CSE 410/565: Computer Security

Instructor: Dr. Ziming Zhao

Academic Integrity

- The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: UB Student Code of Conduct.

- Plagiarism or any form of cheating in homework, or exams is subject to serious academic penalty.

- Any violation of the academic integrity policy will result in a 0 on the homework or exam, and even an **F** or **>F<** on the final grade. And, the violation will be reported to the Dean’s office.

The AI quiz is due this Sunday.

Advantages of Symmetric Key Crypto

- Secure, hard to break
    - Offers good confidentiality
- Fast
    - Many rounds of substitution and transposition
    - Even faster with direct hardware support
        - Intel AES-NI extension provides six instructions to accelerate symmetric block encryption/decryption of 128-bit data blocks using the Advanced Encryption Standard (AES) specified by the NIST publication FIPS 197.

Disadvantage of Symmetric Key Crypto

- The secret key is to be transmitted to the receiving system before the actual message is to be transmitted.

Key Distribution

- Encrypted communication channel between A and B
    - A selects a key and physically delivers it to B.
    - A trusted third party key distribution center (KDC) selects a key and physically delivers it to A and B.

Diffie-Hellman Key Agreement Protocol

- Suggested ‘Public Key Cryptosystem’
- Presented ‘Diffie-Hellman key agreement protocol’

644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

***Abstract*—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.**

The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as private courier or registered mail. A private conversation

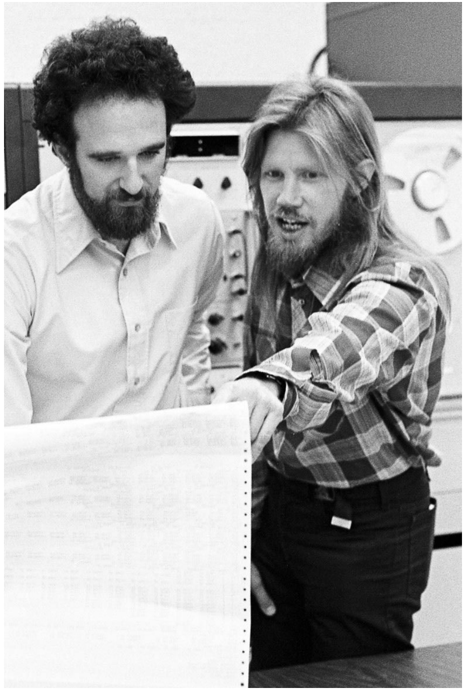
Diffie-Hellman Key Agreement Protocol

- The Diffie-Hellman key agreement protocol (1976) was the first practical method for **establishing a shared secret** over **an unsecured** communication channel.

- The point is to **agree** on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.

- Also called 'Diffie-Hellman key exchange protocol'

Whitfield Diffie and Martin Hellman



1977

Recipients of the 2015 ACM A.M. Turing Award

Applications

- Diffie-Hellman key agreement is currently used in many protocols, namely:
    - Internet Protocol Security (IPSec)
    - Secure Sockets Layer (SSL)/Transport Layer Security (TLS)
    - Public Key Infrastructure (PKI)

Diffie-Hellman Key Agreement Protocol

Steps

1. Alice and Bob agree on a prime number p and a base g. p and g can be public
2. Alice chooses a secret number a, and sends Bob g^a mod p
3. Bob chooses a secret number b, and sends Alice g^b mod p
4. Alice computes (g^b mod p)^a mod p
5. Bob computes (g^a mod p)^b mod p
6. Both Alice and Bob can use this number as their key.

Example

Steps

1. Alice and Bob agree on p = 23 and g = 5.
2. Alice chooses a = 6 and sends 5^6 mod 23 = 8.
3. Bob chooses b = 15 and sends 5^15 mod 23 = 19.
4. Alice computes 19^6 mod 23 = 2.
5. Bob computes 8^15 mod 23 = 2.
6. Then 2 is the shared secret.

In reality, much larger values of a, b, and p are required. An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

Why it is secure?

- Suppose p is a prime of around 300 digits, and a and b at least 100 digits each.
- Discovering the shared secret given g, p, g a mod p and g b mod p would take longer than the lifetime of the universe, using the best known algorithm.
- This is called the discrete logarithm problem.

