Security (CS4028)

Lecture 4. Hash Functions and Message Authentication Codes

Chunyan Mu

chunyan.mu@abdn.ac.uk

The School of Natural and Computing Sciences

Lots of slides taken from Matthew Collinson's

Schedule

	Week	Lecture 1	Lecture 2	Tutorial
	1	Intro to course & security	Intro to Crypto	
⇒	2	Symmetric Crypto	Hash	Math for crypto
	3	Asymmetric Crypto-1	Asymmetric Crypto-2	Symmetric Crypto
	4	Signatures	Zero Knowledge Proof	Asymmetric Crypto
	5	Certificates	Authentication	Signature & certificates
	6	Access Control	AC models	Authentication
	7	Information flow	Management	Access control
	8	Protocols	Communications	Concepts & management
	9	Network security	Network security	Protocols and communications
	10	Advanced topics	Advanced topics	Network & mgt
	11	Revision		

Lecture 4: Hash functions and MAC

Last lecture

- Symmetric cryptography
- DES in details

This lecture

- Hash functions
- MAC

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Hash Functions in general

An old idea

- Used in CS for a long time, not just security
- A hash function *h* on bit strings has two properties:
  (1) **Ease of computation**:
    - given x, it is easy to compute the **hash value** $h(x)$

  (2) **Compression**: h takes inputs of *any* (variable) length, and maps to outputs of some *fixed* length L:

$$h : \{0,1\}^* \rightarrow \{0,1\}^L$$

Hash Functions in general

Example: Hashing byte-blocks into a single byte by XOR

- Take a string of bits.
- Pad to length a whole number of bytes.
- Split it into bytes.
- For each n in the range $n = 1$ to 8, take the bitwise XOR of all the bits in position n of all of the input bytes.
- Output the resulting bytes.

Hash Functions in general

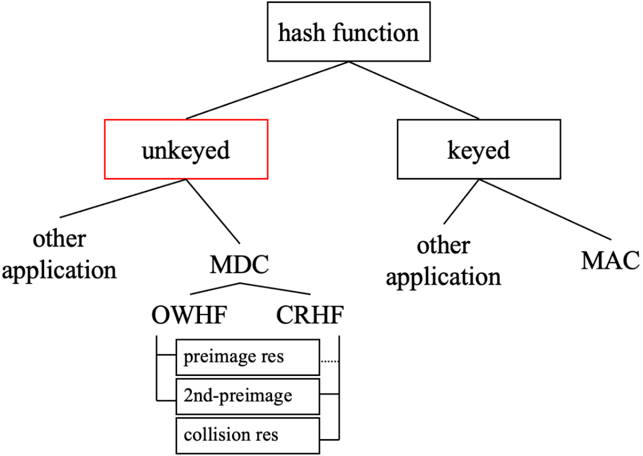
Example: Hashing byte-blocks into a single byte by XOR

- Take a string of bits.
- Pad to length a whole number of bytes.
- Split it into bytes.
- For each n in the range $n = 1$ to 8, take the bitwise XOR of all the bits in position n of all of the input bytes.
- Output the resulting bytes.

Remark

- This is an 8-bit hash function with a **block size** of one byte.
- This hash is not good for security!

Hash functions in general



keys

- Hash functions come in keyed and un-keyed variants.

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Properties of Hash Functions

Pre-image

- For a given hash value y, any input x such that $h(x) = y$ is called pre-image of y.
- A hash function is pre-image resistant if, for any given hash values, it is hard to find (compute) an input that yields it, given only the hash value.

Properties of Hash Functions

One-way functions

- A function $h : X \rightarrow Y$ is one-way if it:
  - There is an efficient algorithm for computing the value $h(x)$ for any given x in X, and
  - It is preimage resistant: there is no efficient algorithm that finds a preimage for y in Y. That is to say, we don't know how to find x such that $h(x) = y$.
- Notes
  - Sometimes don't want it to be too efficient.
  - The '*no*' in the last point is a bit subtle. It means something like 'no known'. Sometimes it means '*no known, and there is reason to believe there is none, given what people currently think about computational complexity*'.

