Mobile and Wireless Security
CSCE 496/896

Lecture # 4
Basics of cryptography and security

Instructor: Nirnimesh Ghose
Computer Science and Engineering

UNIVERSITY of NEBRASKA–LINCOLN

Diffie-Hellman Protocol (1976)

Pick X_A

Primitive $g^{X_A} \bmod p$ →

Pick X_B

Key $K_{AB} = (g^{X_B} \bmod p)^{X_A}$

← Primitive $g^{X_B} \bmod p$

Key $K_{AB} = (g^{X_A} \bmod p)^{X_B}$

Why is Diffie-Hellman Secure?

Discrete Logarithm (DL) problem:

Given → $g^{X_A} \bmod p$, g and p; it is hard to extract X_A.

There is no efficient algorithm to perform the operation.

This is not enough for DH to be secure.

Computational Diffie-Hellman (CDH) problem:

Given → $g^{X_A} \bmod p$ and $g^{X_B} \bmod p$; it is hard to compute $g^{X_A X_B} \bmod p$.

It is easy when at least X_A or X_B is known.

Decisional Diffie-Hellman (DDH) problem:

Given → $g^{X_A} \bmod p$ and $g^{X_B} \bmod p$;

it is hard to differentiate between

$g^{X_A X_B} \bmod p$ and $g^r \bmod p$ for any r random number.

Diffie-Hellman unsecure against MitM Attack

Pick X_A — $g^{X_A} \bmod p$ → Pick X_M — $g^{X_M} \bmod p$ → Pick X_B

$K_{AM} = (g^{X_M} \bmod p)^{X_A}$ ← $g^{X_M} \bmod p$ — $K_{AM} = (g^{X_A} \bmod p)^{X_M}$ ← $g^{X_B} \bmod p$

$K_{MB} = (g^{X_B} \bmod p)^{X_M}$

$K_{MB} = (g^{X_M} \bmod p)^{X_B}$

Lecture Set Overview

Hashes

Basic concepts for hash functions

Security uses

SHA-1

Definition

A function $h, y = h(x)$ which has, as a minimum, the following two properties:

- Compression - h maps an input x of arbitrary finite bit length, to an output $h(x)$ of fixed bit length n.
- Ease of computation - given h and an input $x, h(x)$ is easy to compute.

Unkeyed hash :

- MDC (Manipulation Detection Code)
  - OWHF (One Way Hash Function)
  - CFHF (Collision-Free Hash Function)

Keyed hash :

- MAC (Message Authentication Code)

Classification of Hash Functions

MDC

- Dedicated hash functions: MD class, SHS, HAVAL
- Block cipher-based: MDC-2, MDC-4
- Modular arithmetic: MASH-1, MASH-2

MAC

- Block cipher-based: DES-CBC MAC
- Hash function-based: HMAC

Requirements of Hash Functions

Compression

One-way function

- Preimage resistance: Given y, it is computationally infeasible to compute x with $y = h(x)$
- Second Preimage resistance: Given x and $h(x)$, it is computationally infeasible to compute x' with $h(x) = h(x')$

Collision-free

- It is computational infeasible to find a pair $(x, x'), x \neq x'$ satisfying $h(x) = h(x')$.

Efficiency

- Easy to compute $h(x)$ for a given x and $h(\cdot)$.

Requirements of Hash Function

Collision resistance implies second-preimage resistance

Collision resistance does not guarantee preimage resistance

Let $g(\cdot)$ be a collision resistance hash function with n-bit output, define

$h(x) = 1 \,||\, x$, if x has bit length n

$h(x) = 0 \,||\, g(x)$, otherwise

$h(\cdot)$ is a collision resistant hash function with $(n+1)$-bit output

not preimage resistant; easily to find an image

i.e., when we see $1||\, x$, we know x is the input!

Birthday Paradox

The Birthday Phenomenon:

How many people need to be in a room such that the possibility that there are at least two people with the same birthday is greater than 0.5?

For simplicity, we don't care about February 29, and assume that each birthday is equally likely

If there are over 23 people in a room, then the probability is greater than 0.5 that two people will have the same birthday (out of 365 days in the year)

Birthday Paradox

Formal description:

Assume a target space Y with N possible outcomes; assume a function $F(\cdot)$ mapping input values from a source space X to Y with a uniform distribution.