Fix a prime p. Let a, b be nonzero integers (mod p). The problem of finding x such that $a^x = b \pmod{p}$ is called the discrete logarithm problem

Attacks on Diffie-Hellman

- A man-in-the-middle attack is a type of cyberattack where a malicious actor inserts him/herself into a conversation between two parties, impersonates both parties and gains access to information that the two parties were trying to send to each other.

- A man-in-the-middle attack allows a malicious actor to intercept, send and receive data meant for someone else, or not meant to be sent at all, without either outside party knowing until it is too late.

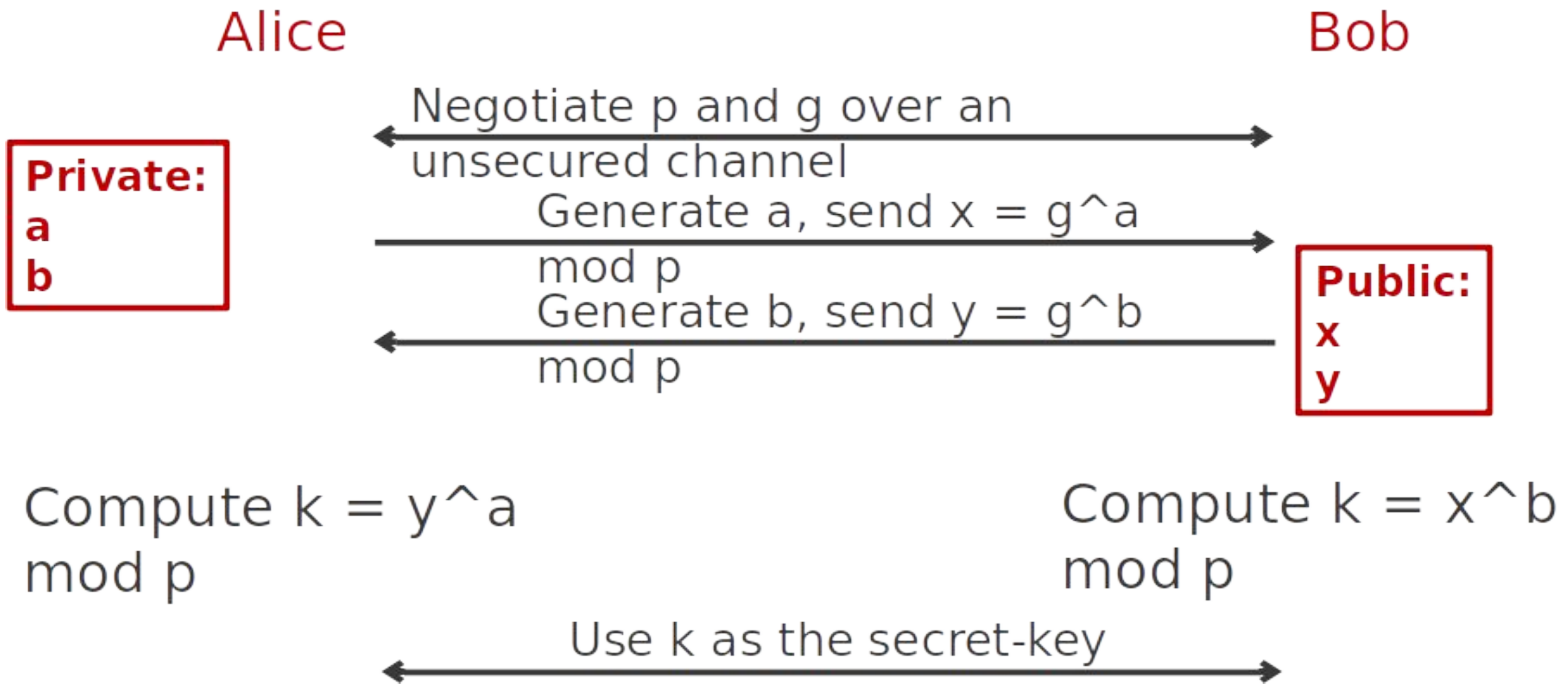
Attacks on Diffie-Hellman

- In the real world, the Diffie-Hellman key exchange is rarely used by itself. The main reason behind this is that it provides no authentication, which leaves users vulnerable to man-in-the-middle attacks.

