2CSDE93- Blockchain Technology

Cryptographic Primitives

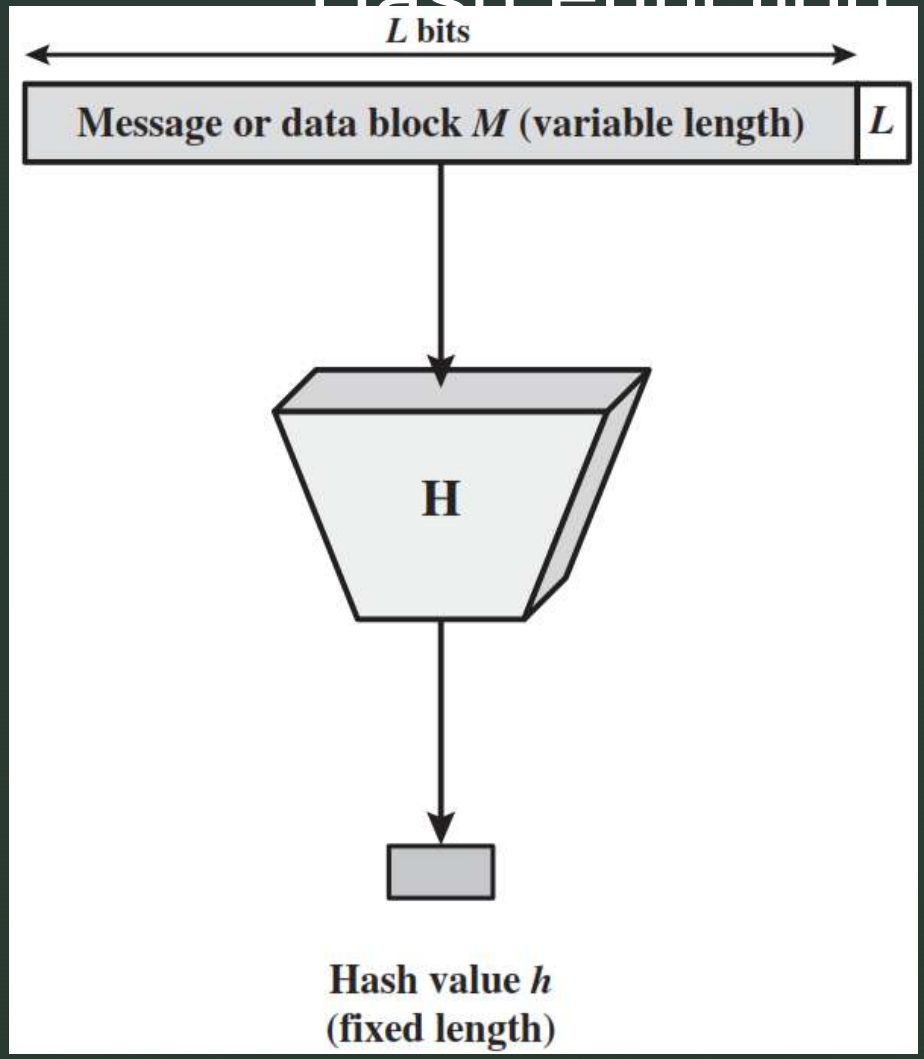
Outline

- Cryptographic Hash Functions
- Simple hash functions: Secure Hash Algorithm (SHA)
- Message Authentication
- MAC (Message Authentication Code)
- Digital Signature
- Key Management
- Key Exchange
- Conclusion

HASH FUNCTIONS

Hash Function

- A hash function **H** accepts a variable-length block of data **M** as input and produces a fixed-size hash value **h = H(M)**.
- A “good” hash function has the property that the results of applying a change to any bit or bits in **M** results, with high probability, in a change to the **hash code**.



Applications of Cryptographic Hash Functions

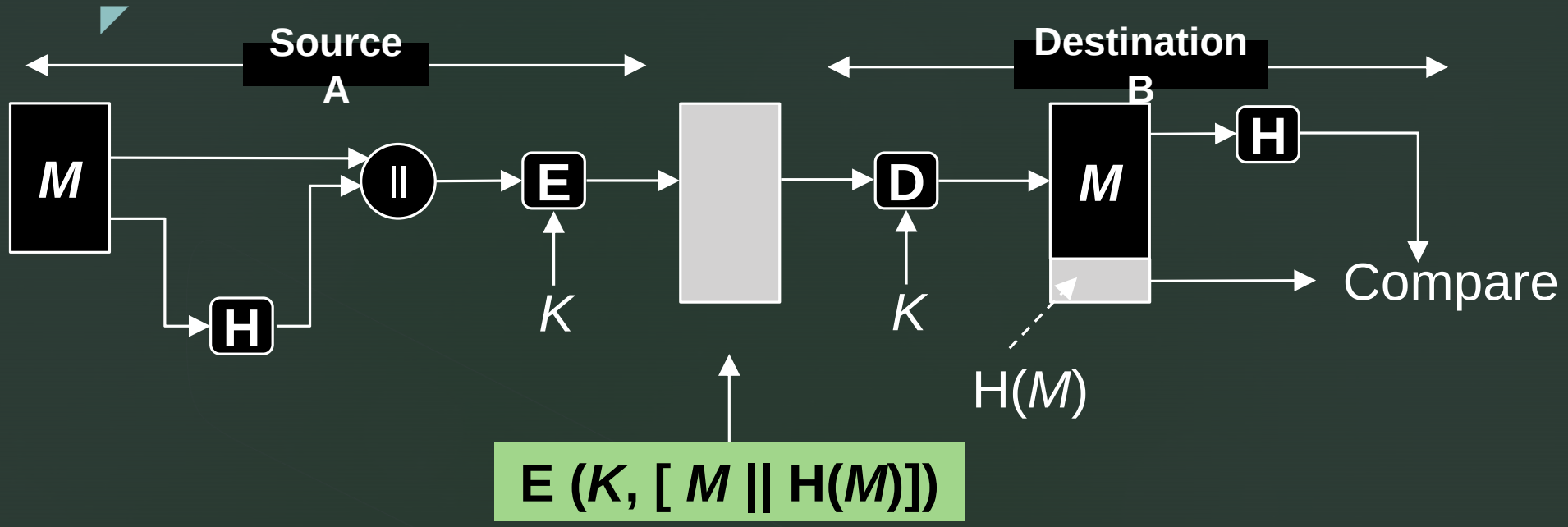
1. Message authentication
2. Digital Signature
3. One-way password file

Message Authentication

1. Message Authentication

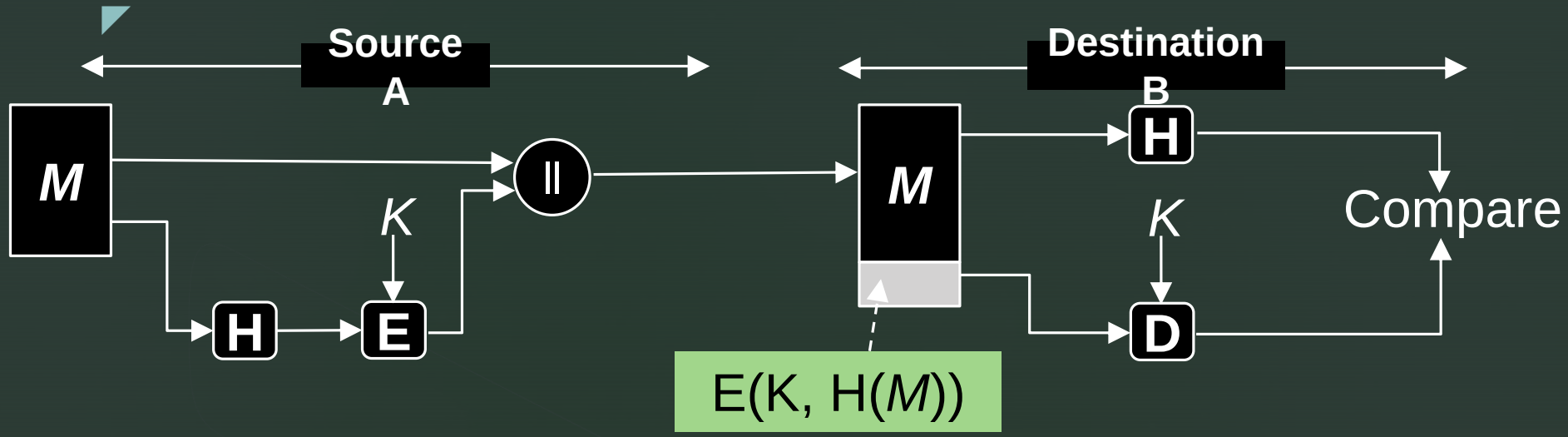
- **Message authentication** is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay).
- When a hash function is used to provide message authentication, the **hash function value** is often referred to as a **message digest**.

