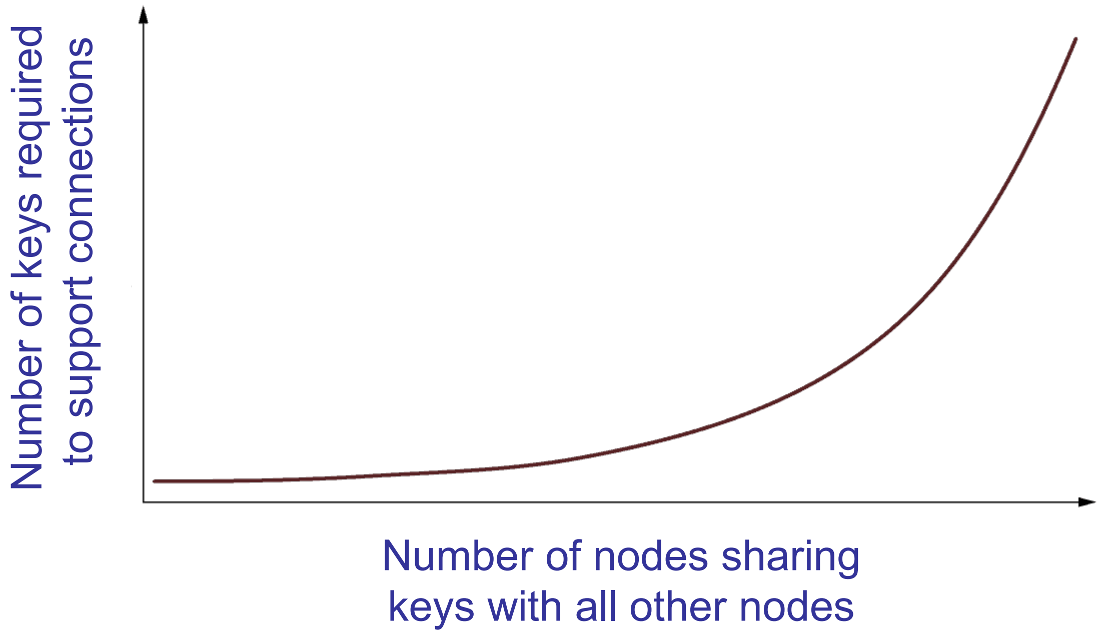
Sender (S)
$K_{private\text{-}S}$
K_X
Receiver (R)
$K_{private\text{-}R}$
K_X
K_X
$K_{public\text{-}S}$
$K_{public\text{-}R}$
Public Keys available online

Key Distribution Issues

- If many individuals want to share data with many others, they must exchange unique symmetric keys with each individual so that none of the others can access the data
  - This leads to a rapid increase in the number of keys that must be shared because it causes a many-to-many mapping of keys
- It also becomes more difficult to manage the use of those keys to avoid attempting to use the wrong key to establish a new secure connection with another host



Unique pairs of keys
between seven nodes



As more hosts connect to each other, the number of unique keys required to make connections will grow exponentially

Key Management

- Even though the number of possible keys is large, most nodes aren’t in constant contact with all of the other nodes
  - However, they would still have to generate and maintain a set of keys for all other nodes
- Also, when a symmetric key is used over a long period of time, the large volume of data encrypted with this key makes it easier to use cryptanalysis to determine the key
  - If the key is discovered or stolen, all of the data encrypted with that key is potentially exposed

Session Keys

- To reduce this risk, a temporary key, called a *session key*, can be used to make a secure connection between two nodes
- When the nodes need to connect a new session key is generated and shared and when the connection ends it is discarded
- A new session key will be created and shared the next time a connection is made
  - this reduces the amount of data that uses the same key and reduces the number of keys each node has to keep track of

Hierarchical Key Control

- One solution to managing key sharing is to use a Key Distribution Center (KDC)
- For communication among entities within the same local domain, a local Key Distribution Center can handle key distribution for all users in that domain
  - If entities in different domains need to share keys, the corresponding local KDC's can communicate through a global KDC
- This hierarchical arrangement can be extended to three or more layers

Controlling Key Usage

- The use of a key hierarchy and automated key distribution techniques greatly reduces the number of keys that must be manually managed and distributed
- It also may be desirable to impose some control on the way in which automatically distributed keys are used
  - For example, in addition to separating master keys from session keys, we may wish to define different types of session keys on the basis of their use

Session Key Lifetime

For connection-oriented protocols one choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session

A security manager must balance competing considerations:

