CS 5602 Introduction to Cryptography
Lecture 24
Hash Functions &
Message Authentication Codes

George Markowsky
Computer Science Department
Missouri University of Science & Technology

1

Chapter Goals

- All material not otherwise attributed is from Smart's Book
- To understand the properties of cryptographic hash functions.
- To understand how existing deployed hash functions work.

2

Introduction

- In many situations we do not wish to protect the confidentiality of information, we simply wish to ensure the integrity of information.
- That is we want to guarantee that data has not been tampered with.
- In this chapter we look at two mechanisms for this, the first using cryptographic hash functions is for when we want to guarantee integrity of information after the application of the function.
- The other mechanism we look at is the use of a message authentication code.
- Hash functions can also be considered as a special type of manipulation detection code, or MDC.

3

What is Hashing?

- In its most general sense, a hash function is just a function f: X → Y that has special properties
- The exact nature of those properties depends on the application
- Generally, |X| >> |Y| and in many cases X is infinite
- If f is a bijection, then f can be thought of as "encryption"

4

What is Hashing?

- For security applications you want it to be hard for reverse f, in other words if f: X → Y and y ϵ Y, you want it to be difficult to find x ϵ X such that f(x) = y
- If it is easy for a given y to find x such that f(x) = y, then f is not suitable for security applications although it might be handy for other applications

5

A Simple Example

- Consider the function f: $\mathbb{N}$ → range(7) given by f(n) = n % 7
- We have seen that % has many useful properties, but it is clear that given a y ϵ range(7), it is very easy to find an x such that x % 7 = y

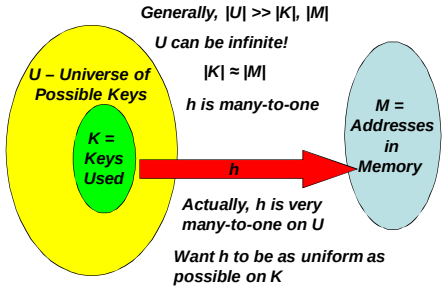
6

Data Retrieval

- Hashing can be used to provide quick access to data by calculating an address to store the data
- By quick, we mean $\Theta(1)$ average time for operations
- Used to store information about variables in compilers, etc.
- For a small universe can use direct addressing – one storage location for each possible key
- The basic idea is given by the following picture

7

The Big Picture



8

Small Universes

- It doesn't take long for universes to grow
- In particular, if you allow only _, A..Z, and 0..9 to be in identifiers, you can have 37^5 = 69,343,957 possible identifiers
- If you add a..z to the mix and distinguish case you have 63^5 = 992,436,543 distinct identifiers
- You don't want to store them all!

9

Collisions

- This is an important consideration for all types of hashing
- Whenever we have x1 ≠ x2 but h(x1) = h(x2), we call this a **collision**
- The process of handling collisions is called **collision resolution**
- This can be done in a variety of ways
- Collision resolution is not important for cryptography – we want collision avoidance!

10

Cryptographic Hash Function

- A **cryptographic hash function** h is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output, the output is often called a **hashcode** or hash value.
- Hash functions are used a lot in computer science, but the crucial difference between a standard hash function and a cryptographic hash function is that a cryptographic hash function should at least have the property of being one-way.
- In other words given any string y from the range of h, it should be computationally infeasible to find any value x in the domain of h such that h(x) = y.

11

Cryptographic Hash Function

- Another way to describe a hash function which has the one-way property is that it is preimage resistant.
- Given a hash function which produces outputs of n bits, we would like a function for which finding preimages requires $O(2^n)$ time.
- In practice we need something more than the one-way property.
- A hash function is called collision resistant if it is infeasible to find two distinct values x and x' such that h(x) = h(x').
- It is harder to construct collision resistant hash functions than one-way hash functions due to the **Birthday Paradox.**

12

Collisions

- To find a collision of a hash function f, we can keep computing $f(x_1)$, $f(x_2)$, $f(x_3)$, . . . until we get a collision.
- If the function has an output size of n bits then we expect to find a collision after $O(2^{n/2})$ iterations.
- This should be compared with the number of steps needed to find a preimage, which should be $O(2^n)$ for a well-designed hash function.
- Hence to achieve a security level of 80 bits for a collision resistant hash function we need roughly 160 bits of output.

