Lecture A6:

Hashing for Message Authentication

4th Year Course- CCSIT, UoA

Lecture goals

To understand:

- ❑ What is a hash function?
- ❑ Different ways to use hashing for message authentication
- ❑ The one-way and collision-resistance properties of secure hash functions
- ❑ Structure of cryptographically secure hash functions
- ❑ SHA Series of Hash Functions
- ❑ Message Authentication Codes (MACs)

What is a Hash Function? (1)

- A hash function takes a **variable sized input** message and produces a **fixed-sized output**. The output is usually referred to as the **hash code** or the hash value or the **message digest**.
- For example, the SHA-512 hash function takes for input messages of length up to 2^{128} bits and produces as output a 512-bit message digest (MD). [SHA stands for Secure Hash Algorithm].
- We can think of the hash code as a **fixed-sized fingerprint** of a variable-sized message.

What is a Hash Function? (2)

- Message digests produced by the most commonly used hash functions range in length from 160 to 512 bits depending on the algorithm used.
- Since a message digest depends on all the bits in the input message, any alteration of the input message during transmission would cause its message digest to not match with its original message digest. This can be used to check for forgeries, unauthorized alterations, etc.

Different Ways to Use Hashing for Message Authentication (1)

- Figures 1 and 2 show six different ways in which you could incorporate message hashing in a communication network. These constitute different approaches to protect the hash value of a message. No authentication at the receiving end could possibly be achieved if both the message and its hash value are accessible to an adversary.
- In the symmetric-key encryption based scheme shown in Figure 1(a), the message and its hash code are concatenated together to form a composite message that is then encrypted and placed on the wire. The receiver decrypts the message and separates out its hash code, which is then compared with the hash code calculated from the received message. The hash code provides authentication and the encryption provides confidentiality.

Different Ways to Use Hashing for Message Authentication (2)

- The scheme shown in Figure 1(b) is a variation on Figure 1(a) in the sense that only the hash code is encrypted. This scheme is efficient to use when confidentiality is not the issue but message authentication is critical. Only the receiver with access to the secret key knows the real hash code for the message. So the receiver can verify whether or not the message is authentic.