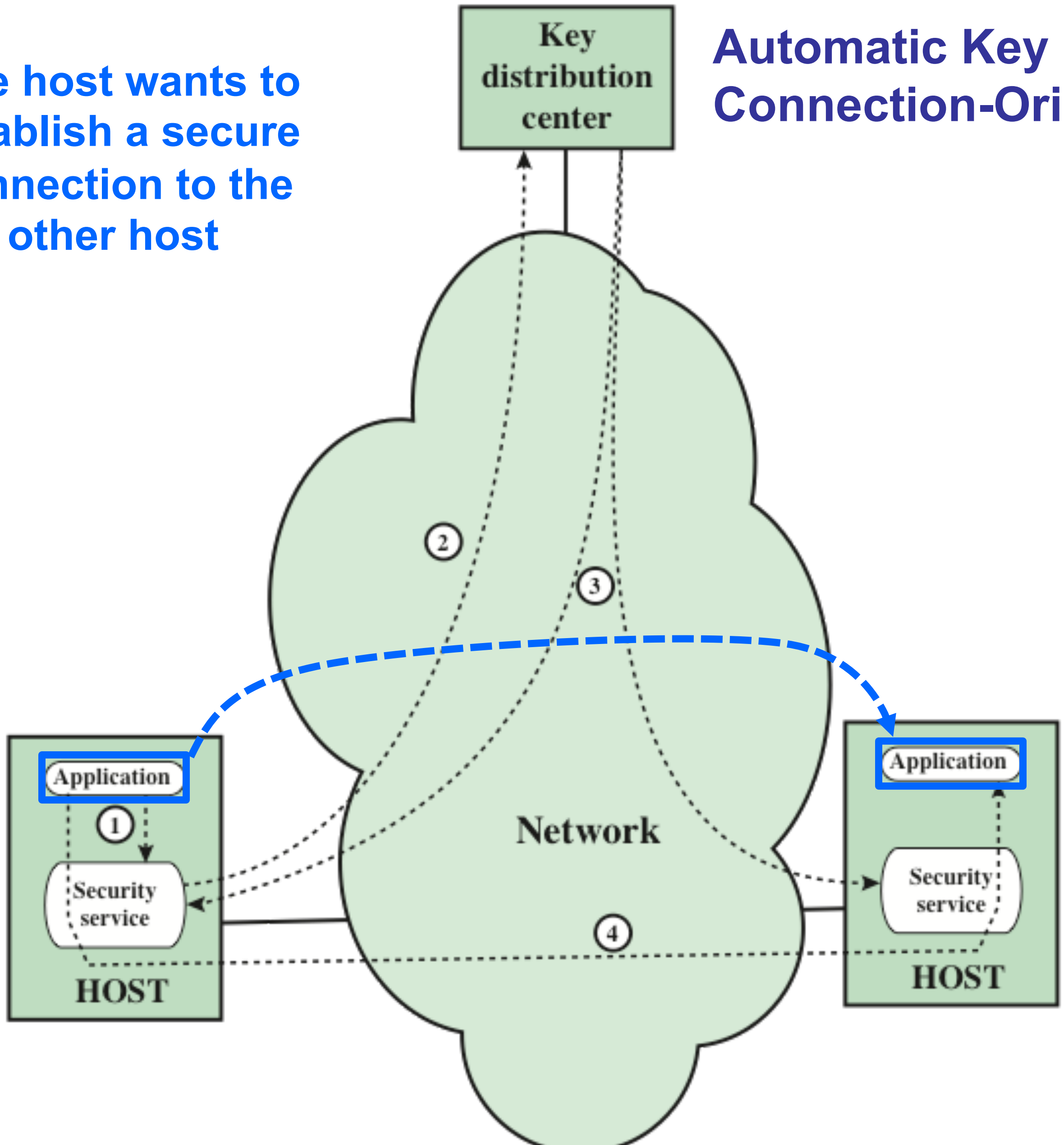
For a connectionless protocol there is no explicit connection initiation or termination, thus it is not obvious how often one needs to change the session key

The more frequently session keys are exchanged, the more secure they are

The distribution of session keys delays the start of any exchange and places a burden on network capacity