13

Collisions

- But still that is not enough; a cryptographic hash function should also be **second preimage resistant**.
- This is the property that given m it should be hard to find an $m' \neq m$ with $h(m') = h(m)$.
- Whilst this may look like collision resistance, it is actually related more to preimage resistance.
- In particular a cryptographic hash function with n-bit outputs should require $O(2^n)$ operations before one can find a second preimage.

14

Properties of Cryptographic Hashes

- In summary a cryptographic hash function needs to satisfy the following three informal properties:

1. **Preimage Resistant:** It should be hard to find a message with a given hash value.
2. **Collision Resistant:** It should be hard to find two messages with the same hash value.
3. **Second Preimage Resistant:** Given one message it should be hard to find another message with the same hash value.

- Has some lemmas that I think are sloppily formulated
- Uses a||b to mean the bits of a are concatenated to the bits of b

15

Designing Hash Functions

- To be effectively collision free a hash value should be at least 128 bits long, for applications with low security, but preferably its output should be 160 bits long.
- However, the input size should be bit strings of (virtually) infinite length.
- In practice designing functions of infinite domain is hard, hence usually one builds a so called compression function which maps bits strings of length s into bit strings of length n, for s > n, and then chains this in some way so as to produce a function on an infinite domain.
- We have seen such a situation before when we considered modes of operation of block ciphers.

20

The Merkle–Damgard Construction

- Suppose f is a compression function from s bits to n bits, with s > n, which is believed to be collision resistant.
- We wish to use f to construct a hash function h which takes arbitrary length inputs, and which produces hash codes of n bits in length.
- The resulting hash function should be collision resistant.
- The standard way of doing this is to use the Merkle–Damgard construction described in Algorithm 10.1

21

The Merkle–Damgard Construction

Algorithm 10.1: Merkle–Damgård Construction

$l = s - n$
Pad the input message m with zeros so that it is a multiple of l bits in length
Divide the input m into t blocks of l bits long, $m_1, \ldots, m_t$
Set H to be some fixed bit string of length n.
for $i = 1$ **to** t **do**
  $H = f(H \| m_i)$
end
return (H)

22

The Merkle–Damgard Construction

- In this algorithm the variable H is usually called the internal state of the hash function.
- At each iteration this internal state is updated, by taking the current state and the next message block and applying the compression function.
- At the end the internal state is output as the result of the hash function.
- Algorithm 10.1 describes the basic Merkle–Damgard construction, however it is almost always used with so called **length strengthening**.
- In this variant the input message is preprocessed by first padding with zero bits to obtain a message which has length a multiple of L bits (using L instead of l).

23

The Merkle–Damgard Construction

- Then a final block of L bits is added which encodes the original length of the unpadded message in bits.
- This means that the construction is limited to hashing messages with length less than 2^L bits.
- To see why the strengthening is needed consider a "baby" compression function f which maps bit strings of length 8 into bit strings of length 4 and then apply it to the two messages m_1 = 0b0, m_2 = 0b00.
- Whilst the first message is one bit long and the second message is two bits long, the output of the basic Merkle–Damgard construction will be $h(m_1) = f(0b00000000) = h(m_2)$, i.e. we obtain a collision.

24

The Merkle–Damgard Construction

- However, with the strengthened version we obtain the following hash values in our baby example
- $h(m_1) = f(f(0b00000000)||0b0001)$,
- $h(m_2) = f(f(0b00000000)||0b0010)$.
- These last two values will be different unless we just happen to have found a collision in
- Another form of length strengthening is to add a single one bit onto the data to signal the end of a message, pad with zeros, and then apply the hash function. Our baby example in this case would become
- $h(m_1) = f(0b01000000)$,
- $h(m_2) = f(0b00100000)$.

25

The Merkle–Damgard Construction

- Yet another form is to combine this with the previous form of length strengthening, so as to obtain
- $h(m_1) = f(f(0b01000000)||0b0001)$,
- $h(m_2) = f(f(0b00100000)||0b0010)$.