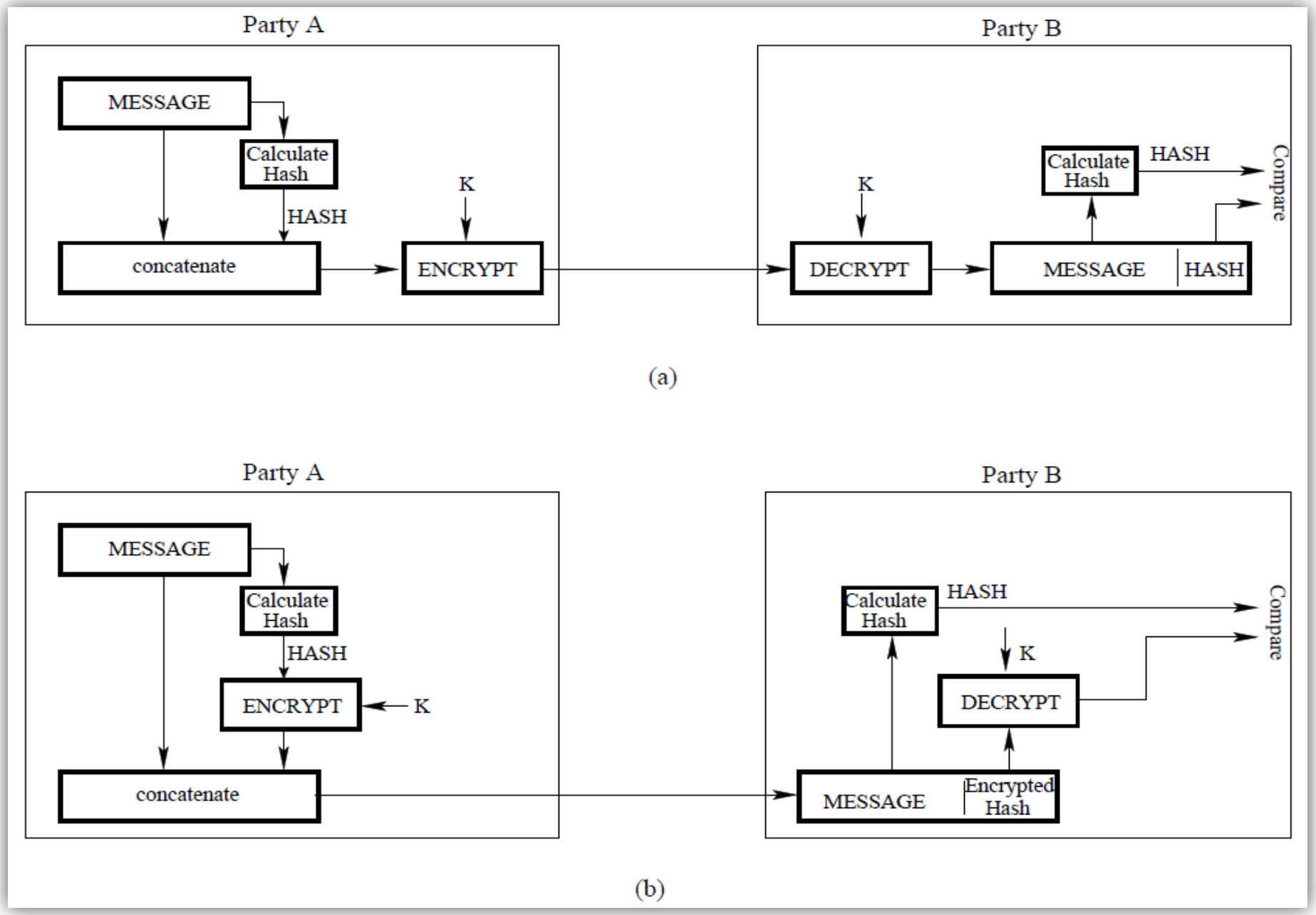

Party A
MESSAGE
Calculate Hash
HASH
K
concatenate
ENCRYPT
Party B
K
DECRYPT
Calculate Hash
HASH
Compare
MESSAGE
HASH
(a)
Party A
MESSAGE
Calculate Hash
HASH
ENCRYPT
K
concatenate
Party B
Calculate Hash
HASH
Compare
K
DECRYPT
MESSAGE
Encrypted Hash
(b)

Figure 1 (a & b)

Different Ways to Use Hashing for Message Authentication (3)

- The scheme in Figure 1(c) is a **public-key encryption version** of the scheme shown in Figure 1(b). The hash code of the message is encrypted with the sender's private key. The receiver can recover the hash code with the sender's public key and authenticate the message as indeed coming from the alleged sender. **Confidentiality again is not the issue here**. The sender encrypting with his/her private key the hash code of his/her message constitutes **the basic idea of digital signatures**.

- If we want to add symmetric-key based confidentiality to the scheme of Figure 1(c), we can use the scheme shown in Figure 2(a). This is a commonly used approach when **both confidentiality and authentication are needed**.