Automatic Key Distribution for Connection-Oriented Protocol

One host wants to establish a secure connection to the other host



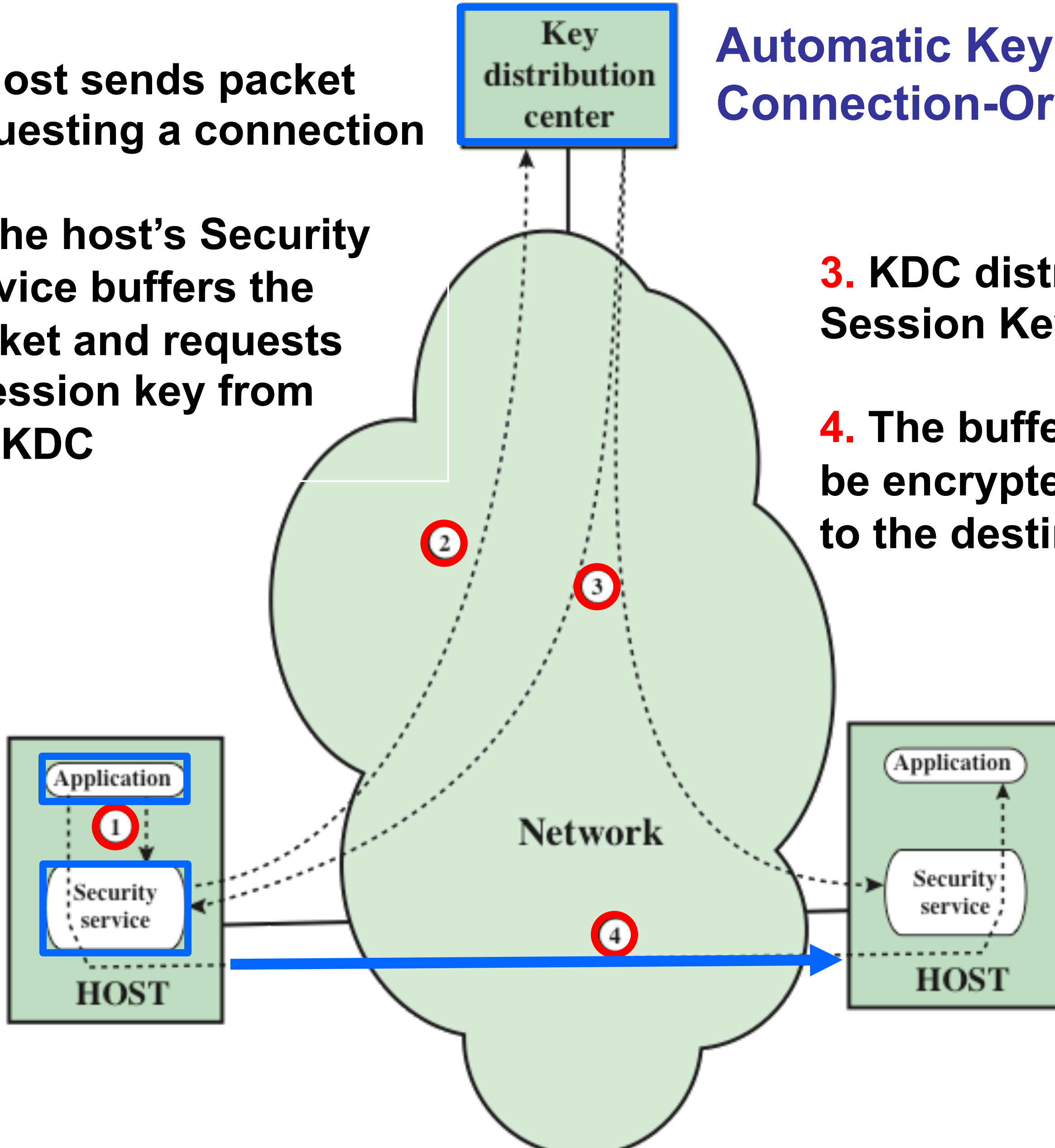
Automatic Key Distribution for Connection-Oriented Protocol

1. Host sends packet requesting a connection

2. The host's Security Service buffers the packet and requests a session key from the KDC

3. KDC distributes a Session Key to both hosts

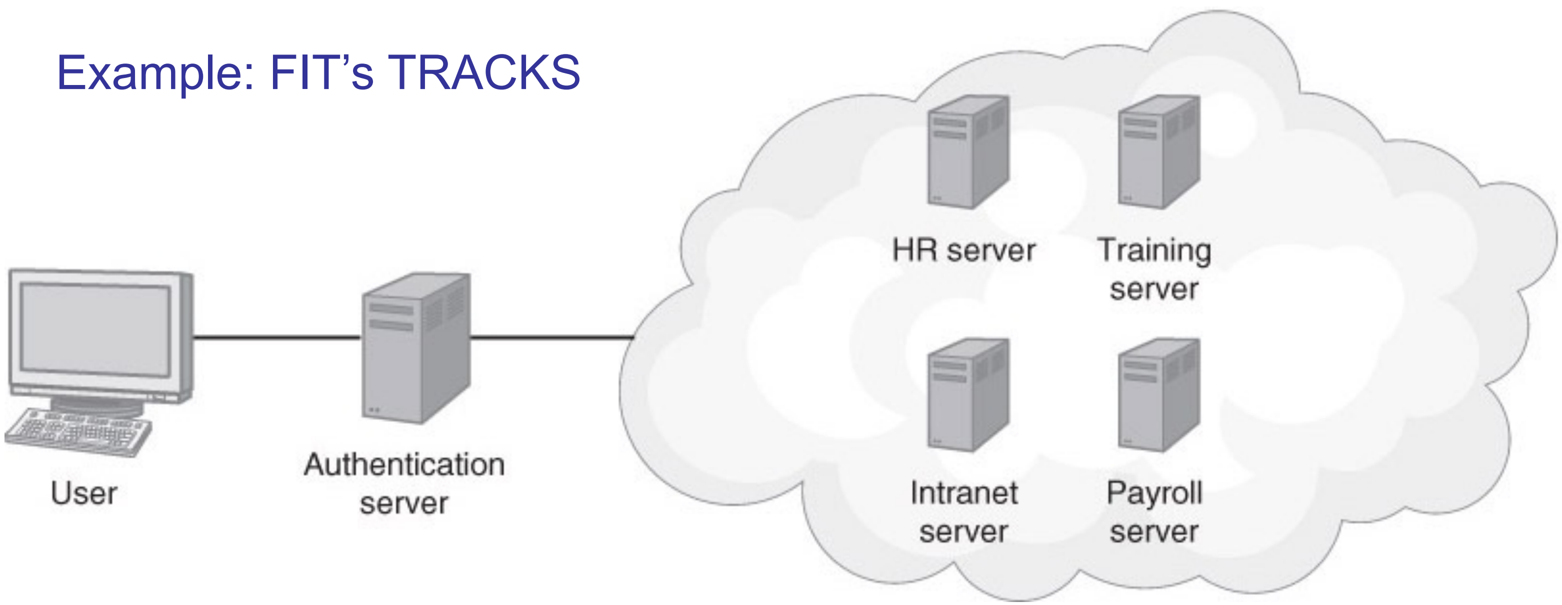
4. The buffered packet can be encrypted and transmitted to the destination host



Single Sign-On (SSO)

- **Single sign-on** (SSO) — Process that allows a user to log into a central authority and then access other sites and services for which he or she has credentials.