26

The MD4 Family

- A basic design principle when designing a compression function is that its output should produce an avalanche affect, in other words a small change in the input produces a large and unpredictable change in the output.
- This is needed so that a signature on a cheque for 30 pounds cannot be altered into a signature on a cheque for 30 000 pounds, or vice versa.
- This design principle is typified in the MD4 family which we shall now describe.
- Several hash functions are widely used, they are all iterative in nature. The three most widely deployed are MD5, RIPEMD-160 and SHA-1.
- The MD5 algorithm produces outputs of 128 bits in size, whilst RIPEMD-160 and SHA-1 both produce outputs of 160 bits in length.

27

The MD4 Family

- Recently NIST has proposed a new set of hash functions called SHA-256, SHA-384 and SHA-512 having outputs of 256, 384 and 512 bits respectively, collectively these algorithms are called SHA-2.
- All of these hash functions are derived from an earlier simpler algorithm called MD4.
- The following slide summarizes the facts about this family

28

The MD4 Family

- The seven main algorithms in the MD4 family are
- **MD4**: This has 3 rounds of 16 steps and an output bitlength of 128 bits.
- **MD5**: This has 4 rounds of 16 steps and an output bitlength of 128 bits.
- **SHA-1**: This has 4 rounds of 20 steps and an output bitlength of 160 bits.
- **RIPEMD-160**: This has 5 rounds of 16 steps and an output bitlength of 160 bits.
- **SHA-256**: This has 64 rounds of single steps and an output bitlength of 256 bits.
- **SHA-384**: This is identical to SHA-512 except the output is truncated to 384 bits.
- **SHA-512**: This has 80 rounds of single steps and an output bitlength of 512 bits.

29

The MD4 Family

- We discuss MD4 and SHA-1 in detail, the others are just more complicated versions of MD4, which we leave to the interested reader to look up in the literature.
- In recent years a number of weaknesses have been found in almost all of the early hash functions in the MD4 family, for example MD4, MD5 and SHA-1.
- Hence, it is wise to move all application to use the SHA-2 algorithms.

30

MD4

- In MD4 there are three bit-wise functions of three 32-bit variables
- f(u, v, w) = (u ∧ v) ∨ ((¬u) ∧ w),
- g(u, v, w) = (u ∧ v) ∨ (u ∧ w) ∨ (v ∧ w),
- h(u, v, w) = u ⊕ v ⊕ w.
- Throughout the algorithm we maintain a current hash state
- (H_1, H_2, H_3, H_4)
- of four 32-bit values initialized with a fixed initial value,
- H_1 = 0x67452301,
- H_2 = 0xEFCDAB89,
- H_3 = 0x98BADCFE,
- H_4 = 0x10325476.

31

MD4

- There are various fixed constants (y_i, z_i, s_i), which depend on each round. We have
- y_j = 0 for 0 ≤ j ≤ 15,
- y_j = 0x5A827999 16 ≤ j ≤ 31,
- y_j = 0x6ED9EBA1 32 ≤ j ≤ 47,
- and the values of z_i and s_i are given by following arrays,
- $z_{0...15}$ = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
- $z_{16...31}$ = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15],
- $z_{32...47}$ = [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15],
- $s_{0...15}$ = [3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19],
- $s_{16...31}$ = [3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13],
- $s_{32...47}$ = [3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15].

32

MD4

- The data stream is loaded 16 words at a time into X_j for 0 ≤ j < 16.
- The length strengthening method used is to first append a one bit to the message, to signal its end and then to pad with zeros to a multiple of the block length.
- Finally the number of bits of the message is added as a separate final block.
- We then execute the steps in Algorithm 10.2 for each 16 words entered from the data stream.