Properties of Hash Functions

One-way hash functions

- A one-way hash function (OWHF) is a hash function that is preimage resistant.
- Check you understand: Convince yourself that a one-way hash function is a one-way function that is also a hash function.

Applications of OWHFs

Integrity checks on files and programs

- Distribute a file, x, and publish its hash value (message digest, checksum): $h(x)$.
- If x is modified to some x', then it will (for a good hash function) give a new: $h(x') \neq h(x)$.
- So the value of the hash will not match.
- No secret key is required.
- In this usage, hashes are also known as modification detection codes (MDC), message integrity codes (MIC).
- Can think of hash value as being like a digital fingerprint.

Applications of OWHFs

Authentication

- You want to verify that someone, say A, has a copy x of a particular file (for which you also have a copy, x'). You don't want them to actually send x itself, perhaps because of its size or because they want to keep it a secret and the channel is insecure.
- You can just ask A to send the hash value, $h(x)$ to you.
- You can calculate $h(x')$.
- If $h(x) = h(x')$, then you have a degree of confidence that A really does have genuine access to the file

Applications of OWHFs

Storage of authentication data

- Allows us to avoid storing the full (secret) authentication data.
- Only have to store complementary data
  - E.g. password hashes in password files.

Applications of OWHFs

Other requirements

- One might want to use hashes for other purposes.
- Example 1. Bob takes the fact that $h(x)$ and x (supplied by Alice) match and then passes $h(x)$ to Charlie, telling him Alice's input x can be trusted if it matches $h(x)$.
  - Problem: maybe Alice can compute x and x' with

$$h(x) = h(x')$$

- Example 2. Bob uses $h(x)$ from Alice as a proof by him of authenticity of x.
  - Problem: given $h(x)$ and x, maybe Bob can compute x' s.t.

$$h(x) = h(x')$$

Collisions

- A collision is a pair of inputs x and x' such that:

$$h(x) = h(x')$$

- Apply the pigeonhole principle to hashes
  - There are more strings of variable length than there are strings of the given fixed length L.
  - Each string gets a (unique) hash value.
  - Thus, some input strings must get the the same hash value.
  - There must be some collisions.

Collisions for good hashes

- Hash value should be a kind of fingerprint of the original (pre-image) value.
- It should be unlikely that there is any other given input value hashes to the same value.
    - Given x and x', it should be very unlikely that $h(x) = h(x')$.
    - L needs to be big enough to allow for this.
- There should not be too many collisions. If it had absolutely no collisions, then it would not really be hashing.
- A hash function does not absolutely guarantee that a particular hash value came from a particular input; it can only give a very high-degree of confidence.

Collisions-resistance and pre-image resistance

- A hash is collision-resistant if it is hard to find any two inputs with the same hash value.
  - They must exist, but they are hard to search for by computational and other cryptological means.
  - The (awful) terminology collision-free is also comm
- Recall: a hash is pre-image resistant if, for any given hash value, it is hard to find an input that yields it, given only the hash value.
- Note the difference between computational problems:
  - finding $\underline{\text{any two}}$ x and x' that collide, and
  - finding $\overline{\text{an } x'}$ that collides with a $\underline{\text{given}}$ x (with hash value $h(x)$).

2^{nd} and pre-image resistance

- Sometimes both the input and its hash value are known to a potential attacker.
  - Attacker knows both x and its hash value $h(x)$.
  - See Yuval's attack on signatures below.
- It still needs to be difficult for the attacker to find another input with the same hash value.
- A hash is called 2^{nd} pre-image resistant if it is hard to find $x' \neq x$ such that $h(x') = h(x)$, given any x and $h(x)$.

Recap: computational problems

- Pre-image Resistant: It should be hard to find a message with a given hash value.
- Collision Resistant: it should be hard to find two messages with the same hash value.
- 2^{nd} Pre-image Resistant: it should be hard to find another message with the same hash value.

Relationships

- Collision resistant implies 2nd preimage resistant. (Check you understand: prove this.)
- A hash that is not preimage resistant is unlikely to be collision resistant (although this is not impossible).
- A hash that is not preimage resistant is unlikely to be 2^{nd} preimage resistant.