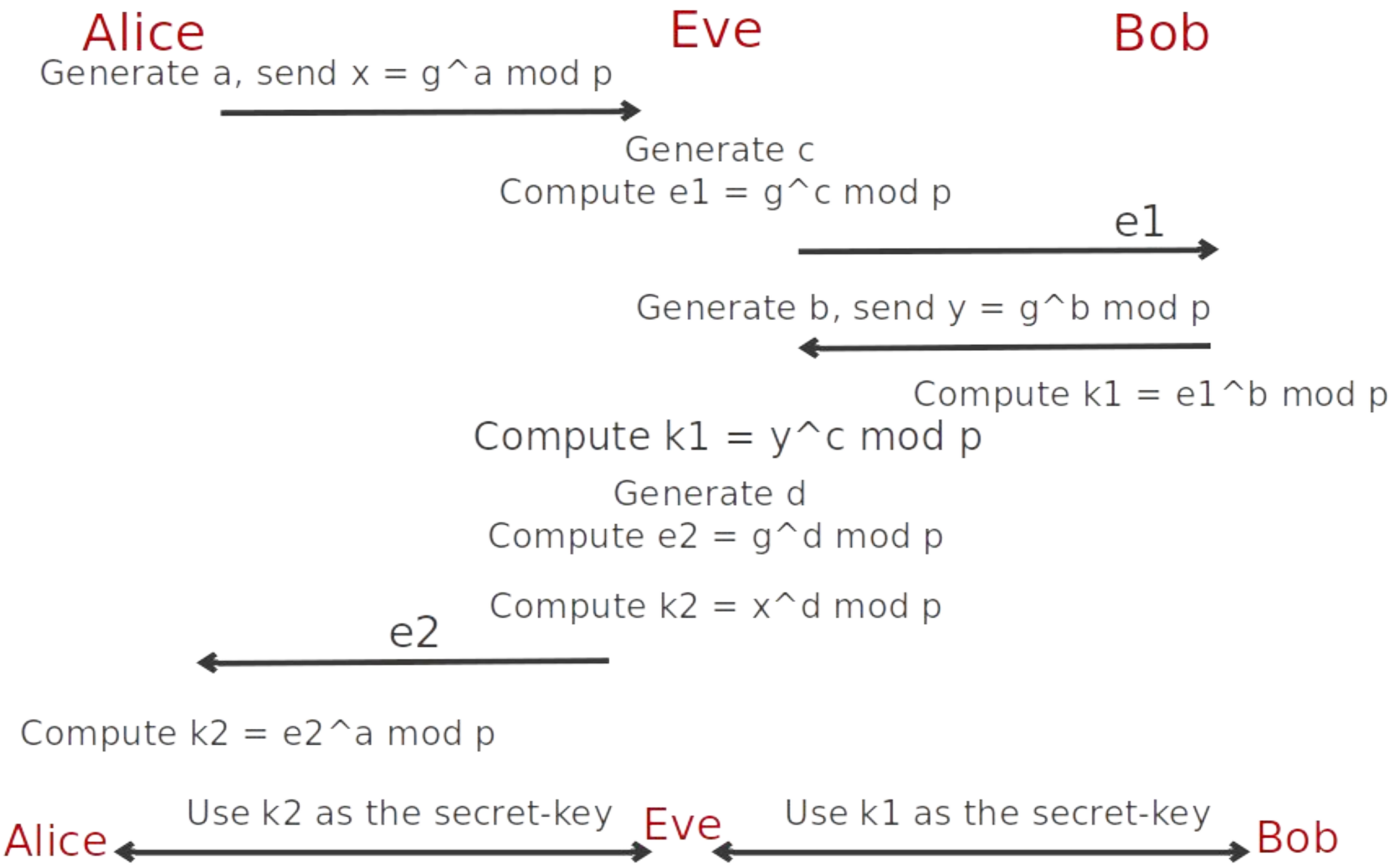
- For this reason, the Diffie-Hellman key exchange is generally implemented alongside some means of authentication. This often involves using digital certificates and a public-key algorithm, such as RSA, to verify the identity of each party.

Diffie-Hellman

The Diffie-Hellman key agreement scheme is based on six numbers (*p*, *g*, *a*, *b*, *x*, and *y*)



MiM on Diffie-Hellman



Data Integrity and Hash Functions

Outline

- So far we discussed encryption as means to data confidentiality protection
- Next, we will talk about data integrity protection
  - this covers message authentication codes
  - we also discuss hash functions as a tool for integrity protection and other applications
- Everything we are discussing so far assumes a computationally limited adversary
  - doesn't have unlimited resources, can't search through the key space, etc.