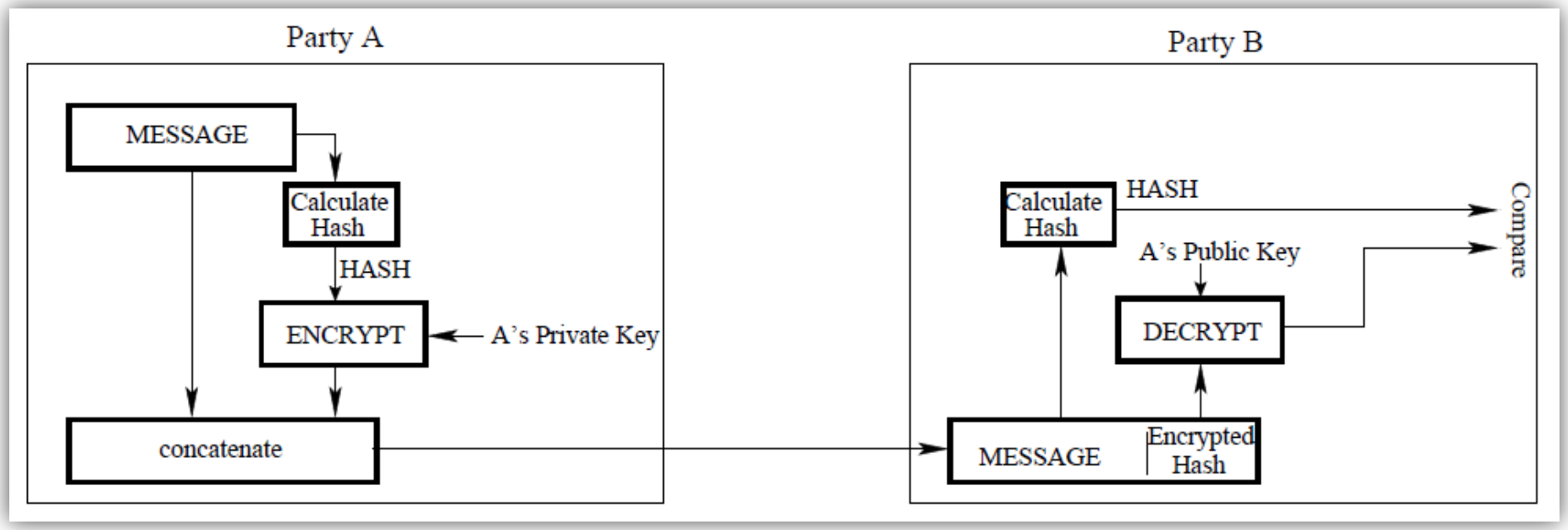


Figure 1 (c)

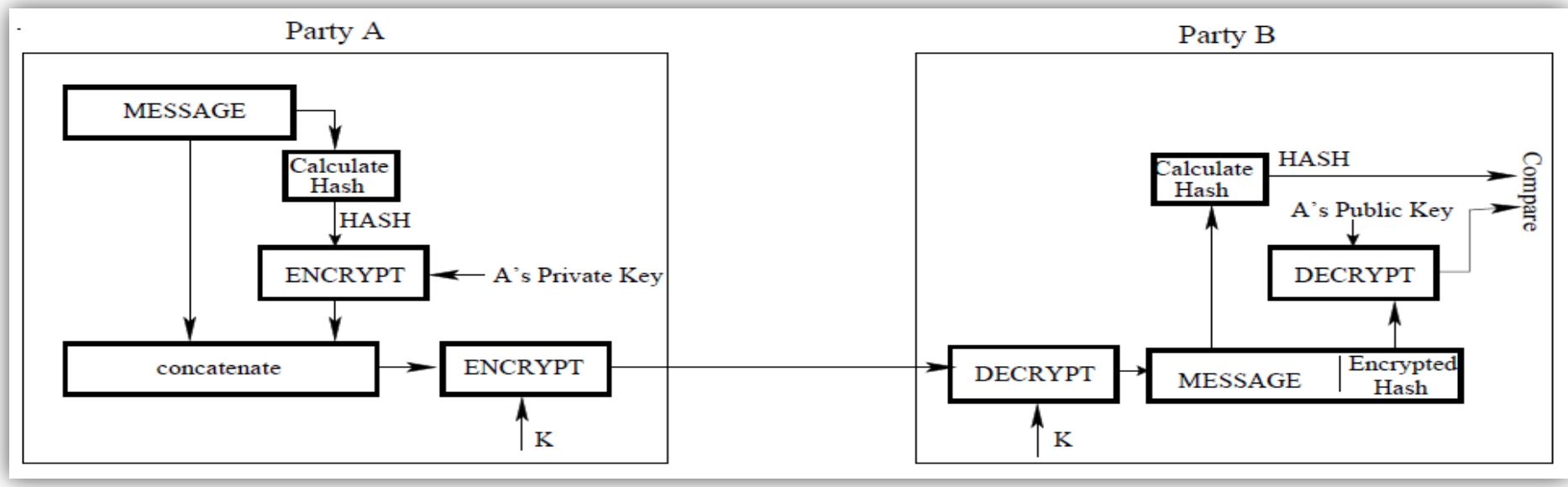
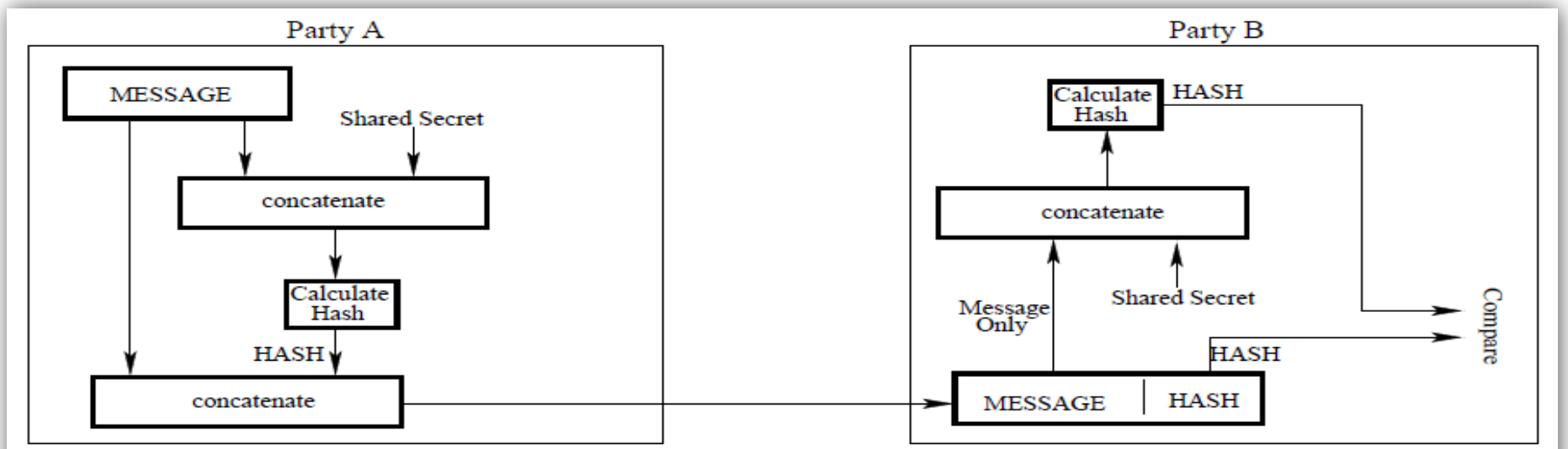