Algorithm 10.2: MD4 Overview

$(A, B, C, D) = (H_1, H_2, H_3, H_4)$
Execute Round 1
Execute Round 2
Execute Round 3
$(H_1, H_2, H_3, H_4) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$

33

MD4

- After all data has been read in, the output is the concatenation of the final value of H_1, H_2, H_3, H_4.
- The details of the rounds are given by Algorithm 10.3 where ≪denotes a bit-wise rotate to the left:

Algorithm 10.3: Description of the MD4 round functions

Round 1
for $j = 0$ **to** 15 **do**
  $t = A + f(B, C, D) + X_{z_j} + y_j$
  $(A, B, C, D) = (D, t \lll s_j, B, C)$
end
Round 2
for $j = 16$ **to** 31 **do**
  $t = A + g(B, C, D) + X_{z_j} + y_j$
  $(A, B, C, D) = (D, t \lll s_j, B, C)$
end
Round 3
for $j = 32$ **to** 47 **do**
  $t = A + h(B, C, D) + X_{z_j} + y_j$
  $(A, B, C, D) = (D, t \lll s_j, B, C)$
end

34

SHA-1

- We use the same bit-wise functions f, g and h as in MD4. For SHA-1 the internal state of the algorithm is a set of five, rather than four, 32-bit values
- (H_1, H_2, H_3, H_4, H_5).
- These are assigned with the initial values
- H_1 = 0x67452301,
- H_2 = 0xEFCDAB89,
- H_3 = 0x98BADCFE,
- H_4 = 0x10325476,
- H_5 = 0xC3D2E1F0.

35

SHA-1

- We now only define four round constants y_1, y_2, y_3, y_4 via
- y_1 = 0x5A827999,
- y_2 = 0x6ED9EBA1,
- y_3 = 0x8F1BBCDC,
- y_4 = 0xCA62C1D6.
- The data stream is loaded 16 words at a time into X_j for $0 \le j < 16$, although note the internals of the algorithm uses an expanded version of X_j with indices from 0 to 79.
- The length strengthening method used is to first append a one bit to the message, to signal its end and then to pad with zeros to a multiple of the block length.
- Finally the number of bits of the message is added as a seperate final block.

36

SHA-1

- We then execute the steps in Algorithm 10.4 for each 16 words entered from the data stream.
- Note the one bit left rotation in the expansion step, an earlier algorithm called SHA (now called SHA-0) was initially proposed by NIST which did not include this one bit rotation.
- This was however soon replaced by the new algorithm SHA-1. It turns out that this single one bit rotation improves the security of the resulting hash function quite a lot.

Algorithm 10.4: SHA-1 Overview

$(A, B, C, D, E) = (H_1, H_2, H_3, H_4, H_5)$
/* Expansion */
for $j = 16$ **to** 79 **do**
  $X_j = ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \ll 1)$
end
Execute Round 1
Execute Round 2
Execute Round 3
Execute Round 4
$(H_1, H_2, H_3, H_4, H_5) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$

37

SHA-1

- The details of the rounds are given by Algorithm 10.5.
- After all data has been read in, the output is the concatenation of the final value of H_1, H_2, H_3, H_4, H_5.

Algorithm 10.5: Description of the SHA-1 round functions

Round 1
for $j = 0$ **to** 19 **do**
  $t = (A \ll 5) + f(B, C, D) + E + X_j + y_1$
  $(A, B, C, D, E) = (t, A, B \ll 30, C, D)$
end
Round 2
for $j = 20$ **to** 39 **do**
  $t = (A \ll 5) + h(B, C, D) + E + X_j + y_2$
  $(A, B, C, D, E) = (t, A, B \ll 30, C, D)$
end
Round 3
for $j = 40$ **to** 59 **do**
  $t = (A \ll 5) + g(B, C, D) + E + X_j + y_3$
  $(A, B, C, D, E) = (t, A, B \ll 30, C, D)$
end
Round 4
for $j = 60$ **to** 79 **do**
  $t = (A \ll 5) + h(B, C, D) + E + X_j + y_4$
  $(A, B, C, D, E) = (t, A, B \ll 30, C, D)$
end

38

Hash Functions and Block Ciphers

- One can also make a hash function out of an n-bit block cipher, E_K.
- There are a number of ways of doing this, all of which make use of a constant public initial value IV .
- Some of the schemes also make use of a function g which maps n-bit inputs to keys.
- We first pad the message to be hashed and divide it into blocks
- x_0, x_1, . . . , x_t,
- of size either the block size or key size of the underlying block cipher, the exact choice of size depending on the exact definition of the hash function being created.

