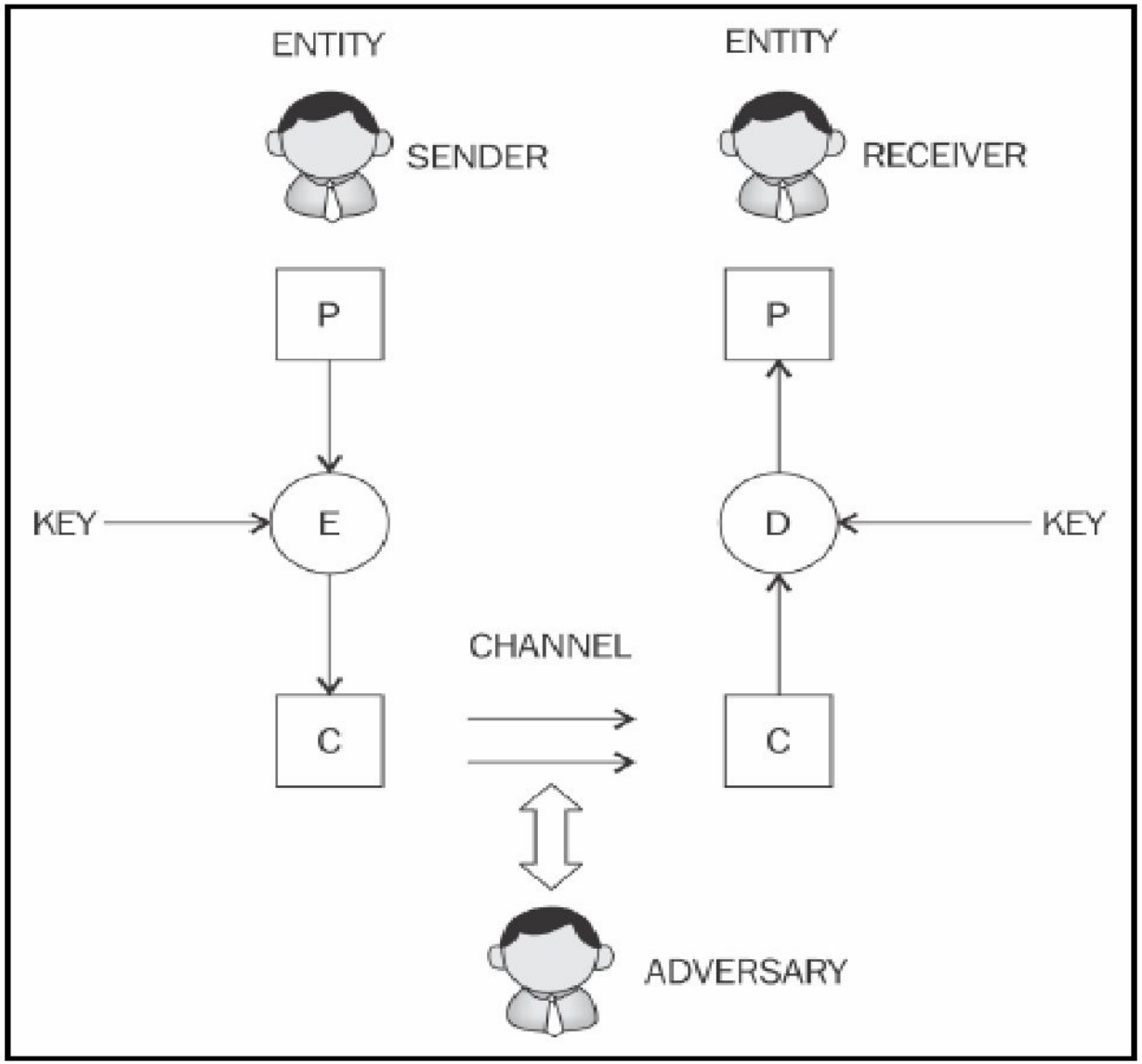
Cryptography and Technical Foundations

Cryptographic primitives

- Cryptographic primitives are the basic building blocks of a security protocol or system.
- A **security protocol** is a set of steps taken in order to achieve required security goals by utilizing appropriate security mechanisms.
- Various types of security protocols are in use, such as **authentication protocols**, **nonrepudiation protocols**, and **key management protocols**.

Generic cryptography model



Cryptographic primitives

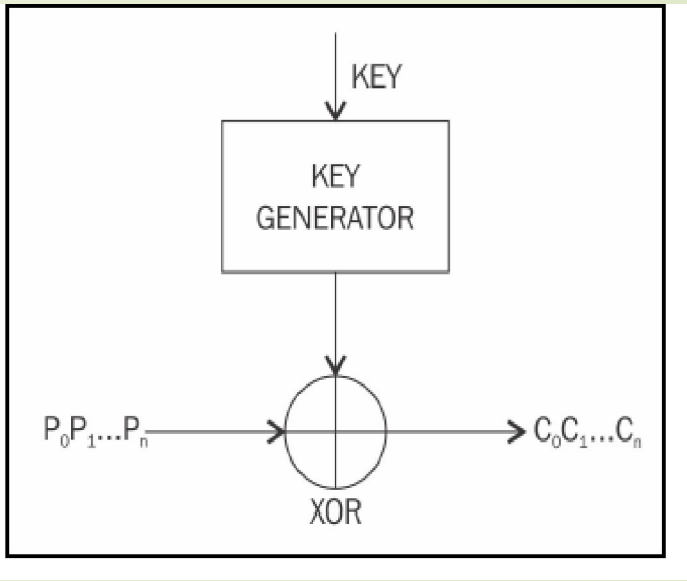
- **Entity**: It is either a person or a system that sends, receives, or performs operations on data
- **Sender**: Sender is an entity that transmits the data
- **Receiver**: Receiver is an entity that takes delivery of the data
- **Adversary**: This is an entity that tries to circumvent the security service
- **Key**: A key is some data that is used to encrypt or decrypt data
- **Channel**: Channel provides a medium of communication between entities

Symmetric cryptography

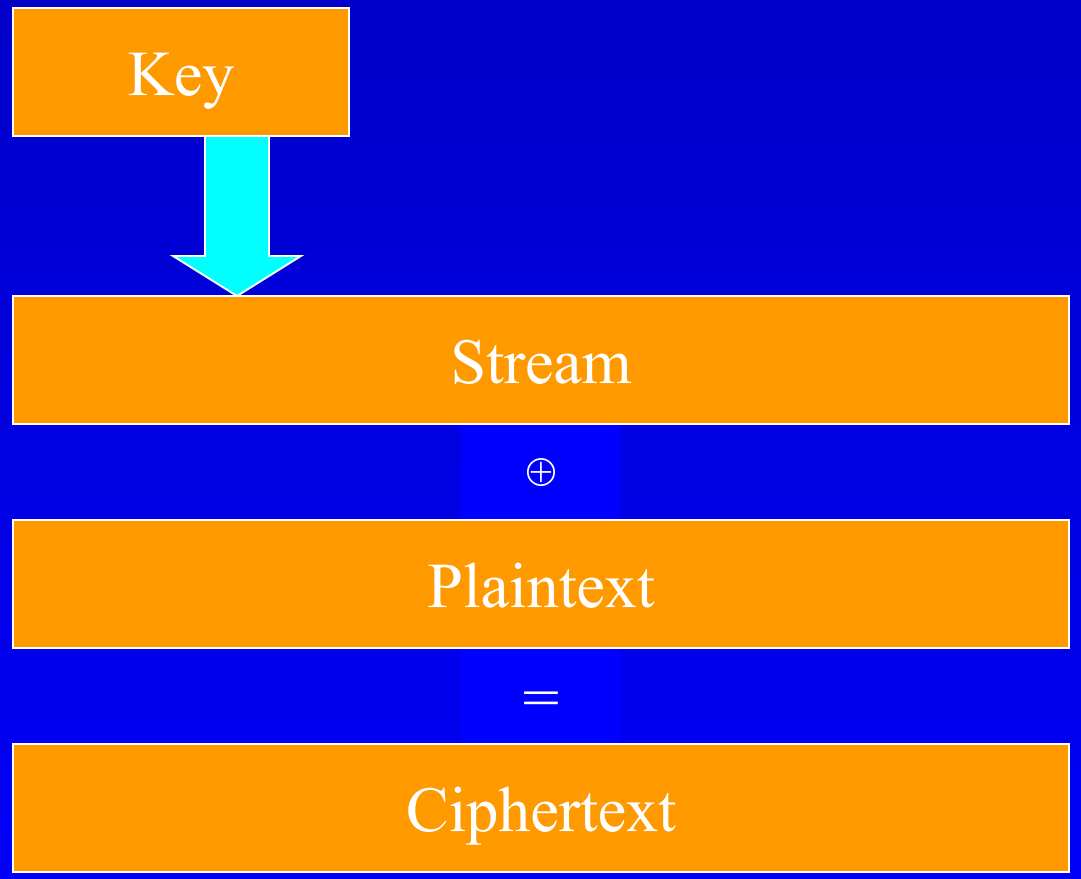
- Symmetric cryptography refers to a type of cryptography whereby the **key that is used to encrypt the data is the same for decrypting the data**
- The key must be established or agreed on before the data exchange between the communicating parties
- Also known as a shared key cryptography **(secret key Cryptography)**
- There are two types of symmetric ciphers
  - Stream ciphers
    - RC4 and A5
  - Block ciphers.
    - Data Encryption Standard (DES) and Advanced Encryption Standard (AES)

Stream ciphers

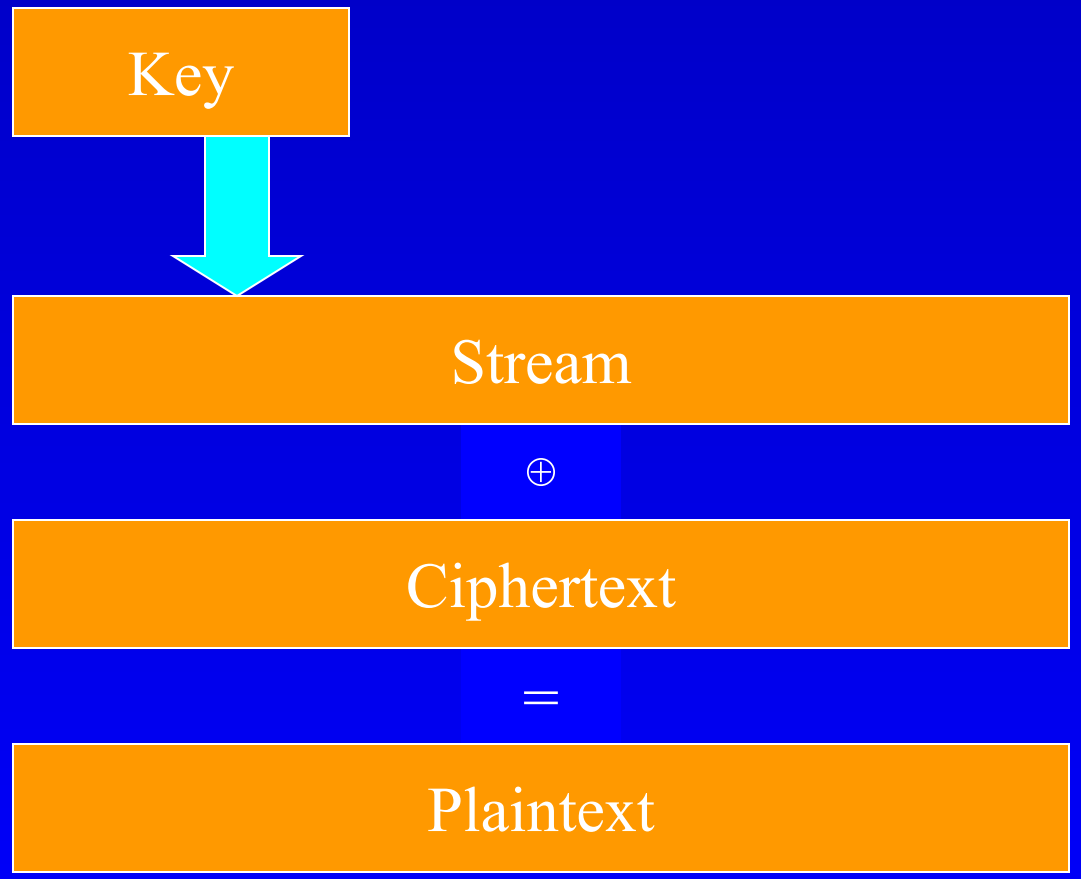
- Encryption algorithms on a bit-by-bit basis to plain text using a key stream.
- There are two types of stream ciphers:
  - Synchronous
    - Synchronous stream ciphers are ones where key stream is dependent only on the key
  - Asynchronous.
    - Asynchronous stream ciphers have a key stream that is also dependent on the encrypted data.
- Encryption and decryption are basically the same function because they are simple modulo 2 additions or XOR operation



Example of Stream Encryption

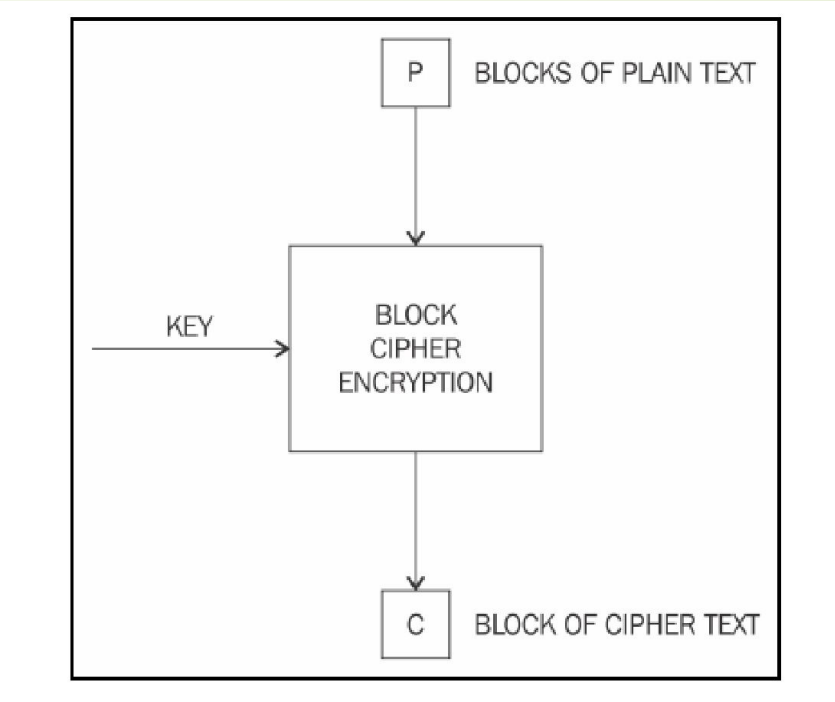


Example of Stream Decryption



Block ciphers

- Encryption algorithms that break up a text to be encrypted (plain text) into blocks of fixed length
- Apply encryption block by block.
- Block ciphers are usually built using a design strategy known as Fiestel cipher.



Block ciphers

- Fiestel ciphers are based on the Fiestel network,
- This structure is based on the idea of combining multiple rounds of repeated operations to achieve desirable cryptographic properties knows as confusion and diffusion.
- Fiestel networks operate by dividing data into two blocks (left and right) and process these blocks via keyed round functions.

Block cipher encryption modes

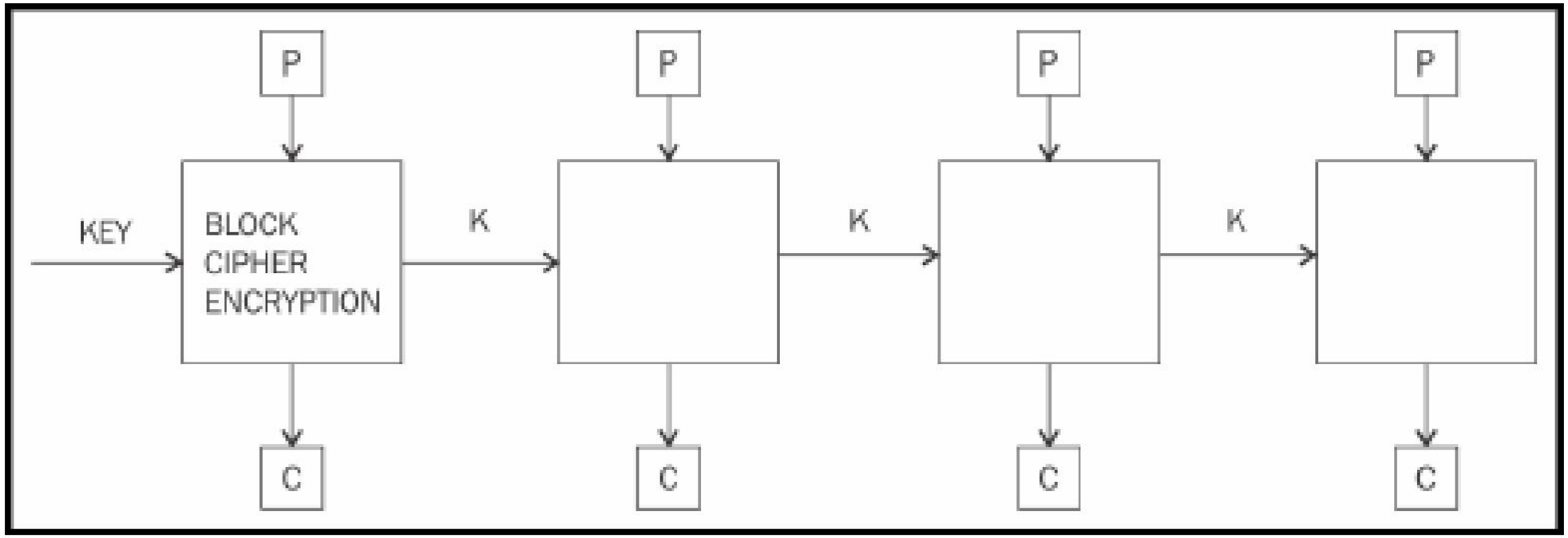
- **Block encryption mode**
  - In this mode, plaintext is divided into blocks of fixed length depending on the type of cipher used and then the encryption function is applied on each block.
- **Keystream generation modes**
  - In this mode, the encryption function generates a keystream that is then XORed with the plaintext stream in order to achieve encryption.
- **Message authentication modes**
  - In this mode, a message authentication code is computed as a result of an encryption function.
  - MAC is basically a cryptographic checksum that provides an integrity service.
  - The most common method to generate MAC using block ciphers is CBC-MAC, where some part of the last block of the chain is used as a MAC.

