Contemporary Cryptography：Principles and Practice

7. Application of Symmetric Encryption

Fuyou Miao，Wenchao Huang

Web：http://staff.ustc.edu.cn/~huangwc/crypto.html

Email: mfy@ustc.edu.cn，huangwc@ustc.edu.cn

Key Points

- **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 (主密钥)**, which are infrequently used and are **long lasting**
    - **session keys (会话密钥)**, which are generated and distributed for temporary use between two parties.
- A capability with application to a number of cryptographic functions is **random or pseudorandom number generation (随机数/伪随机数发生器)**
    - The principle requirement for this capability is that the generated number **stream** be **unpredictable**

Outline

1. Key distribution
    - A Hierarchy of Keys
    - A Key Distribution Scenario
    - Session Key Lifetime
    - A Transparent Key Control Scheme
    - Decentralized Key Control
    - Controlling Key Usage
2. Pseudorandom number generation

1. Key distribution

- Question: How to exchange the symmetric key for two parties?
    - The key must be **protected** from access by others
    - **Frequent key changes** are usually desirable to limit the amount of data compromised if an attacker learns the key
- Answer: **Key distribution** technique

1. Key distribution

- Possible Solutions of key distribution for A and B

1. A can select a key and physically deliver it to B.

2. A 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.

1. Key distribution

- Possible Solutions of key distribution for A and B

    1. A can select a key and physically deliver it to B.

    2. A third party can select the key and physically deliver it to A and B.

- Observation

    - Suited for **link encryption**

    - But **not** for **end-to-end** encryption

1. Key distribution

- If there are N hosts,
  - the number of required keys is $[N(N-1)]/2$
- 1000 nodes
  - 50,000 keys!

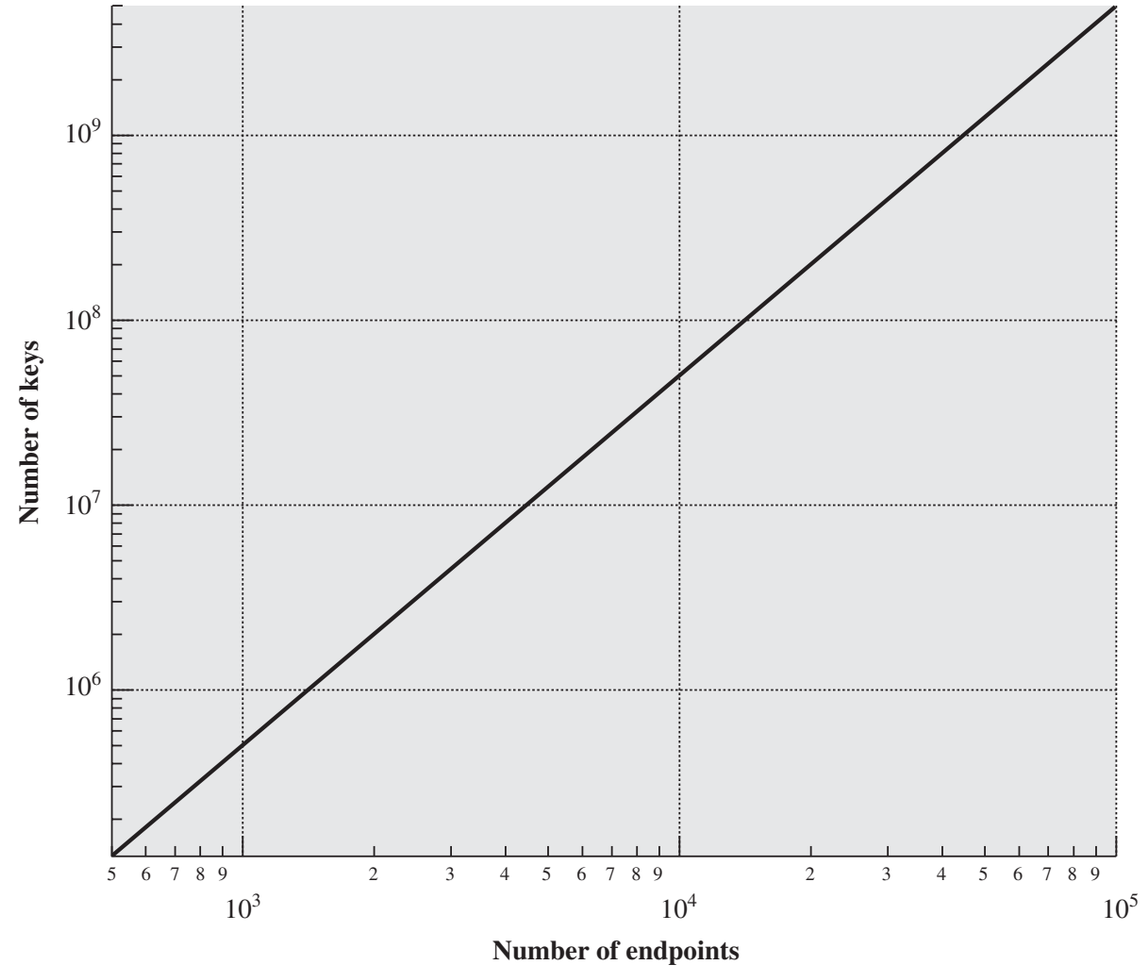


Figure 14.1 Number of Keys Required to Support Arbitrary Connections between Endpoints

1. Key distribution

- Possible Solutions of key distribution for 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.
- Observation
    - Possibility for either link encryption or end-to-end encryption
    - Problems
        - If an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed
        - The initial distribution of potentially millions of keys still must be made

1. Key distribution

- Possible Solutions of key distribution for A and B
  - 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.
- Observation
  - For end-to-end encryption, some variation on option 4 has been widely adopted
  - A **key distribution center** is responsible for distributing keys to pairs of users

1. Key distribution
(1) A hierarchy of keys

- Communication between <u>end systems</u> 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 <u>key distribution center</u> and an <u>end system</u> or user.