Then, if R input values from X are picked at random, such that $R \geq 1.18\sqrt{N}$, then the probability that there are two inputs (out of R) mapping to the same output value will be > 0.5

Reduce the attack complexity from $O(2^{N-1})$ to $O(2^{N/2})$

Instead of exhaustive search, the attacker could find the same hash value for two different messages with bit length N.

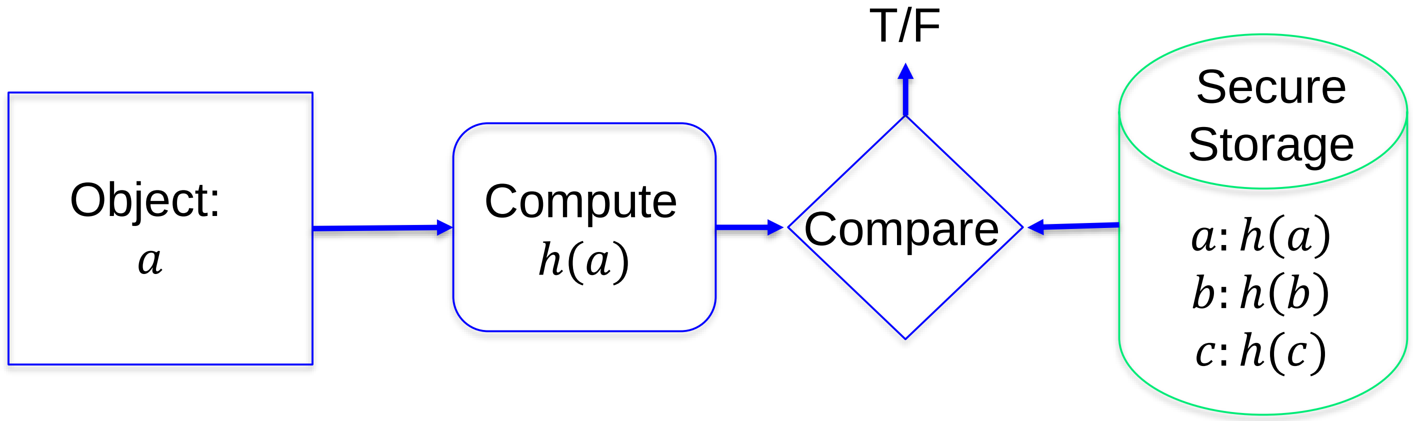
Applications: Object Identifier

Computing hash of files / documents / emails

Keeping the hash values in a secure place, while Files / documents / emails may reside in non-secure places

Before using an object, the system computes a hash of the object and compares it with the stored hash

If the hash values do not match – the object was probably changed in an unauthorized manner