Message authentication method - 1



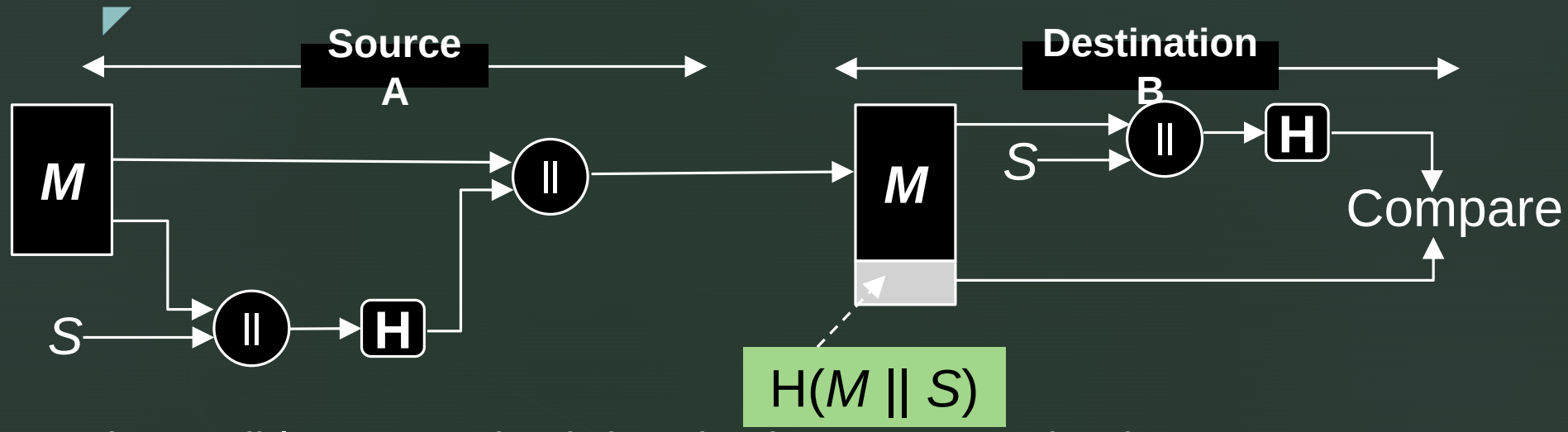
- Only A and B share the secret key, the message must have come from A and has not been altered.
- The hash code provides the structure required to achieve authentication.
- Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

Message authentication method - 2



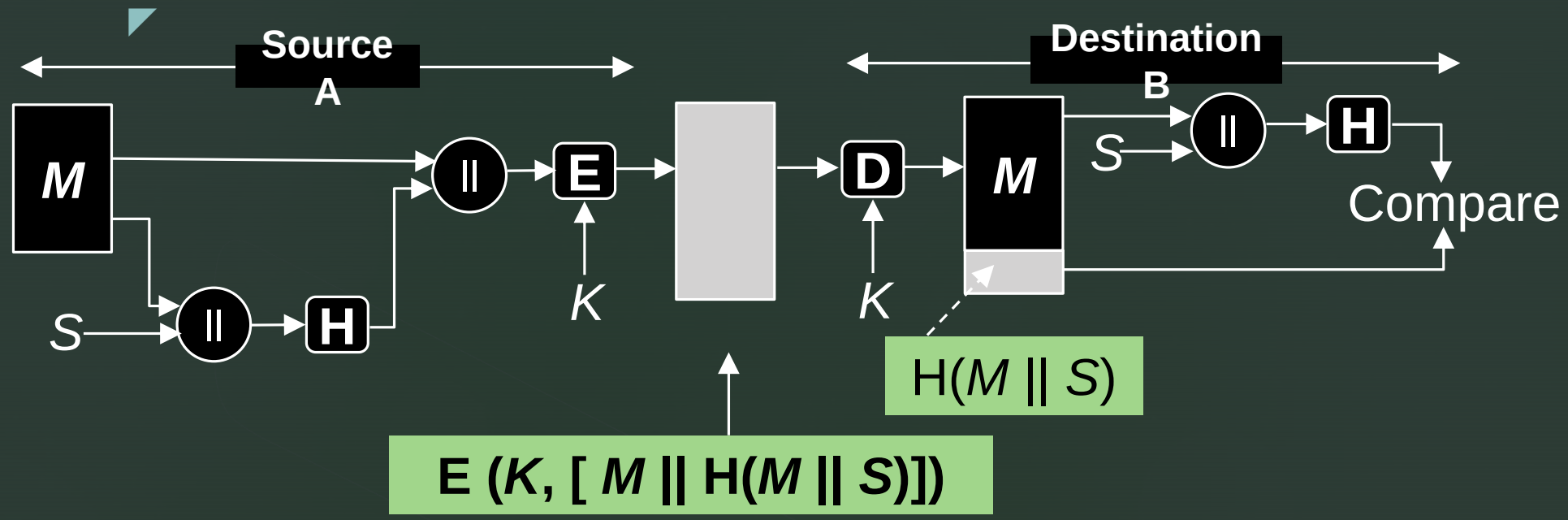
- Only the hash code is encrypted, using symmetric encryption.
- This reduces the processing burden for those applications that do not require confidentiality.

Message authentication method - 3



- It is possible to use a hash function but no encryption for message authentication.
- A and B share a common secret value S.
- A computes the hash value over the concatenation of M and S and appends the resulting hash value to M.
- Because B possesses S, it can recompute the hash value to verify.
- An opponent cannot modify an intercepted message.

Message authentication method - 4



- Confidentiality can be added to the approach of method (3) by encrypting the entire message plus the hash code.

MESSAGE AUTHENTICATION CODE (MAC)

MAC (Message Authentication Code)

- More commonly, message authentication is achieved using a **MAC** also known as **keyed hash function**.
- MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.
- A **MAC** function takes as input a secret key and a data block and produces a hash value, referred to as the **MAC**.
- The combination of hashing and encryption results in an overall function that is, in fact, a MAC (Method -2 in previous slide).