Avalanche

- Avalanche effect: a small change to the input value (usually) changes the output value a lot.
- Moreover, in a good hash function, a single bit change will lead to approximately half of the output values changing, on average.
  - Our toy XOR example above is not a good hash function for security purposes.
  - Ideally, an attacker should have subjective probability of approximately $1/2^L$ for any particular output (of length L) with a previously untested input.
- Idea: if this wasn't the case, an attacker could use this information to tune a search for collisions.

Avalanche in a 8-bit hash (digest)

$$\boxed{\text{01010101010S}} \longrightarrow \boxed{\textit{hash}} \longrightarrow \boxed{\text{10111011}}$$
$$\boxed{\text{01010100010S}} \longrightarrow \boxed{\textit{hash}} \longrightarrow \boxed{\text{00011110}}$$

- S is some fixed binary string.
- One digit of input changes.
- About half of the output bits change.

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Attacks on Hashes

Colliding Birthdays

Recall the birthday paradox. Think about the three questions.

1. How likely is it that a randomly chosen other person has the same birthday as you?
2. How likely is it that there is another person in this class who has the same birthday as you?
3. How likely is it that there are any two people in the class who have the same birthday?

Attacks on Hashes

Colliding Birthdays: Q1

- Q1: How likely is it that a randomly chosen other person has the same birthday as you?
- We need assumptions:
  - "*random*" above means "*uniformly at random*"
  - People are equally likely to have been born on any day of the year.
- A: probability of collision $= 1/365 \approx 0.003$.
  - Note: probability of no collision $= 364/365$.

Attacks on Hashes

Colliding Birthdays: Q2

- Q2: How likely is it that there is another person in this class who has the same birthday as you?
- N people in the room:
  - Below, $N = 23$, for example
- There are $N - 1$ people with whom you might share a birthday.
  - There are 22 possible people with whom you might share a birthday.
  - Assume people's birthdays are independent.
  - Gives a $(364/365)^{22} \approx 0.94$ probability of <u>no</u> collision for somebody else with you.
  - So only about 0.06 probability of collision.

Attacks on Hashes

Birthday Paradox and Hashes

- Q3: How likely is it that there are any two people in the class who have the same birthday?
- There are $N \times (N-1)/2$ pairs of people in the class
  - $N = 23$ gives 253 pairs
  - Probability of <u>no</u> collision $=$

$$(1 - 1/365) \times (1 - 2/365) \times \cdots \times (1 - (23-1)/365).$$

  - Probability of collision ≈ 0.51.

Attacks on Hashes

Birthdays: sampling

- Drawing random elements with replacement from a set of N elements, a repeat is likely after approximately $\sqrt{N} = N^{1/2}$ selections.
- If $N = 2^L$ elements (the size of our space of hash values), then a repeat is likely after approximately $(N^{1/2}) = 2^{L/2}$ selections.

Attacks on Hashes

Birthday attack

- Usable against any un-keyed hash function
- Based on birthday paradox
- Running time $\mathcal{O}(2^{L/2})$, where L is the number of output bits in the hash
- Good hash functions will:
  - Have sufficiently large L
  - Be such that finding a collision will not be much better than $2^{L/2}$
  - And sometimes meet more requirements

Attacks on Hashes

Yuval's Birthday attack example

- An attack on use of hash for integrity.
- 64-bit hash, so $L = 64$.
- If machine hashes a million messages per second, finding a collision for a given hash value takes about $600,000$ years. Finding a pair takes about an hour.
- Mallory has two contracts, one that pays him £1000, the other £$100,000$. The first is to be signed with the hash. He makes $2^{L/2} = 2^{32}$ minor variations in each (e.g., spaces/control chars), and finds a pair with the same hash. Later claims second document was signed, not first.