Figure 2 (a)

Different Ways to Use Hashing for Message Authentication (4)

- A very different approach to the use of hashing for authentication is shown in Figure 2(b). In this scheme, nothing is encrypted. However, the sender appends a secret string *S* , known also to the receiver, to the message before computing its hash code. Before checking the hash code of the received message for its authentication, the receiver appends the same secret string *S* to the message. Obviously, it would not be possible for anyone to alter such a message, even when they have access to both the original message and the overall hash code.
- Finally, the scheme in Figure 2(c) shows an extension of the scheme of Figure 2(b) where we have added symmetric-key based confidentiality to the transmission between the sender and the receiver.



(b)

(c)

Figure 2 (b & c)

When is a Hash Function Secure? (1)

- A hash function is called secure if the following two conditions are satisfied:
  1. If it is computationally infeasible to find a message that corresponds to a given hash code. This is sometimes referred to as **the one-way property of a hash function**.
  2. If it is computationally infeasible to find two different messages that hash to the same hash code value. This is also referred to as the **strong collision resistance property of a hash function**.

When is a Hash Function Secure? (2)

- If you use n bits to represent the hash code, there are only 2^n distinct hash code values. If we place no constraints whatsoever on the messages, then obviously there will exist multiple messages giving rise to the same hash code.
- But then considering messages with no constraints whatsoever **does not represent reality** because messages are not noise — they must possess considerable structure in order to be intelligible to humans.
- **Collision resistance refers to the likelihood that two different messages possessing certain basic structure so as to be meaningful will result in the same hash code.**

Simple Hash Functions (1)

- Practically all algorithms for computing the hash code of a message view the message as a **sequence of** *n*-bit **blocks**.
- The message is processed one block at a time in an **iterative fashion** to produce an *n*-bit hash code.
- Perhaps the **simplest hash function** consists of starting with the first *n*-bit block, **XORing it bit-by-bit** with the second *n*-bit block, XORing the result with the next *n*-bit block, and so on. We will refer to this as **the XOR hash algorithm**.
- With this algorithm, every bit of the hash code represents **the parity** at that bit position if we look across all of the *n*-bit blocks. For that reason, the hash code produced is also known as **longitudinal parity check**.
- The hash code generated by the XOR algorithm can be useful as a **data integrity check** in the presence of **completely random transmission errors**.