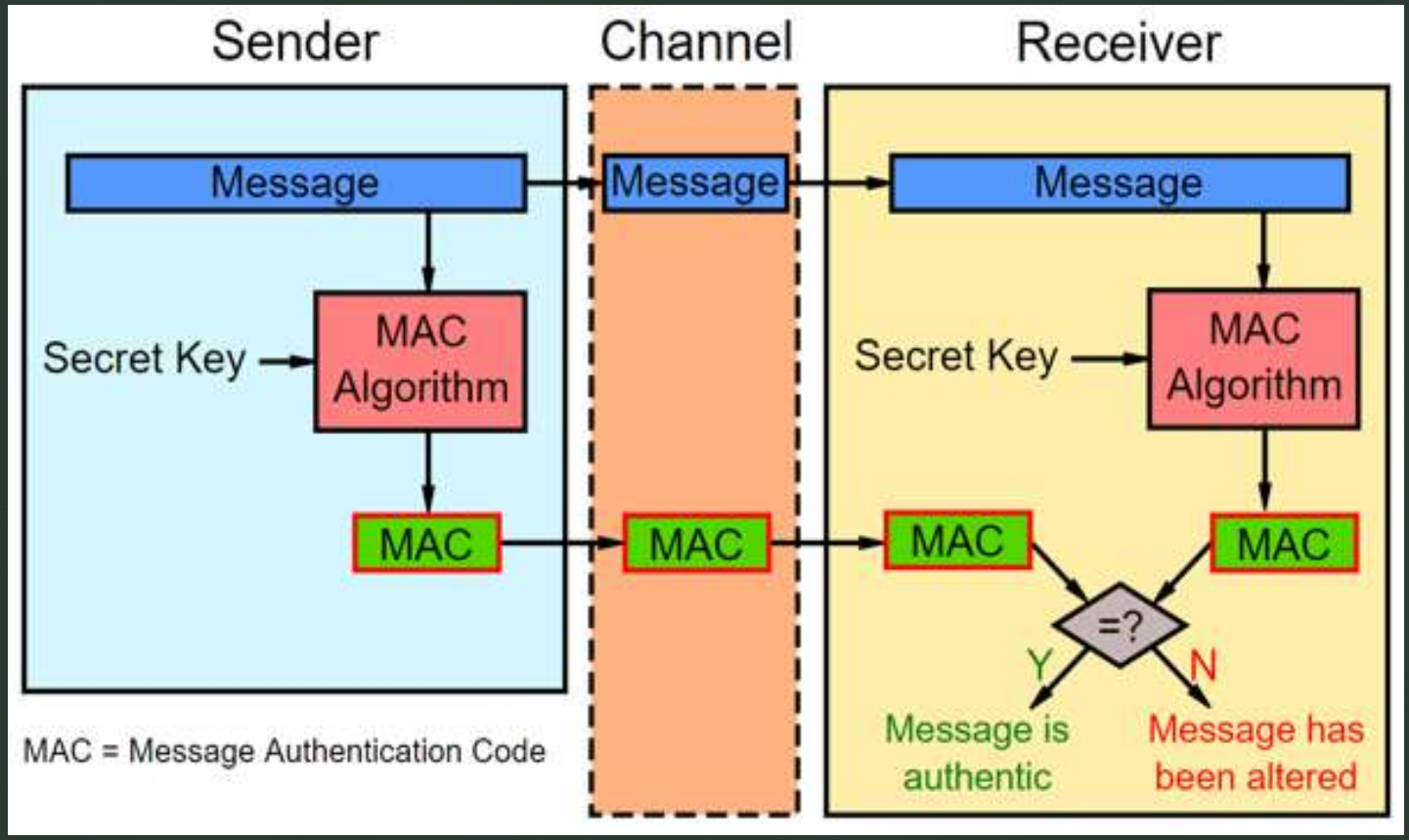
What is MAC?

- A message authentication code is an algorithm which takes as input a **message and a secret key** and produces a fixed-sized output which can be later on verified to match the message.
- The verification also requires the **same secret key**. Contrary to hash functions where everything is known and attackers are fighting against mathematics, MAC make sense in models where there are **entities with knowledge of a secret.**
- What we expect from a good MAC is **unforgeability**: it should be infeasible to compute a pair ***message+MAC*** value which successfully verifies with a given key *K* without knowing *K* exactly and in its entirety.

Hash vs MAC- What's the difference??

- The main difference is conceptual: while **hashes** are used to guarantee the **integrity of data**, a **MAC** guarantees **integrity AND authentication**.
- This means that a **hashcode** is blindly generated from the message without any kind of external input: what you obtain is something that can be used to check if the message got any alteration during its travel.
- A **MAC** instead uses a **private key as the seed** to the hash function it uses when generating the code: this should assure the receiver that, not only the message hasn't been modified, but also who sent it is what we were expecting: otherwise an attacker couldn't know the private key used to generate the code.

A flow scenario of MAC generation and verification

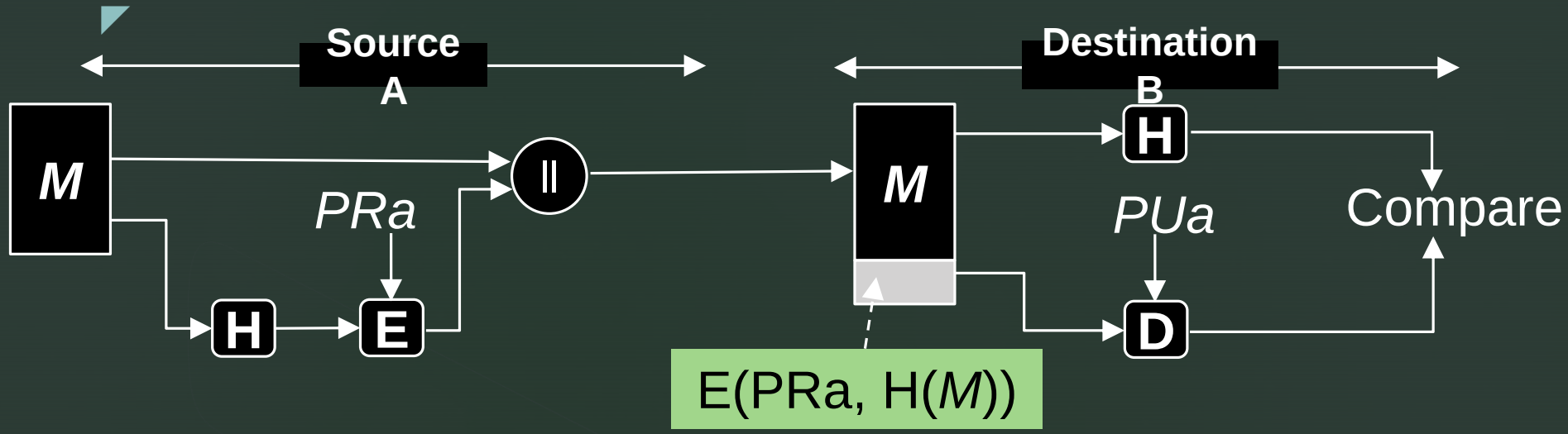


Digital Signature

Digital Signature

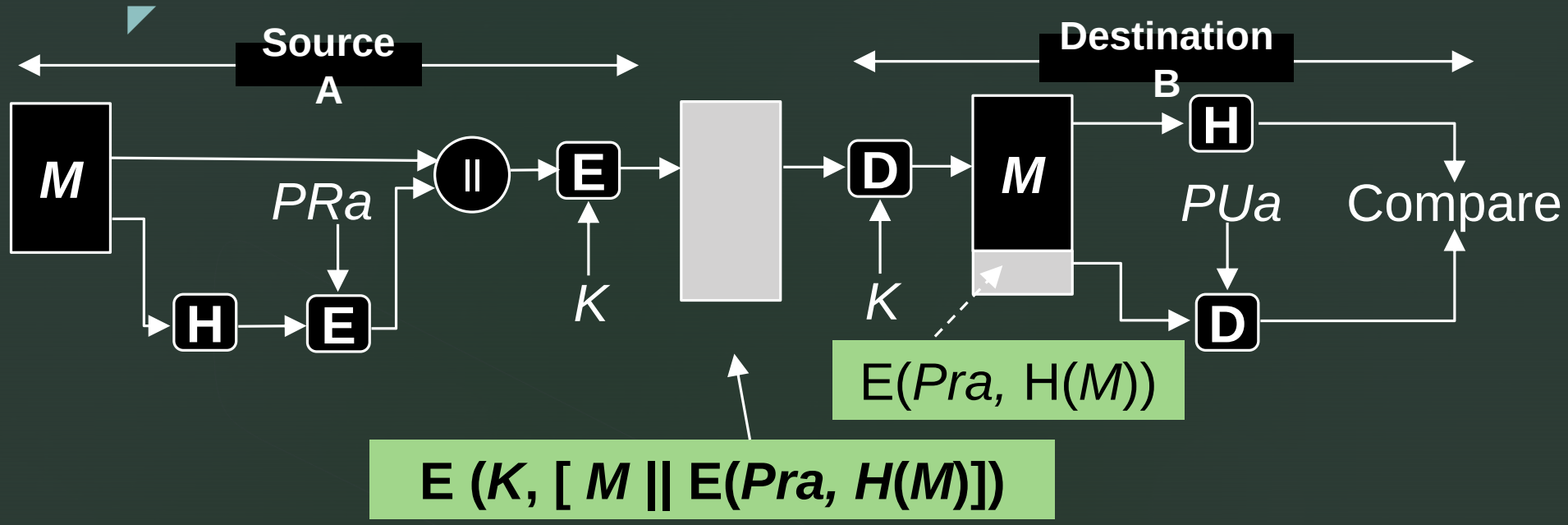
- A **digital signature** is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.
- The operation of the digital signature is similar to that of the **MAC**.
- In the case of the **digital signature**, the hash value of a message is encrypted with a user's **private key**.
- Anyone who knows the user's **public key** can verify the integrity of the message that is associated with the **digital signature**.