– Source: Schneier, Applied Crypto

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Hash Functions

Well known hash functions

- MD5
  - output 128 bits
  - collision resistance completely broken by researchers
- SHA1
  - output 160 bits
  - considered insecure for collision resistance
  - one-wayness still holds
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively

Well known Hash Function

Secure Hash Function (SHA)

Characteristics	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Maximum Message Size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

Well known Hash Function

SHA-512

- Append padding bits, append length, initialise hash buffer
- Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm
- Output the final state value as the resulting hash

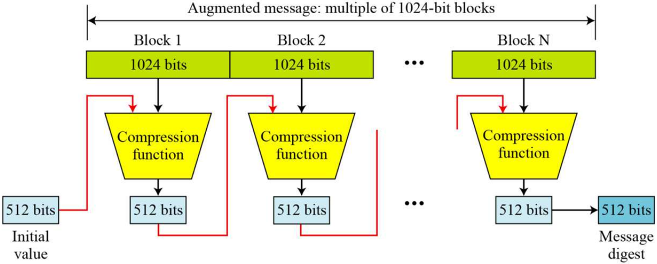


Image taken from: `https://www.scribd.com/presentation/444572482/SHA512`

Well known Hash Function

Example: SHA-512 length restriction

Suppose we need to send a message that is 2^{128} bits in length. How long does it take for a communication network with a data rate of 2^{64} bits per second to send this message?

Well known Hash Function

Example: SHA-512 length restriction

Suppose we need to send a message that is 2^{128} bits in length. How long does it take for a communication network with a data rate of 2^{64} bits per second to send this message?

Solution

A communication network that can send 2^{64} bits per second is not yet available. Even if it were, it would take many years to send this message. This tells us that we do not need to worry about the SHA-512 message length restriction.

Well known Hash Function

Example: SHA-512 length restriction

How many pages are occupied by a message of 2^{128} bits?

Well known Hash Function

Example: SHA-512 length restriction

How many pages are occupied by a message of 2^{128} bits?

Solution ???

Suppose that a character is 32 bit, or 2^5 bits. Each page is less than 2048, or approximately 2^{10} characters . So 2^{128} bits need at least $2^{128}/2^{15} = 2^{113}$ pages. This again shows that we need not worry about the message length restriction.

Well known Hash Function

Example: SHA-512 padding

Do we need padding if the length of the original message is already a multiple of 1024 bits?

Well known Hash Function

Example: SHA-512 padding

Do we need padding if the length of the original message is already a multiple of 1024 bits?

Solution

Yes we do, because we need to add the length field. So padding is needed to make the new block a multiple of 1024 bits.

Well known Hash Function

Example: SHA-512 padding

What is the minimum number of padding bits that can be added to a message?

Well known Hash Function

Example: SHA-512 padding

What is the minimum number of padding bits that can be added to a message?

Solution

The minimum length of padding is 0 and it happens when $(-M - 128) \mod 1024 = 0$. This means that $|M| = -128 = 896$ mod 1024 bits. In other words, the last block in the original message is 896 bits. We add a 128-bit length field to make the block complete.

Well known Hash Function

Example: SHA-512 padding

What is the maximum number of padding bits that can be added to a message?

Well known Hash Function

Example: SHA-512 padding

What is the maximum number of padding bits that can be added to a message?

Solution

The maximum length of padding is 1023 and it happens when $(-|M| - 128) = 1023 \mod 1024$. This means that the length of the original message is $|M| = (-128 - 1023) \mod 1024$ or the length is $|M| = 897 \mod 1024$. In this case, we cannot just add the length field because the length of the last block exceeds one bit more than 1024. So we need to add 897 bits to complete this block and create a second block of 896 bits. Now the length can be added to make this block complete.

Well known Hash Function

MD5

- Designed by Ronald Rivest
- Latest in a series of MD2, MD4
- Produces a 128-bit hash value
- Until recently was the mostly widely used hash algorithm
- Specified as an Internet standard RFC1321

Well known Hash Function

MD5 Implementation

- Pad the message so its length (in bits) is 448 mod 512
- Append a 64-bit length value to the message
- Initialise 4 word (128-bit) MD buffer (A,B,C,D)
- Process message in 16-word (512) blocks
- Output hash value is the final buffer value

Well known Hash Function

Padding and length in MD5

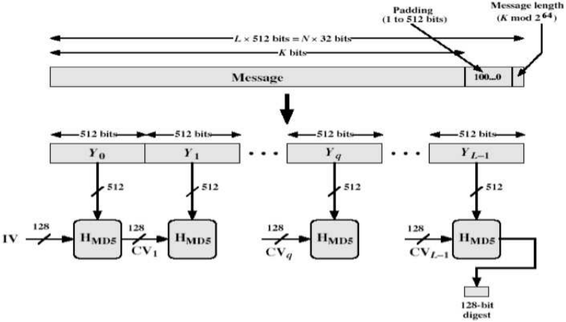


Image taken from:
`https://www.dhsinformatics.com/md5-message-digest-algorithm/`

Cryptography ...