Block cipher encryption modes

- **Cryptographic hashes**
  - Hash functions are basically used to compress a message to a fixed length digest.
  - In this mode, block ciphers are used as a compression function to produce a hash of plain text.
- **Electronic code book**
  - This is a basic mode of operation in which the encrypted data is produced as a result of applying the encryption algorithm one by one separately to each block of plain text.
  - This is the simplest mode but should not be used in practice as it is insecure and can reveal information:

Block encryption modes

- **Electronic code book**

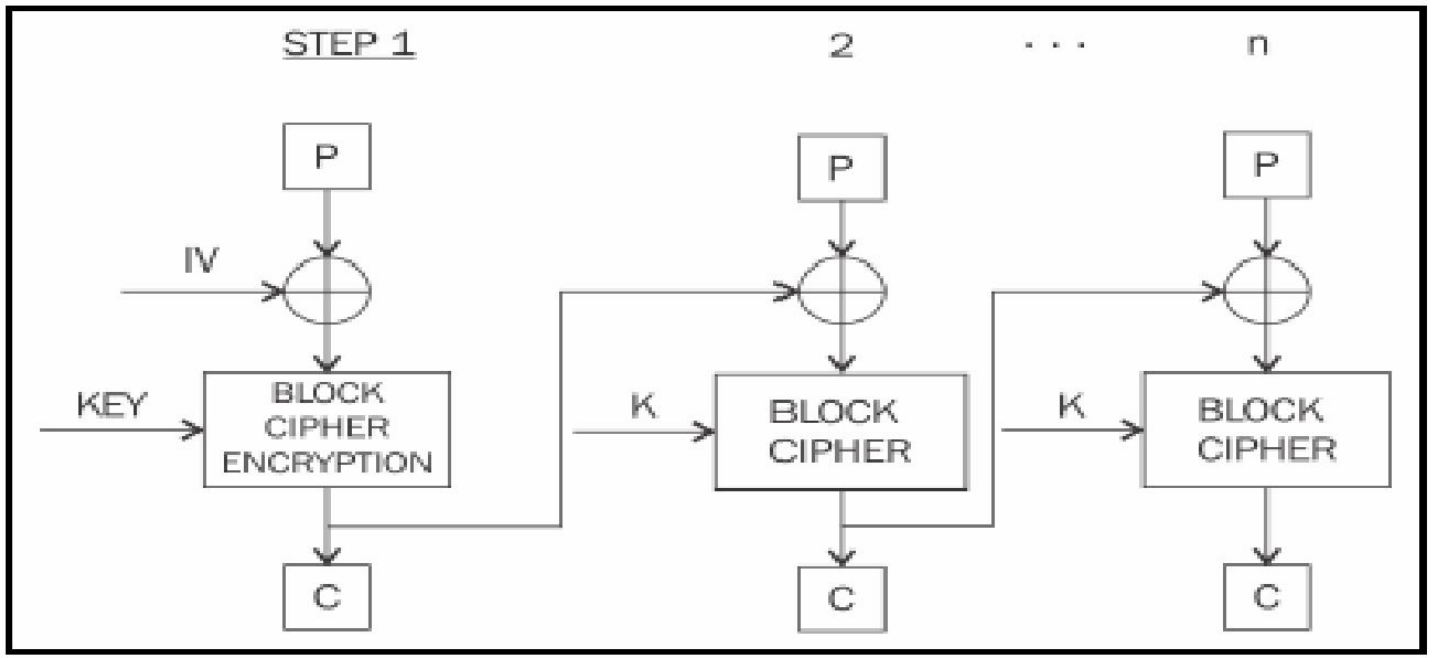


Block cipher encryption modes

- **Cipher block chaining**
  - In this mode, each block of plain text is XORed with the previous encrypted block.
  - The CBC mode uses initialization vector IV to encrypt the first block.
  - It is recommended that IV be randomly chosen

Cipher block chaining

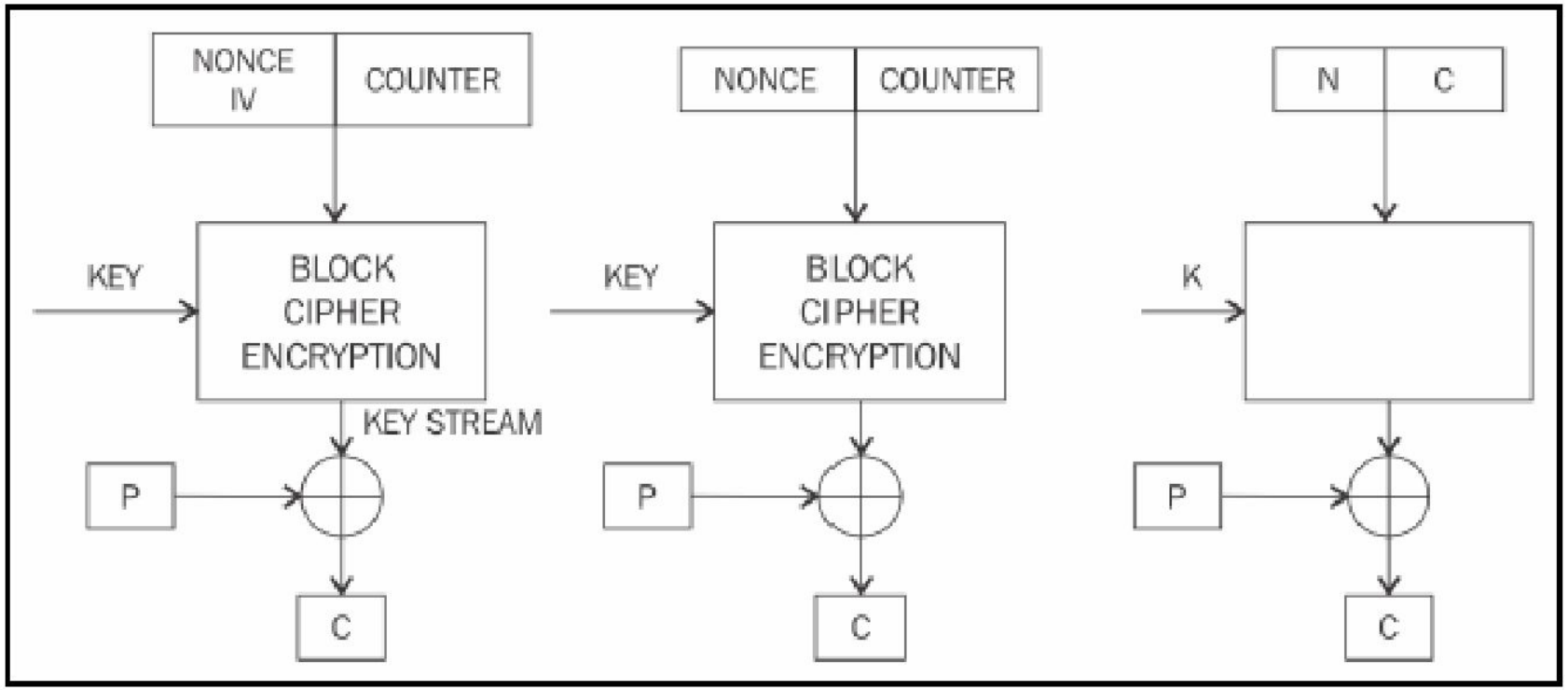
- initialization vector (IV)



Block cipher encryption modes

- **Counter mode**
  - The CTR mode effectively uses a block cipher as a stream cipher.
  - In this case, a unique nonce is supplied that is concatenated with the counter value in order to produce a **key stream**:

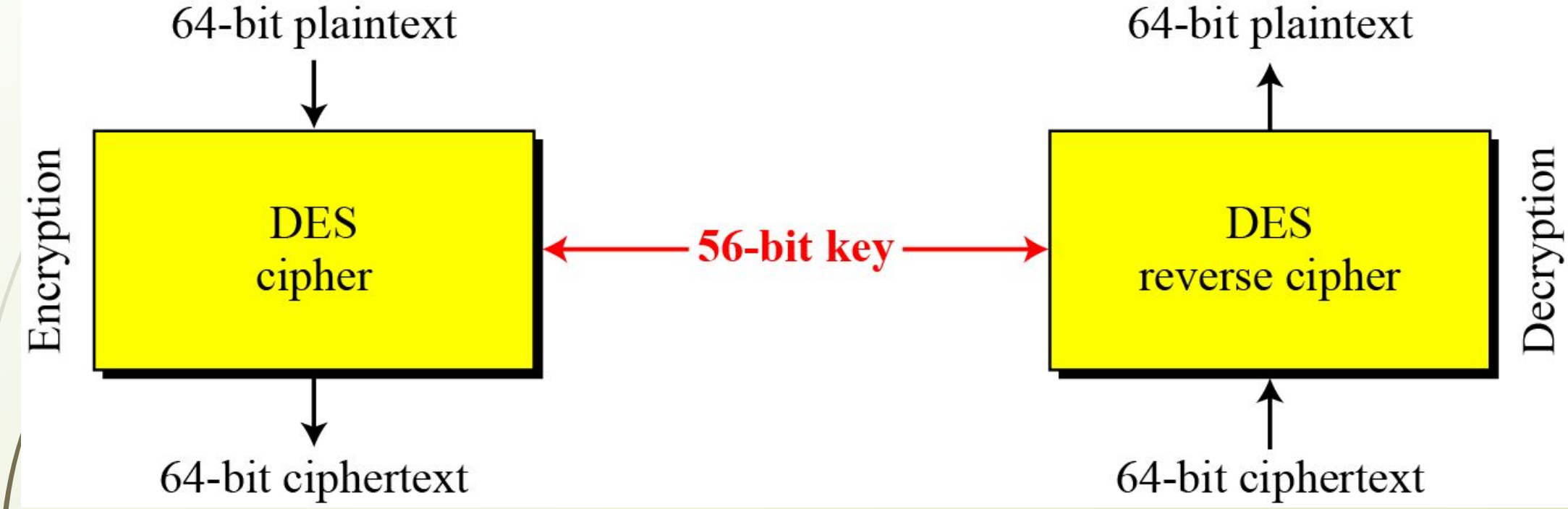
Counter mode



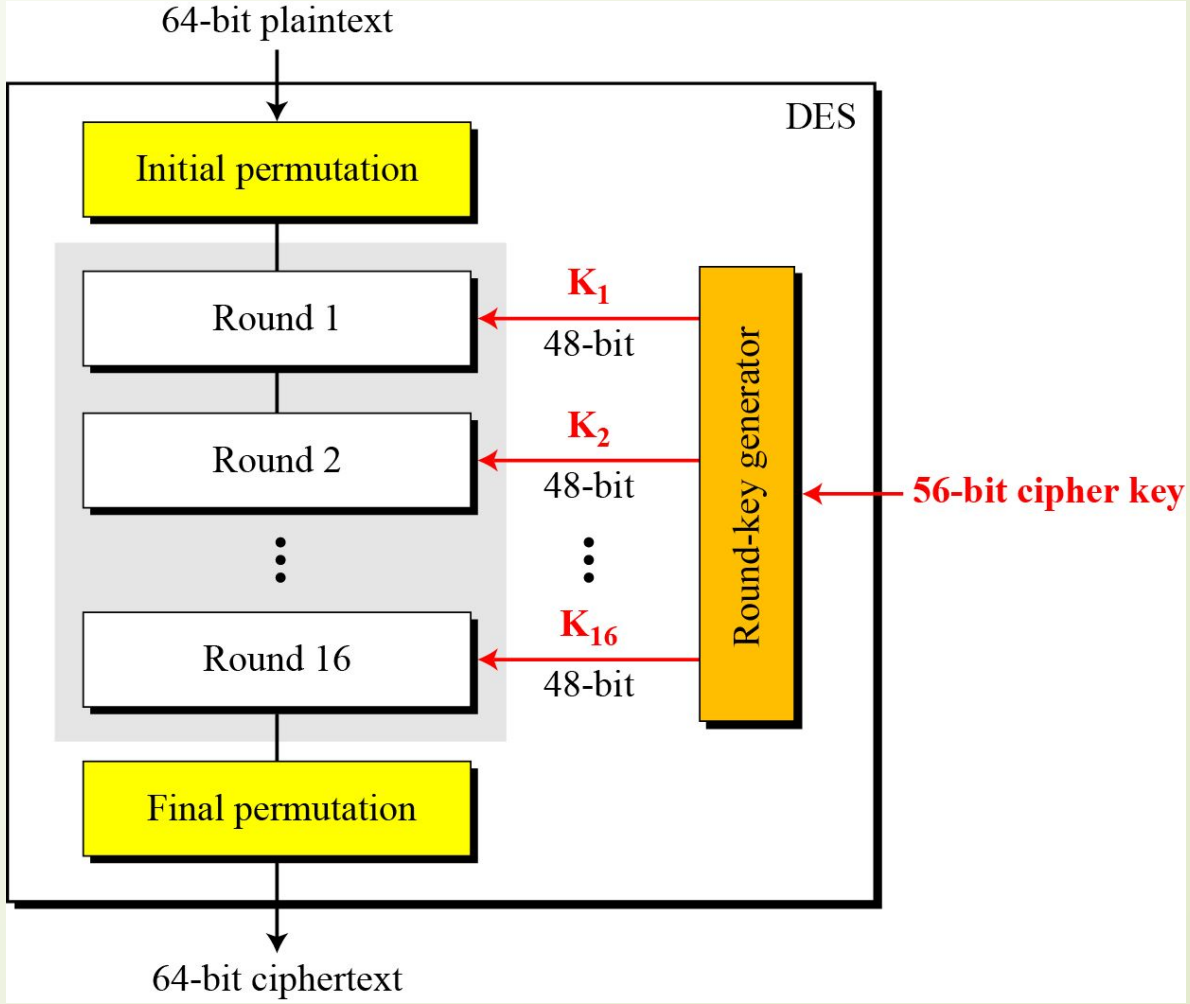
Data Encryption Standard (DES)

- The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
- DES is an implementation of a Feistel Cipher.
- It uses 16 round Feistel structure.
- The block size is 64-bit.
- Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm

Encryption and decryption with DES



General structure of DES



6.2.2 Rounds

DES uses 16 rounds. Each round of DES is a Feistel cipher.

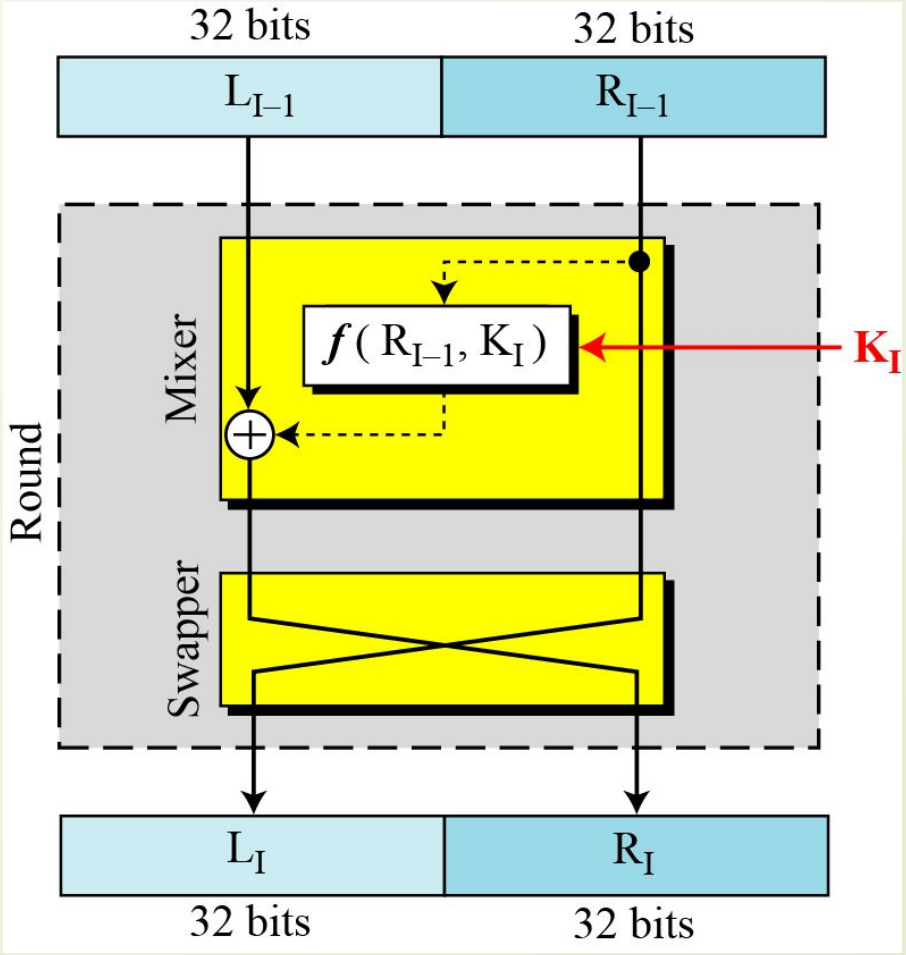


Figure 6.4

A round in DES (encryption site)

DES Function

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

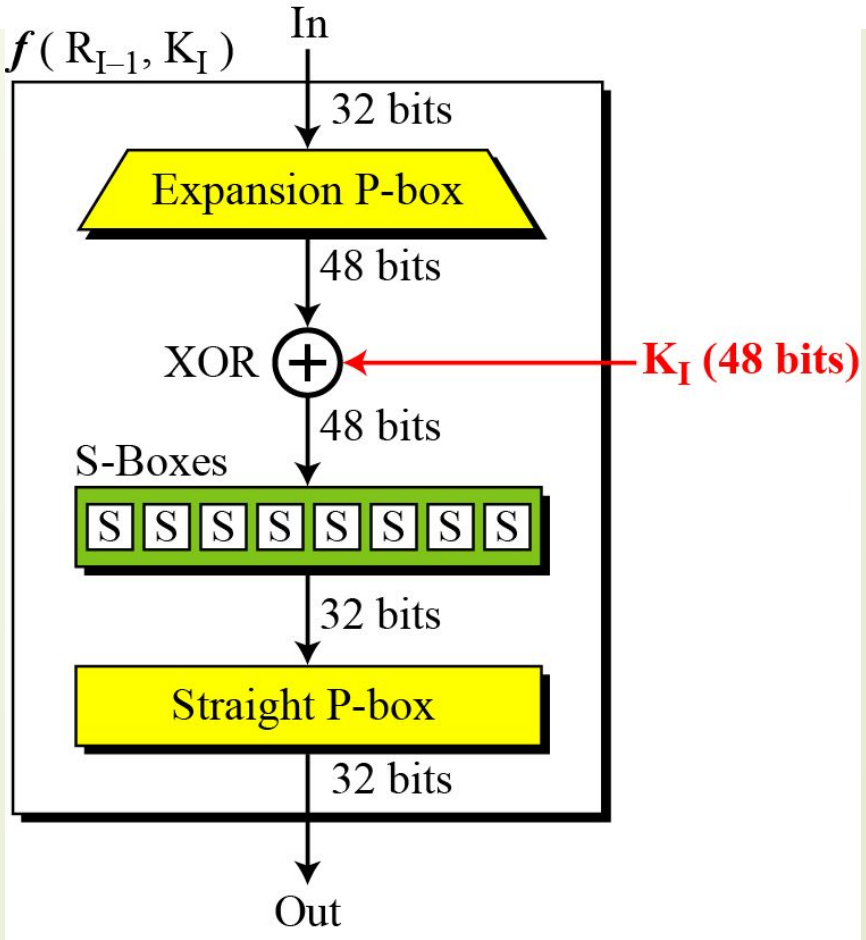


Figure 6.5
DES function

- **Triple DES (3DES)**, which proposed the usage of a 168-bit key using three 56-bit keys and the same number of executions of the DES algorithm
  - thus making brute force attacks almost impossible.
- Slow performance and 64-bit block size, are not desirable.

Advanced Encryption Standard (AES)

- The more popular and widely adopted symmetric encryption algorithm is the Advanced Encryption Standard (AES).
- It is found at least six time faster than triple DES.
- A replacement for DES was needed as its key size was too small.
- With increasing computing power, it was considered vulnerable against exhaustive key search attack.
- Triple DES was designed to overcome this drawback but it was found slow.