Digital Signature method - 1



- The hash code is encrypted, using public-key encryption with the sender's private key.
- This provides authentication.
- It also provides a **digital signature**, because only the sender could have produced the encrypted hash code.

Digital Signature method - 2



- If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key.

Security Requirements

1. Disclosure
2. Traffic analysis
3. Masquerade
4. Content modification
5. Sequence modification
6. Timing modification
7. Source repudiation
8. Destination repudiation

Requirements for hash functions

1. Can be applied to any *sized message M*.
2. Produces *fixed-length output h*.
3. It is *easy to compute h=H(M)* for any *message M*.
4. Given hash value *h* is *infeasible* to find *y* such that (*H(y) = h*)
   - *One-way property*
5. For given block *x*, it is computational infeasible to find

   y ≠ x with *H*(*y*) = *H*(*x*)
   - *Weak collision resistance*
6. It is computationally infeasible to find messages *m1* and *m2* with *H*(*m1*) = *H*(*m2*)
   - *Strong collision resistance*

Simple Hash Function

- The input (message, file, etc.) is viewed as a sequence of n-bit blocks.
- The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block.

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

Where,

C_i = i^{th} bit of the hash code $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = i^{th} bit in j^{th} block

SHA - Secure Hash Algorithm

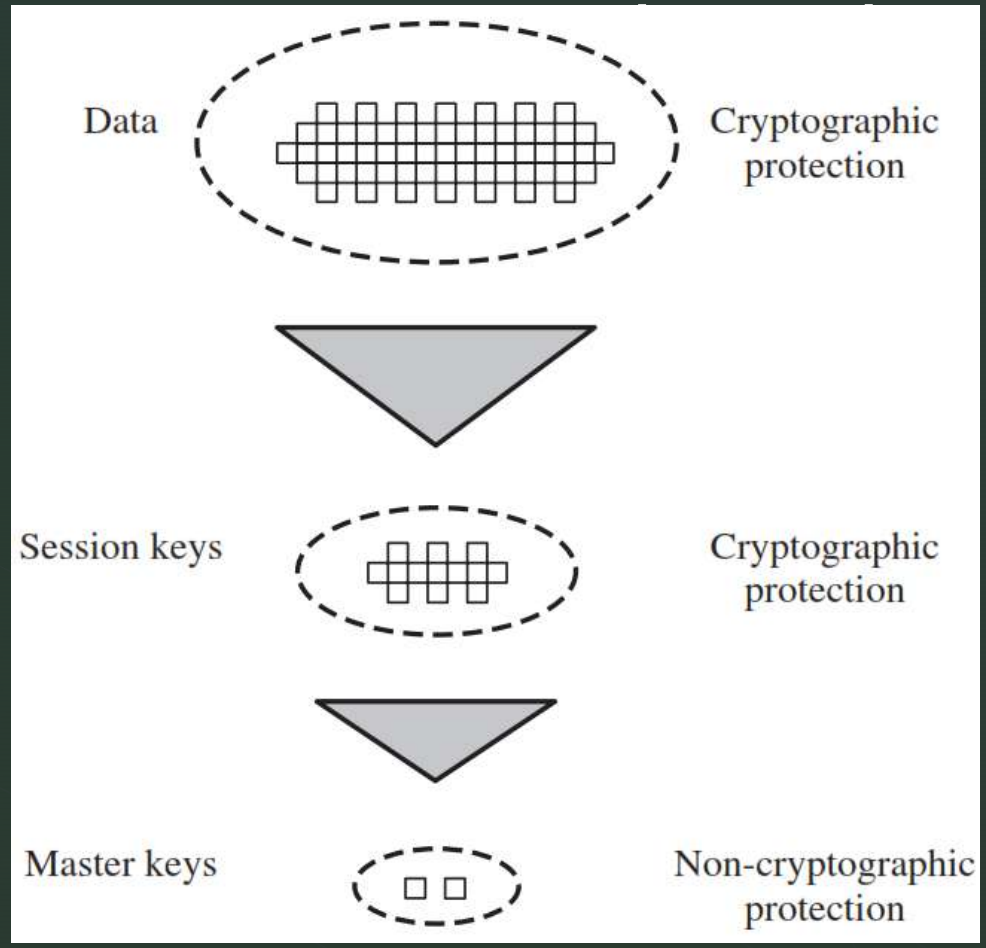
	SHA - 1	SHA - 224	SHA - 256	SHA - 384	SHA - 512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Key Management

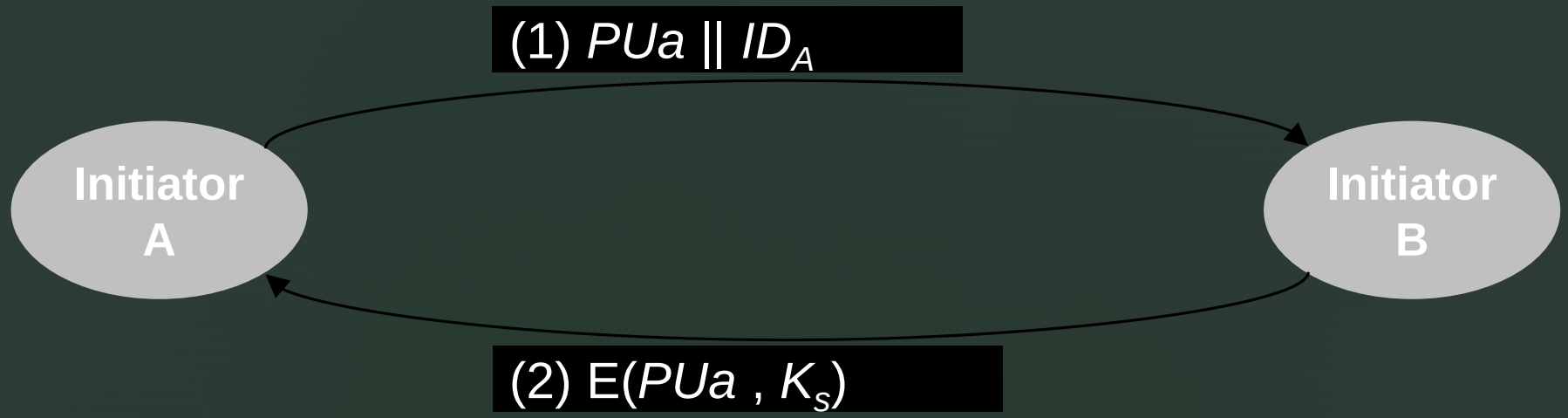
Key Distribution

- **Key distribution** is the function that delivers a key to two parties who wish to exchange secure encrypted data.
- Some sort of mechanism or protocol is needed to provide for the secure distribution of keys.
- Key distribution often involves the use of **master keys**, which are infrequently used and are long lasting, and **session keys**, which are generated and distributed for temporary use between two parties.

- Communication between end systems is encrypted using a temporary key, often referred to as a **session key**.
- Session keys are transmitted in encrypted form, using a **master key** that is shared by the key distribution center and an end system or user