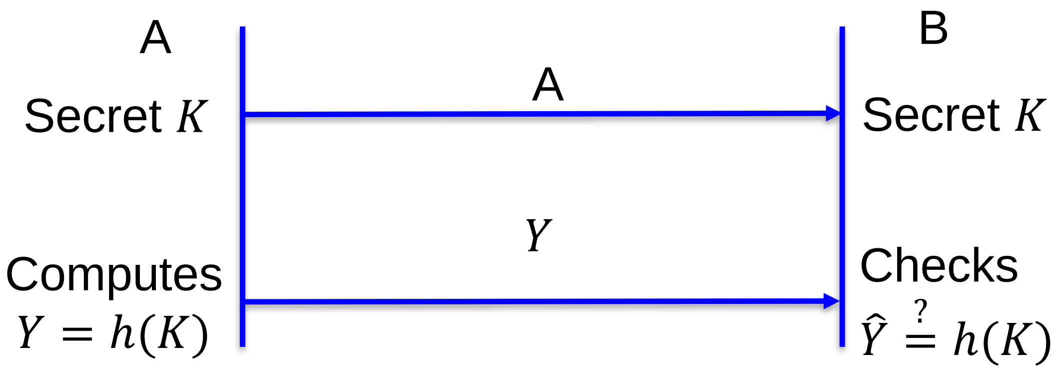


Applications: Entity Authentication

Goal: A wishes to identify & authenticate himself to B

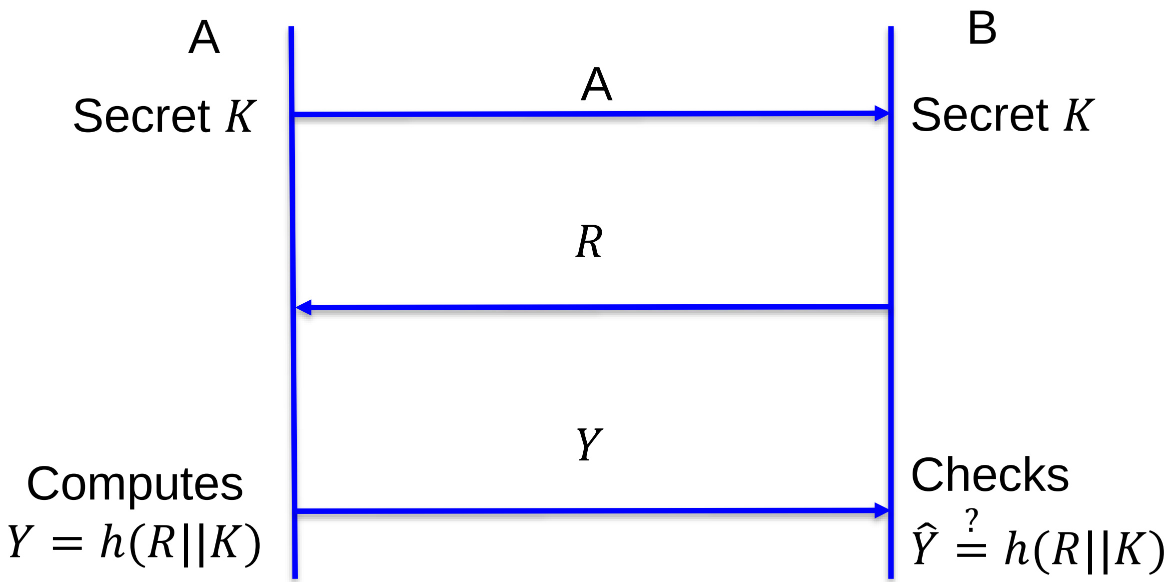
Infrastructure: A and B share a long-lived secret key K

Naive Authentication Protocol



Problem: Above protocol is subject to a “replay attack”

Authentication with Hash



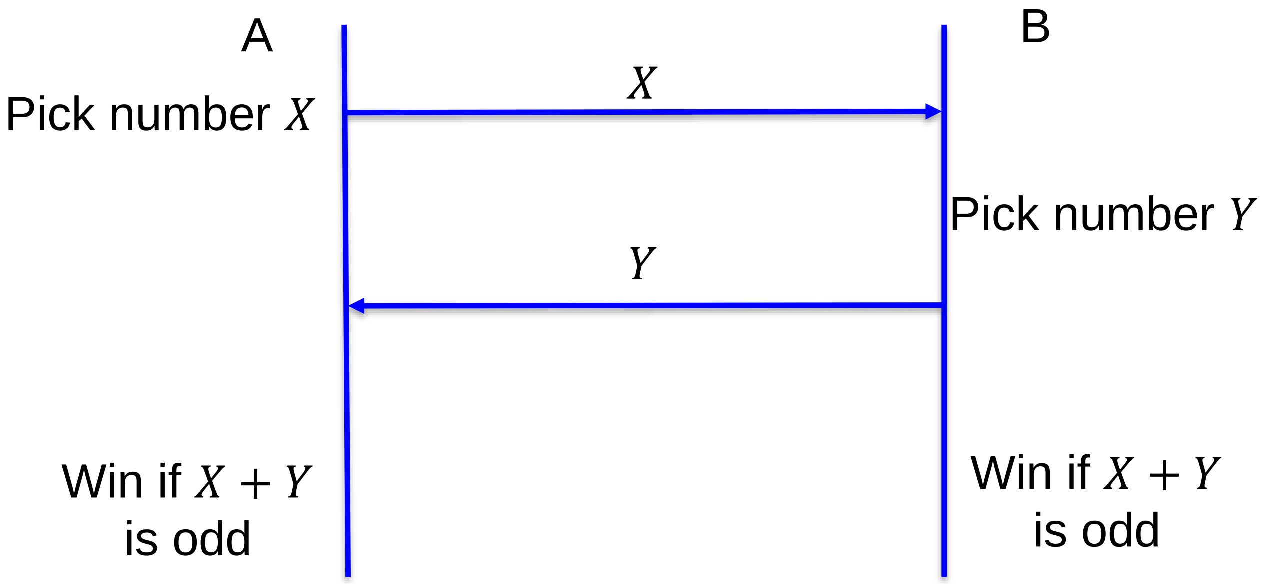
B verifies hash of random number R and secret key K