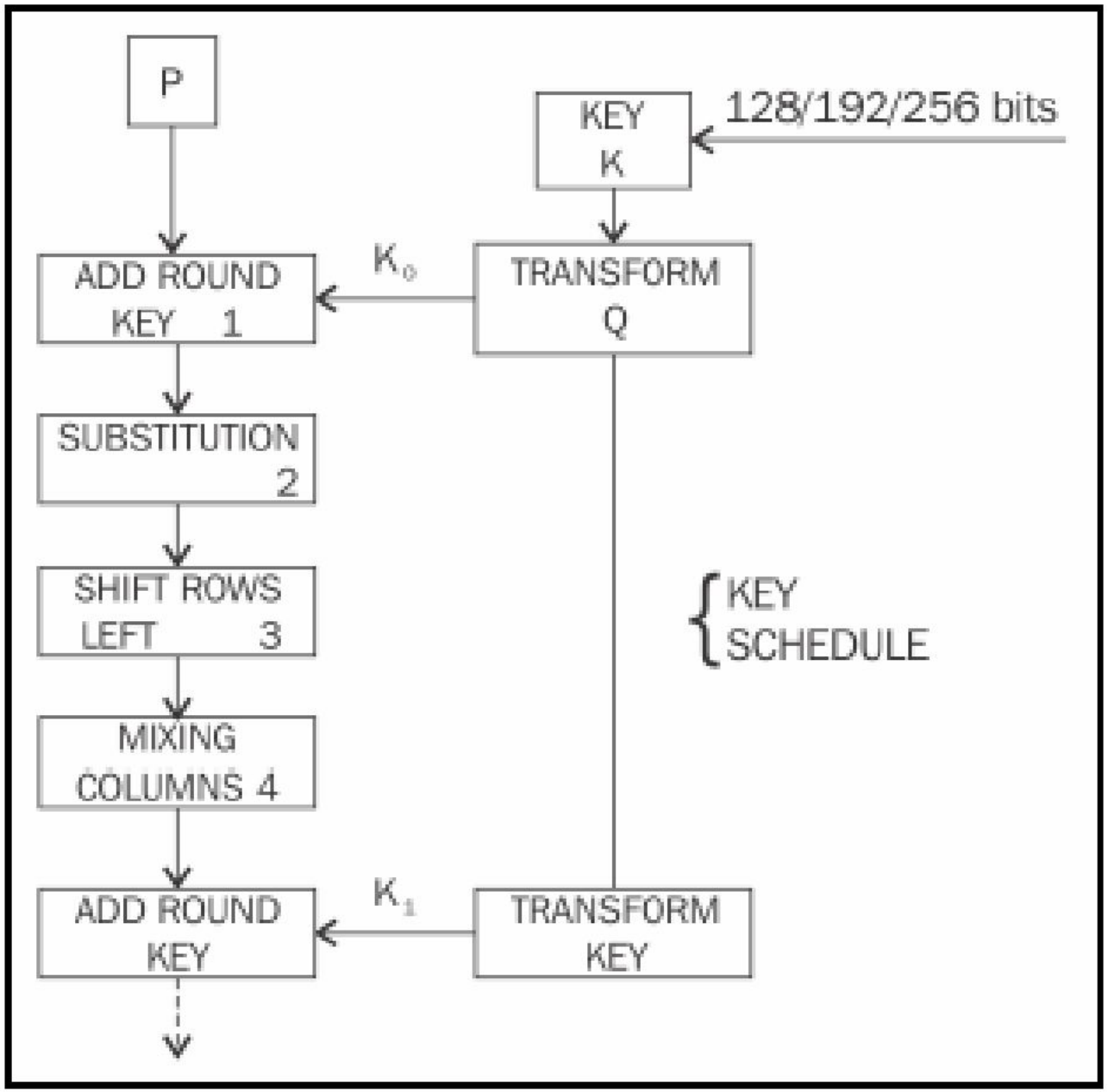
Advanced Encryption Standard (AES)

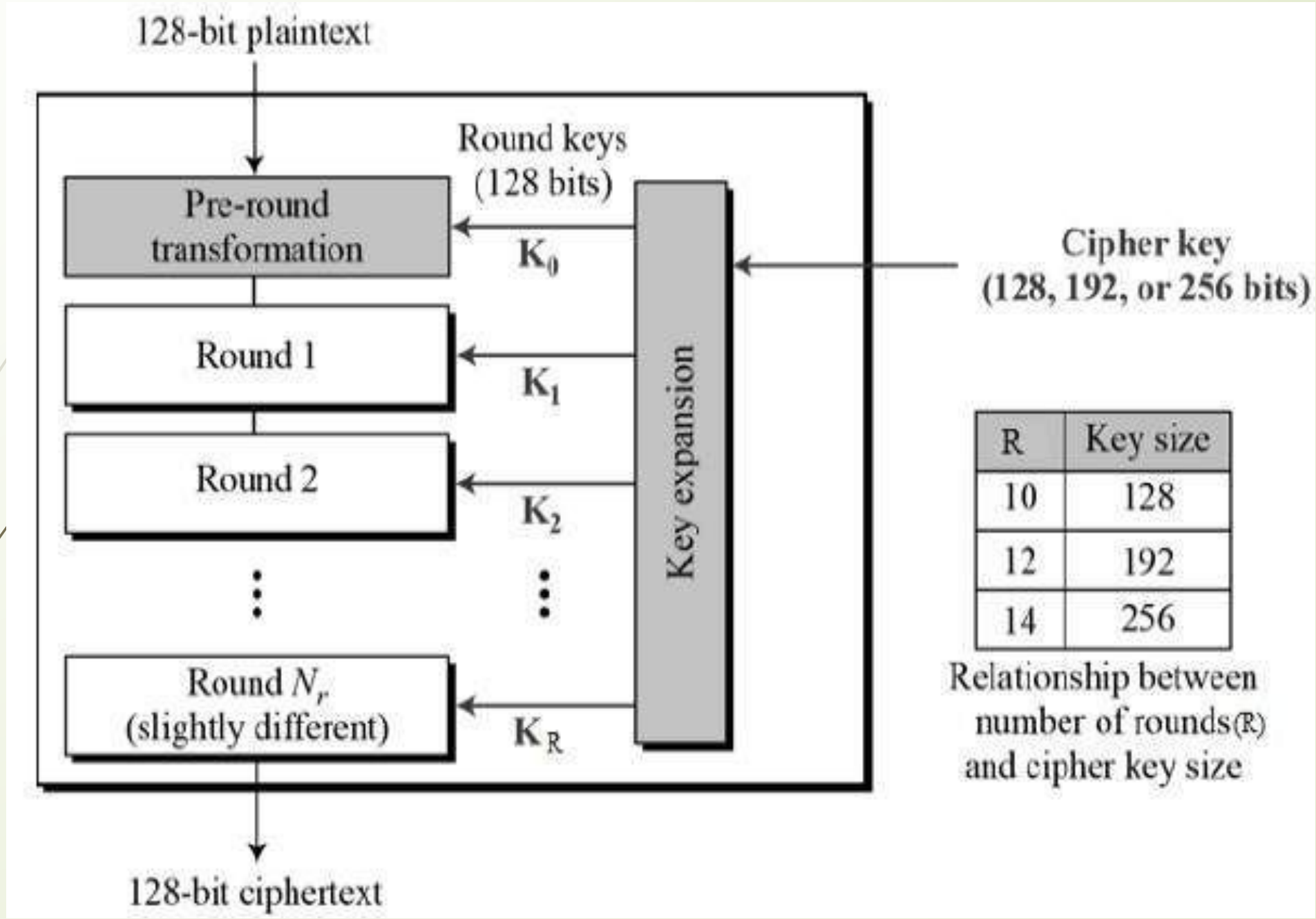
- No attack has been found against AES that is better than the brute force method.
- Original Rijndael allows different key and block sizes of 128-bit, 192-bit, and 256-bits,
- But in the AES standard, only a 128-bit block size is allowed.
- However, key sizes of 128-bit, 192-bit, and 256-bit are allowed

Operation of AES

- AES is an iterative rather than Feistel cipher.
- It is based on ‘substitution–permutation network’.
- It comprises of a series of linked operations,
  - some of which involve replacing inputs by specific outputs (substitutions)
  - others involve shuffling bits around (permutations).
- AES performs all its computations on bytes rather than bits. Hence,
- AES treats the 128 bits of a plaintext block as 16 bytes.
- These 16 bytes are arranged in four columns and four rows for processing as a matrix

Key size	Number of rounds required
128-bit	10 rounds
192-bit	12 rounds
256-bit	14 rounds



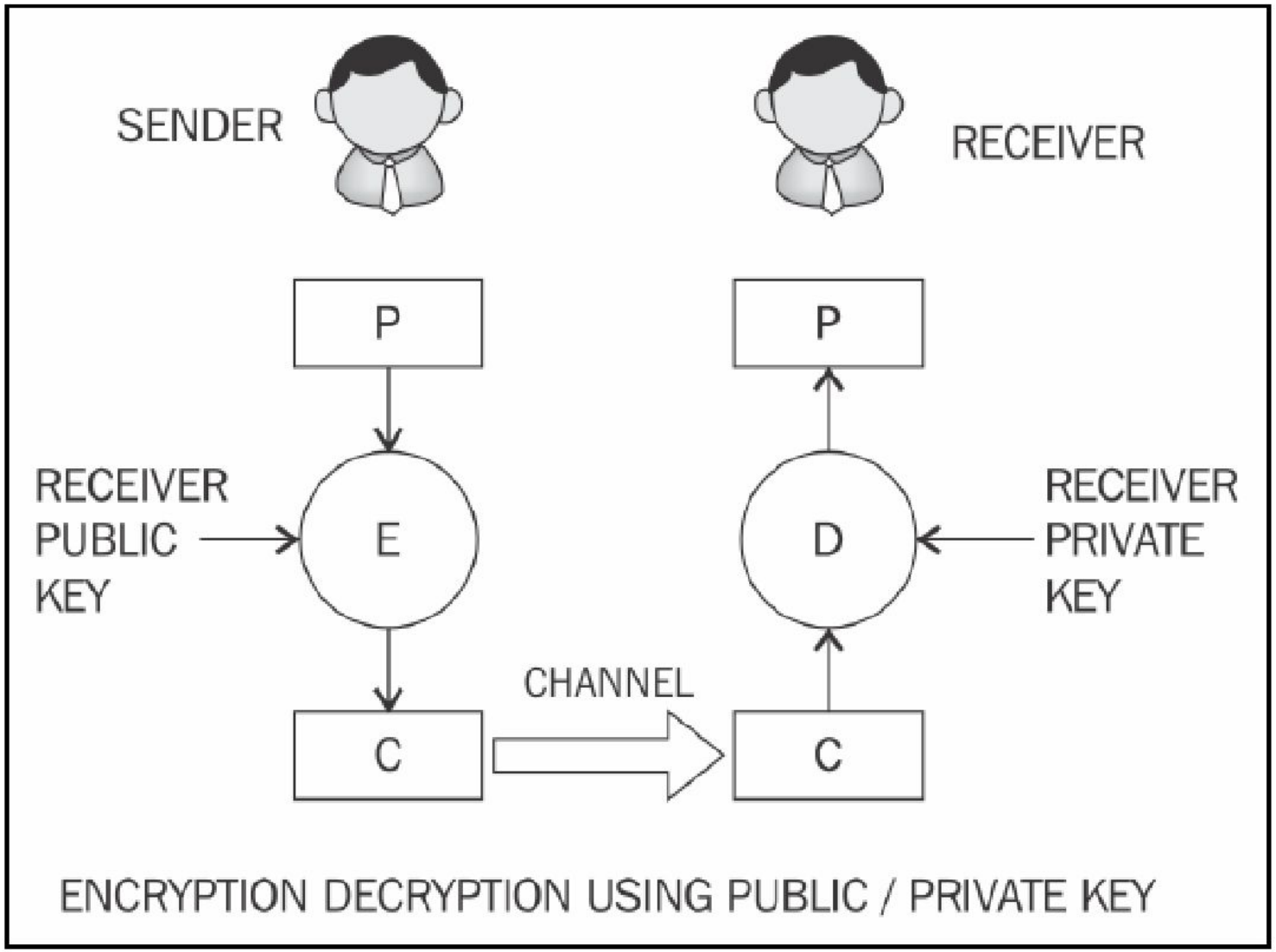


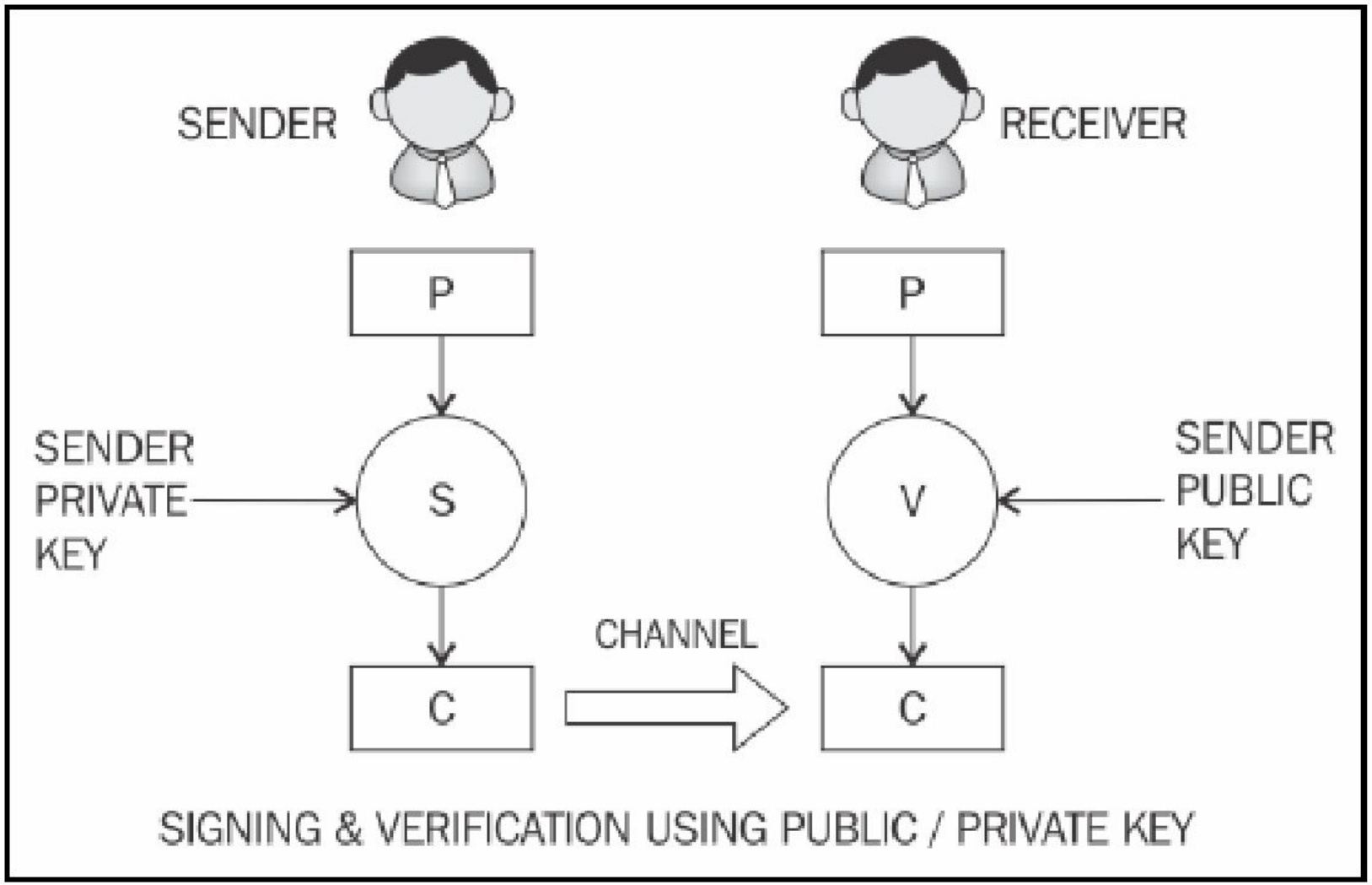
R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Asymmetric cryptography

- Asymmetric cryptography refers to a type of cryptography whereby the key that is used to encrypt the data is different from the key that is used to decrypt the data.
- Also known as public key cryptography
- It uses public and private keys in order to encrypt and decrypt data, respectively.


SENDER
RECEIVER
P
P
RECEIVER
PUBLIC
KEY
E
D
RECEIVER
PRIVATE
KEY
CHANNEL
C
C
ENCRYPTION DECRYPTION USING PUBLIC / PRIVATE KEY



Model of a public key cryptography signature scheme

- Public key cryptography algorithms are based on various underlying mathematical problems
  - **Integer factorization**
    - These schemes are based on the fact that large integers are very hard to factor
  - **Discrete logarithm**
    - This is based on a problem in modular arithmetic that it is easy to calculate the result of modulo function but it is computationally infeasible to find the exponent of the generator.

$$3^2 \bmod 10 = 9$$

  - **Elliptic curves**
    - This is based on the discrete logarithm problem, but in the context of elliptic curves.
    - Elliptic curve is an algebraic cubic curve over a field

$$y^2 = x^3 + ax + b$$

RSA

- RSA was invented in 1977 by *Ron Rivest*, *Adi Shamir*, and *Leonard Adelman*
- This is based on the integer factorization problem, where the multiplication of two large prime numbers is easy but difficult to factor it back to the two original numbers.
- **Modulus generation:**
  - Select p and q very large primes
  - Multiply p and q , $n=p.q$ to generate modulus n
- **Generate co-prime:**
  - Number should satisfy certain conditions, that is, it should be greater than 1 and less than $(p-1)\ (q-1)$. I
  - e must be such a number that no number other than 1 can be divided into e and $(p-1)\ (q-1)$. This is called co-prime, that is, e is the co-prime of $(p-1)(q-1)$.

- **Generate public key:**
  - Modulus generated in step 1 and e generated in step 2 is pair that, together, is a public key.
  - This part is the public part that can be shared with anyone;
  - *p* and *q* need to be kept secret.
- **Generate private key:**
  - Private key called d here and is calculated from p, q and e.
  - Private key is basically the inverse of e modulo (p-1)(q-1).

$$ed = 1 \bmod (p-1)(q-1)$$

Encryption and decryption using RSA

- Plain text *P* is raised to *e* number of times and then reduced to modulo *n*.

$$C = P^e \bmod n$$

- Receiver who has a public key pair (*n*, *e*) can decipher the data by raising *C* to the value of the private key *d* and reducing to modulo *n*

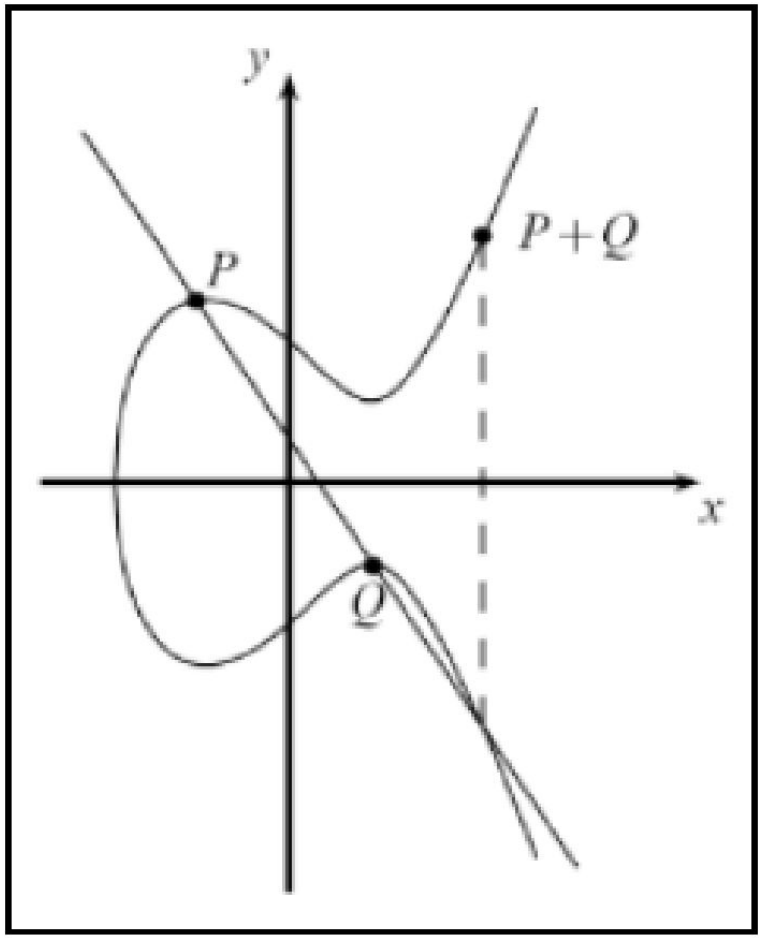
$$P = C^d \bmod n$$

Definition of Elliptic curves

- An elliptic curve over a field ***K*** is a nonsingular cubic curve in two variables, $f(x,y) = 0$ with a rational point (which may be a point at infinity).
- The field ***K*** is usually taken to be the complex numbers, reals, rationals, algebraic extensions of rationals, p-adic numbers, or a **finite field**.
- Elliptic curves groups for cryptography are examined with the underlying fields of F_p (*where p>3 is a prime*) and F_{2^m} (*a binary representation with 2^m elements*).

What is Elliptic Curve Cryptography?

- Implementing Group Operations
  - Main operations - point addition and point multiplication
  - Adding two points that lie on an Elliptic Curve – results in a third point on the curve
  - Point multiplication is repeated addition
  - If P is a known point on the curve (aka Base point; part of domain parameters) and it is multiplied by a scalar k, Q=kP is the operation of adding P + P + P + P… +P (k times)
  - Q is the resulting public key and k is the private key in the public-private key pair



Point addition visualized over R

Generic Procedures of ECC

- Both parties agree to some publicly-known data items
  - The **elliptic curve equation**
    - values of ***a*** and ***b***
    - prime, ***p***
  - The **elliptic group** computed from the elliptic curve equation
  - A **base point**, B, taken from the elliptic group
    - Similar to the generator used in current cryptosystems
- Each user generates their public/private key pair
  - Private Key = an integer, x, selected from the interval [1, p-1]
  - Public Key = product, Q, of private key and base point
    - (Q = x*B)

Elliptic Curve Cryptography

- Components

Private Key	Public Key	Set of Operations	Domain Parameters (Predefined constants)
A random number	Point on a curve = Private Key * G	These are defined over the curve $\mathbf{y^2 = x^3 + ax + b,}$ where $\mathbf{4a^3 + 27b^2 \neq 0}$	G, a, b

Why use ECC?