Example: FIT's TRACKS



Advantages of Single Sign-On

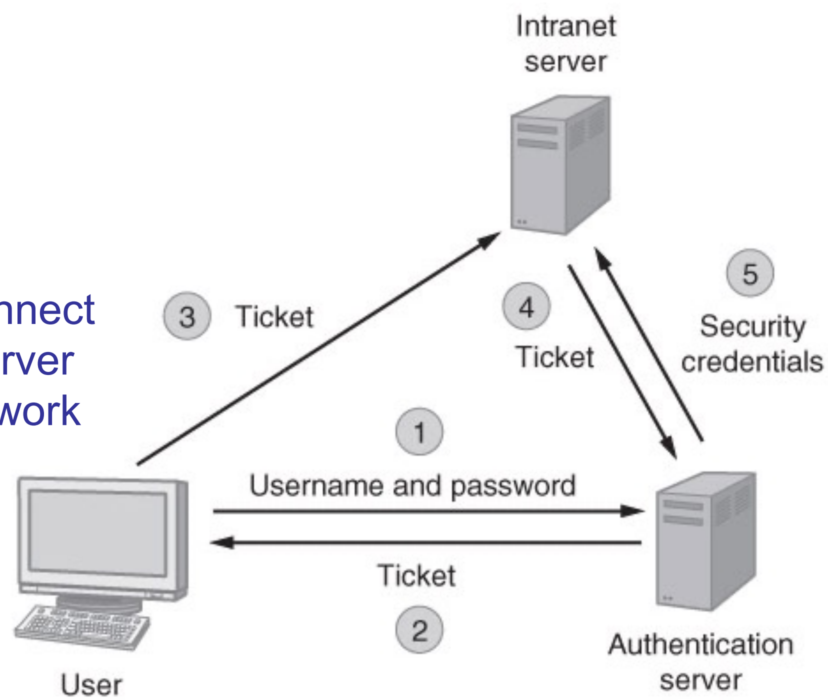
- Fewer username and password combinations for users to remember and manage
  - Which reduces the risks to privacy and security associated with password reuse
- More convenient for users who frequently access multiple machines/systems
  - With fewer calls to the help desk to reset passwords
- Supports centralized management of password compliance and reporting by IT staff

Disadvantages of Single Sign-On

- The primary disadvantage of SSO systems is the potential for a single source of failure.
- If the authentication server fails, users will not be able to log in to other servers.
  - Having a redundant (perhaps cloud-based) authentication server that can immediately be brought online if the primary server fails will reduce the risk of losing access to the system

How Single Sign-On Works

User wants to connect to an intranet server in the same network



Authentication server and Intranet server use encrypted connections to share information

Step 1: User logs into the authentication server using a username and password

Step 2: The authentication server returns the user's ticket

Step 3: User sends the ticket to the intranet server

Step 4: Intranet server sends the ticket to the authentication server

Step 5: Authentication server sends the user's security credentials for that server back to the intranet server