This protocol is sound against sniffing and replay

Applications: Commitment Protocol

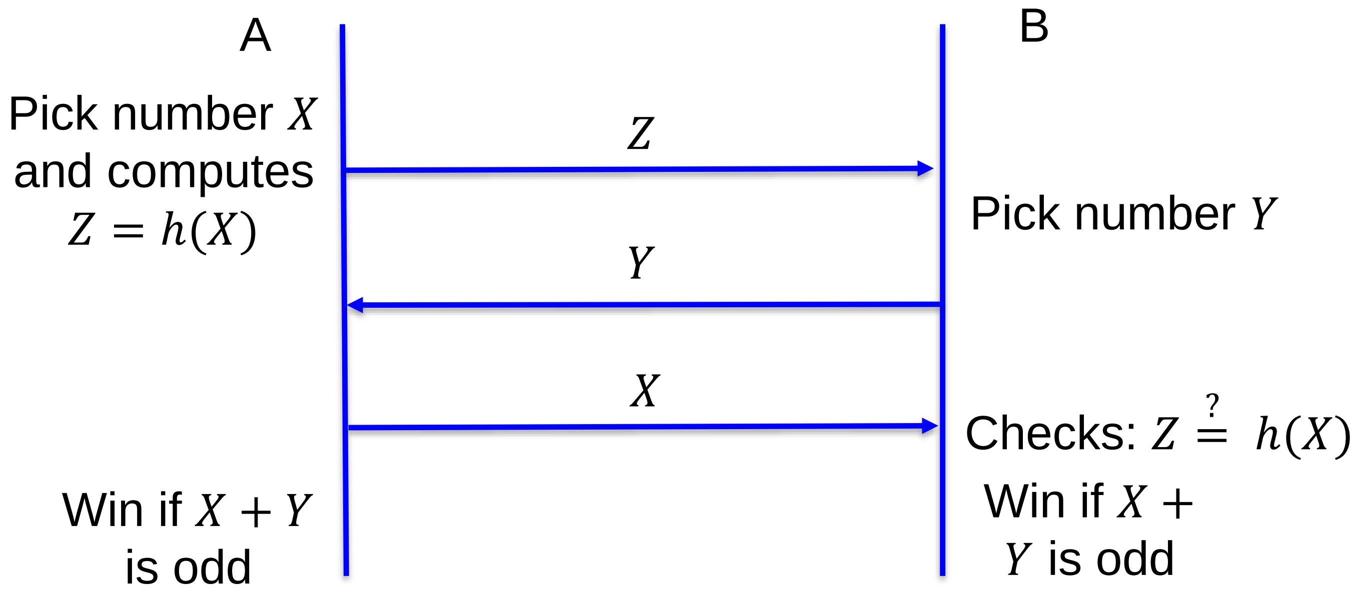
Goal: A and B wish to play "odd or even" over the network

Naive Commitment Protocol



Problem: How can we guarantee that B doesn't cheat?