- How do we analyze Cryptosystems?
  - How difficult is the underlying problem that it is based upon
    - RSA – Integer Factorization
    - DH (Diffie–Hellman) – Discrete Logarithms
    - ECC - Elliptic Curve Discrete Logarithm problem
  - How do we measure difficulty?
    - We examine the algorithms used to solve these problems

Security of ECC

- To **protect** a 128 bit AES key it would take a:
  - RSA Key Size: 3072 bits
  - ECC Key Size: 256 bits
- How do we strengthen RSA?
  - Increase the key length
- **Impractical?**

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Applications of ECC

- Many devices are small and have limited storage and computational power
- Where can we apply ECC?
  - **Wireless communication devices**
  - Smart cards
  - Web servers that need to handle many encryption sessions
  - **Any application where security is needed but lacks the power, storage and computational power that is necessary for our current cryptosystems**

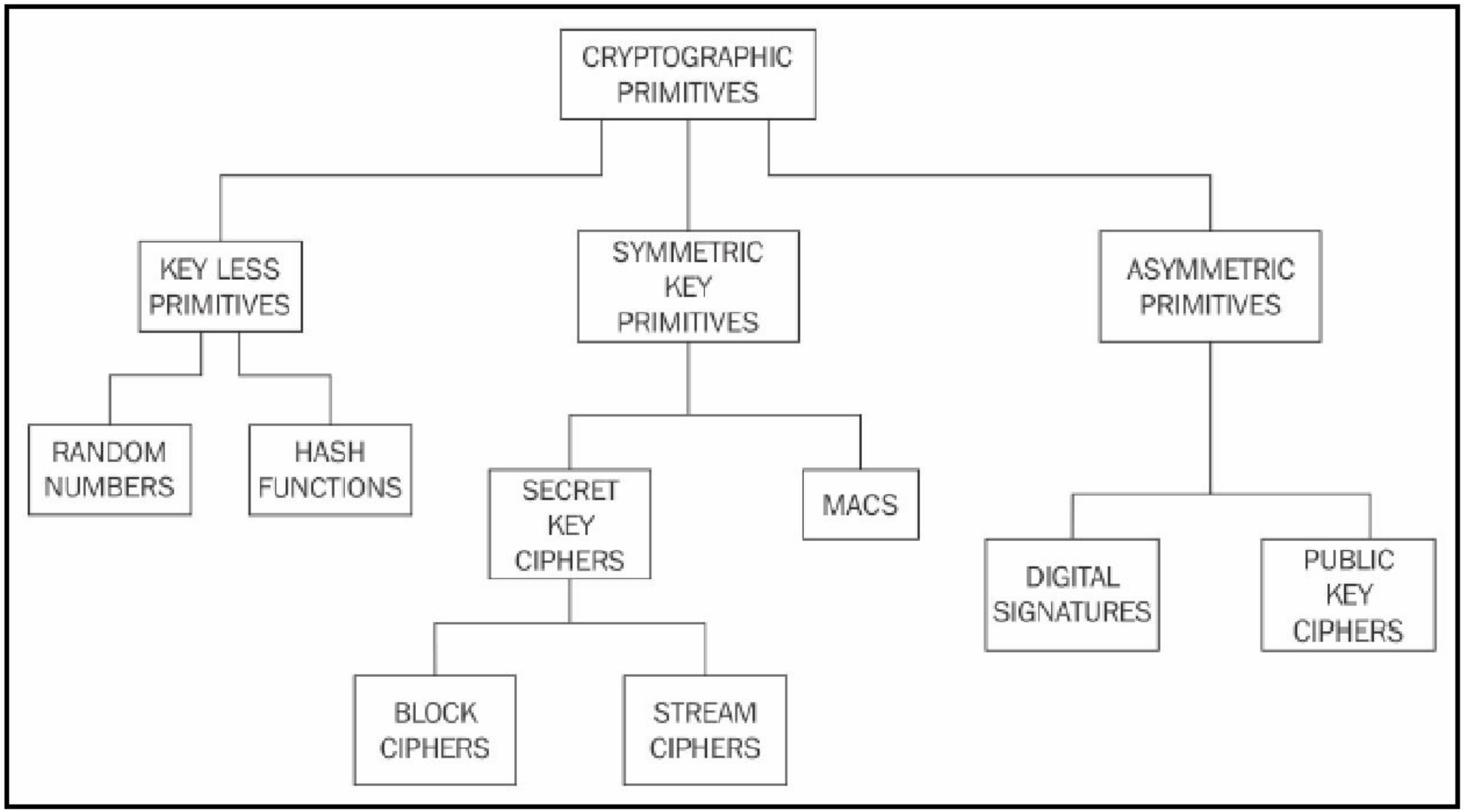
Benefits of ECC

- Same benefits of the other cryptosystems: confidentiality, integrity, authentication and non-repudiation but…
- Shorter key lengths
  - Encryption, Decryption and Signature Verification speed up
  - Storage and bandwidth savings

Summary of ECC

- "**Hard problem**" analogous to discrete log
  - **Q=kP, where Q, P belong to a prime curve**
    - **given k, P → "easy" to compute Q**
    - **given Q, P → "hard" to find k**
  - **known as the elliptic curve logarithm problem**
    - **k must be large enough**
- ECC security relies on elliptic curve logarithm problem
  - compared to factoring, can use much smaller key sizes than with RSA etc
    - **for similar security ECC offers significant computational advantages**

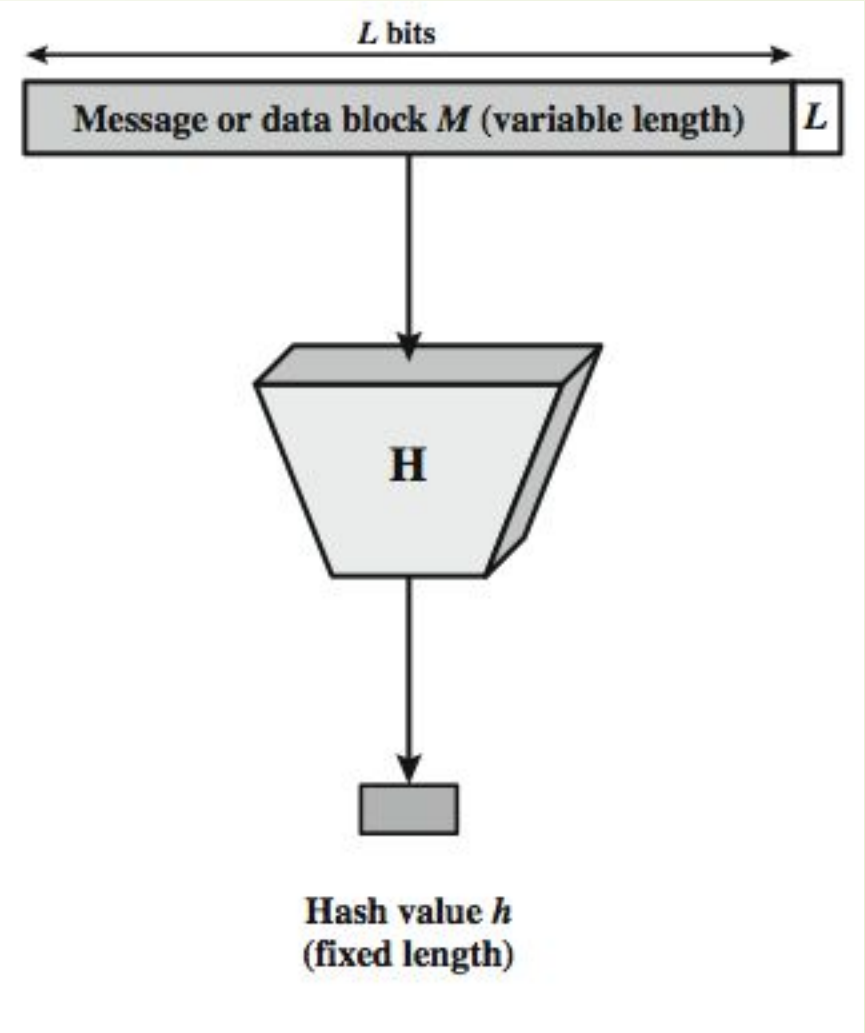
Cryptographic primitives



Hash Functions

- A hash function maps a message of an arbitrary length to a m-bit output
  - output known as the fingerprint or the message digest
- What is an example of hash functions?
  - Give a hash function that maps Strings to integers in [0,2^{32}-1]
- Cryptographic hash functions are hash functions with additional security requirements

Cryptographic Hash Function



Hash functions

- Hash functions are used to create fixed length digests of arbitrarily long input strings.
- Hash functions are keyless and provide the data integrity service.
- They are usually built using iterated and dedicated hash function construction techniques.
- Various families of hash functions are available, such as MD, SHA1, SHA-2, SHA-3, RIPEMD, and Whirlpool.
- Hash functions are commonly used in digital signatures and message authentication codes, such as HMACs.

Hash functions

- **Hash functions** have three security properties, namely
  - pre-image resistance
  - second preimage resistance
  - collision resistance
- **Hash functions** are typically used to provide data integrity services.
- **Hash functions** can be used as one-way functions and to construct other cryptographic primitives, such as MACs and digital signatures.
- Some applications used hash functions as a means of generating **pseudo random numbers** (**PRNGs**).
- Hash functions do not require a key.

Two practical properties of hash functions

- **Compression of arbitrary messages into fixed length digest**
  - This property is concerned with the fact that a hash function must be able to take a long input text of any length and output a fixed length compressed message.
  - Hash functions produce a compressed output in various bit sizes, usually between 128-bits and 512-bits.
- **Easy to compute**
  - Hash functions are efficient and fast one-way functions.
  - Requirement is that they be very quick to compute regardless of the message size.
  - The efficiency may decrease if the message is too big but the function should still be fast enough for practical use.

Security properties of hash functions

- **Pre-image resistance (**one-way property)
  - $h(x) = y$
  - h is the hash function, x is the input, and y is the hash.
  - The first security property requires that y cannot be reverse computed to x.
  - x is considered a *pre-image* of y
- **Second pre-image resistance (**weak collision resistance)
  - This property requires that given x and $h(x)$
  - It is almost impossible to find any other message m ,
  - where ***m != x*** and ***hash of m = hash of x*** *i.e* $h(m) = h(x)$

Security properties of hash functions

- **Collision resistance (**strong collision resistance)
  - This property requires that two different input messages should not hash to the same output.
  - In other words, *h(x) != h(z)*.

Security Requirements for Cryptographic Hash Functions

Given a function $h:X \rightarrow Y$, then we say that h is:

- preimage resistant (one-way):

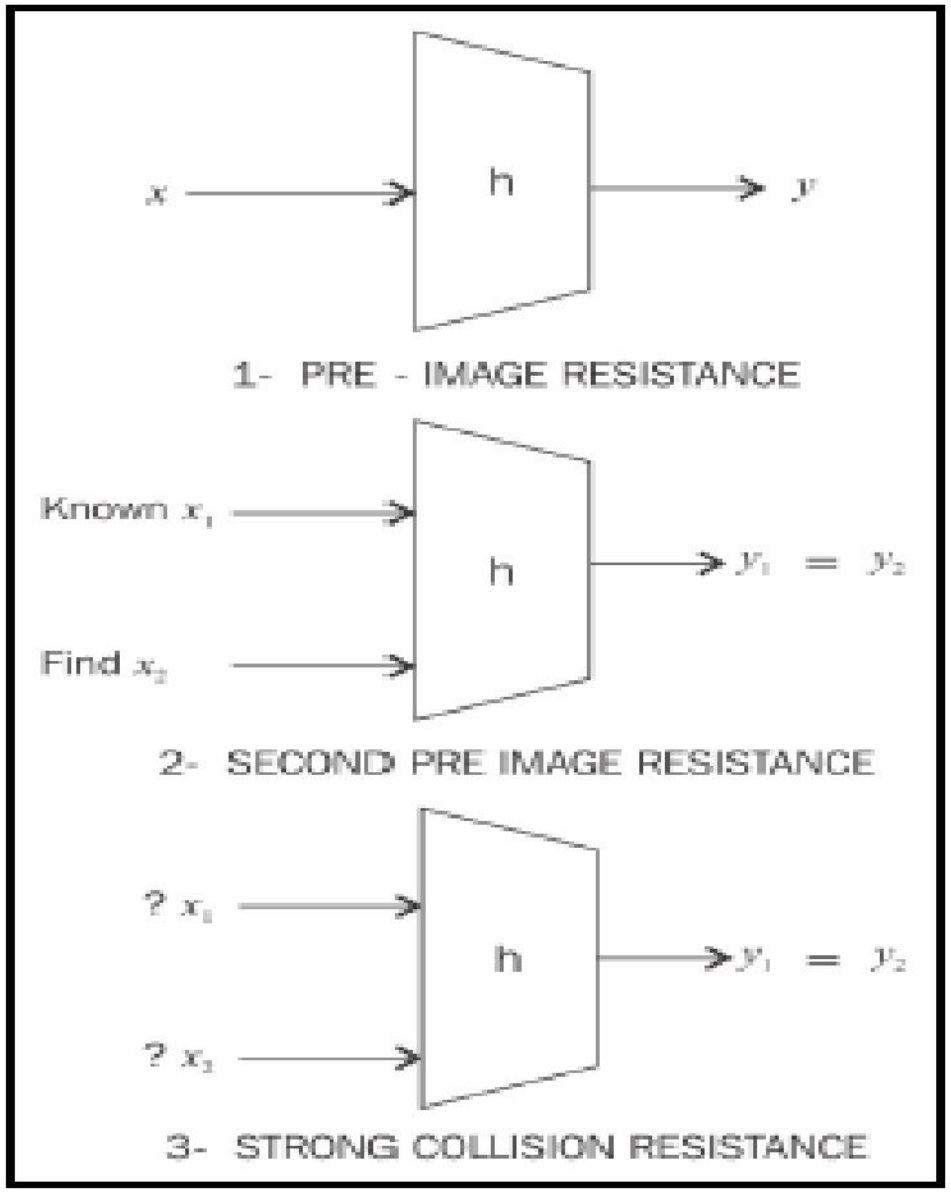
  if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $h(x) = y$

- 2-nd preimage resistant (weak collision resistant):

  if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $h(x') = h(x)$

- collision resistant (strong collision resistant):

  if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $h(x') = h(x)$



Three security properties of hash functions

Attacks on Hash Functions

- have brute-force attacks and cryptanalysis
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

Hash functions

- Hash functions, due to their very nature, will always have some collisions
  - That is where two different messages hash to the same output
  - But they should be computationally infeasible to find.
- A concept known as **avalanche effect** is desirable in all hash functions.
  - Avalanche effect specifies that a small change, even a single character change in the input text, will result in a totally different hash output.

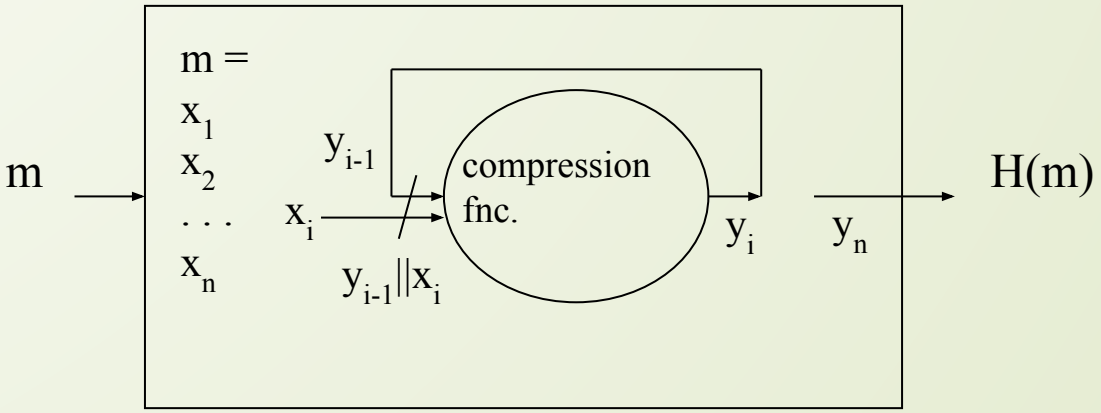
Hash functions

- Hash functions are usually designed by following iterated hash functions approach.
  - The input message is compressed in multiple rounds on a block-by-block basis to produce the compressed output.
  - A popular type of iterated hash function is **Merkle- Damgard construction.**
  - This construction is based on the **idea of dividing the input data into equal sizes of blocks and then feeding them through the compression functions in an iterative manner**.
  - The collision resistance of the property of compression functions ensures that the hash output is also collision-resistant.
  - Compression functions can be built using block ciphers.

Internals of a Hash Function

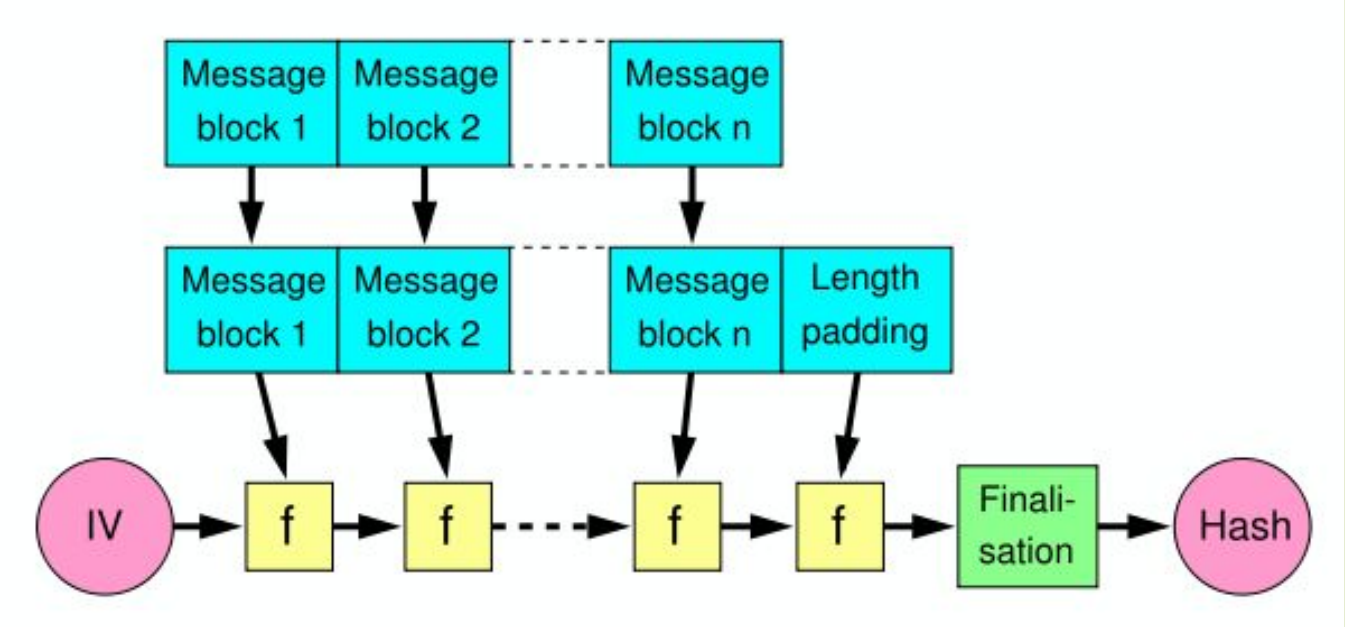
Merkle-Damgard construction:

- A fixed-size “compression function”.
- Each iteration mixes an input block with the prev. output.



Merkle-Damgard Construction for Hash Functions

- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value



$M=m_1 m_2 \ldots m_n; C_0=IV, C_{i+1}=f(C_i,m_i); H(M)=C_n$

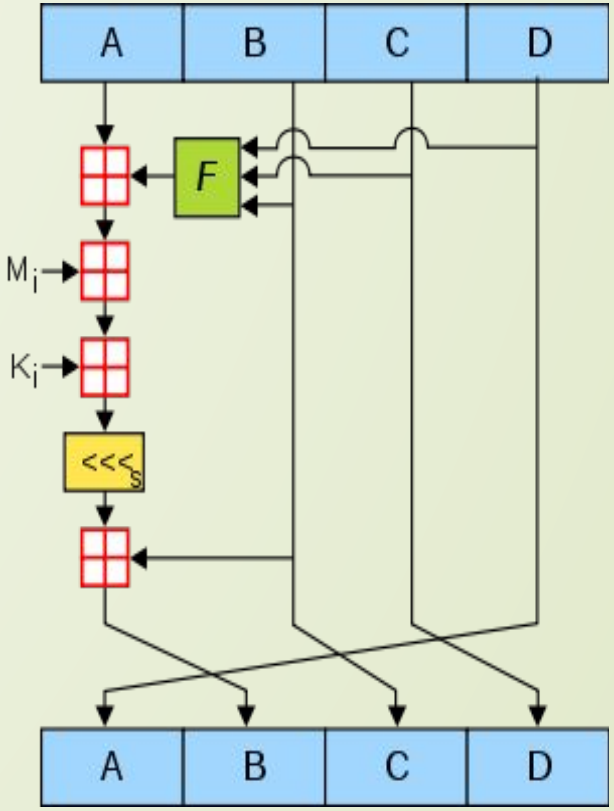
Message Digest (MD)

- Message Digest functions were very popular in early 1990s.
- MD4 and MD5 are members of this category.
- Both MD functions are found to be insecure and not recommended for use any more.
- MD5 is a 128-bit hash function that was commonly used for file integrity checks.
  - *Algorithm takes as input a message of arbitrary length and produces as output a 128-bit 'fingerprint' or 'message digest' of the input.*
  - *It is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest*

MD5

- Rivest, 1991
- Based on Davies-Meyer const.
- Very popular until recently.
  - 2004: First collision attacks
  - 2008: Practical collision attack; SSL cert. with same MD5 hash.
  - 2010: Forged Microsoft MD5 certificates used in Flame malware
- Preimage resistance: Mostly ok.