39

Hash Functions and Block Ciphers

- The output hash value is then the final value of H_i in the following iteration
- H_0 = IV,
- $H_i = f(x_i, H_{i-1})$.
- The exact definition of the function f depends on the scheme being used.
- We present just three, although others are possible.
- **Matyas–Meyer–Oseas** hash
- $f(x_i, H_{i-1}) = E_g(H_{i-1})(x_i) \oplus x_i$.
- **Davies–Meyer** hash
- $f(x_i, H_{i-1}) = E_\alpha(H_{i-1}) \oplus H_{i-1}$, where $\alpha = x_i$.
- **Miyaguchi–Preneel** hash
- $f(x_i, H_{i-1}) = E_g(H_{i-1})(x_i) \oplus x_i \oplus H_{i-1}$.

40

Message Authentication Codes

- Given a message and its hash code, as output by a cryptographic hash function, ensures that data has not been tampered with between the execution of the hash function and its verification, by recomputing the hash.
- However, using a hash function in this way requires the hash code itself to be protected in some way, by for example a digital signature, as otherwise the hash code itself could be tampered with.
- To avoid this problem one can use a form of keyed hash function called a **message authentication code**, or **MAC**.
- This is a symmetric key algorithm in that the person creating the code and the person verifying it both require the knowledge of a shared secret.
- Suppose two parties, who share a secret key, wish to ensure that data transmitted between them has not been tampered with.

41

Message Authentication Codes

- They can then use the shared secret key and a keyed algorithm to produce a check-value, or MAC, which is sent with the data.
- In symbols we compute
- code = $MAC_k(m)$
- where
- MAC is the check function,
- k is the secret key,
- m is the message.
- Note we do not assume that the message is secret, we are trying to protect data integrity and not confidentiality.
- If we wish our message to remain confidential then we should encrypt it before applying the MAC.

42

Message Authentication Codes

- After performing the encryption and computing the MAC, the user transmits
- $e_{k1}(m)||MAC_{k2}(e_{k1}(m))$.
- This is a form of encryption called a **data encapsulation mechanism**, or **DEM** for short.
- Note, that different keys are used for the encryption and the MAC part of the message and that the MAC is applied to the ciphertext and not the message.
- Before we proceed on how to construct MAC functions it is worth pausing to think about what security properties we require.
- We would like that only people who know the shared secret are able to both produce new MACs or verify existing MACs.
- In particular it should be hard given a MAC on a message to produce a MAC on a new message.

43

Producing MACs from Hash Functions

- A collision-free cryptographic hash function can also be used as the basis of a MAC.
- The first idea one comes up with to construct such a MAC is to concatenate the key with the message and then apply the hash function.
- For example
- $MAC_k(M) = h(k||M)$.
- However, this is not a good idea since almost all hash functions are created using methods like the Merkle–Damgard construction.
- This allows us to attack such a MAC as follows: We assume first that the non-length strengthened Merkle–Damgard construction is used with compression function f.

44

Producing MACs from Hash Functions

- Suppose one obtains the MAC c_1 on the t block message m_1
- $c_1 = MAC_k(m_1) = h(k||m_1)$
- We can then, without knowledge of k compute the MAC c_2 on the t + 1 block message $m_1||m_2$ for any m_2 of one block in length, via
- $c_2 = MAC_k(m_1||m_2) = f(c_1||m_2)$.
- Clearly this attack can be extended to appending an m_2 of arbitrary length.
- Hence, we can also apply it to the length strengthened version.
- If we let m_1 denote a t block message and let b denote the block which encodes the bit length of m_1 and we let m_2 denote an arbitrary new block, then from the MAC of the message m_1 one can obtain the MAC of the message $m_1||b||m_2$.
- *You can read the remaining details in the textbook (this is a very poorly written chapter)*

45

Chapter Summary

- Hash functions are required which are both preimage, collision and second-preimage resistant.
- Due to the birthday paradox the output of the hash function should be at least twice the size of what one believes to be the limit of the computational ability of the attacker.
- More hash functions are iterative in nature, although most of the currently deployed ones have recently shown to be weaker than expected.
- A message authentication code is in some sense a keyed hash function.
- MACs can be created out of either block ciphers or hash functions.

46