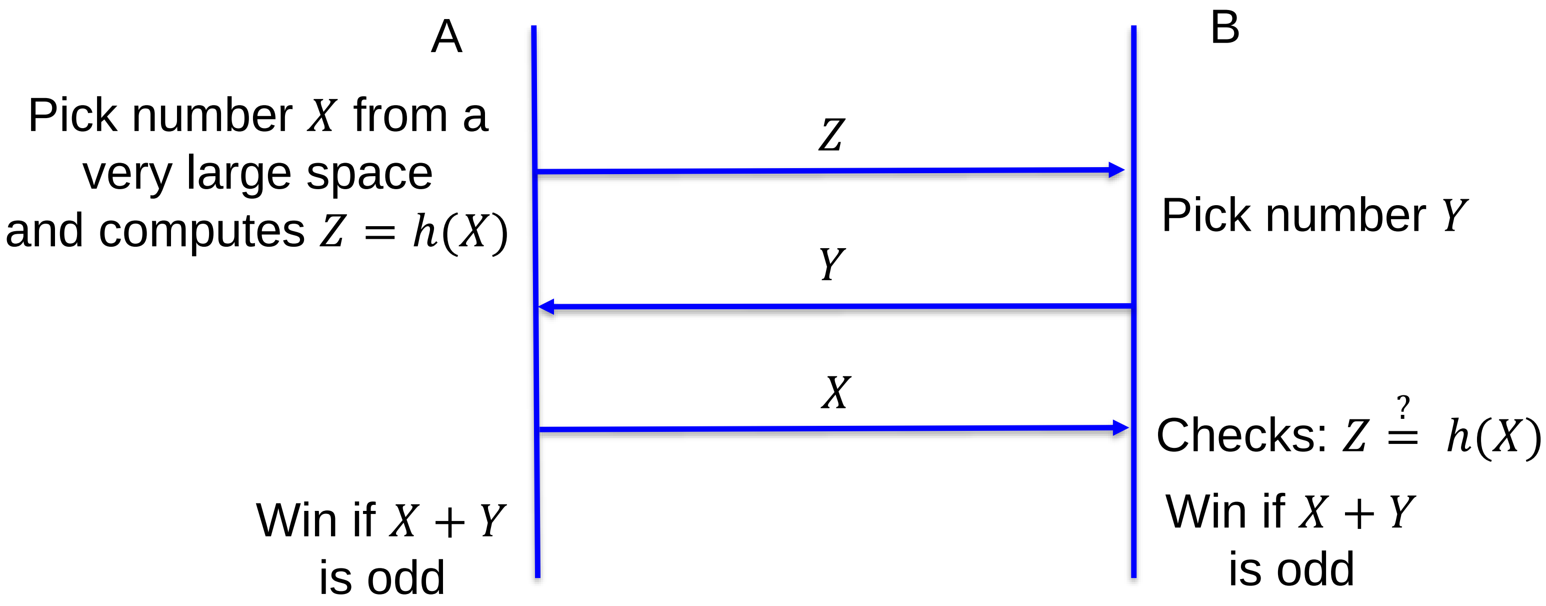
Commitment Protocol with Hash



In this protocol B cannot cheat because B cannot know X before sending Y; on the other hand, A also cannot cheat on the value of X after receiving Y.

Question: What if A always picks small numbers so that B can make a list of all the hash values?

Commitment Protocol with Hash



Dedicated Hash Algorithms

MD2, MD4, MD5

- Message Digest, designed by Ron Rivest
- MD2: 1989, operates on 8-bit octets
- MD4: 1990, operates on 32-bit words
- MD5: 1991, operates on 32-bit words