Data Integrity

- Encryption protects data only from a passive attack
  - we often also want to protect message from active attacks (modification or falsification of data)
  - such protection is called message or data authentication

- Goals of message authentication
  - a message is authentic if it came from its alleged source in its genuine form
  - message authentication allows verification of message authenticity

Message Authentication

- How can message authentication be performed?
  - in addition to the message itself, another token that authenticates the message, often called a tag, is transmitted
  - the cryptographic primitive is called a Message Authentication Code (MAC)

- Message authentication is independent of encryption
  - it can be used with or without encryption
  - a number of applications benefit from message authentication alone

Message Authentication

- What do we want from a message authentication code?
  - a tag should be easy to compute by legitimate parties, but hard to forge by an adversary
- MAC constructions use a secret key
  - a secret key is shared by two communicating parties
  - a MAC cannot be computed (or verified) without the key
- To achieve source authentication and message integrity:
  - the sender computes $t = MAC_k(m)$ and sends (m, t)
  - the receiver recomputes $t' = MAC_k(m)$ for received m and compares it to t

Message Authentication Code

- A MAC scheme is defined by three algorithms:
  - key generation: a randomized algorithm, which on input a security parameter *n*, produces key a *k*
  - MAC generation: a possibly randomized algorithm, which on input a message *m* and key *k*, produces a tag *t*
  - MAC verification: a deterministic algorithm, which on input a message *m*, tag *t*, and key *k*, outputs a bit *b*

Message Authentication Codes

- Properties of MAC algorithms
  - most fundamentally, we desire correctness and security
  - correctness requires that a correctly computed tag will always verify
  - security will be defined later and intuitively requires that it is hard to forge a tag on a new message without the key
- from a performance point of view, we desire (and can achieve) tags of a fixed size (i.e., independent of the message length)

Message Authentication Codes

- Classification of attacks on MACs:
  - known-text attack: one or more pairs $(m_i , Mac_k (m_i))$ are available
  - chosen-text attack: one of more pairs $(m_i , Mac_k (m_i))$ are available for m_i's chosen by the adversary
  - adaptive chosen-text attack: the m_i's are chosen by the adversary, where successive choices can be based on the results of prior queries
- Against which kind of attack do we want to be resilient?

Message Authentication Codes

- Classification of forgeries:
  - selective forgery: an adversary is able to produce a new MAC pair for a message under her control
  - existential forgery: an adversary is able to produce a new MAC pair but with no control of the value of the message
- Resilience against which type would be preferred?
- And, as usual, key recovery is the most damaging attack on MAC

Message Authentication Codes

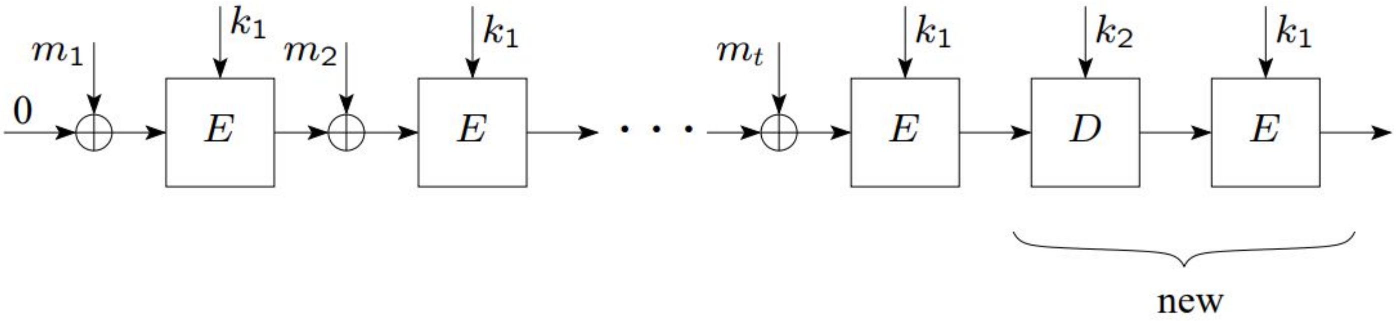
- We desire for a MAC to be existentially unforgeable under an adaptive chosen-message attack
  - an adversary is allowed to query tags on messages of its choice
  - at some point it outputs a pair *(m, t)*
  - the forgery is considered successful if m hasn't been queried before and t is a valid tag for it
  - as with encryption, security guarantees depend on the security parameter
- MACs do not prevent all traffic injections
  - a replayed message will pass verification process
  - it is left to the application to make each message unique