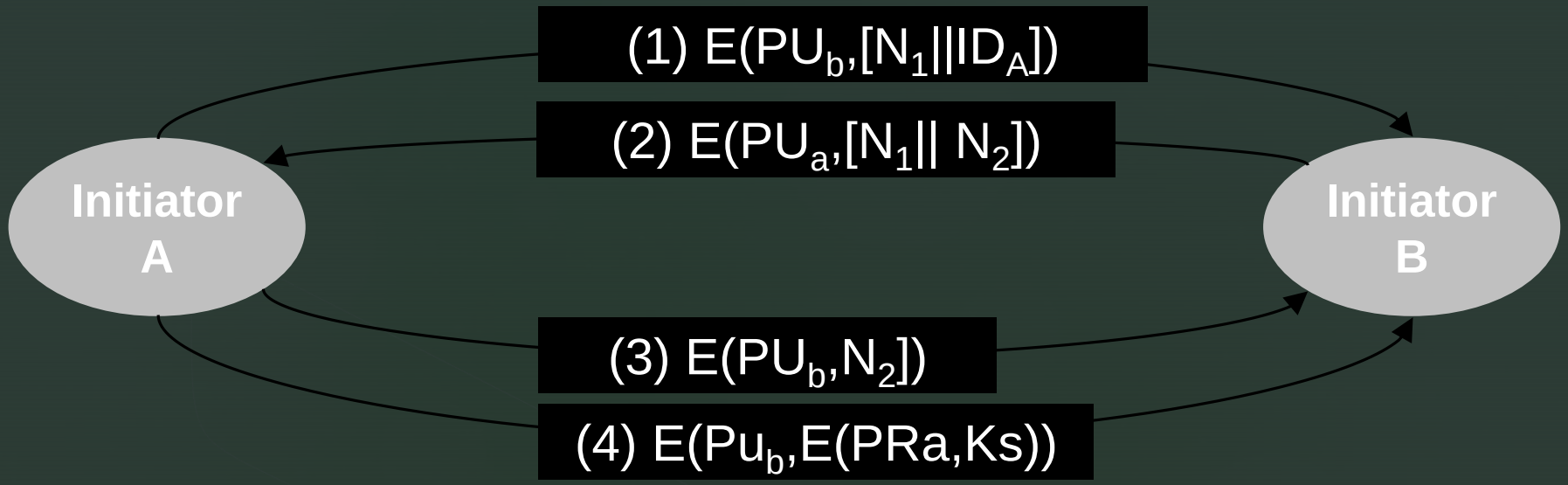


Simple Secret Key Distribution



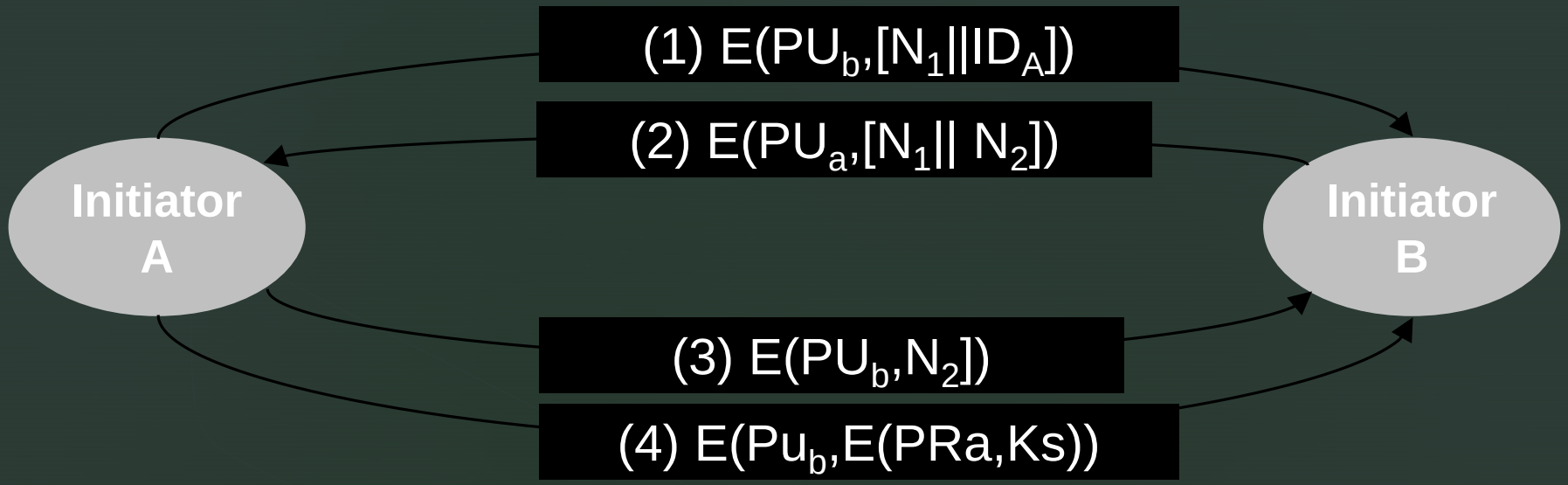
1. **A** generates a public/private key pair **{PUa, PRa}** and transmits a message to **B** consisting of **PUa** and an identifier of **A**, **ID_A**.
2. **B** generates a secret key, **Ks**, and transmits it to **A**, encrypted with **A**'s public key.
3. **A** computes **D(PRa, E(PUa, Ks))** to recover the secret key. Because only **A** can decrypt the message, only **A** and **B** will know the identity of **Ks**.
4. **A** discards **PUa** and **PRa** and **B** discards **PUa**.

Secret Key Distribution with Confidentiality & Authentication



1. **A** uses **B**'s public key to encrypt a message to **B** containing an identifier of **A** (**I_A**) and a nonce (**N_1**), which is used to identify this transaction uniquely.
2. **B** sends a message to **A** encrypted with **PUa** and containing **A**'s (**N_1**) as well as a new nonce generated by **B** (**N_2**). Because only **B** could have decrypted message (1), the presence of **N_1** in message (2) assures **A** that the correspondent is **B**.

Secret Key Distribution with Confidentiality & Authentication

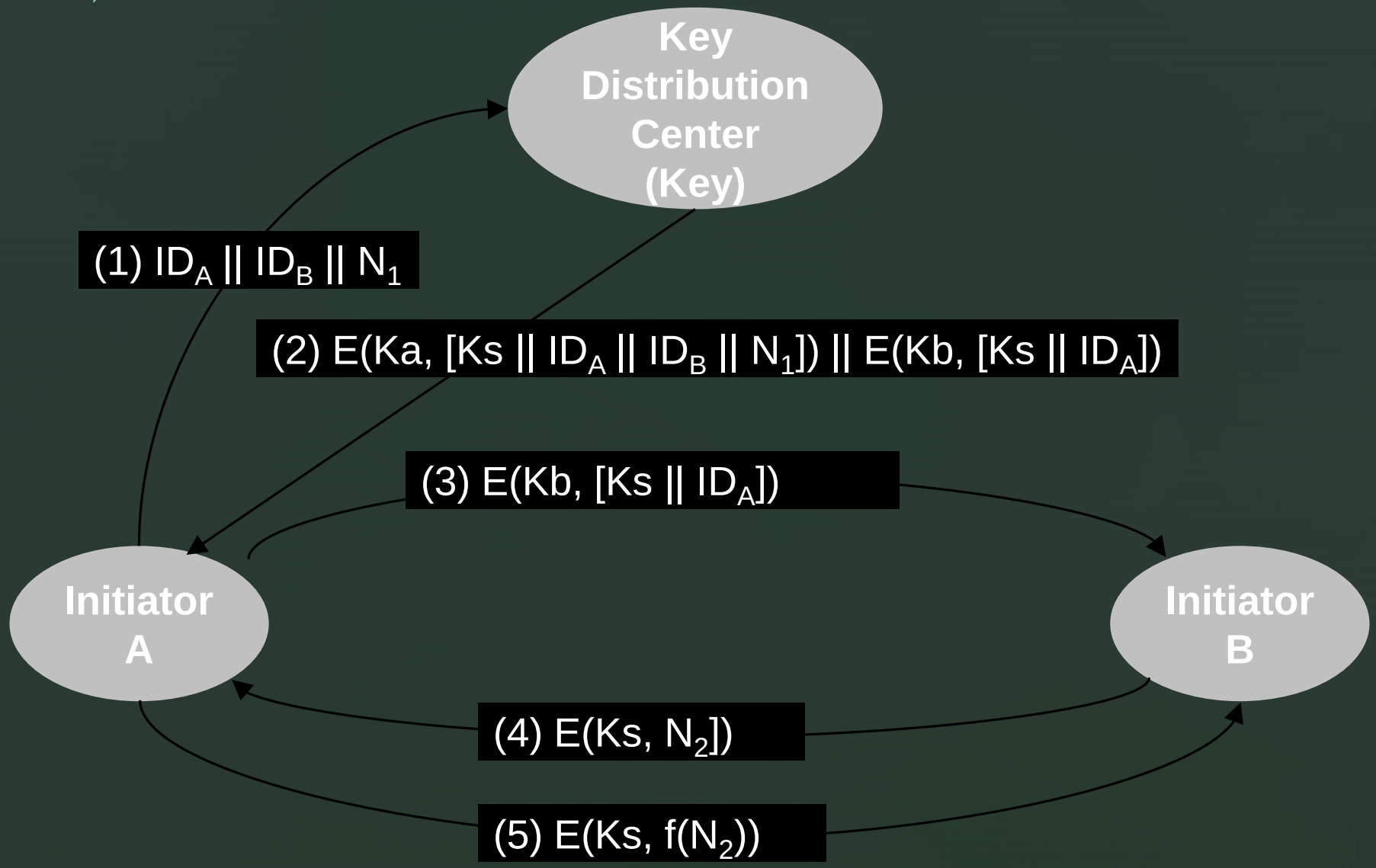


3. **A** returns N_2, encrypted using **B**'s public key, to assure **B** that its correspondent is **A**.
4. **A** selects a secret key **Ks** and sends **M = E(PUb, E(PRa, Ks))** to **B**. Encryption with **B**'s public key ensures that only **B** can read it; encryption with **A**'s private key ensures that only **A** could have sent it.
5. **B** computes **D(PUa, D(PRb, M))** to recover the secret key.