64 rounds of:



MD5 Algorithm

- Step 1 – append padded bits:
- Step 2 – append length:
- Step 3 – Initialize MD Buffer
- Step 4- Process message in 16-word blocks
- Step 5 – output

Secure Hash Algorithms (SHAs)

- **SHA-0**:
  - This is a 160-bit function introduced by NIST in 1993.
- **SHA-1:**
  - SHA-1 was introduced later by NIST as a replacement of SHA-0.
  - This is also a 160-bit hash function.
  - SHA-1 is used commonly in SSL(Secure Sockets Layer) and TLS (Transport Layer Security) implementations.
  - It should be noted that SHA-1 is now considered insecure and is being deprecated by certificate authorities.

Secure Hash Algorithms (SHAs)

- **SHA-2**:
  - This category includes four functions defined by the number of bits of the hash:
  - SHA-224, SHA-256, SHA-384 and SHA-512.
- **SHA-3**:
  - This is the latest family of SHA functions.
  - SHA3-224, SHA3-256, SHA3-384 and SHA3-512 are members of this family.
  - SHA3 is a NIST-standardized version of Keccak.
  - Keccak uses a new approach called *sponge construction* instead of the commonly used Merkle-Damgard transformation.

Secure Hash Algorithms (SHAs)

- **RIPEMD**:
  - RIPEMD is the acronym for *RACE Integrity Primitives Evaluation Message Digest*.
  - It is based on the design ideas used to build MD4.
  - There are multiple versions of RIPEMD, including 128-bit, 160-bit, 256-bit, and 320-bit.
- **Whirlpool**:
  - This is based on a modified version of Rijndael cipher known as W.
  - It uses the Miyaguchi-Preneel compression function, which is a type of one-way function used for the compression of two fixed length inputs into a single fixed length output.
  - It is a single block length compression function:

Applications

- Hash functions have many practical applications ranging from simple file integrity checks and password storage to be used in cryptographic protocols and algorithms.
- Applications: hash tables, distributed hash tables, bloom filters, virus finger printing, peer-to-peer P2P file sharing, and many other applications.
- In blockchain, hash functions play a very vital role.
  - Especially, the **proof of work function uses SHA-256** twice in order to verify the computational effort spent by miners.
  - **RIPEMD 160 is used to produce bitcoin addresses**.

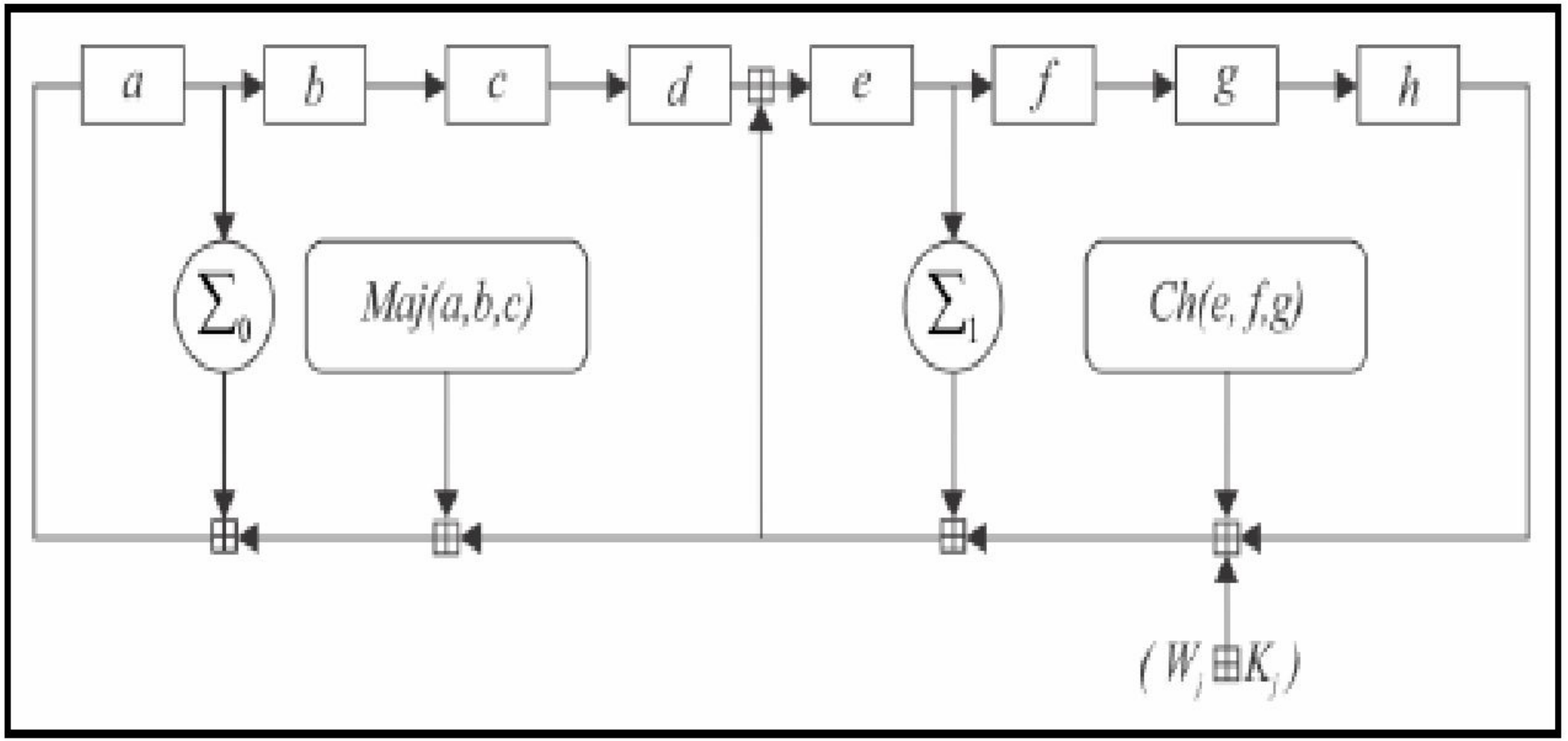
Design of Secure Hash Algorithms (SHA): SHA-256

- SHA-256 has the input message size < 2^64-bits.
- Block size is 512-bits and has a word size of 32-bits.
  - Output is 256-bit digest.
- The compression function processes a 512-bit message block and a 256-bit intermediate hash value.
- There are two main components of this function:
  - compression function
  - message schedule.

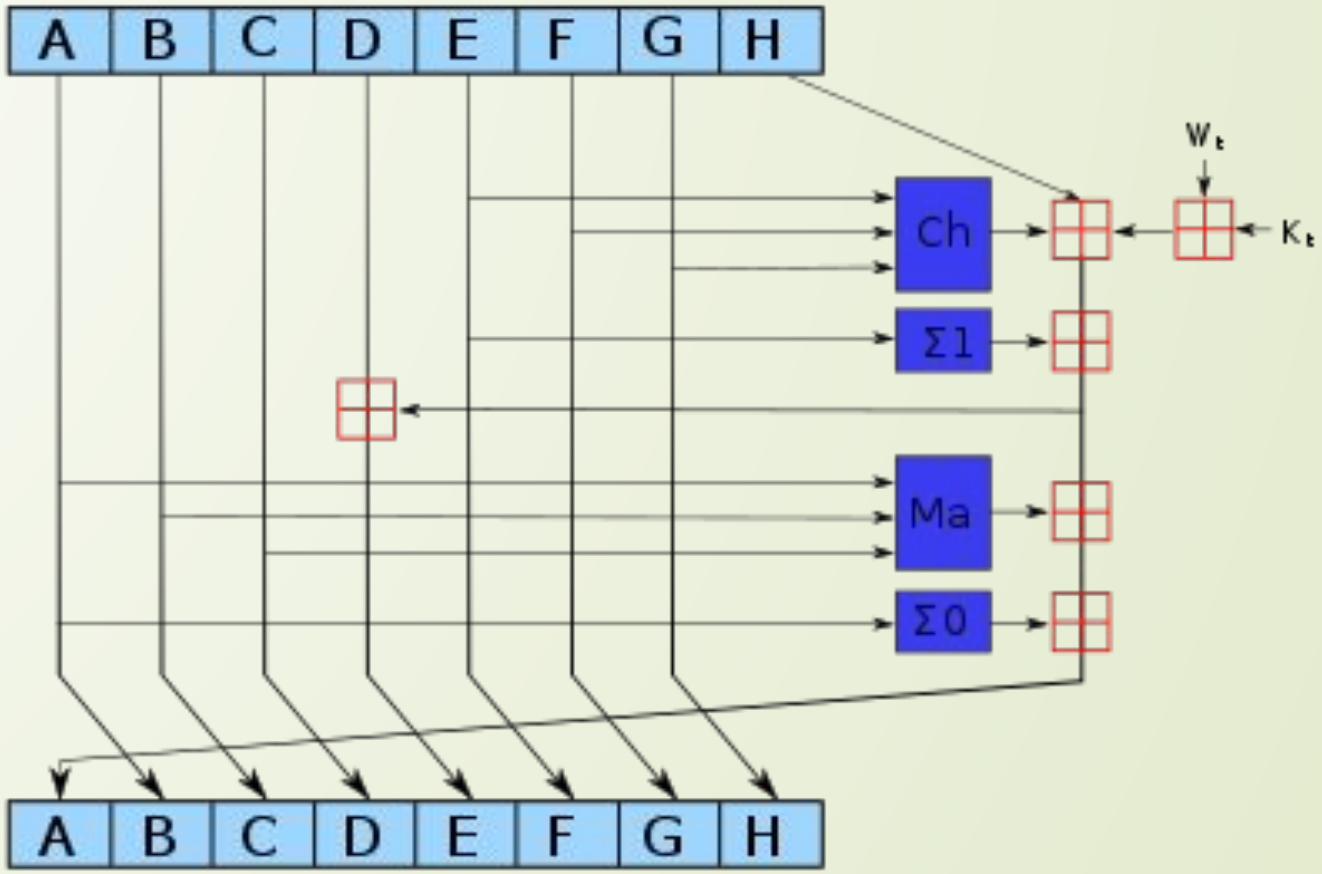
The algorithm works as follows (SHA-256):

- **Pre-processing:**
  - Padding of the message
    - To make the length of a block to 512-bits if it is smaller than the required block size of 512-bits.
  - Parsing the message into message blocks
    - Ensure that the message and its padding is divided into equal blocks of 512-bits.
  - Setting up the initial hash value,
    - which is the eight 32-bit words obtained by taking the first 32-bits of the fractional parts of the square roots of the first eight prime numbers.
    - These initial values are randomly chosen in order to initialize the process
      - Gives a level of confidence that no backdoor exists in the algorithm.

- **Hash computation**:
  - Each message block is processed in a sequence and requires 64 rounds to compute the full hash output.
  - Each round uses slightly different constants to ensure that no two rounds are the same.
  - Message schedule is prepared.
  - Eight working variables are initialized.
  - Intermediate hash value is calculated.
  - Message is processed and the output hash is produced:



one round of SHA 256 compression function


A
B
C
D
E
F
G
H
Ch
Σ1
Ma
Σ0
W_t
K_t
A
B
C
D
E
F
G
H

One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.

The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

~~SHA Versions~~

	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

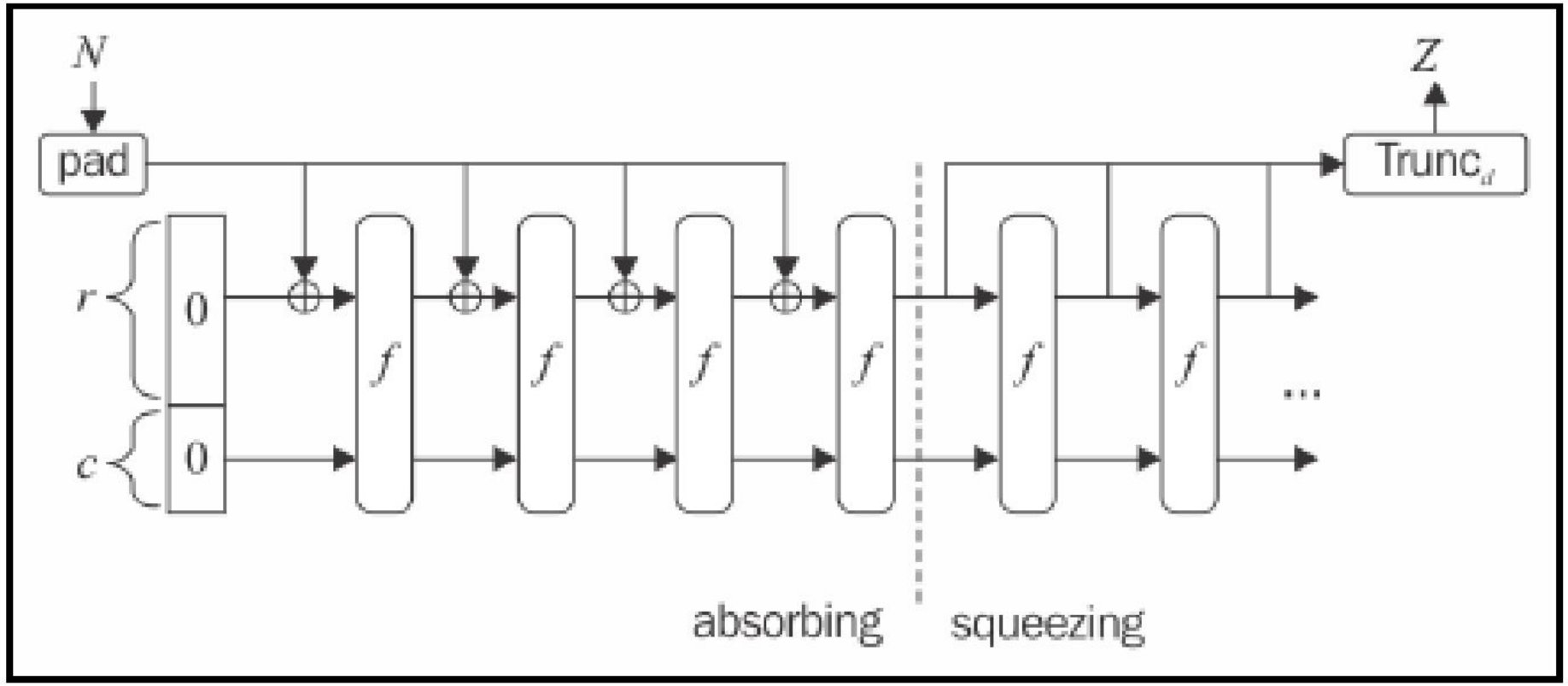
Design of SHA3 (Keccak)

- The structure of SHA-3 is very different from the usual SHA-1 and SHA-2.
- The key idea behind SHA-3 is based on un-keyed permutations
  - as opposed to other usual hash functions constructions that used keyed permutations.
- Keccak also does not make use of the Merkle-Damgard transformation
- New approach called sponge and squeeze construction is used in Keccak,
  - which is basically a random permutation model.

Design of SHA3

- Different variants of SHA3 have been standardized,
  - SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, and SHAKE256.
  - XOF functions that allow the output to be extended to any desired length.

Absorbing and squeezing function in SHA3



Absorbing and squeezing function in SHA3

- As an analogy to sponge, first,
  - Data is absorbed into the sponge after applying padding,
  - Data is then changed into a subset of permutation state using XOR
  - Output is squeezed out of the sponge function that represents the transformed state.
- Rate is the input block size of a sponge function,
- Capacity determines the generic security level:

Message Authentication codes (MACs)

- MACs are sometimes called keyed hash functions
  - Can be used to provide message integrity and authentication.
- In others words, they are used to provide data origin authentication.
- **MACs** are symmetric cryptographic primitives using a shared key between the sender and the receiver.
- MACs can be constructed using block ciphers or hash functions.

MACs using block ciphers

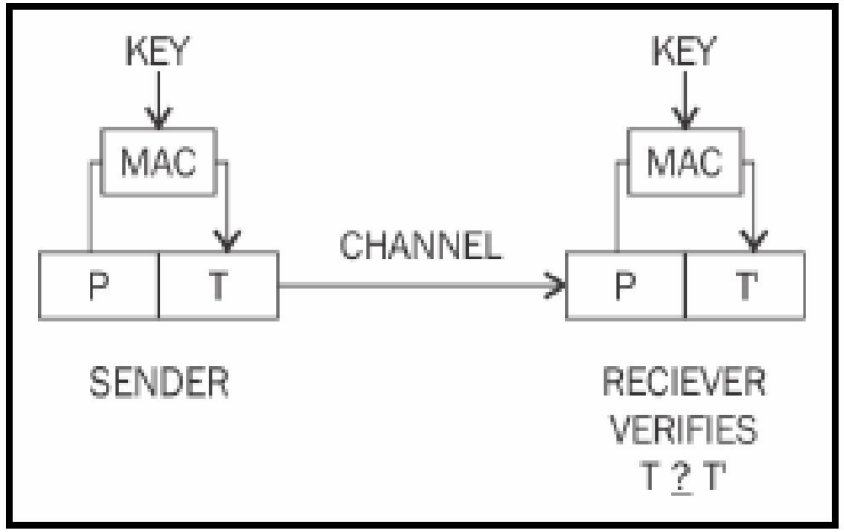
- Block ciphers are used in the **Cipher block chaining mode** (**CBC mode**) in order to generate a MAC.
  - AES in the CBC mode-can be used.
- The MAC of the message is in fact the output of the last round of the CBC operation.
- The length of the MAC output is the same as the block length of the block cipher used to generate MAC.
- MACs are verified simply by computing the MAC of the message and comparing it with the received MAC.
  - If they are the same, then the message integrity is confirmed;
  - otherwise, the message is considered altered.
- MACs work like digital signatures, but they cannot provide the nonrepudiation service due to their symmetric nature.