Message Authentication Codes

- There are two most common (standardized) implementations of MAC functions
  - CBC-MAC: based on a symmetric encryption (e.g., AES) in Cipher Block Chaining (CBC) mode with some modifications
    - varying IV is not permitted
    - only a single block is produced
    - additional security measures are in place to support variable-length messages
  - HMAC: based on a hash function
- We'll discuss the latter and need to look at hash functions first

Message Authentication Codes

- A CBC-MAC variant secure in the presence of variable-length messages

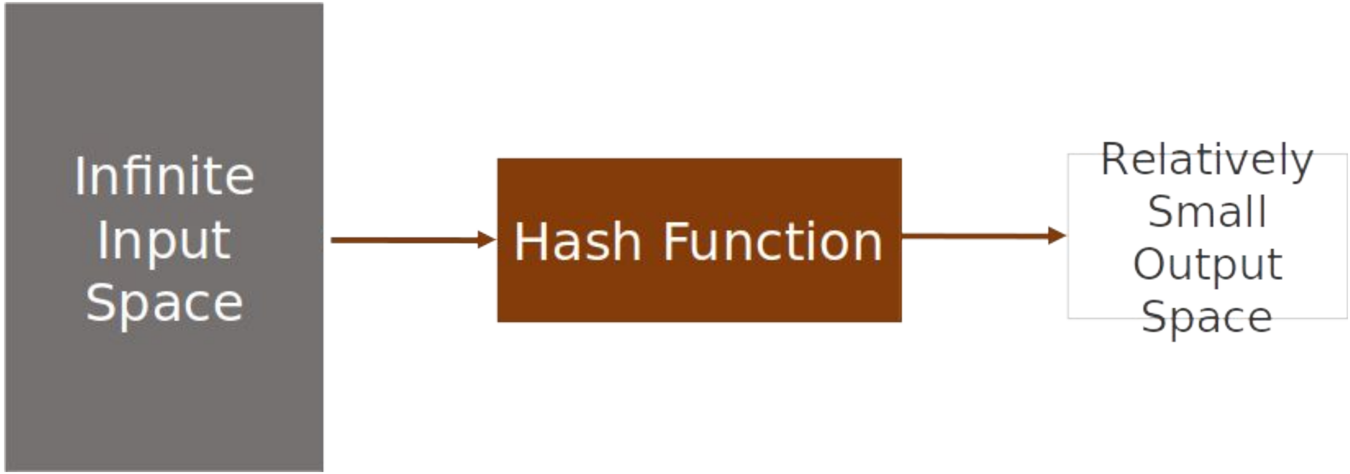


Hash Functions

Hash Functions

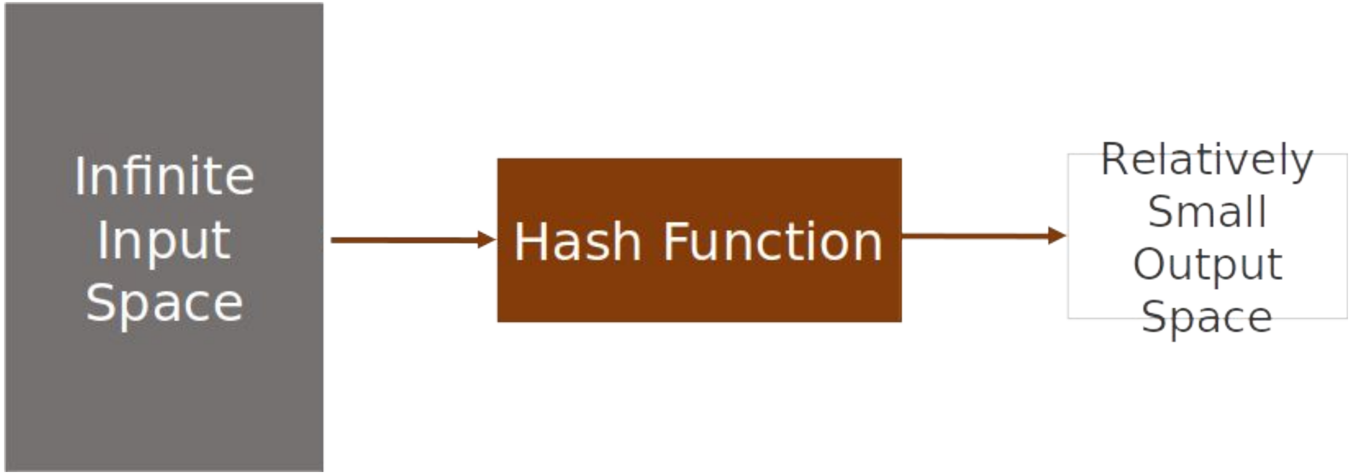
- Just a method of compressing strings
  - E.g., $H : \{0,1\}^* \rightarrow \{0,1\}^{160}$
  - Input is called “message”, output is “digest, hash value”