Symmetric key distribution using symmetric encryption

- Two parties A and B, key distribution can be achieved in a number of ways, as follows:
  1. **A** can select a key and **physically deliver key** to **B**.
  2. Third party can select the key and physically deliver it to **A** and **B**.
  3. If **A** and **B** have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
  4. If **A** and **B** each has an encrypted connection to a third party **C**, **C** can deliver a key on the encrypted links to A and B.

Key Distribution Scenario



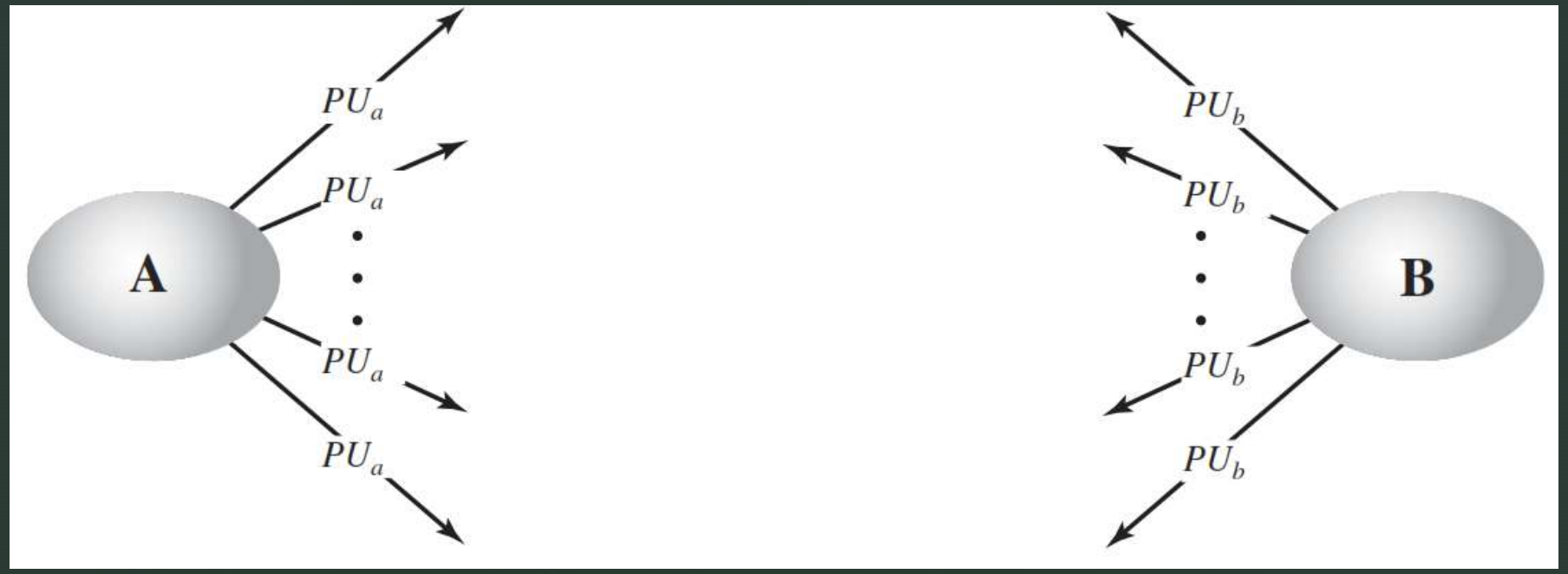
Key Distribution Scenario

1. **A** requests from the KDC a session key to protect a logical connection to **B**. The message includes the identity of **A** and **B** and a unique nonce **N1**.
2. The KDC responds with a message encrypted using **Ka** that includes a one-time session key **Ks** to be used for the session, the original request message to enable **A** to match response with appropriate request, and info for **B**
3. **A** stores the session key for use in the upcoming session and forwards to **B** the information from the KDC for **B**, namely, **E(Kb, [Ks || IDA])**.
4. At this point, a session key has been securely delivered to **A** and **B**, and they may begin their protected exchange.
5. Using the new session key for encryption B sends a nonce $\mathbf{N_2}$ to **A**.
6. Also using **Ks**, **A** responds with $\mathbf{f(N_2)}$. These steps assure **B** that the original message it received (step 3) was not a replay. Note that the actual key distribution involves only steps 1 through 3 but that steps 4 and 5, as well as 3, perform an authentication function.

Distribution of Public Keys

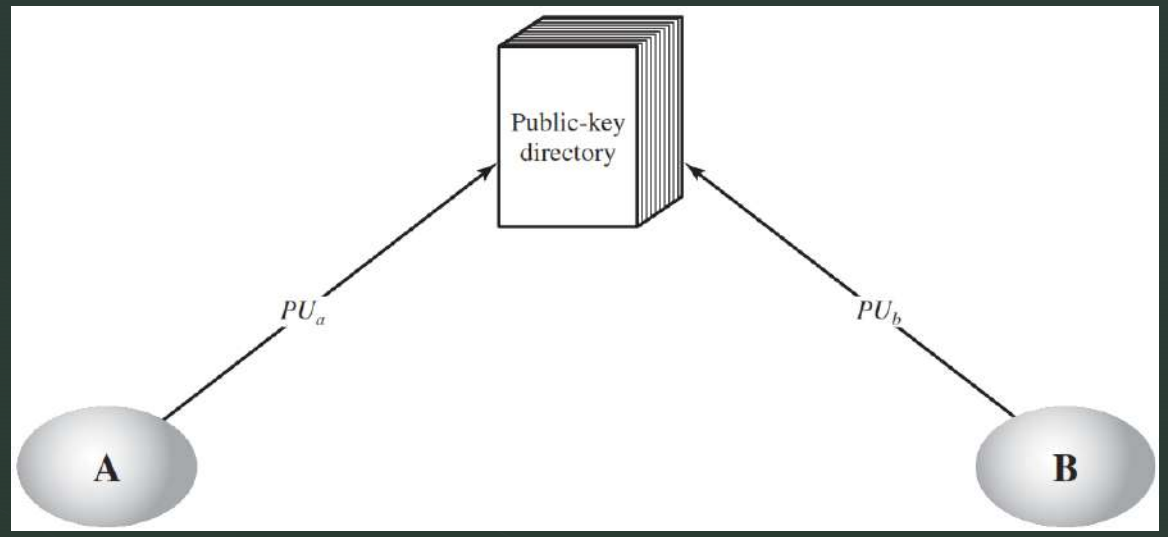
1. Public announcement
2. Publicly available directory
3. Public-key authority
4. Public-key certificates