HMACs (hash-based MACs)

- Similar to the hash function, they produce a fixed length output and take an arbitrarily long message as the input.
- Sender signs a message using MAC
- Receiver verifies it using the shared key.
- The key is hashed with the message using either of the two methods known as secret prefix or the secret suffix method.
  - In the first method, the key is concatenated with the message, that is, the key comes first and the message comes after,
  - In the second method, the key comes after the message:

Secret prefix: M = MACk(x) = h(k||x)

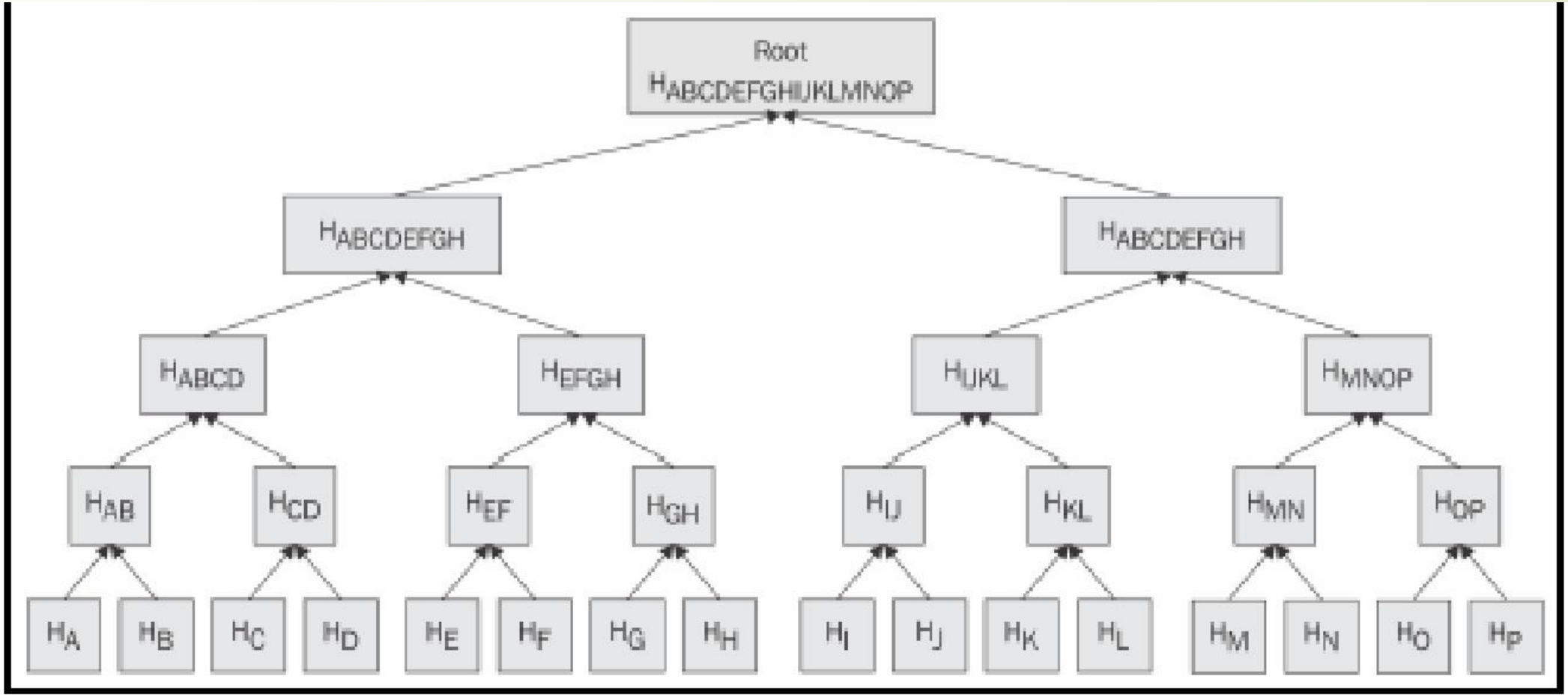
Secret suffix: M=MACk(x) = h(x||k)



Operation of a MAC function

Merkle trees

- The concept of Merkle tree was introduced by *Ralph Merkle*.
- A visualization of Merkle tree makes it easy to understand.
- Merkle trees allow secure and efficient verification of large data sets.
- It is a binary tree in which first,
  - inputs are placed at the leaves (node with no children),
  - values of pairs of child nodes are hashed together in order to produce a value for the parent node (internal node) until a single hash value known as Merkle root is achieved:



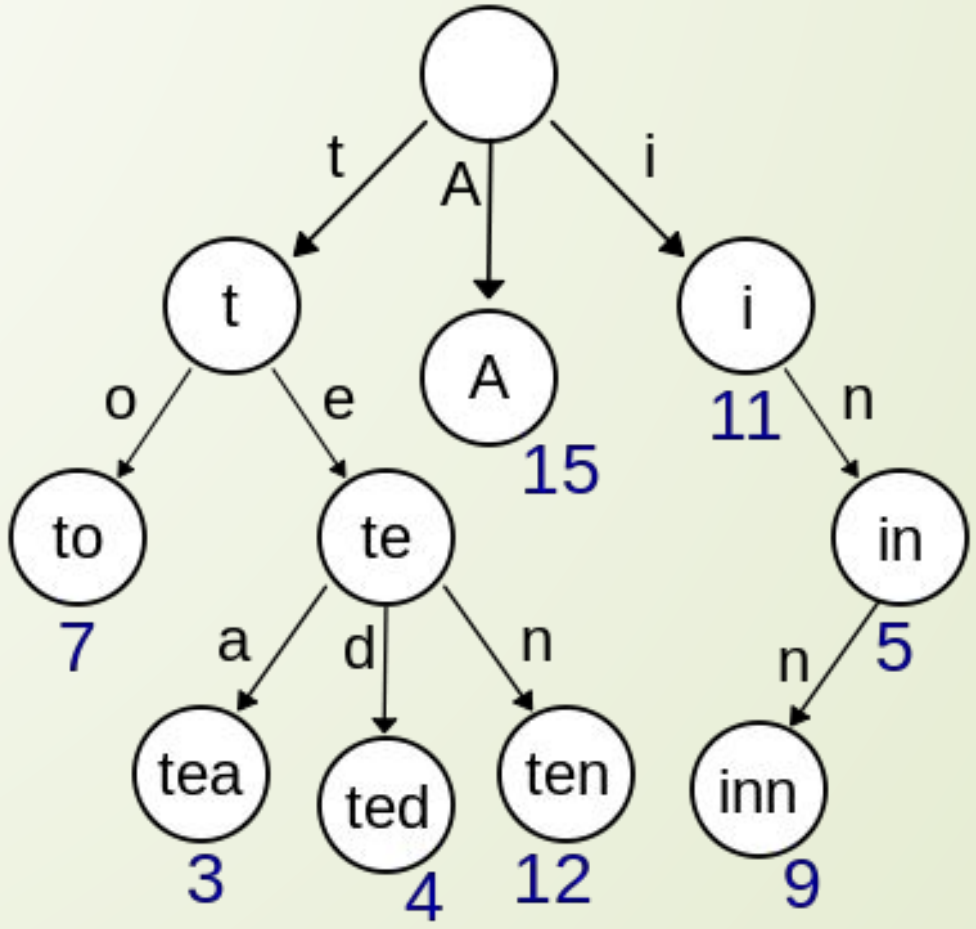
A Merkle tree

Merkle trees

- A Merkle tree, in the most general sense, is a way of hashing a large number of “chunks” of data together which relies on splitting the chunks into buckets,
  - where each bucket contains only a few chunks,
  - then taking the hash of each bucket and repeating the same process,
  - continuing to do so until the total number of hashes remaining becomes only one: the root hash.

Patricia trees

- Trie or a digital tree is an ordered tree data structure used to store a dataset.
- Practical Algorithm to Retrieve Information Coded in Alphanumeric (Patricia), also known as Radix tree
  - Compact representation of a trie in which a node that is the only child of a parent is merged with its parent.
- Merkle-Patricia tree, based on the definitions of Patricia and Merkle, is a tree that has a root node that contains the hash value of the entire data structure.


t
A
i
t
A
i
o
e
11
n
15
to
te
in
7
a
d
n
n
5
tea
ted
ten
inn
3
4
12
9

Patricia trees

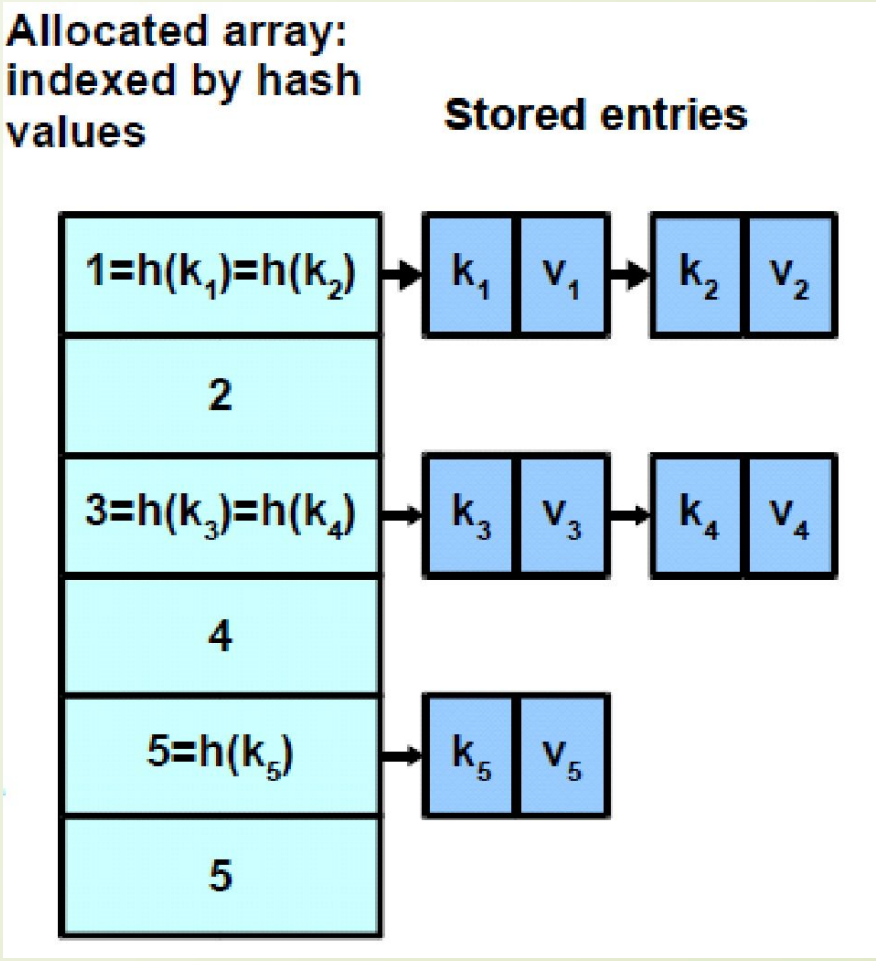
- Patricia trie is a data structure which is also called Prefix tree, radix tree or trie
- Trie uses a key as a path so the nodes that share the same prefix can also share the same path.
- This structure is fastest at finding common prefixes, simple to implement, and requires small memory.
- it is commonly used for implementing routing tables, systems that are used in low specification machines like the router.

Merkle Patricia Trie

- In the MPT, as well as in the Merkle tree, every node has a hash value.
- Each node's hash is decided by the sha3 hash value of its contents.
- This hash is also used as the key that refers to the node.
- Go-ethereum uses levelDB, and parity uses rocksDB to store states.
  - They are key-value storages.
  - Keys and values saved in the storage are not the key-values of the Ethereum state.
  - The value that is stored in the storage is the content of MPT node while the key is the hash of this node.

Hash Tables

- Store arbitrary keys and satellite data (value)
  - put(key,value)
  - value = get(key)
- Lookup must be fast
  - Calculate hash function h() on key that returns a storage cell
  - Chained hash table: Store key there

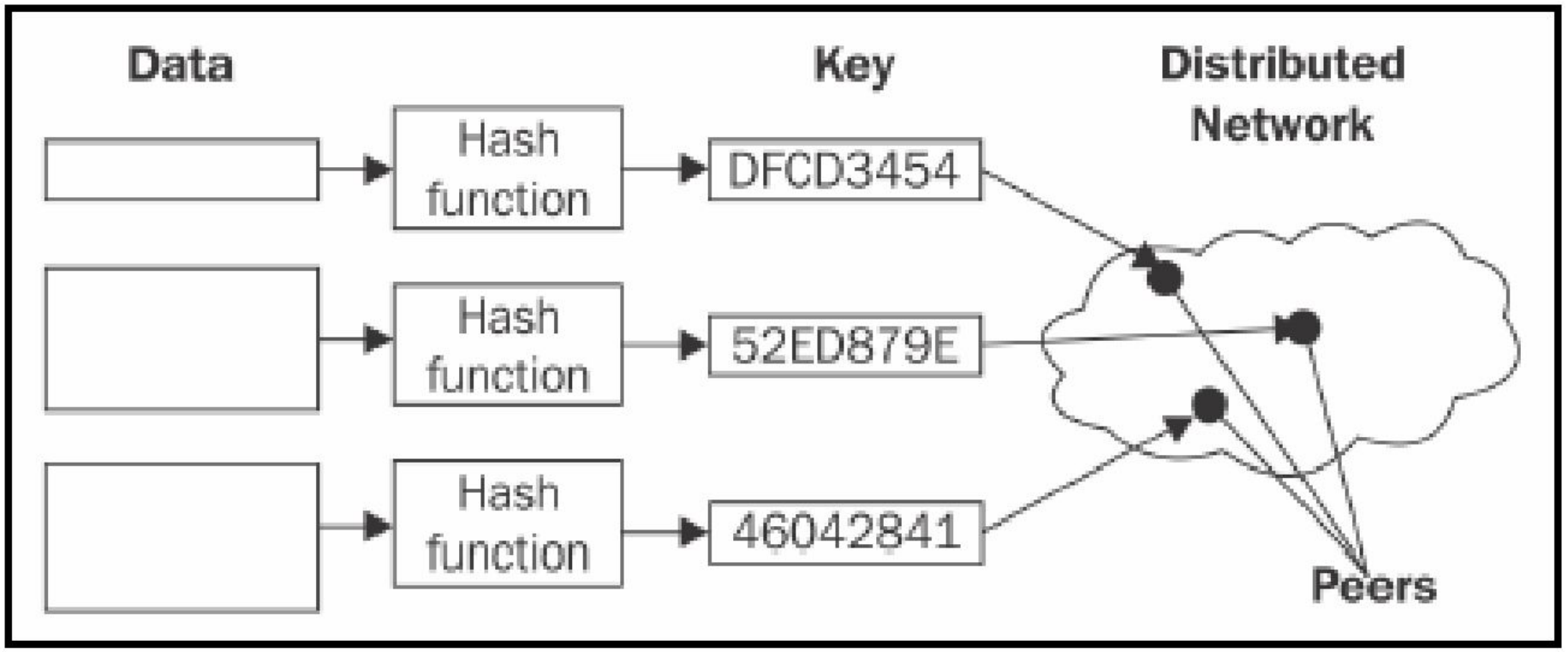


Distributed hash tables (DHTs)

- A hash table is a data structure that is used to map keys to values.
- Internally, a hash function is used to calculate an index into an array of buckets, from which the required value can be found.
- Buckets have records stored in them using a hash key and are organized in a particular order.
- Distributed hash table as a data structure where data is spread across various nodes and nodes are equivalent to buckets in a peer-to-peer to network.

What is a DHT?

- Hash Table
  - data structure that maps “keys” to “values”
  - essential building block in software systems
- Distributed Hash Table (DHT)
  - similar, but spread across many hosts
- Interface
  - insert(key, value)
  - lookup(key)



Distributed hash tables

Distributed hash tables (DHTs)

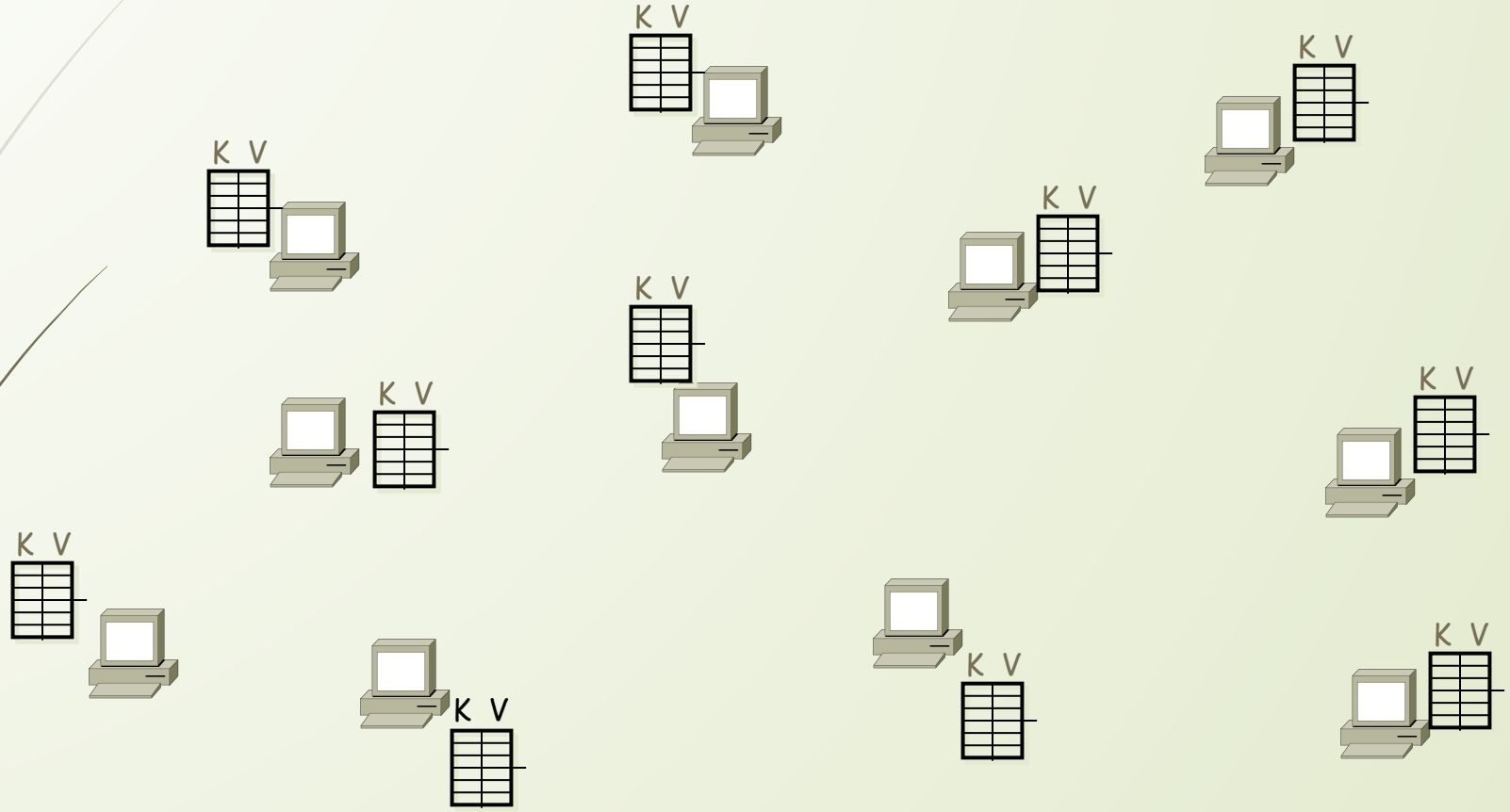
- Data is passed through a hash function, which results in generating a compact key.
- This key is then linked with the data (values) on the peer-to-peer network.
- When users on the network request the data (via the filename), the filename can be hashed again to produce the same key
  - any node on the network can then be requested to find the corresponding data.
- DHTs provides decentralization, fault tolerance, and scalability:

How do DHTs work?

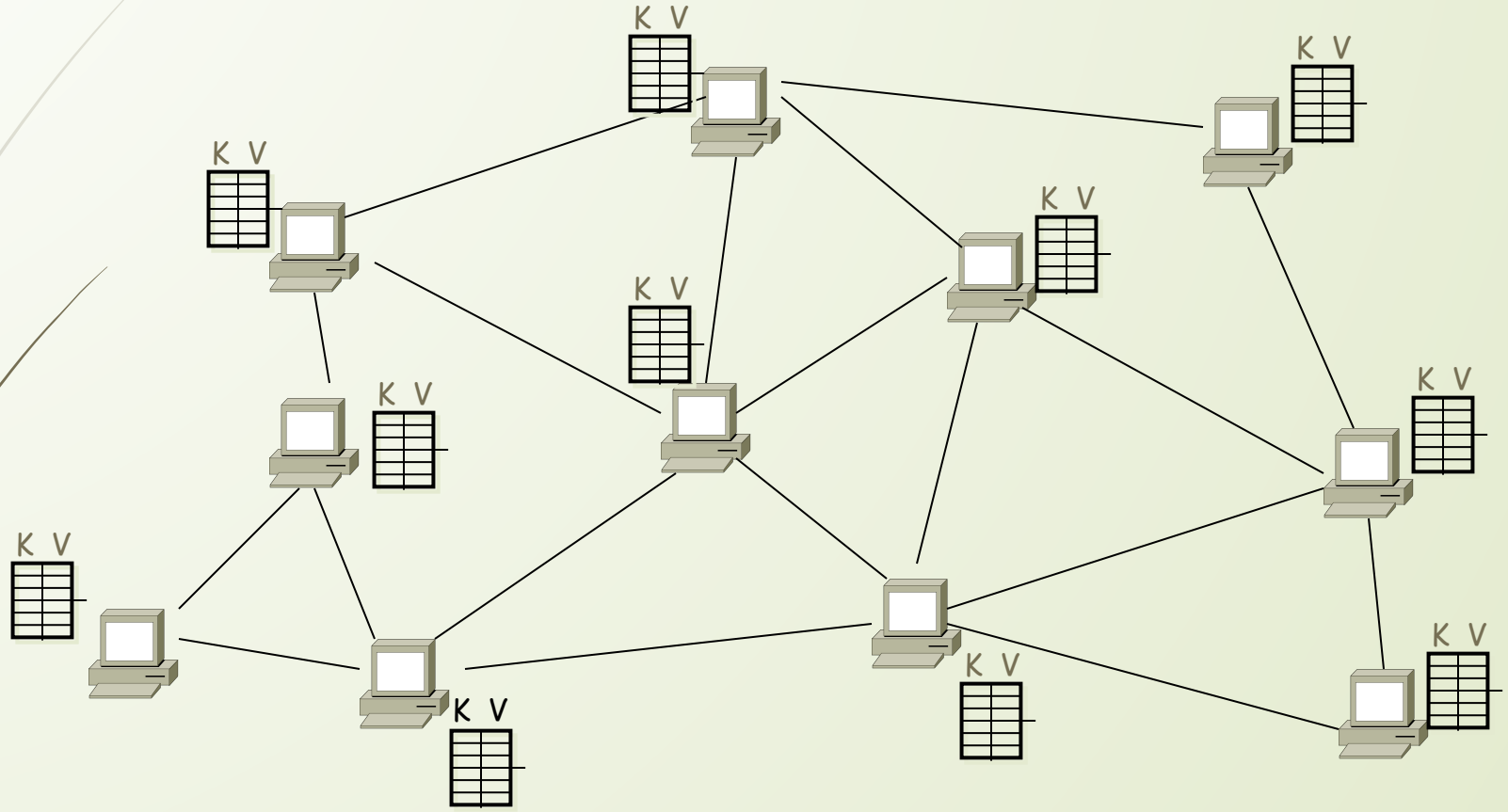
Every DHT node supports a single operation:

- Given *key* as input; route messages to node holding *key*
  - DHTs are *content-addressable*

DHT: basic idea

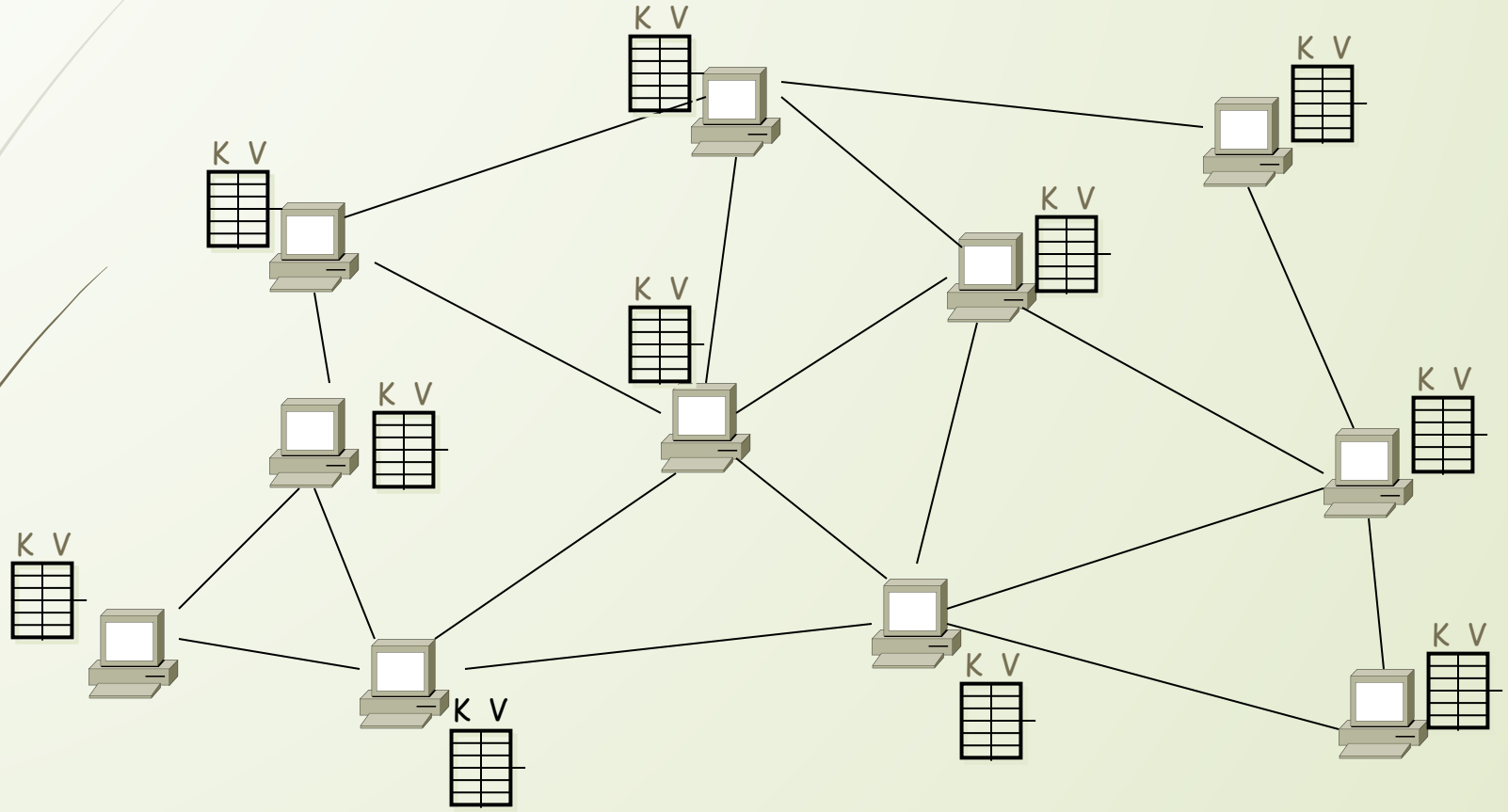


DHT: basic idea



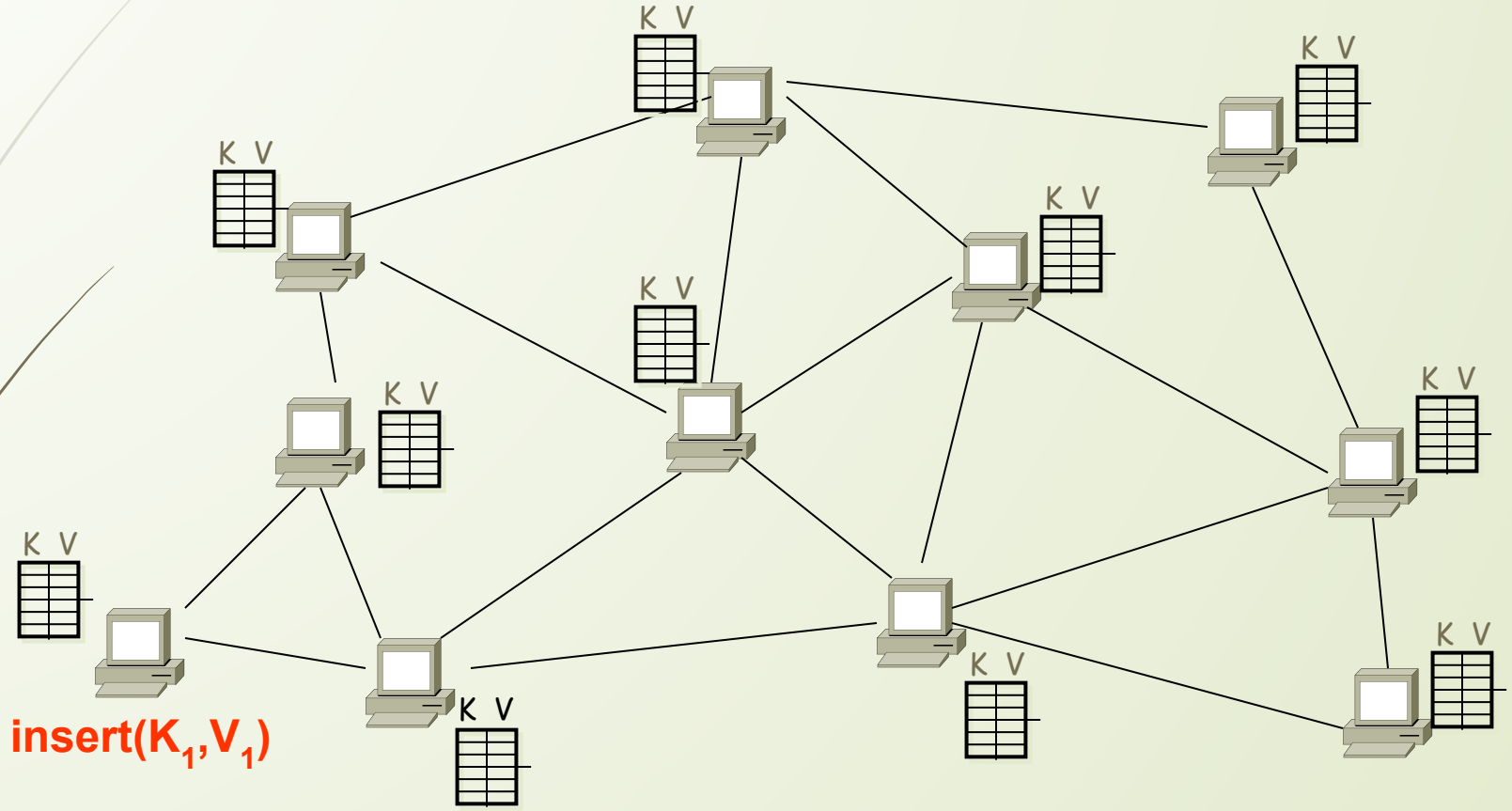
Neighboring nodes are "connected" at the application-level

DHT: basic idea



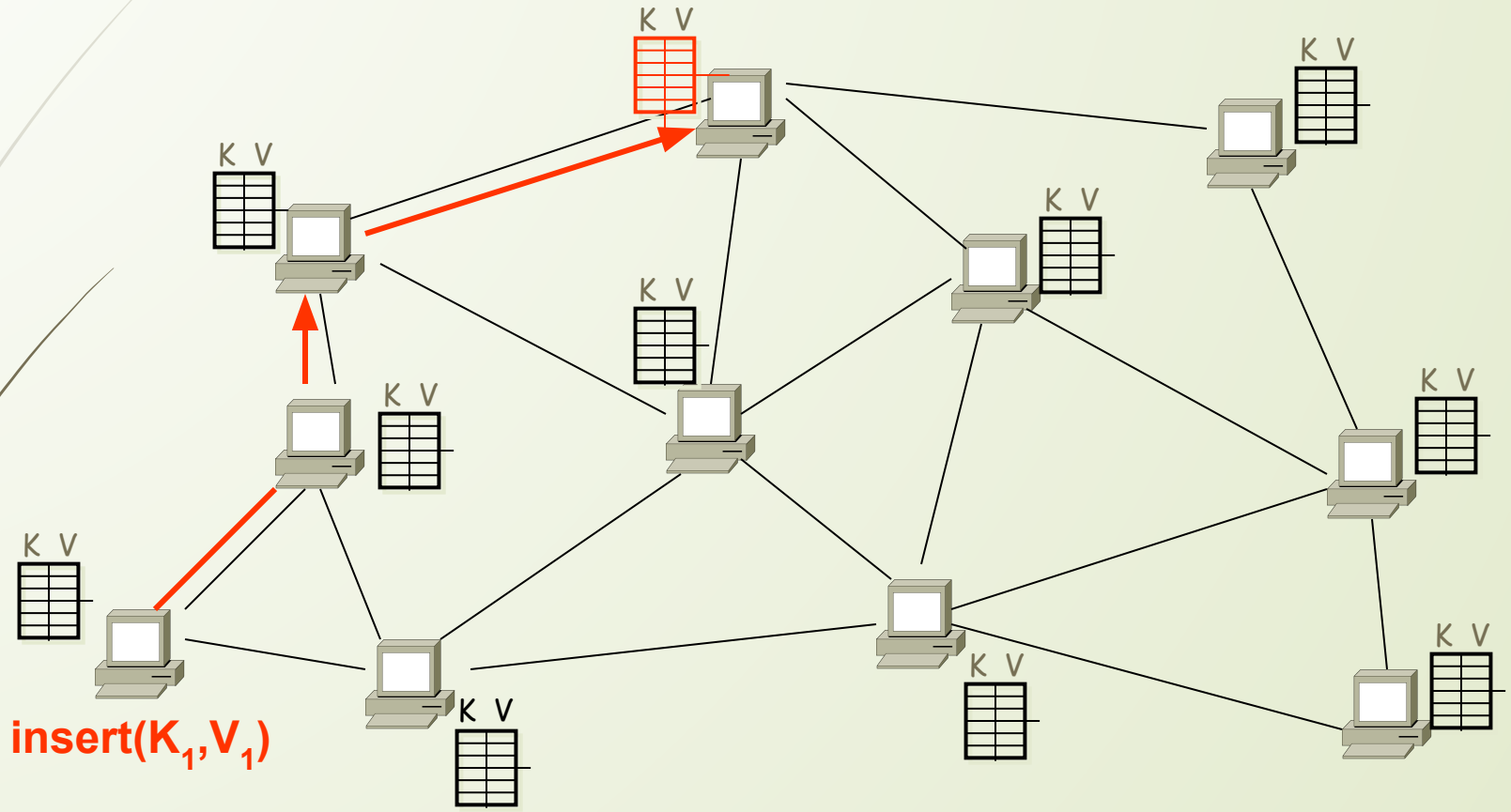
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



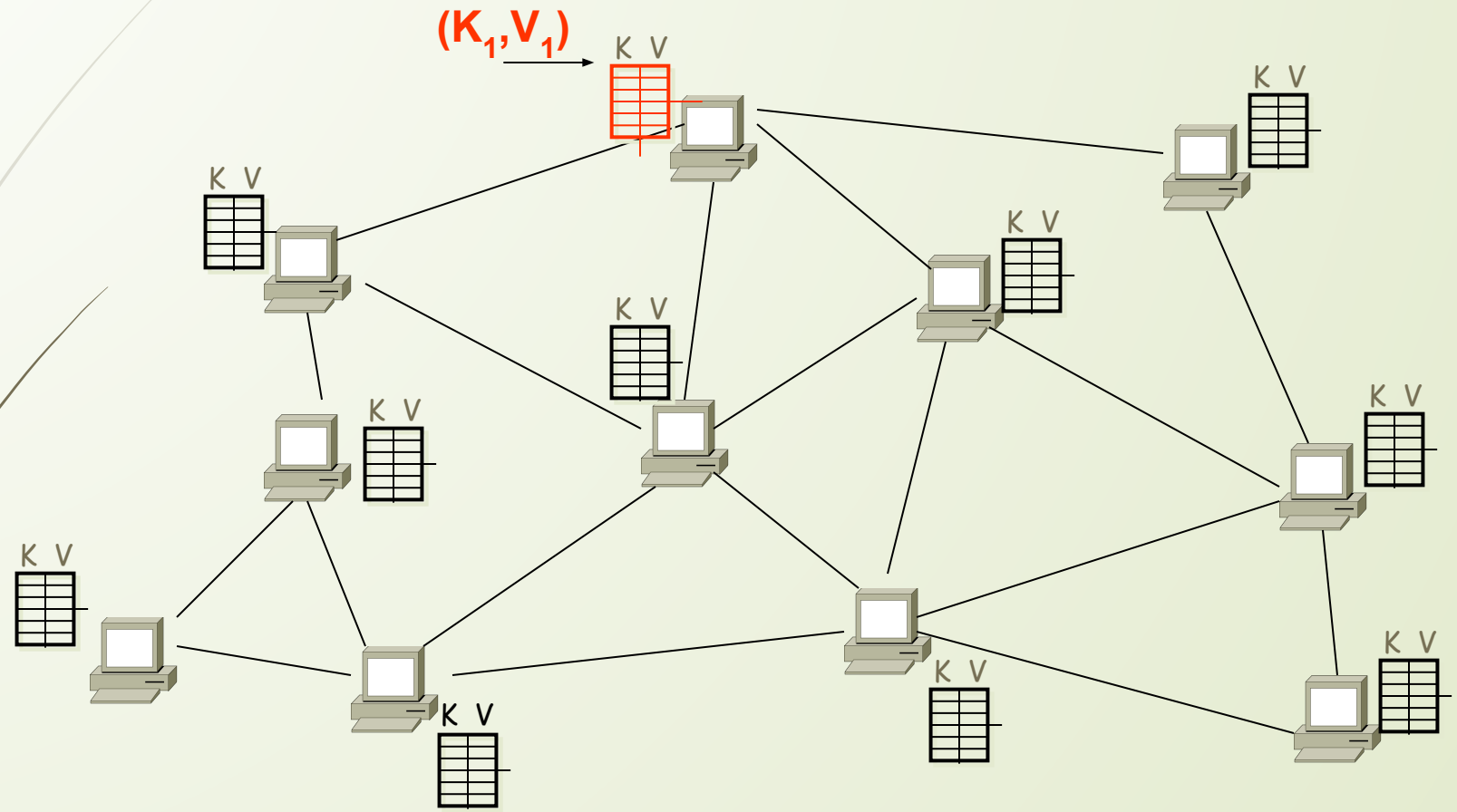
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



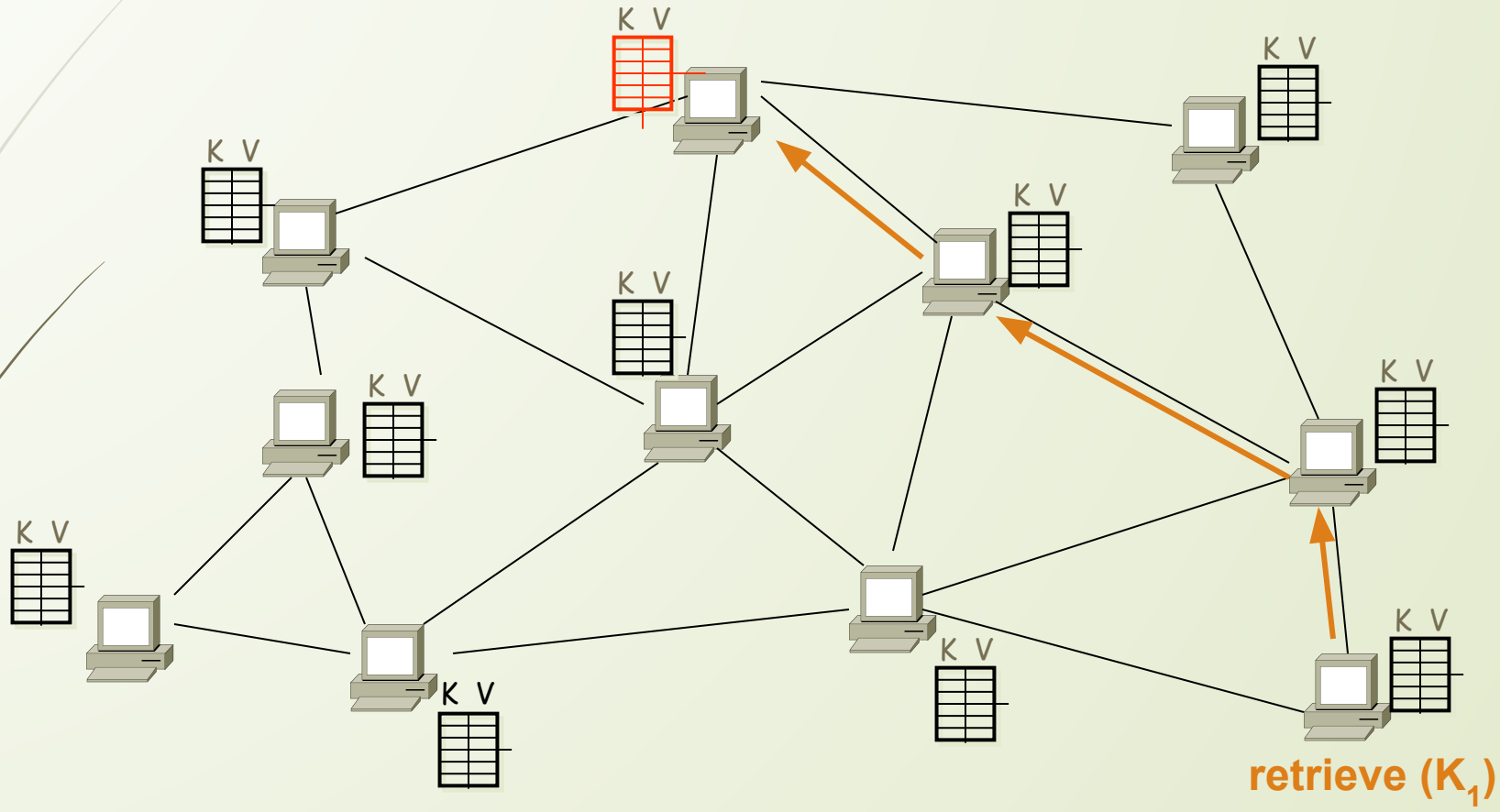
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