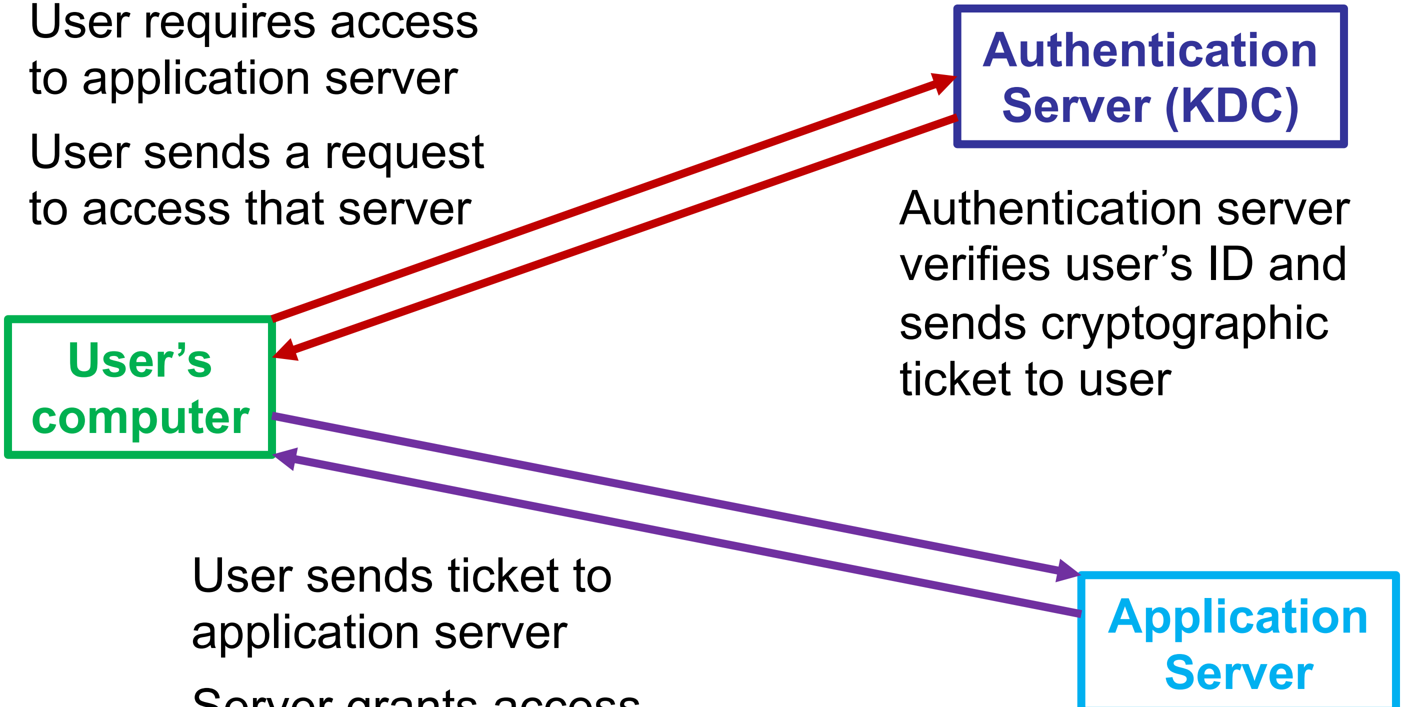
Kerberos

- Authentication in distributed environments
  - can use AES symmetric key encryption or public key encryption
  - credentials are stored on a Kerberos server which authenticates users, manages access
  - authenticated user gets unforgeable "ticket" from server, uses it to access resources
  - uses timestamps to control length of time ticket is valid and prevent reuse of tickets

Kerberos

User requires access to application server

User sends a request to access that server



Authentication server verifies user's ID and sends cryptographic ticket to user

User sends ticket to application server

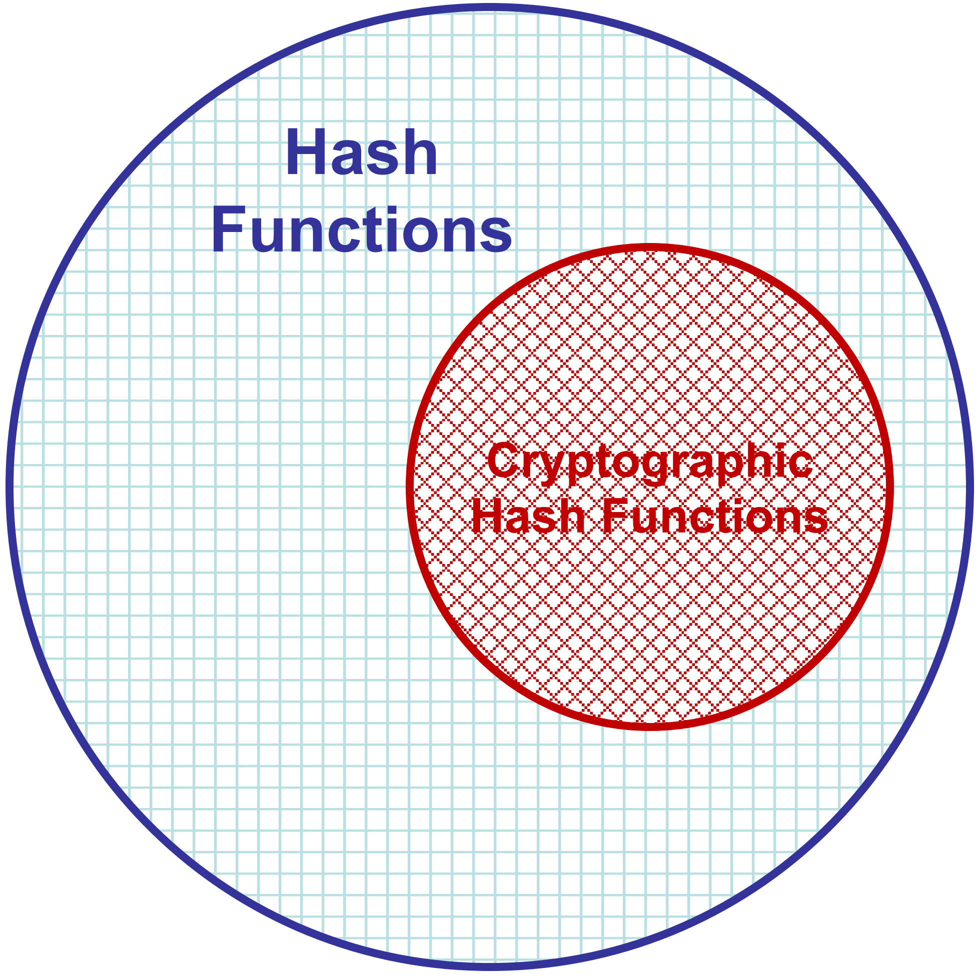
Server grants access

The cryptographic ticket includes the ID of the server, and a session key that is used to access that server