1. Public Announcement



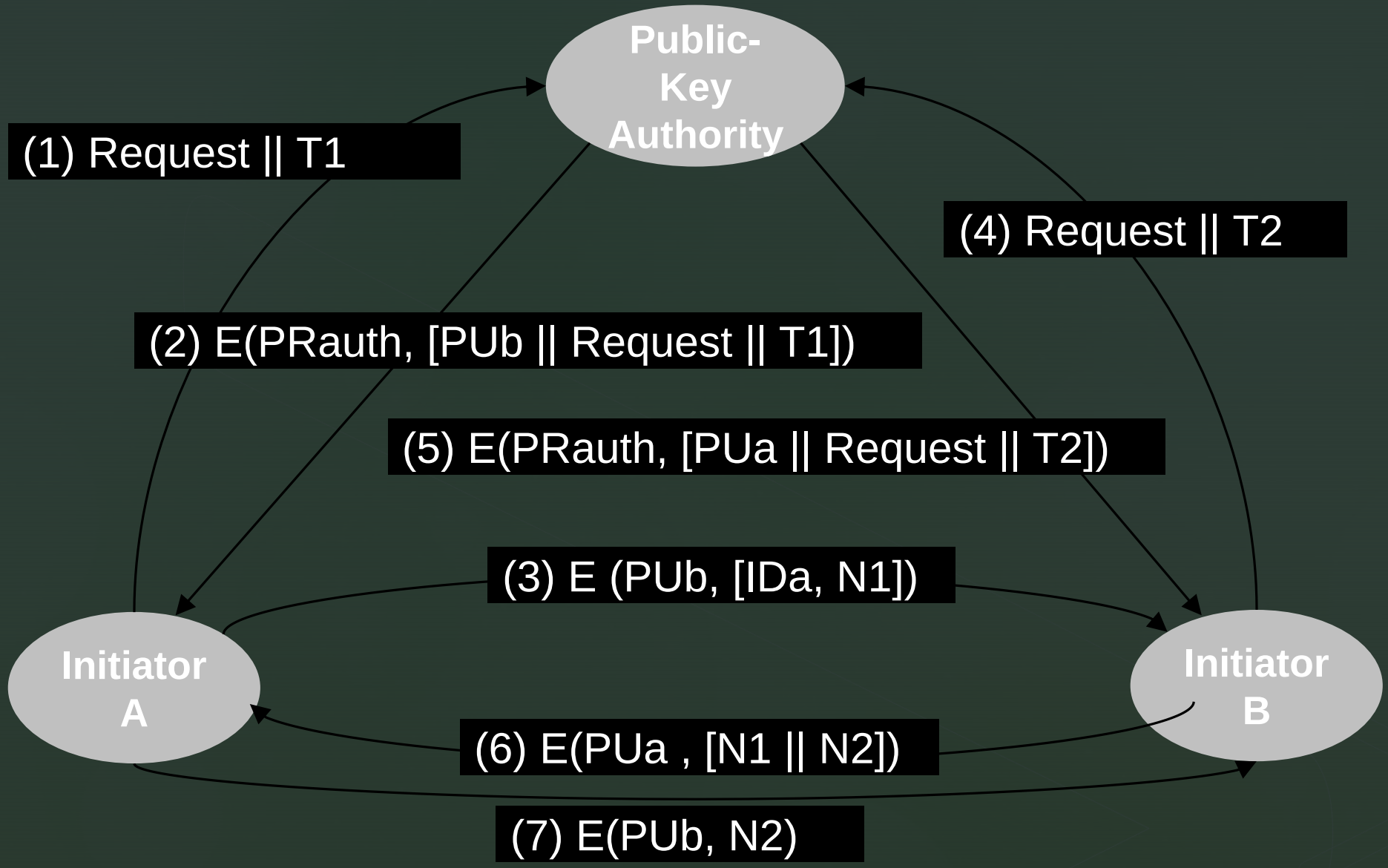
- Some user could pretend to be user A and send a public key to another participant or broadcast such a public key.
- Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication

2. Publicly Available Directory



1. The **authority** maintains a directory with a **{name, public key}** entry for each participant.
2. Each participant registers a public key with the directory authority.
3. A participant may replace the existing key with a new one at any time.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

3. Public-Key Authority



3. Public-Key Authority – Cont…

1. **A** sends a timestamped message to the public-key authority containing a request for the current public key of **B**.
2. The authority responds with a message that is encrypted using the authority's private key .
3. Message contains **PUb**, Original request, Original time stamp **T1**
4. **A** stores **B**'s public key and also uses it to encrypt a message to B containing an identifier of **A(IDa)** and a **nonce(N1)** , which is used to identify this transaction uniquely.
5. **B** retrieves **A**'s public key from the authority in the same manner as **A** retrieved **B**'s public key.

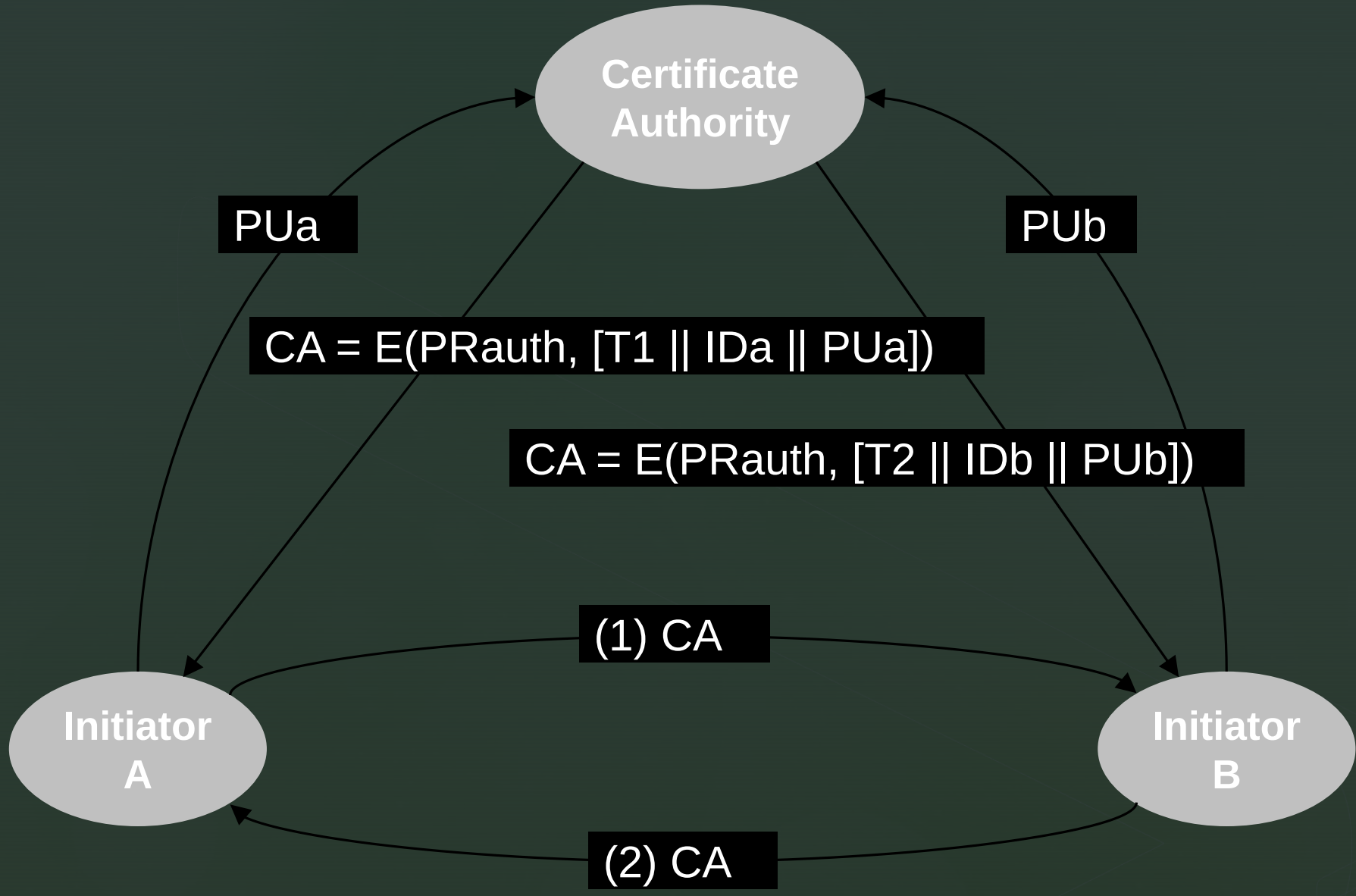
3. Public-Key Authority – Cont…

6. **B** sends a message to **A** encrypted with **PUa** and containing **A**'s **nonce(N1)** as well as a new nonce generated by **B(N2)**. Because only **B** could have decrypted message (3), the presence of **N1** in message (6) assures **A** that the correspondent is **B**.
7. **A** returns **N2**, which is encrypted using **B**'s public key, to assure **B** that its correspondent is **A**.

4. Public-Key Certificates

- Any participant can read a **certificate** to determine the name and public key of the certificate's owner.
- Any participant can verify that the **certificate** originated from the **certificate authority** and is not counterfeit.
- Only the **certificate authority** can create and update certificates.
- Any participant can verify the currency of the certificate.