Hash Functions: one-way



One-way function: It is impossible to calculate *m* from *H(m)*

Hash Functions: one-way



- Some pairs of inputs will be mapped to the same hash value. This is called collision.
- The pigeonhole principle states that if *n* items are put into *m* containers, with $n > m$, then at least one container must contain more than one item.

Hash Functions

- A hash function *h* is an efficiently-computable function that maps an input *x* of an arbitrary length to a (short) fixed-length output *h(x)*
  - hash functions have many uses including hash tables
- We are interested in cryptographic hash functions that must satisfy certain security properties
  - it is computationally hard to invert *h(x)*
  - it is computationally hard to find collisions in *h*
- Other uses of hash functions include
  - password hashing
  - in digital signatures
  - in intrusion detection and forensics

Hash Functions

- h must satisfy the following security properties:
  - Preimage resistance (one-way): given $h(x)$, it is difficult to find x
  - Second preimage resistance (weak collision resistance): given x, it is difficult to find x' such that $x' \neq x$ and $h(x') = h(x)$
  - Collision resistance (strong collision resistance): it is difficult to find any x, x' such that $x' \neq x$ and $h(x') = h(x)$

- Additional properties normally present in cryptographic hash functions:
  - input bits and output bits should not be correlated
  - it should be hard to find any two inputs x and x' such that $h(x)$ and $h(x')$ differ only in a small number of bits
  - given $h(x)$, it should be difficult to recover any substring of the input

Attacks on Hash Functions

- Brute force search attack
  - success solely depends on the length of the hash n
  - difficulty of finding a preimage or a second preimage is 2^n
  - difficulty of finding a collision with probability 0.5 is about $2^{n/2}$
    - this is due to so-called birthday attack that computes hashes of $2^{n/2}$ versions of a message
    - collision resistance is desired for a general-use hash function
- Cryptanalysis attacks are specific to hash function algorithms

Hash Functions: An bad example

H(x) = x mod 8, where x can be any integer

Input: Arbitrarily large integer
Output: {0, 1, 2, 3, 4, 5, 6, 7}, 3 bits

Preimage resistant: Given a hash value h, it's computationally infeasible to find an n that H(n) = h

Given h = 7, we know 7 or 7+8 or 7+16, …

Hash Functions: An bad example

$H(x) = x \bmod 8$, where x can be any integer

Input: Arbitrarily large integer

Output: {0, 1, 2, 3, 4, 5, 6, 7}, 3 bits

2nd preimage resistant: Given m, it's computationally infeasible to find m' such that H(m') = H(m) and m'!=m

Given m = 57, we know 57-8 or 57+8 or 57+16, …

Hash Functions: An bad example

H(x) = x mod 8, where x can be any integer

Input: Arbitrarily large integer

Output: {0, 1, 2, 3, 4, 5, 6, 7}, 3 bits

Strong collision resistant: Computationally infeasible to find m1, m2 such that H(m1) = H(m2)

Any two integers that x = y +/- 8

Hash Functions

- Well known hash function algorithms:
  - MD5
  - SHA-1
  - SHA-2 family (SHA-256, SHA-384, and others)
  - new SHA-3