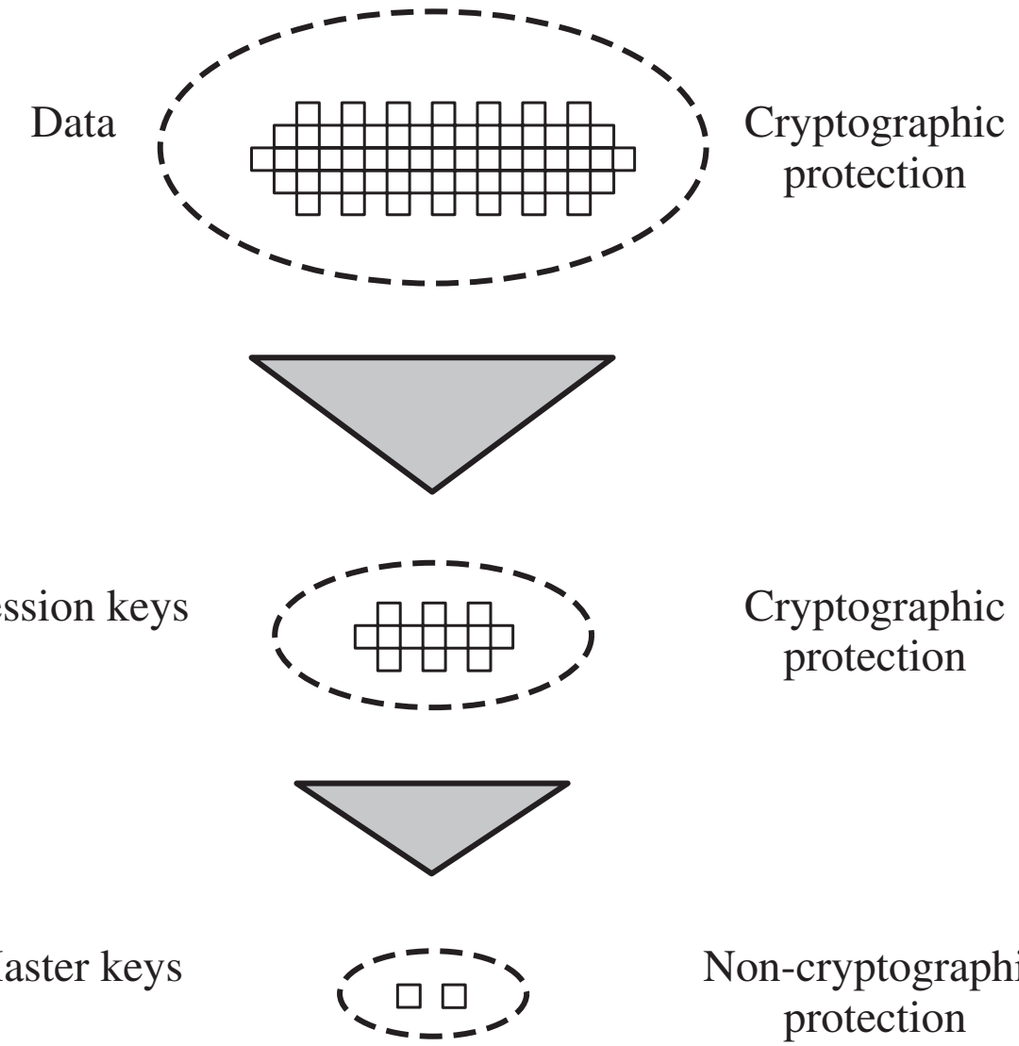


Figure 14.2 The Use of a Key Hierarchy

1. Key distribution (2) A Key Distribution Scenario

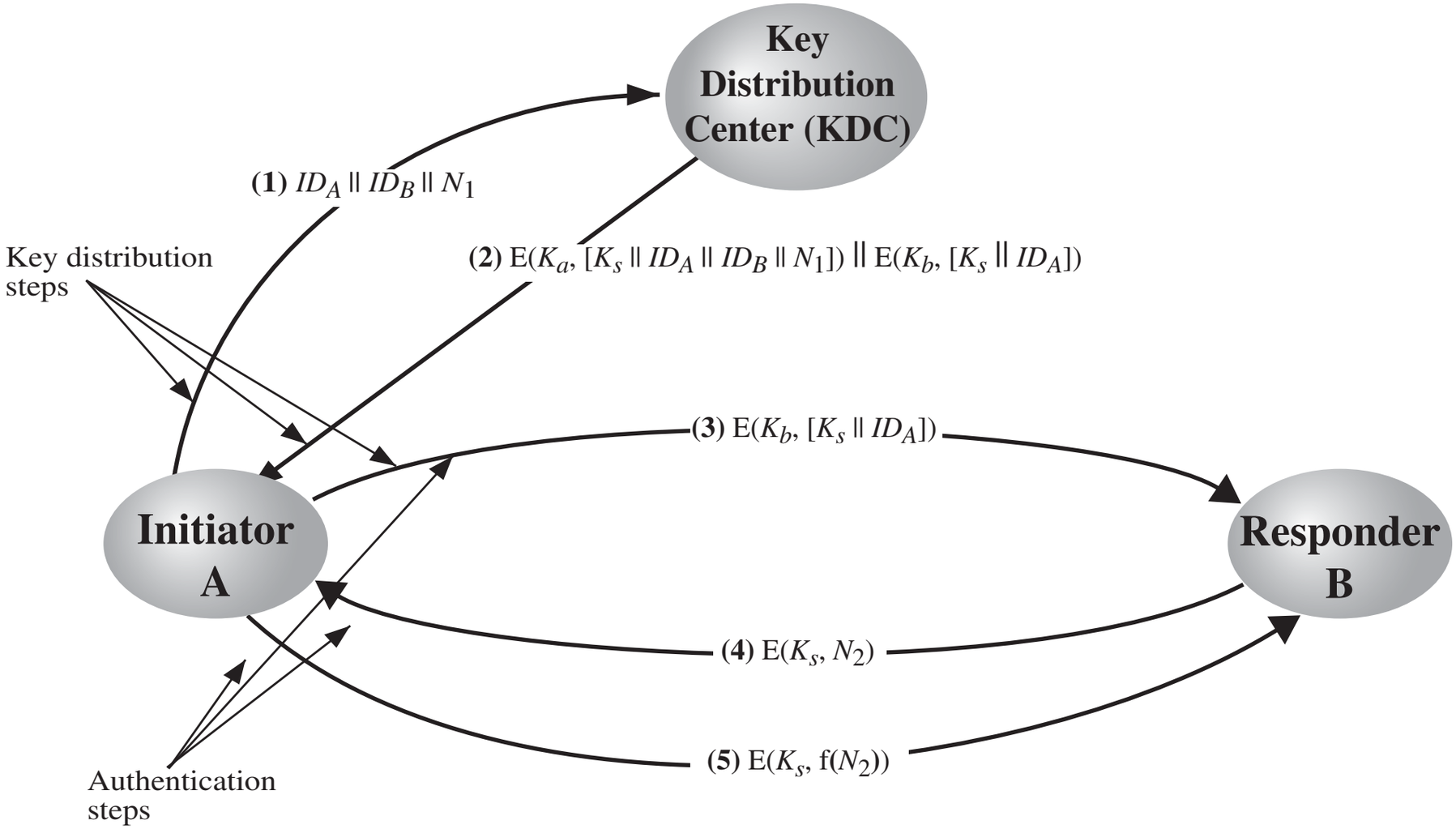


Figure 14.3 Key Distribution Scenario

- (1) A issues a request to the KDC for a session key to protect a logical connection to B
  - N_1 is a nonce