4. Public-Key Certificates – Cont…



4. Public-Key Certificates – Cont…

- Each participant applies to the **certificate authority**, supplying a public key and requesting a certificate.
- For participant A, the authority provides a certificate of the form

$$CA = E(PRauth, [T \| IDa \| PUa])$$

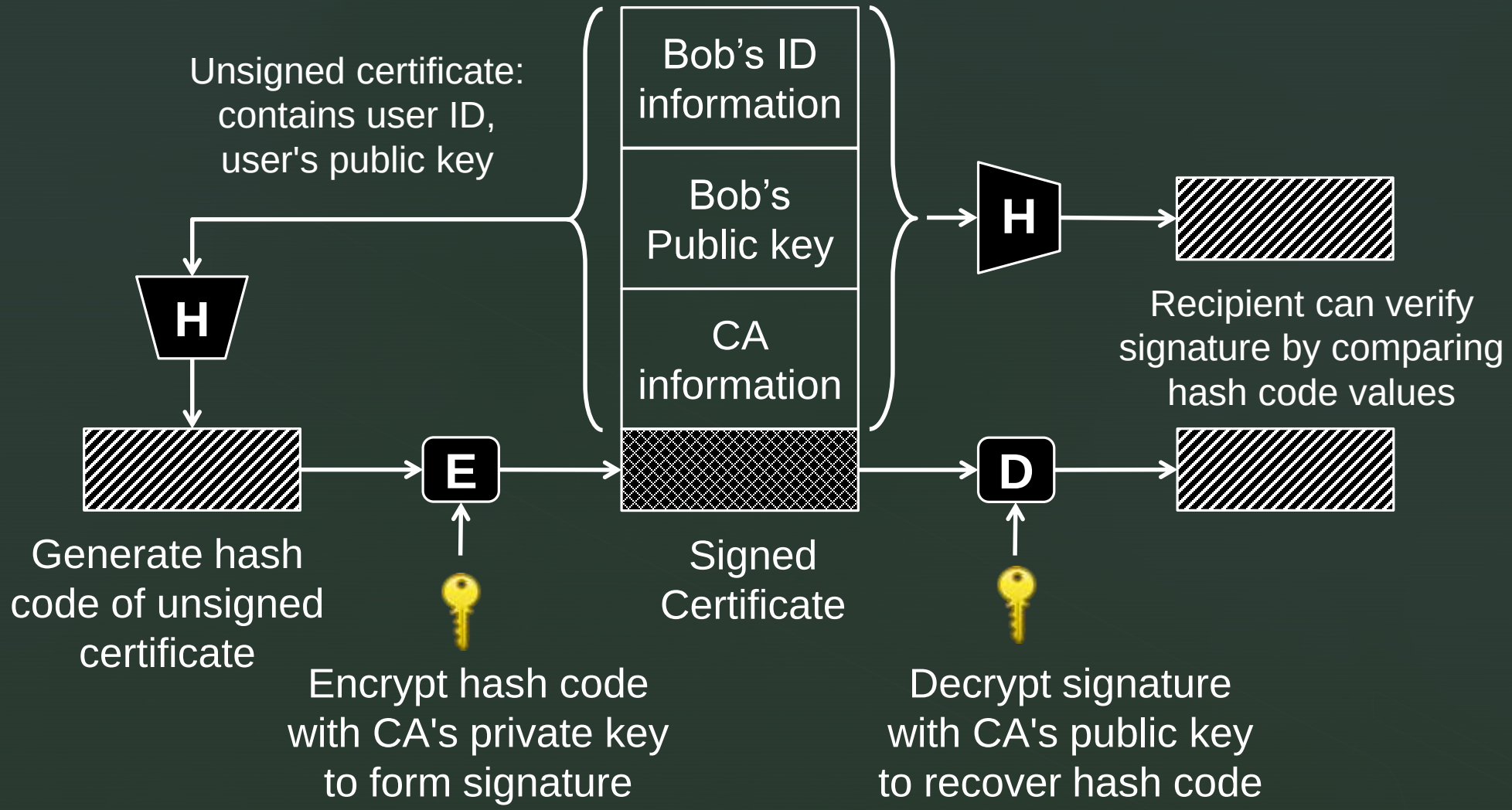
- A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PUauth, CA)$$

$$= D(PUauth, E(PRauth, [T \| IDa \| PUa]))$$

$$= (T \| IDa \| PUa)$$

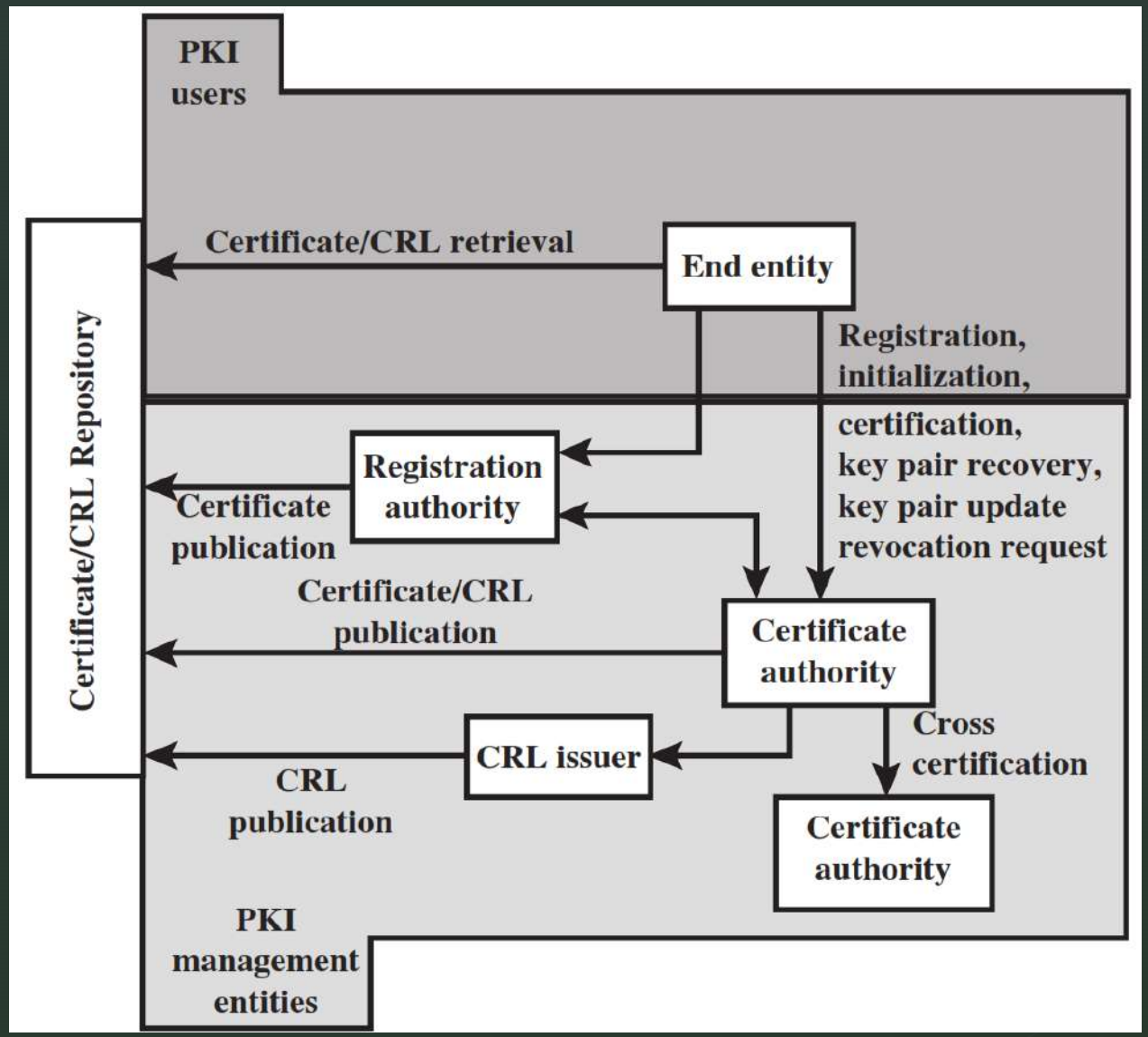
Public-Key Certificate Use



Public key Infrastructure (PKI)

- A **public-key infrastructure (PKI)** is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.
- The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of **public keys**.

Public key Infrastructure (PKI)



Public key Infrastructure (PKI) – Cont…

- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public-key certificate.
- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs).
- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

Thank You