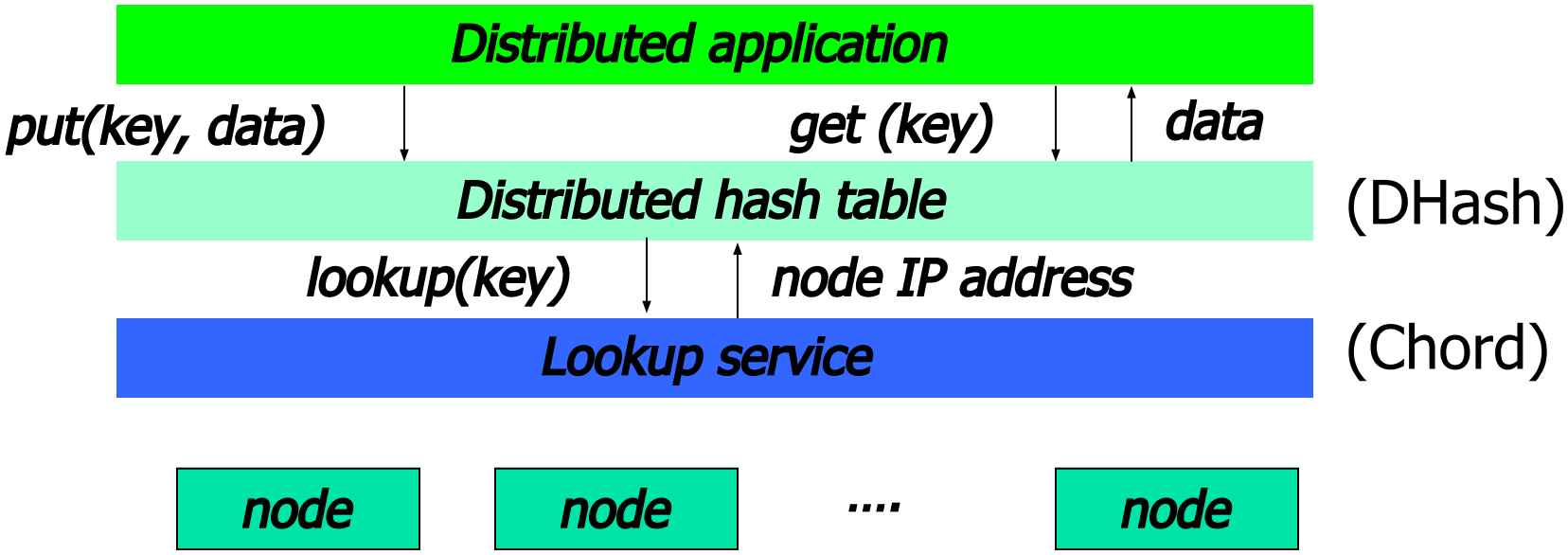
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



Operation: take *key* as input; route messages to node holding *key*

DHT



- Application may be distributed over many nodes
- DHT distributes data storage over many nodes

Digital signatures

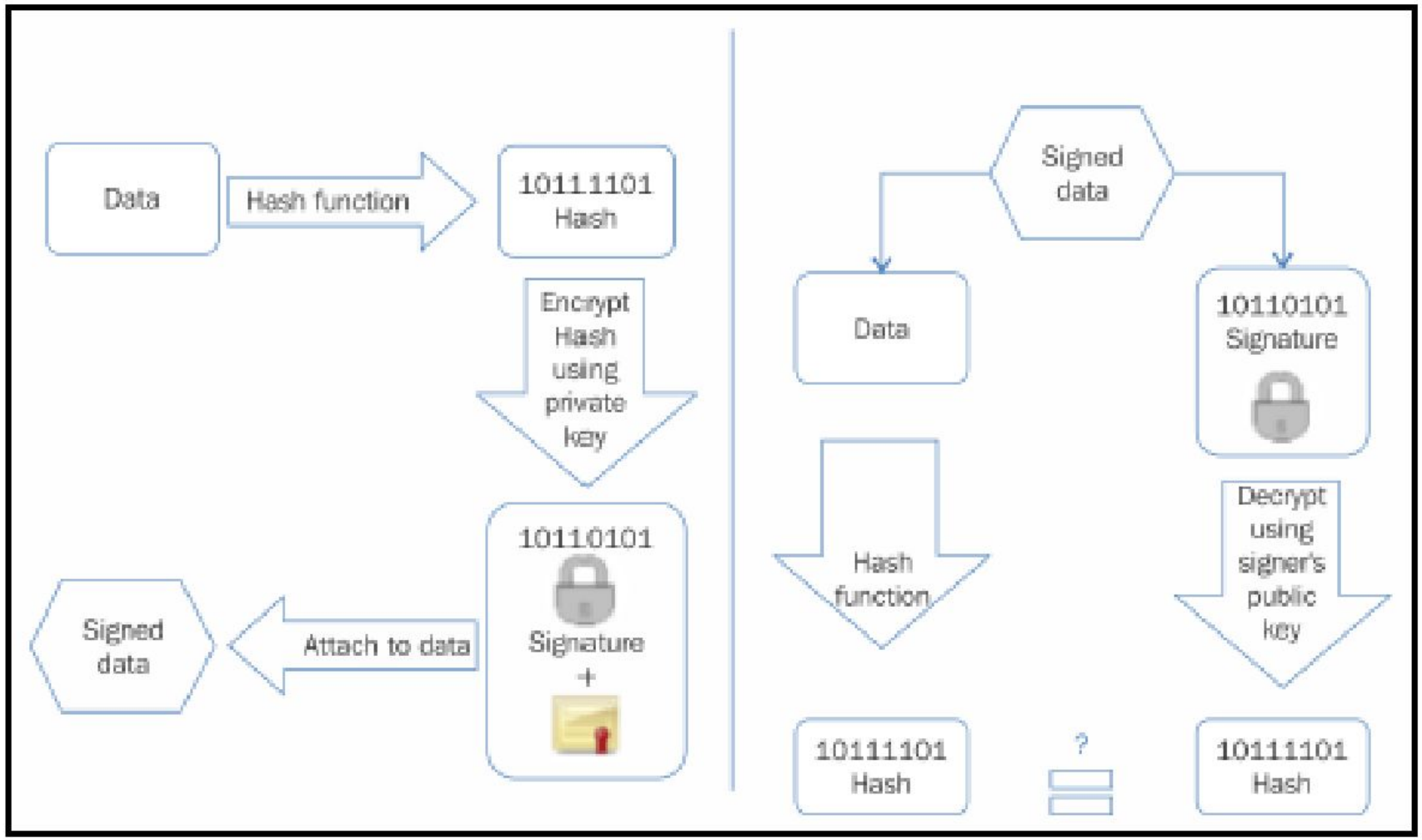
- Digital signatures provide a means of associating a message with an entity from which the message has been originated.
- Digital signatures are used to provide data origin authentication and non- repudiation.
- They are calculated in two steps.

Digital signatures

- Calculate the hash value of the data packet.
  - This will provide the data integrity guarantee as hash can be computed at the receiver's end again and matched with the original hash to check whether the data has been modified in transit.
  - Message signing can work without hashing the data first, but is not considered secure.
- Signs the hash value with the signer's private key.
  - As only the signer has the private key, the authenticity of the signature and the signed data is ensured.

Digital signatures

- Digital signatures have some important properties, such as
  - Authenticity
  - unforgeability,
  - nonreusability.
- **Authenticity** means that the digital signatures are verifiable by a receiving party.
- **Unforgeability** property ensures that only the sender of the message is able to use the signing functionality using the private key.
  - no one else should be able to produce the signed message that has been produced by the legitimate sender.
- **Nonreusability** means that the digital signature cannot be separated from a message and used for another message again.


Data
Hash function
10111101
Hash
Encrypt
Hash
using
private
key
10110101
Signature
+
Attach to data
Signed
data
Signed
data
Data
10110101
Signature
Hash
function
Decrypt
using
signer's
public
key
10111101
Hash
?
10111101
Hash

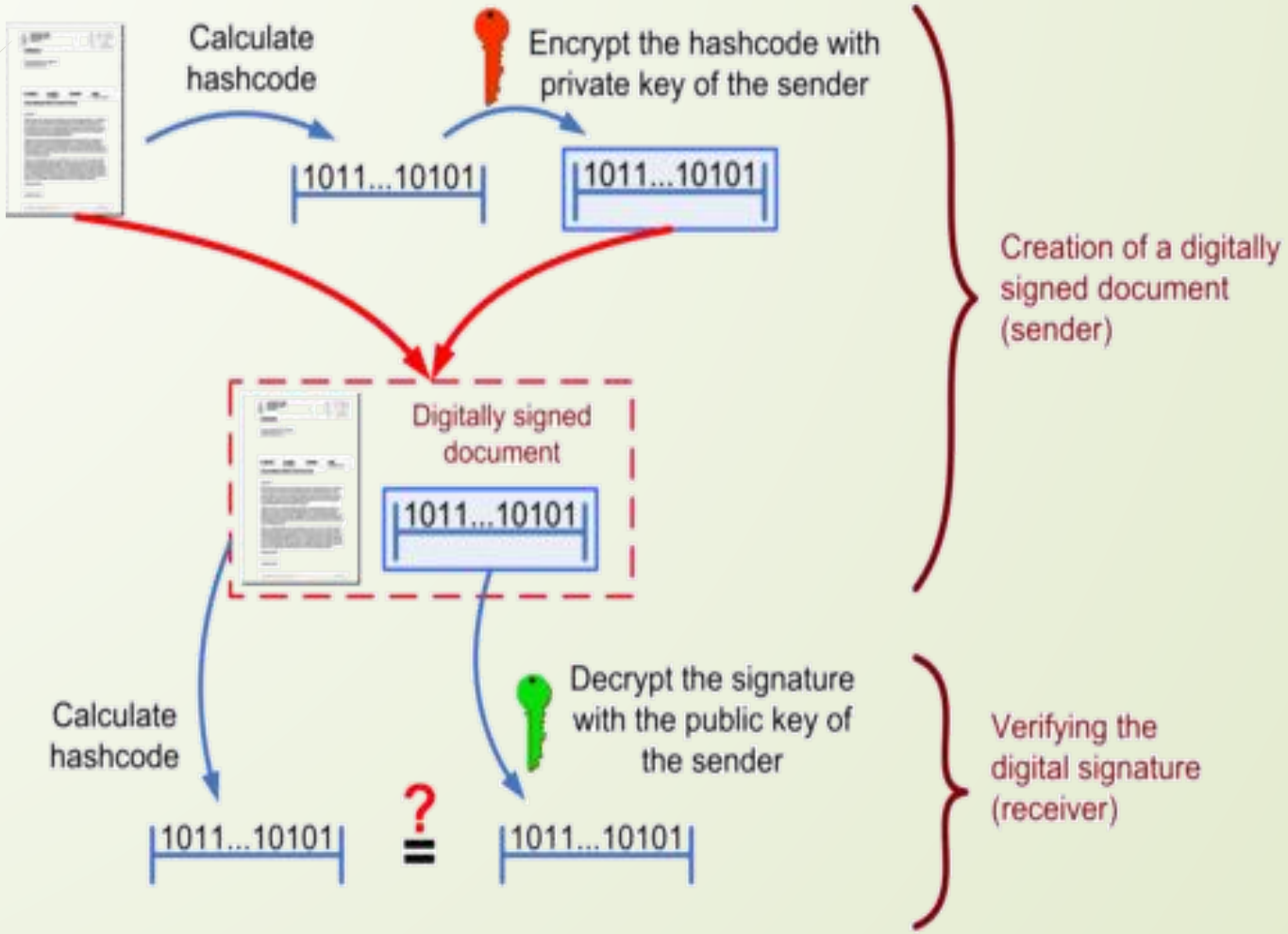
Digital signatures

- If a sender wants to send an authenticated message to a receiver, there are two methods that can be used.
- These two approaches to use digital signatures with encryption are
- **Sign then encrypt**
  - In this approach, the sender digitally signs the data using the private key, appends the signature to the data,
  - Then encrypts the data and the digital signature using the receiver's public key.
- **Encrypt then sign**
  - In this approach, the sender encrypts the data using the receiver's public key
  - Then digitally signs the encrypted data.

Digital signatures

- In practice, a digital certificate that contains the digital signature is issued by a **certificate authority** (**CA**) that associates a public key with an identity.
- Various schemes, such as RSA, Digital Signature Algorithm, and Elliptic Curve Digital Signature Algorithm-based digital signature schemes are used in practice.
- RSA is the most commonly used;
- With the traction of elliptic curve cryptography, ECDSA-based schemes are also becoming quite popular.

Creating and verifying a digital signature



If the calculated hashcode does not match the result of the decrypted signature, either the document was changed after signing, or the signature was not generated with the private key of the alleged sender.

Fully Homomorphic Encryption

The goal

Delegate **processing** of data

without giving away **access** to it

Example 1: Private Search

Delegate PROCESSING of data

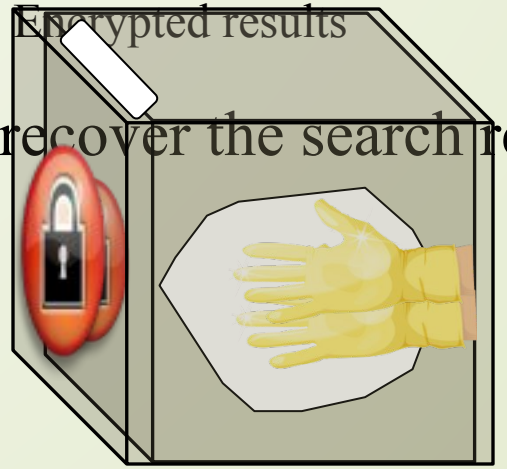
without giving away ACCESS to it

- **You:** Encrypt the query, send to Google

  (Google does not know the key, cannot "see" the query)

- **Google:** Encrypted query → Encrypted results

  (You decrypt and recover the search results)



Example 2: Private Cloud Computing

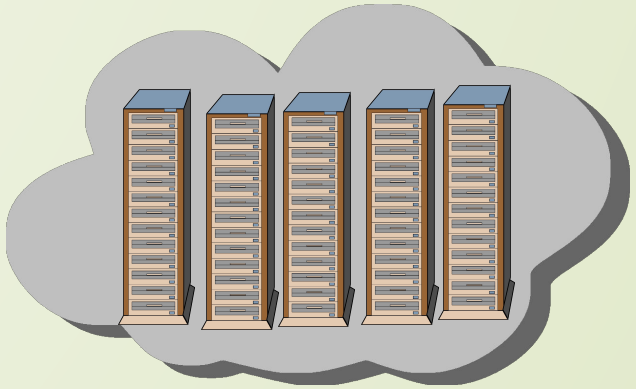
Delegate PROCESSING of data

without giving away ACCESS to it

(Input: x)

Encrypt x

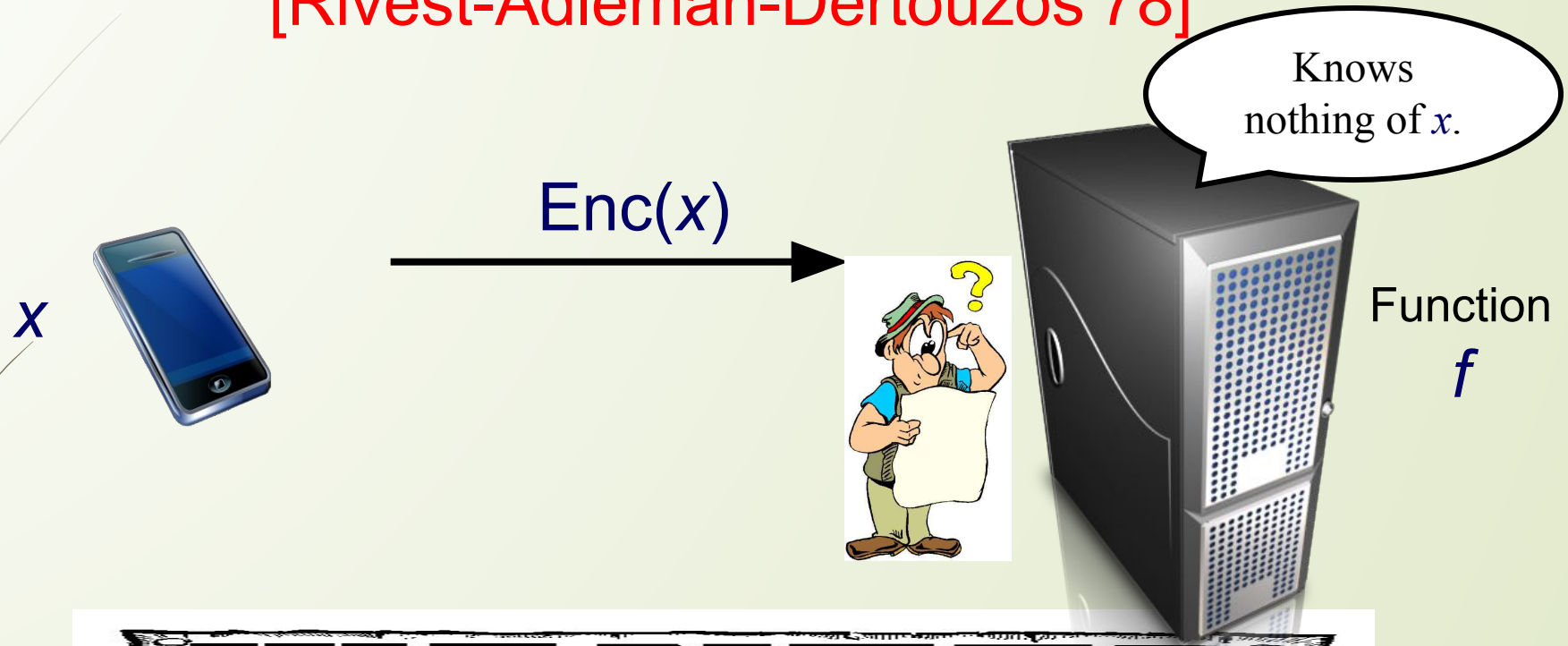
Enc(x), P → Enc(P(x))



(Program: P)

Fully Homomorphic Encryption

[Rivest-Adleman-Dertouzos'78]



WANTED

Eval: f, Enc(x) → Enc$(f(x))$

homomorphic evaluation

Homomorphic encryption

- Usually, public key cryptosystems, such as RSA, are multiplicative homomorphic or additive homomorphic, such as Paillier cryptosystem, and are called **partially homomorphic** systems.
- Additive PHEs are suitable for e-voting and banking applications.
- In 2009, a **fully homomorphic** system was discovered by *Craig Gentry*.
- As these schemes allow the processing of encrypted data without the need for decryption,
  - for example, cloud computing and online search engines.
- Recent development in homomorphic encryption has been very promising
  - researchers are actively working to make it efficient and more practical.
- This is of particular interest in the blockchain technology, because it can solve the problem of confidentiality and privacy in blockchain.

Signcryption

- Signcryption is a public key cryptography primitive that provides all the functions of the digital signature and encryption.
- It was invented by Yuliang Zheng and is now an ISO standard ISO/IEC 29150:2011.
- Traditionally, signature then encrypt or encrypt then sign schemes are used to provide unforgeability, authentication, and nonrepudiation,
  - But with Signcryption, all services of digital signatures and encryption are provided with cost less than that of sign then encrypt schemes.
- Cost (signature & encryption) << Cost (signature) + Cost (Encryption)

Features of Digital Signcryption

- **Unique Unsigncryptability**
  - message *m* of arbitrary length is Signcrypted using Signcryption algorithm
  - This gives you a Signcrypted output *c*
  - The receiver can apply Unsigncryption algorithm on c to verify the message *m*

    This Unsigncryption is unique to the message m and the sender

Features of Digital Signcryption

- **Security**

  - Two security schemes
    - Digital Signature
    - Public Key encryption
  - *likely to be more secure*

  - ensures that the message sent couldn't be forged
  - ensures the contents of the message are confidential
  - ensures non-repudiation

Features of Digital Signcryption

- **Efficiency**

  Computation involved when applying the Signcryption, Unsigncryption algorithms and communication overhead is much smaller than signature-then-encryption schemes

Zero knowledge proofs

- Zero knowledge proofs were introduced by GoldWasser, Micali, and Rackoff.
- These proofs are used to **prove the validity of an assertion without revealing any information** whatsoever about the assertion.
- There are three properties of ZKPs that are required, namely
  - completeness, soundness, and zero-knowledge property.
- **Completeness** ensures that if a certain assertion is true, then the verifier will be convinced of this claim by the prover.
- **Soundness** property makes sure that if an assertion is false, then no dishonest prover can convince the verifier otherwise.
- **Zero-knowledge property**, is the key property of zero knowledge proofs whereby it is ensured that absolutely nothing is revealed about the assertion except whether it is true or false.

Zero knowledge proofs

- Zero knowledge proofs have sparked a special interest among researchers in the blockchain space due to its privacy properties that are very much desirable in financial and many other fields, such as law and medicine.
- A recent example of the successful implementation of the zero knowledge proof mechanism is the Zcash crypto currency.
- In Zcash, a specific type of zero knowledge proof, known as **zero-knowledge Succinct Non-interactive Argument of Knowledge** (**ZK-Snark**), is implemented.

Blind signatures

- Blind signatures were introduced by *David Chaum* in 1982 and are based on public key digital signature schemes, such as RSA.
- The key idea behind blind signatures is to get the message signed by the signer without actually revealing the message.
- This is achieved by disguising or blinding the message before signing it, hence the name blind signatures.
- This blind signature can then be verified against the original message just like a normal digital signature.
- Blind signatures were introduced as a mechanism to allow the development of digital cash schemes.

Encoding schemes

- Other than cryptographic primitives, binary to text encoding schemes are also used in various scenarios.
- The most common usage is to convert binary data into text so that it can be either processed, saved, or transmitted via a protocol that does not support the processing of binary data.
- For example, sometimes, images are stored in the database as base64 encoding, which allows a text field to be able to store a picture.
- Another encoding named base58 was popularized by its usage in bitcoin.

References

- Imran Bashir. “Mastring BlockChain”, Packt
- Web Materials