1. Key distribution (2) A Key Distribution Scenario

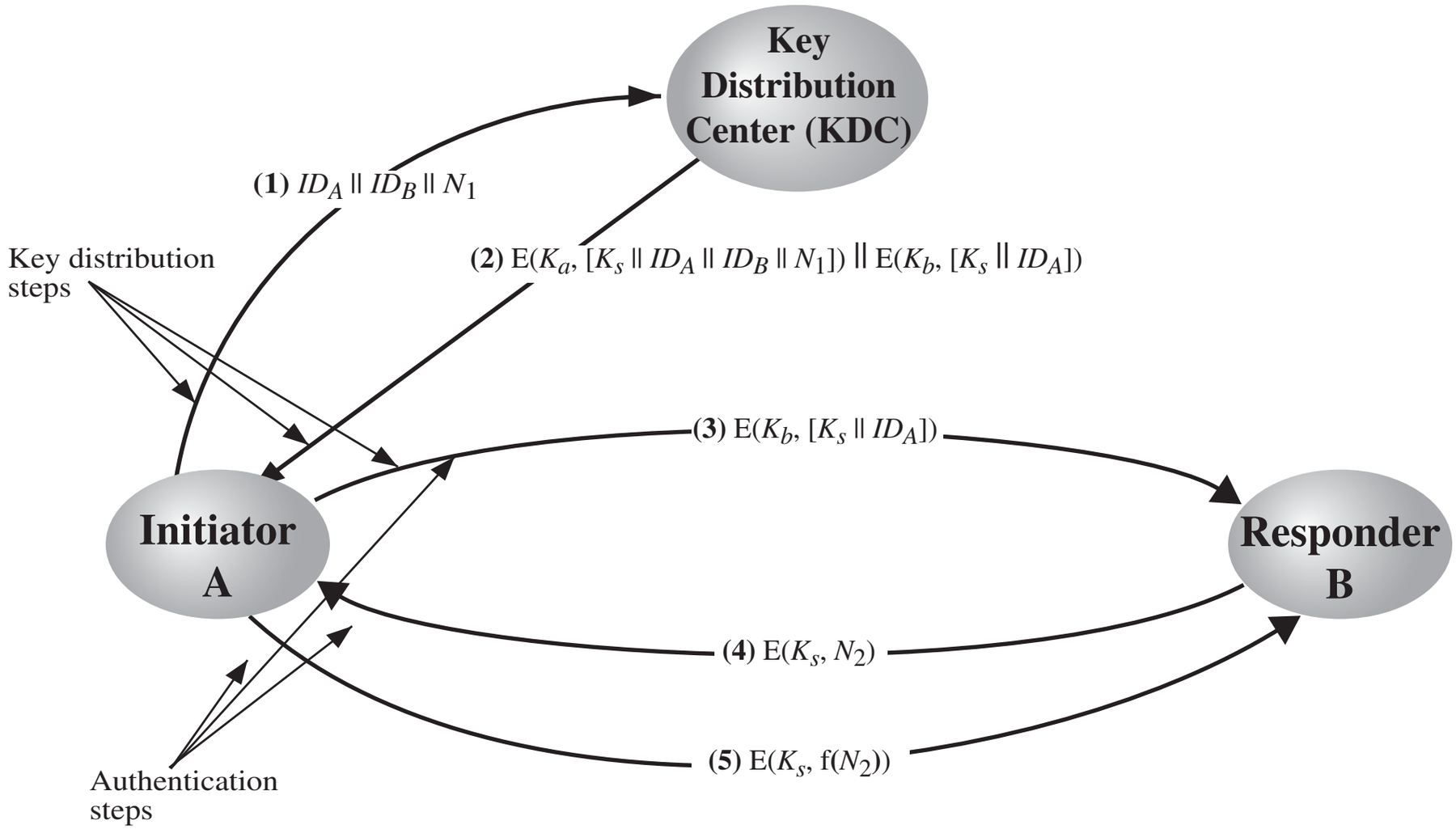


Figure 14.3 Key Distribution Scenario

- (2) The KDC responds with a message encrypted using K_a. For A
  - A can verify that its original request was not altered before reception by the KDC
  - Because of the nonce, that this is not a replay of some previous request

1. Key distribution (2) A Key Distribution Scenario

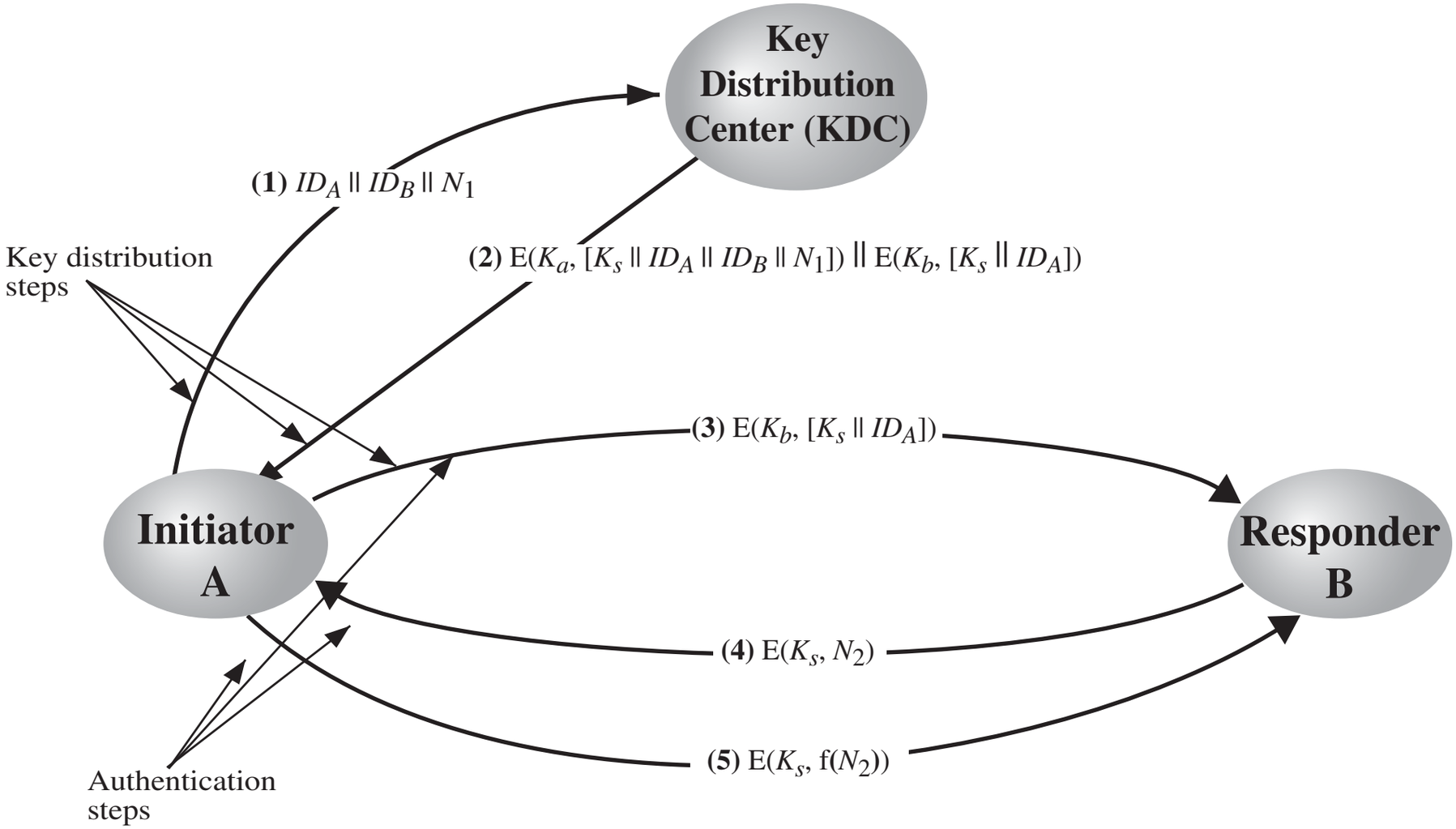


Figure 14.3 Key Distribution Scenario

- (2) The KDC responds with a message encrypted using K_a. For B
  - Establish the connection
  - Prove A's identity