- The cryptographic algorithms in previous lectures provide secrecy, or confidentiality, but not integrity.
- In some cases, we may not even need secrecy but instead must have integrity.

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Message Authentication

What is it?

- A message, file or document is said to be authentic if it is genuine and comes from its alleged source.
- Message authentication is a procedure that allows communicating parties to verify that received messages are authentic.
- Need to verify that:
  - the contents have not been altered
  - the source is authentic
- May need to verify **timeless** of the message (it has not been artificially delayed and replayed) and its **sequence** relative to other messages between the parties.

Message Authentication

Message Authentication is concerned with

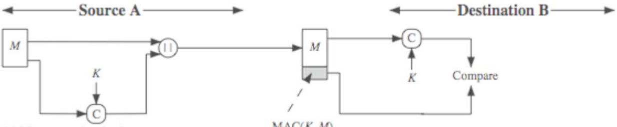
- Protecting the integrity of a message
- Validating the identity of originator

Message Authentication

Definition: Message Authentication Code (MAC)

A function of the message and a secret key that produces a fixed-length value that serves as the authenticator.

- Generated from message + secret key: $MAC = C(K, M)$
- A small fixed-sized block of data
- Appended to message as a signature when sent
- Receiver performs same computation on the message and checks if it matches the MAC



(a) Message authentication

Message Authentication

MAC Properties

- A MAC is a cryptographic checksum $MAC = C(K, M)$
  - condenses a variable-length message M
  - using a secret key K
  - to a fixed-sized authenticator
- A many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult

Message Authentication

Requirements for MAC

Considering different attacks, the MAC needs to satisfy the following:

- knowing a message and MAC, is infeasible to find another message with same MAC
- MACs should be uniformly distributed
- MAC should depend equally on all bits of the message

Message Authentication

Keyed Hash Functions as MAC

- Want a MAC based on a hash function?
  - Hash functions are generally faster
  - Crypto hash function code is widely available
- Need a hashing including a key along with message...
  - but hashing internally has no key!
- Original proposal:
  - KyedHash = Hash(Key | Message)
  - Some weakness were found with this
  - Eventually led to the development of Hash-based Message Authentication Code (HMAC)

Hash Function

Keyed Hash as MAC (HMAC)

- Specified as internet standard RFC2104
- Use hash function (inner hash and outer hash) on the message:

$$\mathrm{HMAC(K,M) = Hash((K^{+} \oplus opad)||Hash((K^{+} \oplus ipad)||M))}$$

- where K^{+} is the key padded out to size
- opad (inner padding), ipad (outer padding) are specified padding constants
    - ipad: 64-byte (512-bit) value mad by repeating the byte $0x36$ (00110110) across the length of the key block
    - opad: 64-byte (512-bit) value mad by repeating the byte $0x5C$ (01011100) across the length of the key block
- $||$ denotes concatenation.

Hash Function

HMAC Overview

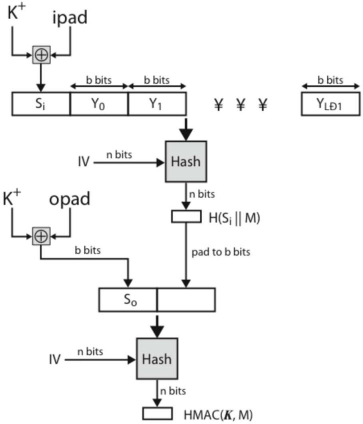


Image taken from:
`https://www.researchgate.net/figure/SRTP-Authentication-Tag-Generation-using-HMAC-SHA1_fig3_228821876`

Outline

Introduction

Properties of hash functions

Attacks on Hashes

Well known hash functions

MAC

Summary

Summary

This lecture

- Hash function
- Message Authentication Code (MAC)

Next lecture

- Public key cryptography