Hash Functions

- A *hash function* is any well-defined procedure or mathematical function which converts a large, possibly variable-sized amount of data into a smaller number
  - those results are usually integer numbers
- The values returned by a hash function are called *hash values*, *hash codes*, *hash sums*, or a *message digest*
- However, they *do not* encrypt the data

Cryptographic Hash Functions



Cryptographic Hash Functions

- Not every hash function is a cryptographic hash function, but every cryptographic hash function is a hash function
- Hash functions
  - Most important property is collision avoidance
  - Mainly used for fast retrieval of data
- Cryptographic Hash functions
  - Provide security, avoid collision, provide an apparently random output, provide diffusion

Cryptographic Hash Functions

- The *ideal* cryptographic hash function has the following properties:
  - it should not be possible to find a message that has a given hash,
  - it should not be possible to modify a message without changing its hash,
  - it should not be possible to find two different messages with the same hash.
- However, poorly designed hash functions do not achieve all three goals

Hash Functions

- The message digest (hash value) of a good hashing algorithm should have a random pattern
  - Randomness: any bit in digest is “1” ~half the time
  - If you change only one bit in the input, the hash algorithm should change half of the digest bits
  - Diffusion: if hash function does not change around half the bits with a slight change of input, then it has poor randomization, and thus a cryptanalyst can make predictions about the input, even when given only the output

Hash Example

- The two messages below differ by only one character, but the hash values are quite different

A hash function applied to:
"The quick brown fox jumps over the lazy dog"
produces the following hash value:

`37c4b87edffc5d198ff5a185cee7ee09`

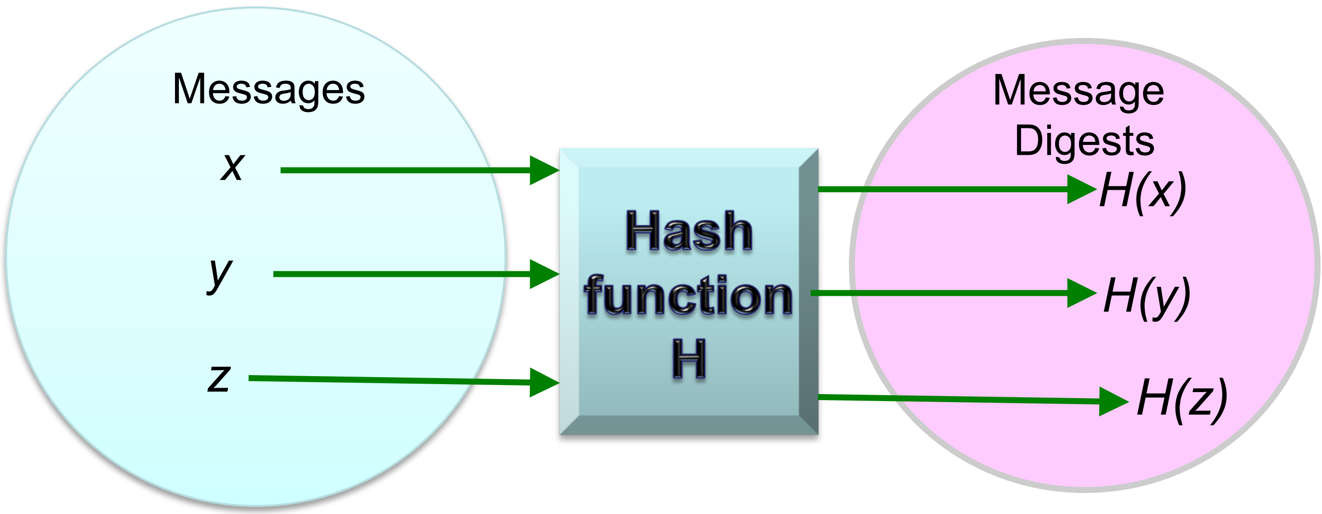
The same hash function applied to the message, but with one letter changed (changing only 1 bit):
"The quick brown fox jumps over the mazy dog"
produces a very different hash value:

`6f40fa4886af7070cf05e2c5c84c4937`

Collisions

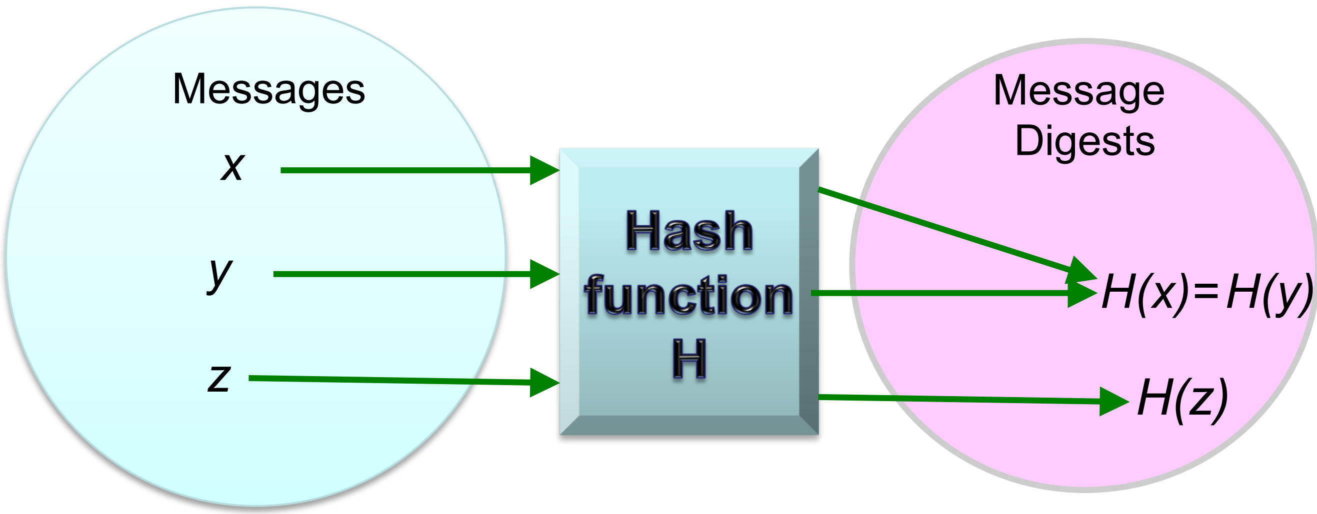
- If a given hash function produces the same output from *two different inputs*, this is called a "*collision*"
- This impacts the usefulness of that hash function because it cannot guarantee which input was used to produce the hash
  – if you could modify a file so that both the original and modified copy produced the same hash, the change would *not be detectable*

Hash Functions



- H is a hash function that is used to produce a *message digest or hash value*

Hash Functions



- H is a hash function that is used to produce a *message digest or hash value*
- Collisions: H(*x*) = H(*y*) for messages *x* and *y*
  - two distinct files should not have the same hash value, if they do then the hash value is not useful

Some Applications of Hashing

- Integrity in Software Distribution
  - For a *Good File* and Hash(*Good File*), it should not be possible for an attacker to create a *Tampered File* such that:

    Hash(Good File) = Hash(Tampered File)
- This is why you often see an *MD-5 hash* shown on a webpage for software downloads:
  - download the file and calculate the MD-5 hash
  - compare with the hash code shown at the website
  - if they match, your download was not corrupted (?)

Standard Hash Functions

- MD5 (Message-Digest algorithm 5)
  - It was designed by Ron Rivest (RSA) in 1991 as an improvement of MD4, widely used for authentication
- RIPEMD-160
  - 160-bit variant of MD-5
  - RIPEMD-128, RIPEMD-256, and RIPEMD-320
- SHA-1 (Secure Hash Algorithm)
  - originally 160-bit output
  - US government (NIST) standard as of 1995 - 2003
    - hash algorithm for the Digital Signature Standard (DSS)
- SHA-2
  - can produce 256, 384 and 512 bit outputs

Message Digest 5 (MD5)

- MD5 (1991) replaced MD4 (which was broken by 1995), both designed by Ron Rivest (RSA)
  - MD5 produces a 128-bit digest
    - In theory: 1 in 2^{64} chance of two msgs with same digest
  - MD5 has also been shown to be susceptible to *collision attacks* where a message could be modified in a way that creates the same hash as the original
    - 2004: produced 2nd message with same hash in < one hour
    - 2005: produced two different certificates with same hash
    - 2010: produced same hash with 512-bit messages
    - 2012: created fake Microsoft digital signatures (uses MD5)

MD5 Collisions

- Both of these messages (binary data) produce the MD5 hash: **79054025255fb1a26e4bc422aef54eb4**
  - analysis of the MD5 algorithm allowed researchers to determine that specific bits could be modified to produce the same hash