1. Key distribution (2) A Key Distribution Scenario

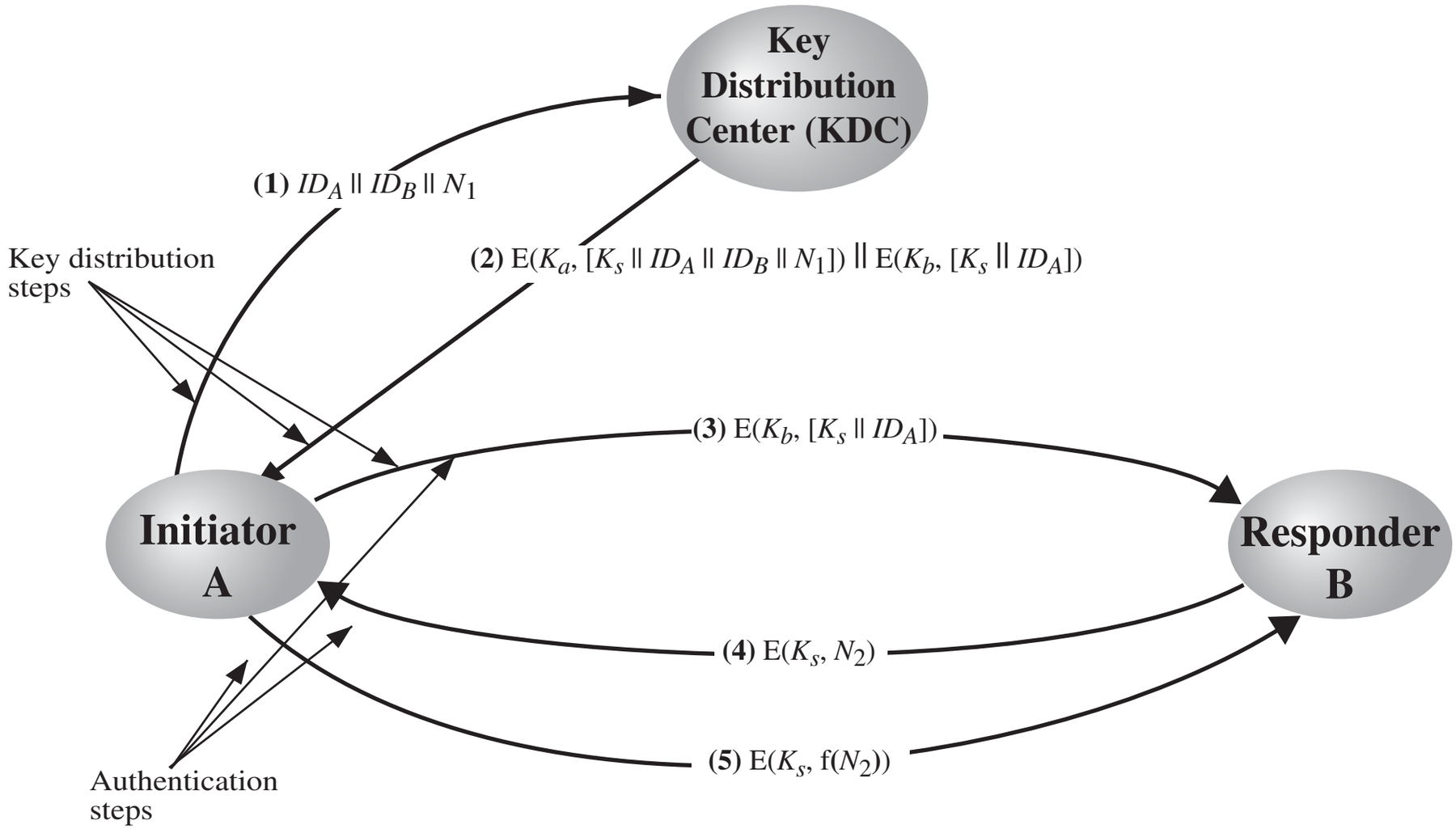


Figure 14.3 Key Distribution Scenario

- (3)
  - A stores the session key for use in the upcoming session and
  - A forwards to B the information that originated at the KDC for B.

1. Key distribution (2) A Key Distribution Scenario

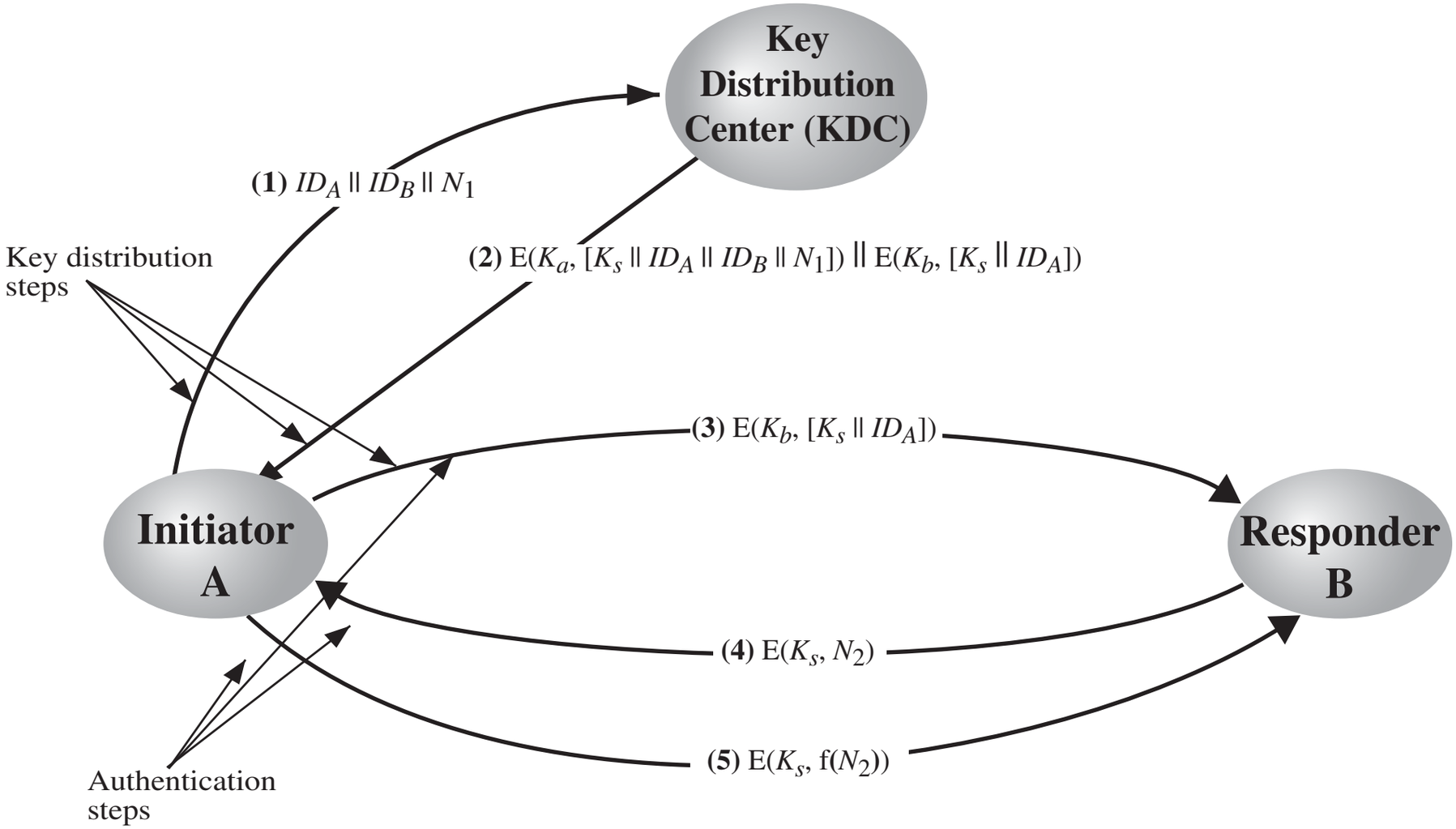


Figure 14.3 Key Distribution Scenario

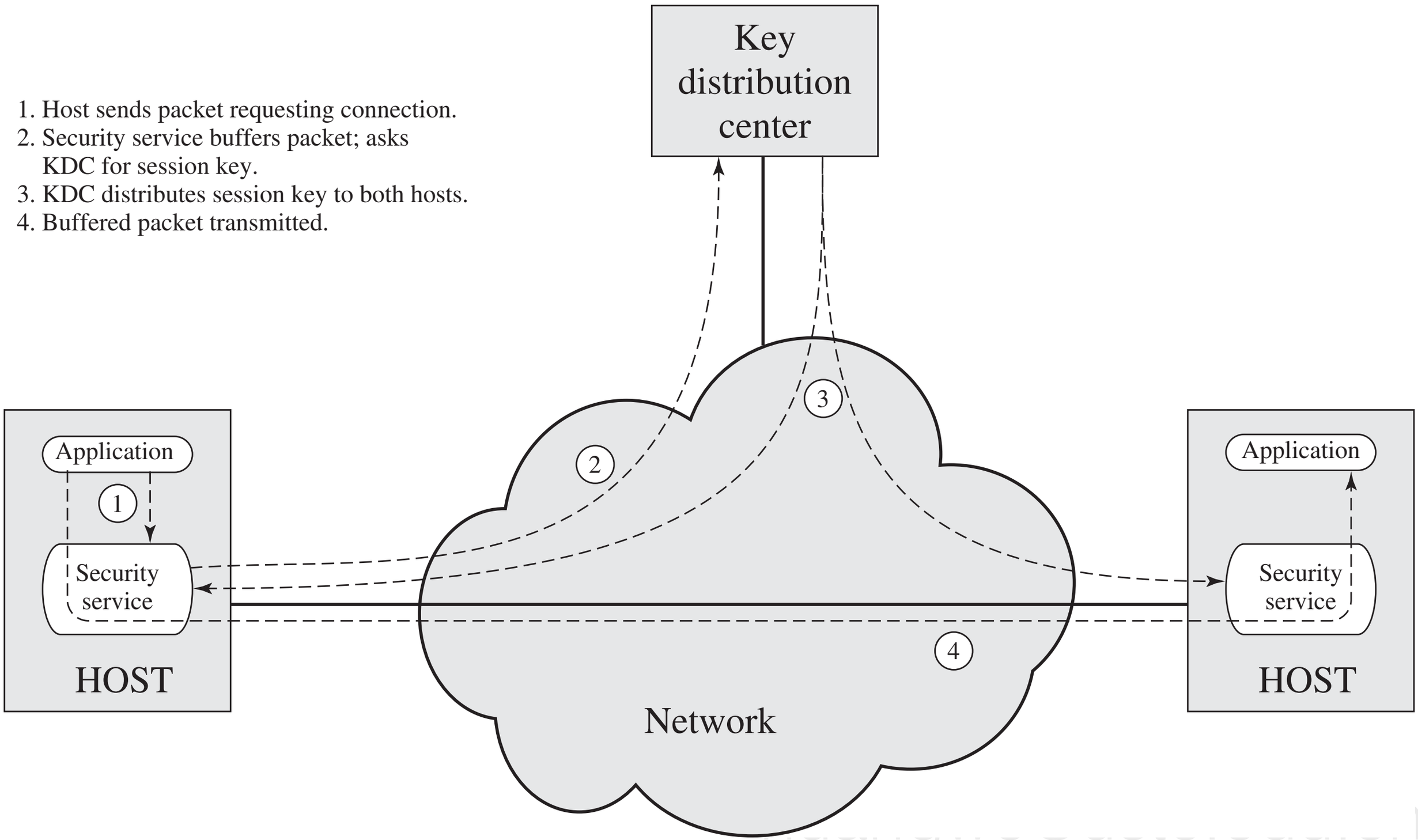
- (4)(5) assure B that the original message it received (step 3) was <u>not a replay</u>
  - Note: Authentication: (3), and (4,5)