Simple Hash Functions (2)

- But, in the presence of an adversary trying to deliberately tamper with the message content, the XOR algorithm is useless for message authentication. An adversary can modify the main message and add a suitable bit block before the hash code so that the final hash code remains unchanged.
- Another problem with this simple algorithm is its somewhat reduced collision resistance for structured documents. Ideally, one would hope that, with an n-bit hash code, any particular message would result in a given hash code value with a probability of $1/2^n$. But now consider the case when the characters in a text message are represented by their ASCII codes. Since the highest bit in each byte for each character will always be 0, you can see that some of the n bits in the hash code will predictably be 0 with the simple XOR algorithm. This obviously reduces the number of unique hash code values available to us, and thus increases the probability of collisions.

Structure of Cryptographically Secure Hash Functions (1)

- A hash function is cryptographically secure if it strictly one-way, in the sense that it lets us compute the hash code for a message, but does not let us figure out a message for a given hash code.
- Most secure hash functions are based on the structure proposed by Merkle. This structure forms the basis of SHA series of hash functions and also the Whirlpool hash function.
- The input message is partitioned into *L* blocks, each of size *b* bits. If necessary, the final block is padded suitably so that it is of the same length as others.
- The final block also includes the total length of the message whose hash function is to be computed. This step enhances the security of the hash function since it places an additional constraint on the counterfeit messages.
- Merkle's structure, shown in Figure 3, consists of *L* stages of processing, each stage processing one of the *b*-bit blocks of the input message.

Structure of Cryptographically Secure Hash Functions (2)

- Each stage of the structure in Figure 3 takes **two inputs**, the *b*-bit block of the input message meant for that stage and the *n*-bit output of the **previous stage**.
- For the *n*-bit input, the first stage is supplied with a special *N* -bit pattern called the **Initialization Vector** (IV).
- The function *f* that processes the two inputs, one *n* bits long and the other *b* bits long, to produce an *n* bit output is usually called the **compression function**. That is because, usually, $b > n$, so the output of the *f* function is **shorter** than the length of the input message segment.
- The function *f* itself may involve **multiple rounds** of processing of the two inputs to produce an output.
- The precise nature of *f* depends on **what hash algorithm** is being implemented.