- Normally hash function algorithms are iterated
  - they use a compression function
  - the input is partitioned into blocks
  - a compression function is used on the current block m_i and the previous output h_{i-1} to compute

$$h_i = f(m_i , h_{i-1})$$

Hash Function Algorithms

- Families of customized hash functions
  - MD2, MD4, MD5 (MD = message digest)
    - all have 128-bit output
    - MD4 and MD5 were specified as internet standards in RFC 1320 and 1321
    - MD5 was designed as a strengthened version of MD4 before weaknesses in MD4 were found
    - collisions have been found for MD4 in 2^{20} compression function computations (90s)
    - in 2004 collisions for many MD5 configurations were found
    - MD5 (and all preceding versions) are now too weak and not to be used

Hash Function Algorithms

- Secure Hash Algorithm (SHA)
  - SHA was designed by NIST and published in FIPS 180 in 1993
  - In 1995 a revision, known as SHA-1, was specified in FIPS 180-1
    - it is also specified in RFC 3174
  - SHA-0 and SHA-1 have 160 bit output and MD4-based design
  - In 2002 NIST produced a revision of the standard in FIPS 180-2
  - SHA-2 hash functions have length 256, 384, and 512 to be compatible with the increased security of AES
    - they are known as SHA-256, SHA-384, and SHA-512
  - Also, SHA-224 was added to compatibility with 3DES

Hash Function Algorithms

- Security of SHA
  - brute force attack is harder than in MD5 (160 bits vs. 128 bits)
  - SHA performs more complex transformations that MD5
    - it makes finding collisions more difficult
  - in 2004 collisions in SHA-0 were found in < 2^{40}
  - in 2005 collisions have been found in "reduced" SHA-1 (2^{33} work)
  - finding collisions in the full version of SHA-1 is estimated at < 2^{69}
  - several other attacks followed and SHA-1 is considered too weak
  - SHA-2 is a viable option, but has the same structure as in SHA-1 (security weaknesses may follow)

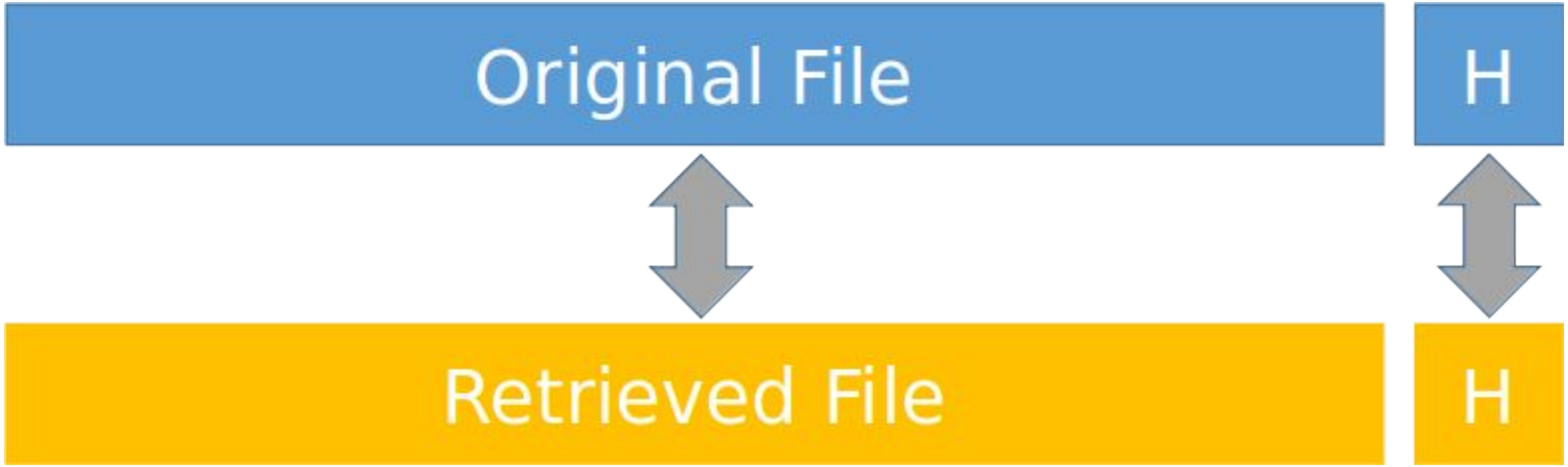
Hash Function Algorithms

- SHA-3
    - search for SHA-3 family was announced by NIST in 2007
        - it was required to support digests of 224, 256, 384, and 512 bits and messages of at least $2^{64} - 1$ bits
    - the winner, Keccak, was announced in 2012 and the SHA-3 standard was released in 2015 as NIST’s FIPS 202
    - Keccak is a family of sponge functions
        - it is a mode of operation that builds a function mapping variable-length input to variable-length output using a fixed-length permutation and a padding rule
        - SHA-3 can be used with one of seven Keccak permutations
        - the design is distinct from other widely used techniques