1. Key distribution
(3) Session Key Lifetime

- The more frequently session keys are exchanged, the more secure they are
  - For connection-oriented protocols, one obvious 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
    - Change the session key periodically, for long lifetime connection
- For a connectionless protocol
  - The most secure approach is to use a new ses- sion key for each exchange
  - A better strategy is to use a given session key for a certain fixed period only or for a certain number of transaction

1. Key distribution
(4) A Transparent Key Control Scheme

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.



1. Key distribution
(5) Decentralized Key Control

- The key control can be **decentralized**

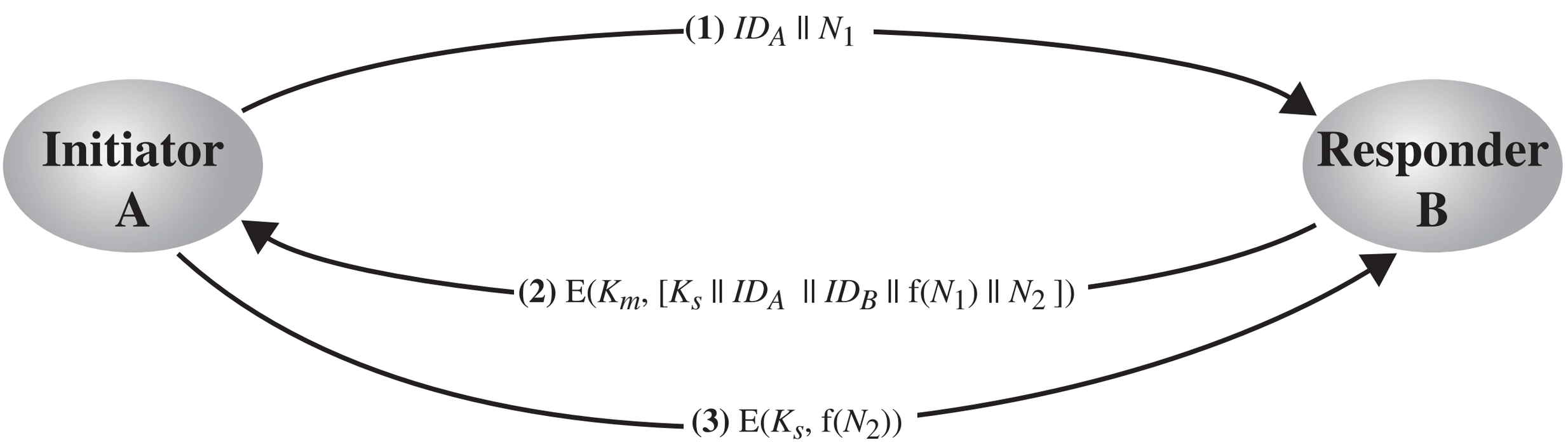


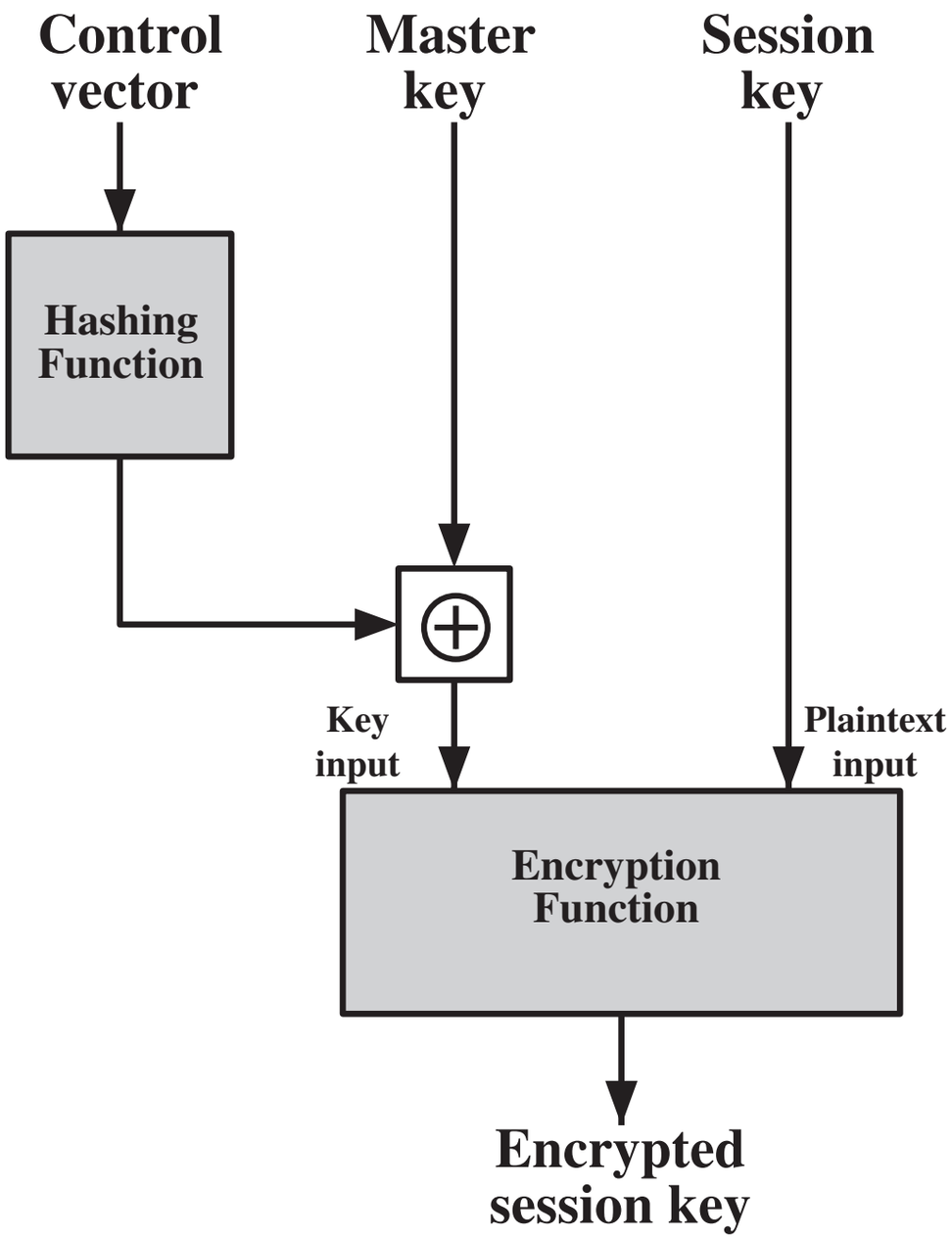
Figure 14.5 Decentralized Key Distribution