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```

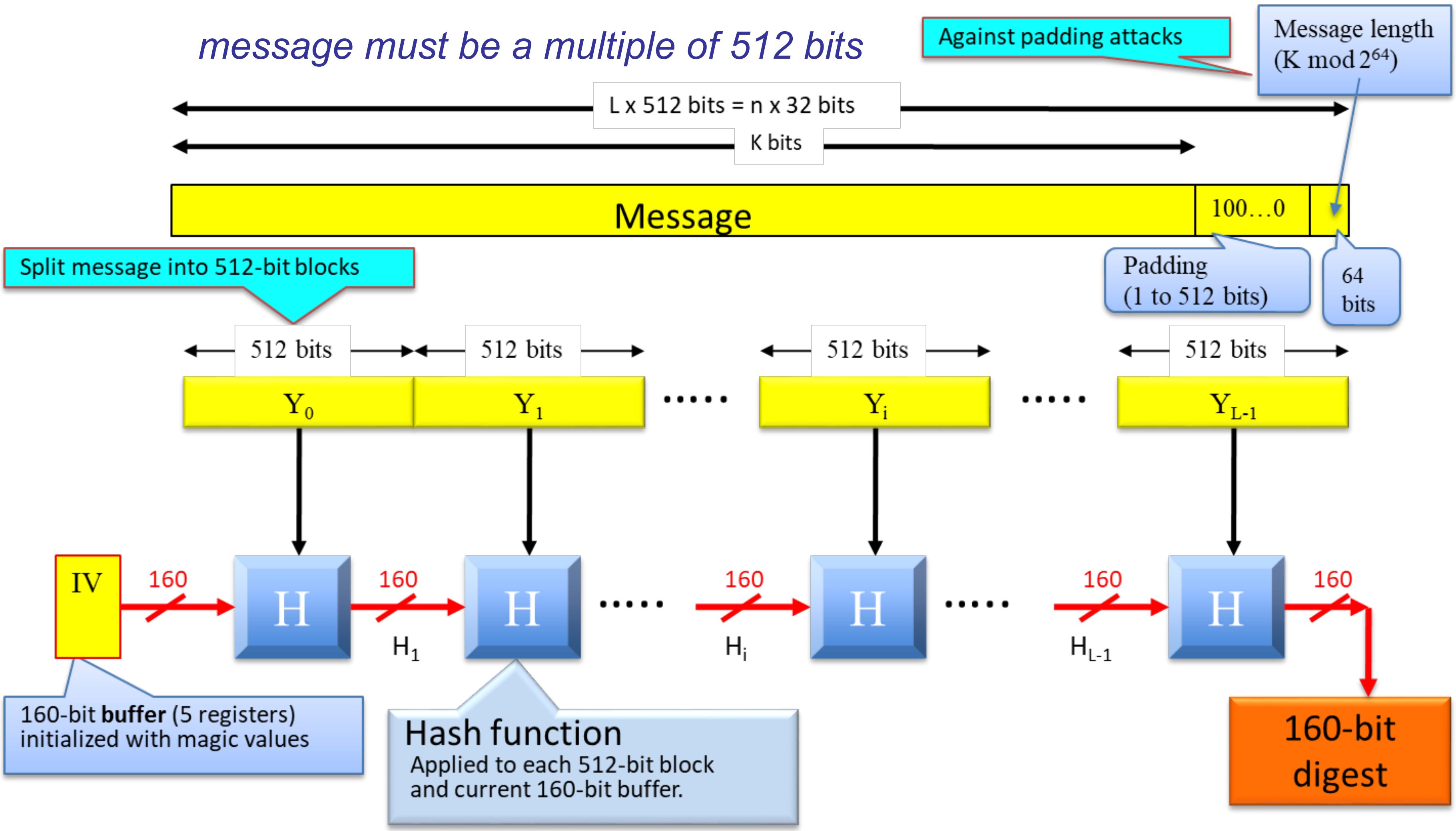
Secure Hash Algorithms (SHA)

- Family of cryptographic hash functions designed by NSA and published by NIST for use by the United State government
  - SHA-0 (published as SHA in 1993) included flaws that caused it to be replaced quickly
  - SHA-1 (1995) design is similar to MD5, but with 160-bit digest
  - SHA-2 (2001) several different digest sizes
  - SHA-3 (2015) new design, NIST approved

SHA-1

- SHA-1 generates a 160-bit digest
  - 1 in 2^{80} chance of two messages with same digest, but it is really $2^{63} \sim= 10^{24}$, which is quite feasible with current hardware
- Since 2005, *no longer considered secure*
  - web browsers no longer accept certificates that are based on SHA-1
  - in 2017 Google was able to produce two different PDF files with same SHA-1 hash
  - still used for error checking in data files

Basic Structure of SHA-1



SHA 2

- SHA-2 can produce different digest sizes
  - Standard sizes: 224, 256, 384 and 512 bits
- Published in 2001 and recommended by NIST to replace SHA-1
- More secure than SHA-1, but still less than ideal for some applications
  - Susceptible to *length extension attacks*
    - add data to the end of a message and compute a new valid hash based on that longer message

SHA 3

- Released by NIST in 2015
- Uses different algorithms from SHA-1 & 2
  – Not intended to replace SHA-2, but to provide an alternate with different strengths
- Produces 224, 256, 384 or 512 bit hashes
- The examples below used the same input text

SHA2-256:

bd1e994f0ba615d8b0f0674d22fffcdfd86bf1827f09dc983dde584a4b6b1899

SHA3-256:

e74bcdd564a5d4cfe7452c55d1e209dddbc29d7a8d2690d6a8a91edfc2ee3fc9

SHA Properties

- SHA-1 and SHA-256: a message of any length < 2^{64} bits
- SHA-384 and SHA-512: message of any length < 2^{128} bits

Algorithm Name	Max. Message Size (bits)	Block Size (bits)	Word size (bits)	Digest size (bits)	Collision resistance (bits)
SHA-1	2^{64}	512	32	160	80
SHA-256	2^{64}	512	32	256	128
SHA-384	2^{128}	1024	64	384	192
SHA-512	2^{128}	1024	64	512	256
SHA-512/224	2^{128}	1024	64	224	112
SHA-512/256	2^{128}	1024	64	256	128