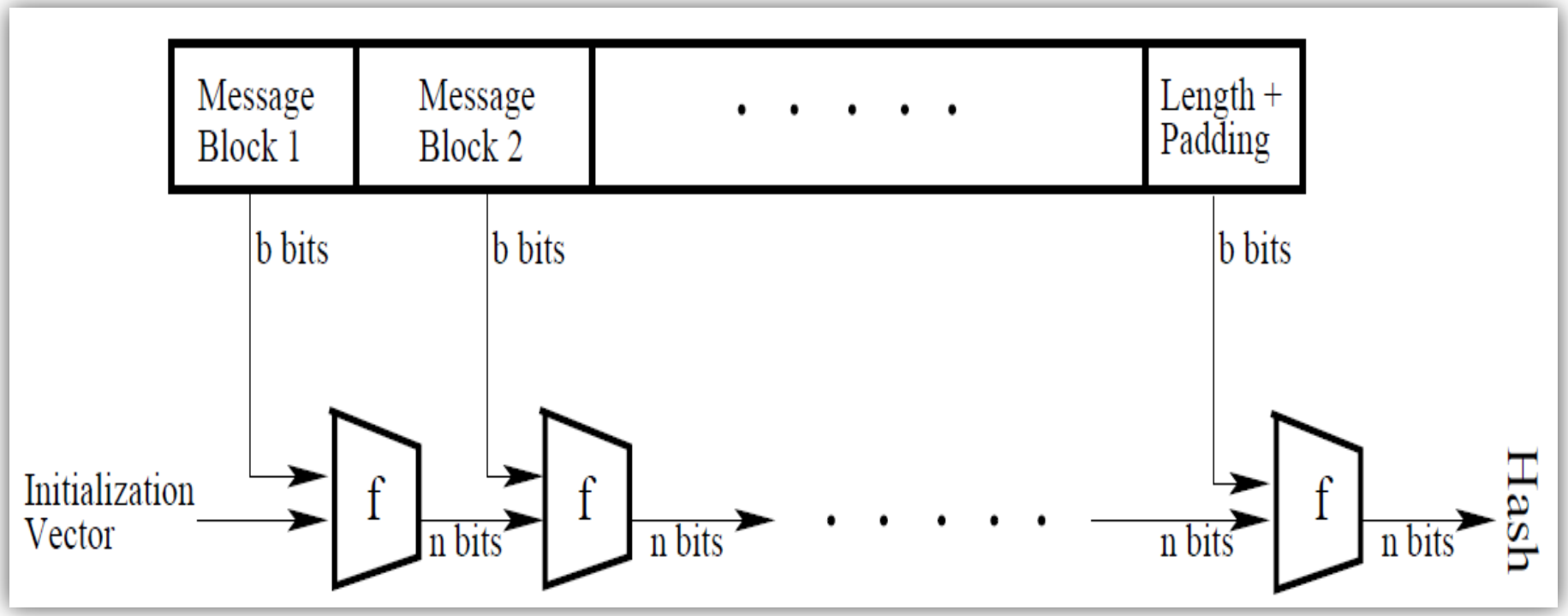


Figure 3: Merkle's structure

The SHA Family of Hash Functions (1)

- SHA (Secure Hash Algorithm) refers to a family of NIST-approved cryptographic hash functions.
- The most commonly used hash function from the SHA family so far is SHA-1. It is used in many applications and protocols that require secure and authenticated communications. SHA-1 is used in SSL/TLS, PGP, SSH, S/MIME, and IPSec.
- The following table shows the various parameters of the different SHA hashing functions.

Algorithm	*Message Size* (bits)	*Block Size* (bits)	*Word Size* (bits)	*Message Digest Size* (bits)	*Security* (bits)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

The SHA Family of Hash Functions (2)

- Here is what the different columns of the above table stand for:

1. The column *Message Size* shows the upper bound on the size of the message that an algorithm can handle.

2. The column heading *Block Size* is the size of each bit block that the message is divided into. [Recall from the previous section that an input message is divided into a sequence of *b*-bit blocks. Block size for an algorithm tells us the value of *b* in Figure 3].

3. The *Word Size* is used during the processing of the input blocks.

4. *Message Digest Size* refers to the size of the hash code produced.

5. Finally, the *Security* column refers to how many messages would have to be generated before one can be found with the same hash code with a probability of 0.5 .

The SHA Family of Hash Functions (3)

- In general, for a secure hash algorithm producing *n*-bit hash codes, one would need to come up with $2^{n/2}$ messages in order to discover a collision with a probability of 0.5 . That's why the entries in the last column of the table are half in size compared to the entries in the *Message Digest Size*.
- The algorithms SHA-256, SHA-384, and SHA-512 are collectively referred to as SHA-2.
- Also note that SHA-1 is a successor to MD5 that used to be a widely used hash function.
- SHA-1 was cracked in year 2005. It was shown that it is possible to come up with a collision for SHA-1 with only 2^{69} operations, far fewer than the security level of 2^{80} that is associated with this hash function.

Message Authentication Codes (MACs) (1)

- Just as a hash code is a fixed-size fingerprint of a variable-sized message, so is a **message authentication code** (MAC).
- A MAC is also known as a **cryptographic checksum** and as an **authentication tag**.
- A MAC can be produced by **appending a secret key to the message** and **then hashing** the composite message. The resulting hash code is the MAC.
- A MAC produced with a **hash function** is also referred to by **HMAC**.
- More **sophisticated ways** of producing a MAC may involve an **iterative procedure** in which a pattern derived from the key is added to the message, the composite hashed, another pattern derived from the key added to the hash code, the new composite hashed again, and so on.

Message Authentication Codes (MACs) (2)

- A MAC can also be based on a block cipher or a stream cipher. The block-cipher based DES-CBC MAC is widely used in various standards.
- Another way to generate a MAC would be to compress the message into a fixed-size signature and to then encrypt the signature with an algorithm like DES. The output of the encryption algorithm becomes the MAC value and the encryption key the secret that must be shared between the sender and the receiver of a message.
- Assuming a collision-resistant hash function, the original message and its MAC can be safely transmitted over a network without worrying that the integrity of the data may get compromised. A recipient with access to the key used for calculating the MAC can verify the integrity of the message by re-computing its MAC and comparing it with the value received.

Finally . . .

- Acknowledgment: These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-A6**.