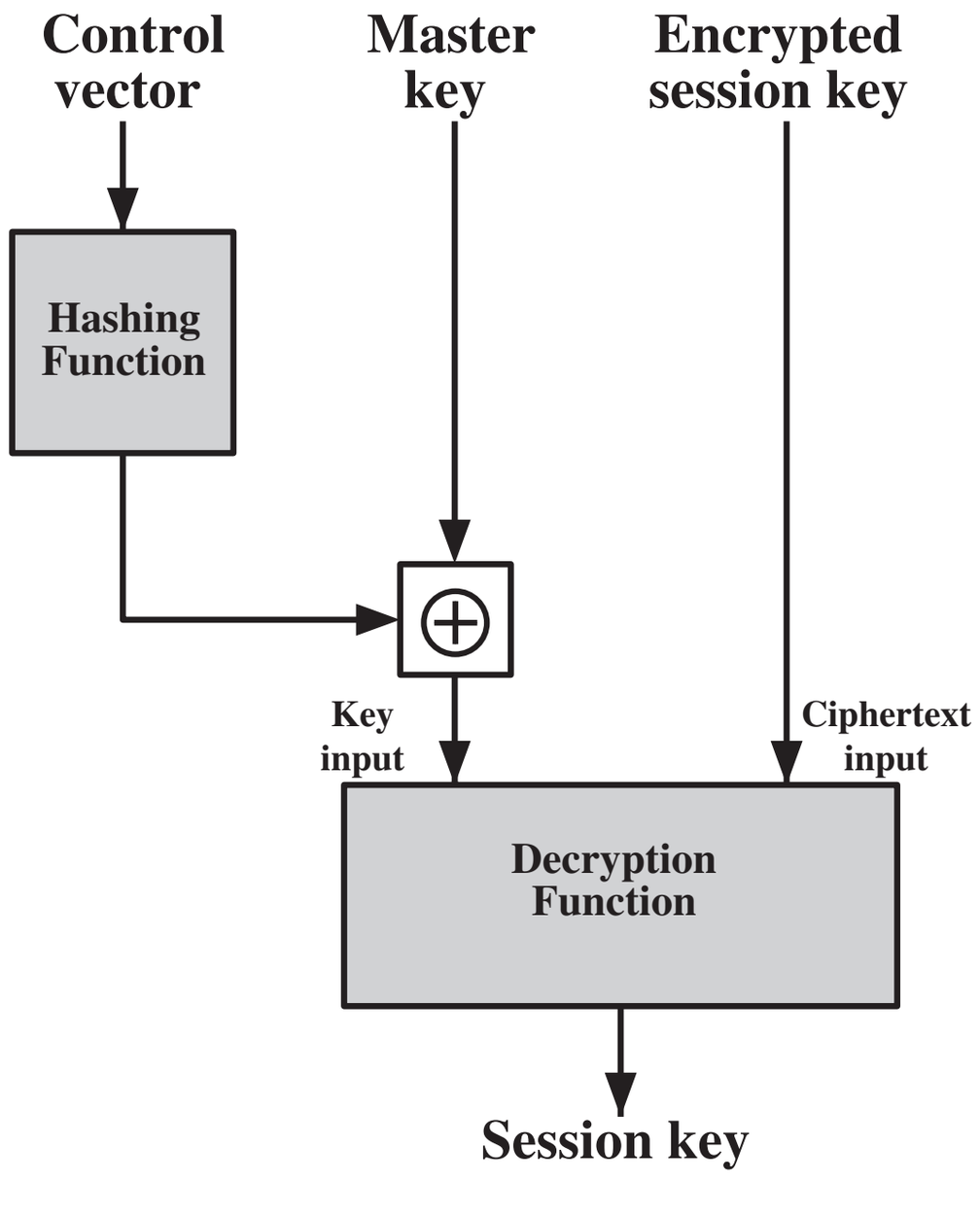
1. Key distribution
(6) Controlling Key Usage

- Different types of session keys on the basis of use
  - Data-encrypting key: for general communication across a network
  - PIN-encrypting key: for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
  - File-encrypting key: or encrypting files stored in publicly accessible locations
- The value of separating keys by type
  - E.g., misuse the master key

1. Key distribution

- A flexible scheme: Use of the control vector





2. Pseudorandom Number Generation

- **The Use** of Random Numbers

  - Key distribution and reciprocal authentication schemes, e.g., nonce

  - Session key generation

  - Generation of keys for the RSA public-key encryption algorithm

  - Generation of a bit stream for symmetric stream encryption

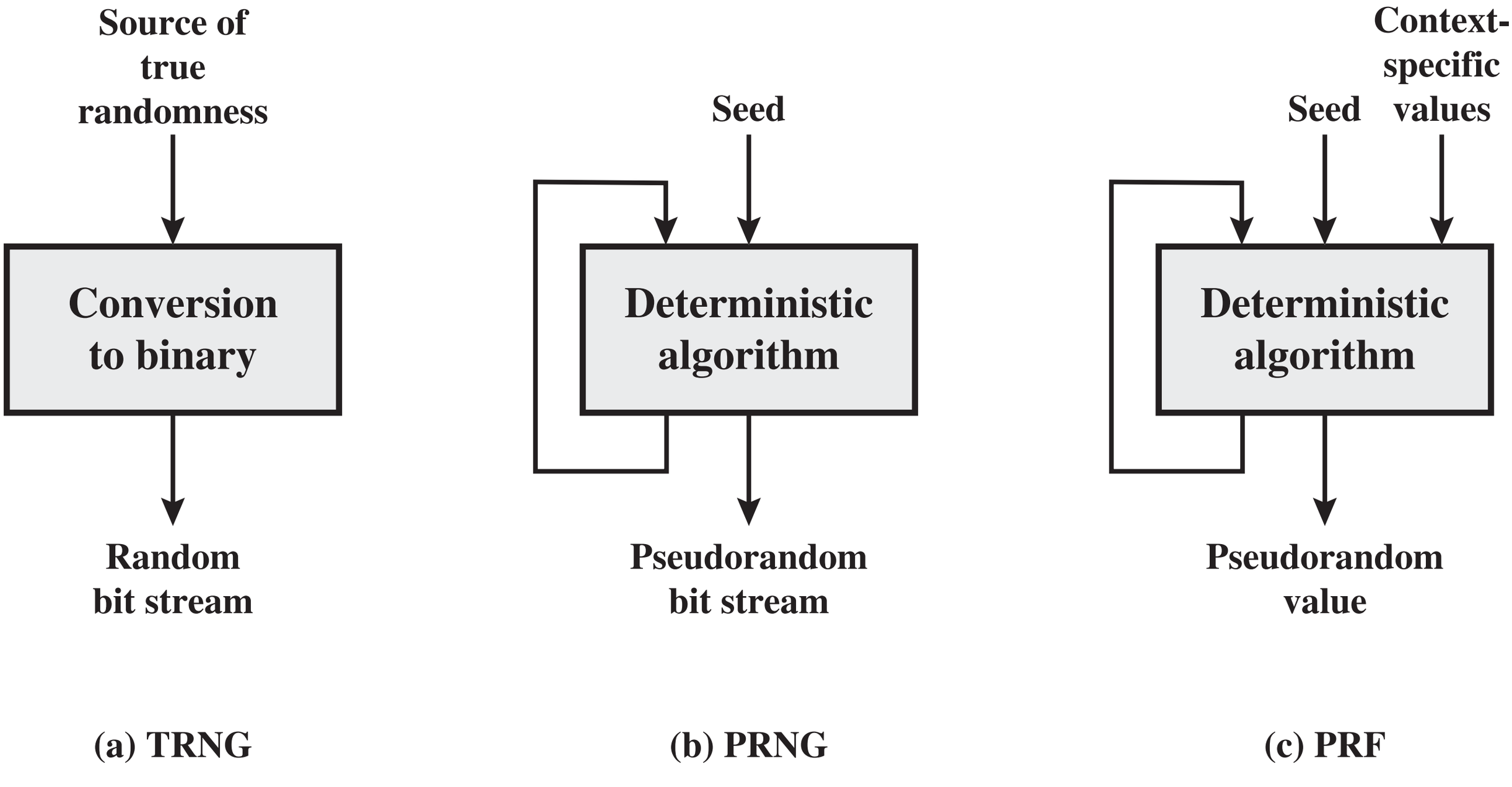
2. Pseudorandom Number Generation

- **Requirements**
  - **Randomness**
    - Uniform distribution: the frequency of occurrence of ones and zeros should be approximately equal.
    - Independence: no one subsequence in the sequence can be inferred from the others.
- **Unpredictability**
  - The successive members of the sequence are unpredictable

2. Pseudorandom Number Generation

- Techniques
  - TRNG = true random number generator
  - PRNG = pseudorandom number generator
  - PRF = pseudorandom function

2. Pseudorandom Number Generation



2. Pseudorandom Number Generation

- Randomness requirements
  - NIST SP 800-22 (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications
    - **Uniformity:** At any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely
    - **Scalability:** Any test applicable to a sequence can also be applied to subsequences extracted at random.
    - **Consistency:** The behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed or an TRNG on the basis of an output produced from a single physical output

2. Pseudorandom Number Generation

- Test of randomness (SP 800-22)

  - **Frequency test**: determine whether the number of ones and zeros in a sequence is approximately the same as would be expected for a truly random sequence

  - **Runs test:** The focus of this test is the total number of runs in the sequence, where a **run** is an uninterrupted sequence of identical bits bounded before and after with a bit of the opposite value

  - **Maurer's universal statistical test**: detect whether or not the sequence can be significantly compressed without loss of information.