Secure Hash Algorithm, proposed by NIST

- SHA: 1993
- SHA-1: 1995
- SHA-256, SHA-512

SHA-1

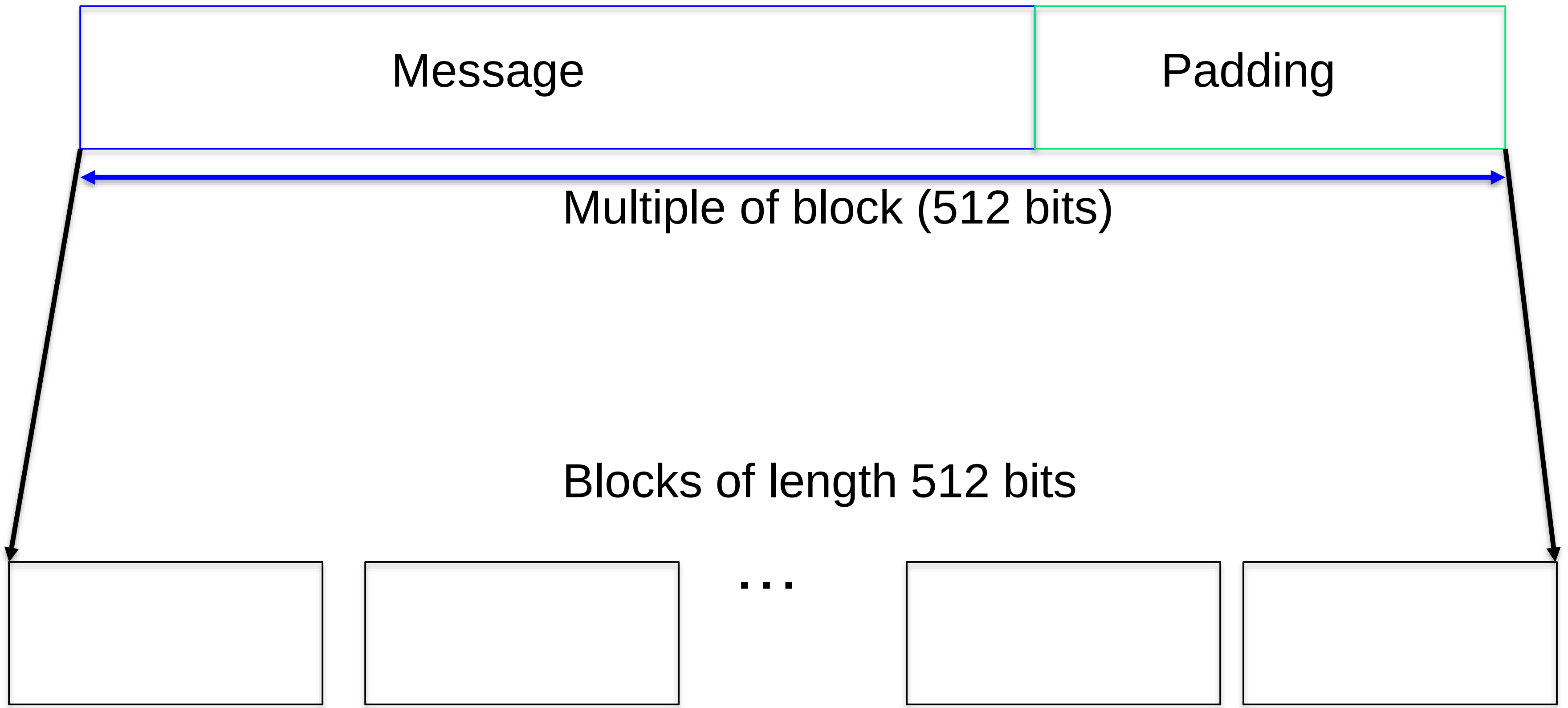
Input: arbitrary number of bits ($\leq 2^{64}$)

Output: 160 bits

Step 1: Pad the message to be a multiple of 512 bits (16 words, 64 octets)

Step 2: Process the message, 512 bits at a time, to produce the message digest

Message padding



Comparison of MD5 and SHA-1

SHA-1 160-bit, MD5 128-bit; SHA-1 is more secure against brute-force attacks

MD5 is considered broken in year 2004

SHA-1 involves more stages and bigger buffer;
SHA-1 executes much slower than MD5

SHA-1 is considered “broken” in year 2005

Keyed Hash

$h(m|Key)$ or $h(Key|m)$?

Keyed hash *h(Key|m)?*

A feature of message digest algorithms

In order to computer the message digest through chuck n, all that you need to know is the message digest through chunk $n-1$, plus the chuck n of the padded message.

An Attack

Someone gets m, and $h(Key|m)$

He first pads m according the used hash function, and then adds another message m' at the end. The result is $m|pad|m'$.

$h(Key|m|pad\ |m')$ can be calculated from $h(Key|m)$, which is the intermediate digest.

Soln: Use $h(m|Key)$