Application of Hash: Integrity Check

- Integrity: Prevent/detect/deter improper modification of information

File Integrity Check



Back to Message Authentication

- How do we construct a MAC from a hash function *h* and key *k*?
  - consider defining $Mac_k(m) = h (k \mid\mid m)$
    - knowledge of the key is required for efficient computation and verification
    - one-way property of *h* makes key recovery difficult
  - unfortunately, this construction is not as secure as we would like
    - iterative nature of hash function computation gives room for easy forgeries
- HMAC is a more complex construction with provable security

MAC Algorithms

- Hash-Based MAC – HMAC

- Goals:
  - use available hash functions without modifications
  - preserve the original performance of the hash function
  - use and handle keys in a simple way
  - allow replacement of the underlying hash function
  - have a well-understood cryptographic analysis of its strength

HMAC

- HMAC
  - $HMAC_k(x) = h((K \oplus opad) \,||\, h((K \oplus ipad)||x))$
  - K is the key *k* padded to a full block (≥ 512 depending on hash function)
  - *ipad* = 0x3636…36 and *opad* = 0x5C5C…5C are fixed padding constants
- HMAC is efficient to compute
  - the entire message is hashed only once
  - the second time *h* is called on only two blocks

HMAC

- HMAC Security
  - security is related to that of the underlying hash function
    - we want $k_1 = h(K \oplus opad)$ and $k_2 = h(K \oplus ipad)$ to be rather independent and close to random
    - then HMAC is existentially unforgeable under an adaptive chosen-message attack for messages of any length
  - HMAC provides greater security than the security of the underlying hash function
  - no known practical attacks if a secure hash function is used according to the specifications

Confidentiality + Integrity

- How do we use a MAC in combination with encryption?
  - message authentication
    - $A \xrightarrow{m, \mathsf{Mac}_k(m)} B$
  - encrypt and authenticate
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m), \mathsf{Mac}_{k_2}(m)} B$
  - authenticate then encrypt
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m, \mathsf{Mac}_{k_2}(m))} B$
  - encrypt then authenticate
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m), \mathsf{Mac}_{k_2}(\mathsf{Enc}_{k_1}(m))} B$

Confidentiality + Integrity

- Analysis of prior constructions:
  - encrypt and authenticate
    - transmitting $Mac_{k2}(m)$ may leak information about m
  - authenticate then encrypt
    - has a chosen-ciphertext attack against the general version, which has been successfully applied in practice
  - encrypt then authenticate
    - satisfies the definition of authenticated encryption and is CCA-secure

- The keys k_1 and k_2 must be different!

Authenticated Encryption

- Do I have to use encryption and MAC separately or are there authenticated encryption modes?
    - recently, authenticated encryption modes have been proposed
- Some good reads:
    - https://blog.cryptographyengineering.com/2012/05/19/how-to-choose-authenticated-encryption/
    - https://stackoverflow.com/questions/1220751/how-to-choose-an-aesencryption-mode-cbc-ecb-ctr-ocb-cfb

Authenticated Encryption

- Good options to consider:
  - Offset Codebook (OCB) mode
    - state of the art in authenticated encryption
    - proposed internet standard
    - has licensing restrictions
    - see http://web.cs.ucdavis.edu/˜rogaway/ocb/ocb-faq.htm for more information
  - Galois/Counter Mode (GCM)
    - does not have licensing restrictions
    - can be used as an alternative for commercial software

Summary

- We so far covered
  - symmetric encryption, block ciphers
  - encryption standards (DES, AES)
  - message authentication codes
  - hash functions (MD5, SHA-1, SHA-2, SHA-3)
- More to come
  - public key cryptography
  - pseudo-random number generators