2. Pseudorandom Number Generation

- Seed requirements (SP800-90)
  - The seed is generated by a TRNG
- Discussion
  - The seed that serves as input to the PRNG must be secure
  - TRNG may <u>not</u> be able to generate bits at a <u>sufficient</u> <u>rate</u>

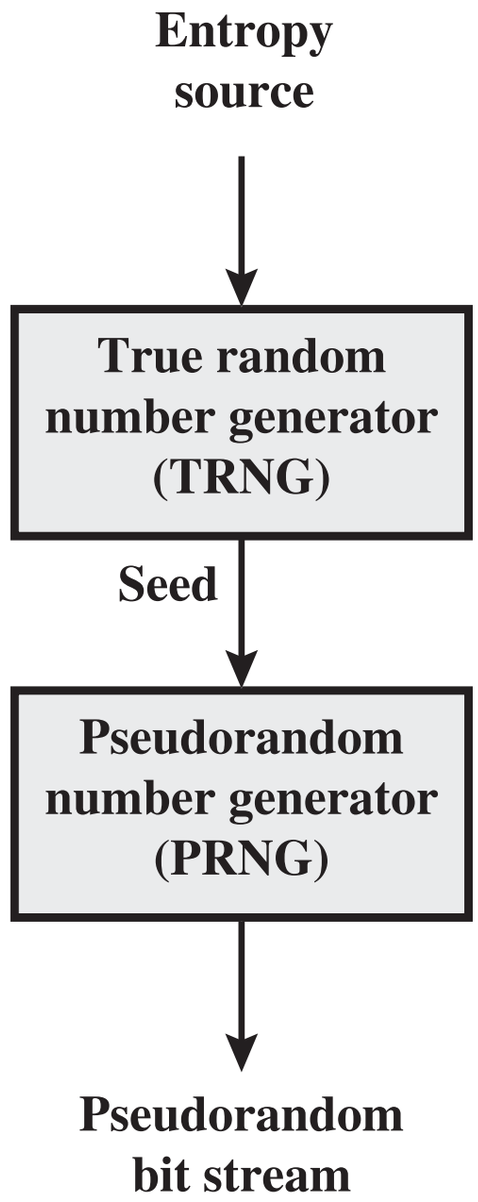


Figure 7.2 Generation of Seed Input to PRNG

2. Pseudorandom Number Generation

- Types of PRNGS
  - Algorithms for PRNGS
    - (1) Linear Congruential Generators
    - (2) Blum Blum Shub Generator
  - PRNG using a block cipher
    - (3) Using modes of operation
    - (4) ANSI X9.17 PRNG

2. Pseudorandom Number Generation
(1) Linear Congruential Generators

- Algorithm: Linear Congruential Generators (线性同余)
  - Parameters
    - m，modulus, 模数，m>0
    - a，multiplier, 乘数，0≤a≤m
    - c，increment, 增量，0≤c≤m
    - X_0，starting value, or seed, 初始值或种子，$0 \le X_0 \le m$
- Iterative equation：$X_{n+1} = (aX_n + c) \bmod m$

2. Pseudorandom Number Generation
(1) Linear Congruential Generators

- Algorithm: Linear Congruential Generator (线性同余)
  - Iterative equation： $X_{n+1} = (aX_n + c) \bmod m$
  - Try 1: a=7, c=0, m=32, X_0=1
    - generated sequence {7, 17, 23, 1, 7, …}
    - Obviously not satisfactory**: period=4**
- Try2: a=5, c=0, m=32, X_0=1
  - Generated sequence: {1, 5, 25, 29, 17, 21, 9, 13, 1, …}
  - not satisfactory either, **period=8**

2. Pseudorandom Number Generation
(1) Linear Congruential Generators

- Algorithm: Linear Congruential Generator (线性同余)
    - Three tests
        1. The function should be a **full-period** generating function.
            - That is, the function should generate **all the numbers** between 0 and m **before repeating**.
        2. The generated sequence should appear **random**.
        3. The function should implement efficiently with **32-bit arithmetic**.

2. Pseudorandom Number Generation (1) Linear Congruential Generators

- Algorithm: Linear Congruential Generator (线性同余)
  - Solution：
    - We would like *m* to be very large, so that there is the potential for producing a long series of distinct random numbers.
      - Try 3，a convenient prime value $m=2^{31}-1$，and
        - $X_{n+1} = (aX_n) \bmod (2^{31}-1)$
        - Only a handful of multipliers pass all three tests
- Pros:
  - If the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from a sequence drawn at random
- Discussion: it is desirable to make the actual sequence used nonreproducible, e.g, solution: **restart** the sequence after some time

2. Pseudorandom Number Generation (2) Blum Blum Shub Generator

- Blum Blum Shub （BBS）发生器
  - Choose two large prime numbers p, q, satisfying

$$p \equiv q \equiv 3 (\mathrm{mod}\ 4)$$

  - Let n=pq, choose a random number s, satisfying gcd(s,n)=1
  - produces a sequence of bits B

$$\begin{aligned} X_0 &= s^2 \bmod n \\ \textbf{for}\, i &= 1 \ \textbf{to}\ \infty \\ X_i &= (X_{i-1})^2 \bmod n \\ B_i &= X_i \bmod 2 \end{aligned}$$

2. Pseudorandom Number Generation
(2) Blum Blum Shub Generator

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

$$p \equiv q \equiv 3 (\text{mod } 4)$$

$X_0 = s^2 \bmod n$

for $i = 1$ **to** ∞

$X_i = (X_{i-1})^2 \bmod n$

$B_i = X_i \bmod 2$

2. Pseudorandom Number Generation
(2) Blum Blum Shub Generator

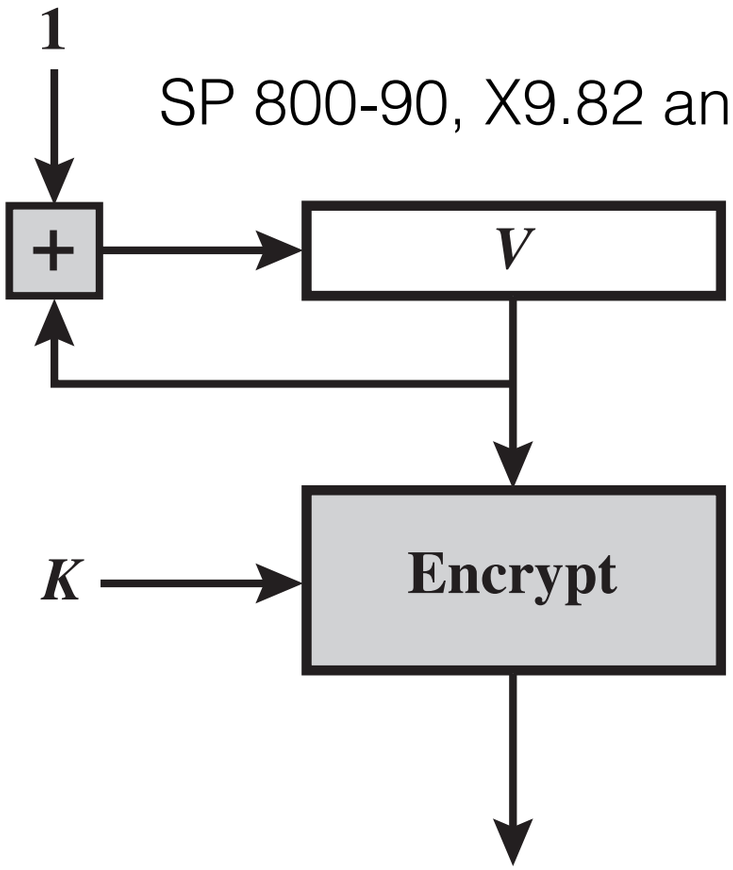
$$p \equiv q \equiv 3 (\mathrm{mod}\ 4)$$

$$X_0 = s^2 \bmod n$$
$$\textbf{for}\ i = 1\ \textbf{to}\ \infty$$
$$X_i = (X_{i-1})^2 \bmod n$$
$$B_i = X_i \bmod 2$$

- Properties
  - based on the difficulty of factoring *n*
  - said to pass the **next-bit test**
    - given the first k bits of the sequence, there is not a practical algorithm that can even allow you to state that the next bit will be 1 (or 0) with probability greater than 1/2

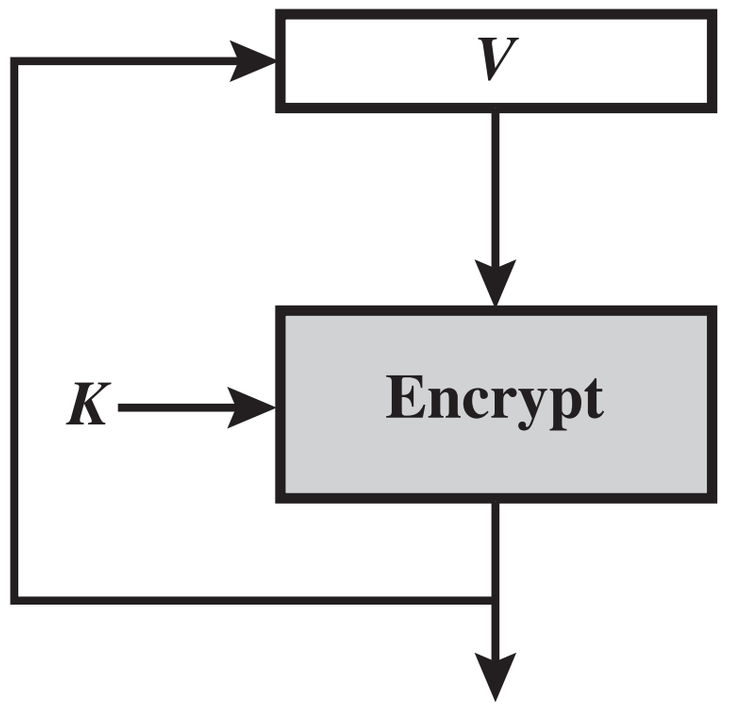
2. Pseudorandom Number Generation
(3) Using modes of operation

- CTR, OFB mode



SP 800-90, X9.82 and RFC 4086

(a) CTR mode

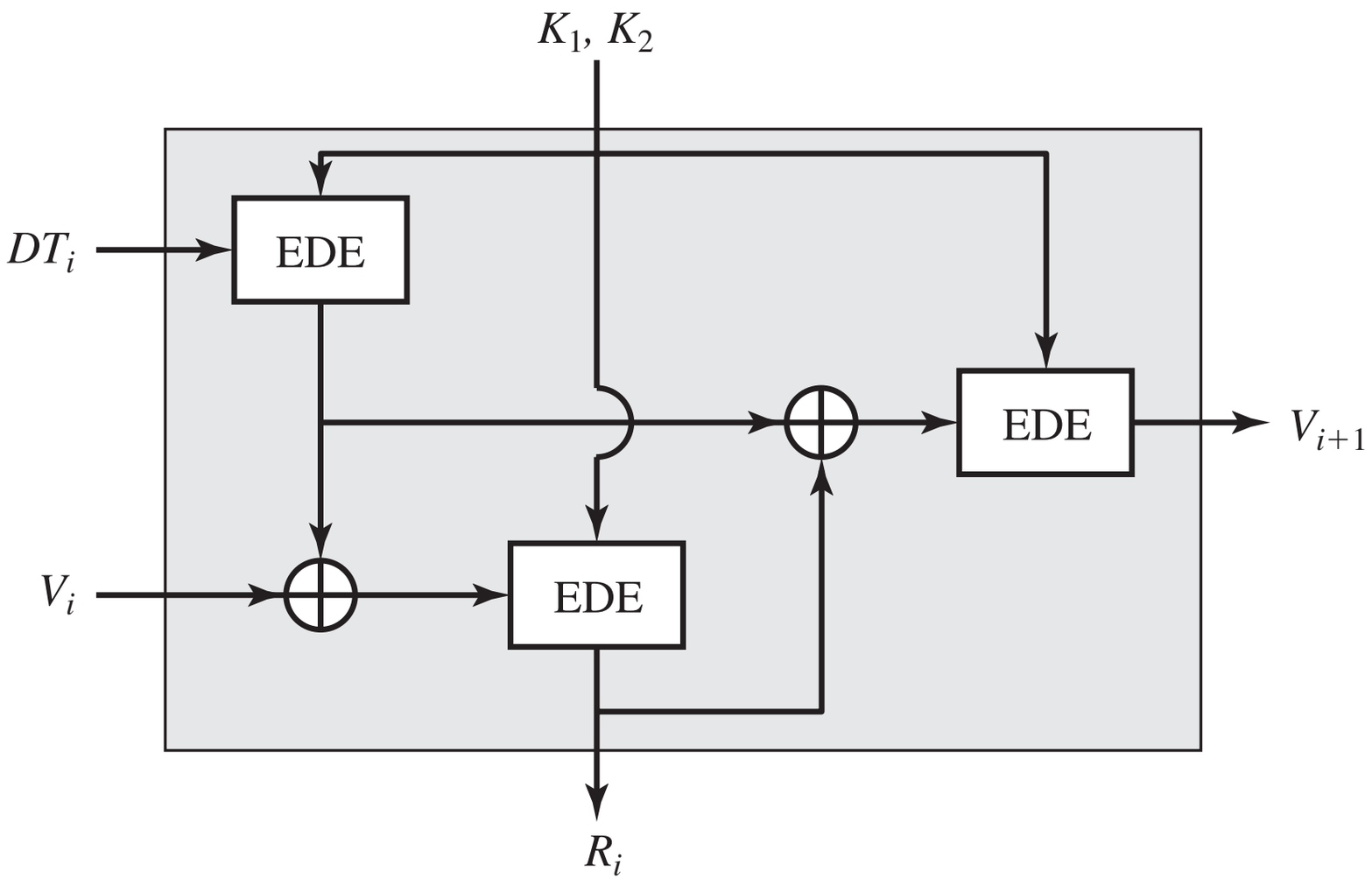


X9.82 and RFC 4086

(b) OFB mode

2. Pseudorandom Number Generation
(4) PRNG: ANSI X9.17

- One of the **strongest** (cryptographically speaking) PRNGs



- Encryption 3DES2
- On stage i:
  - K_1,K_2: keys
  - DT_i: Date/time value
  - V_i: Seed value
  - R_i: Pseudorandom number produced

Homework

7.1 If we take the linear congruential algorithm with an additive component of 0,

$$X_{n+1} = (aX_n) \bmod m$$

Then it can be shown that if m is prime and if a given value of a produces the maximum period of $m - 1$, then a^k will also produce the maximum period, provided that k is less than m and that k and $m - 1$ are relatively prime. Demonstrate this by using $X_0 = 1$ and $m = 31$ and producing the sequences for $a^k = 3, 3^2, 3^3$, and 3^4.

7.2 **a.** What is the maximum period obtainable from the following generator?

$$X_{n+1} = (aX_n) \bmod 2^4$$

b. What should be the value of a?

c. What restrictions are required on the seed?

7.4 With the linear congruential algorithm, a choice of parameters that provides a full period does not necessarily provide a good randomization. For example, consider the following two generators:

$$X_{n+1} = (6X_n) \bmod 13$$

$$X_{n+1} = (7X_n) \bmod 13$$

Write out the two sequences to show that both are full period. Which one appears more random to you?