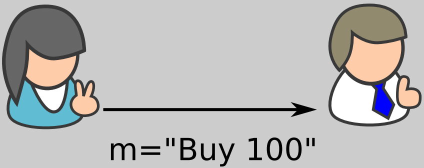
50.020 Security

L3 - Hash Functions

Nils Ole Tippenhauer

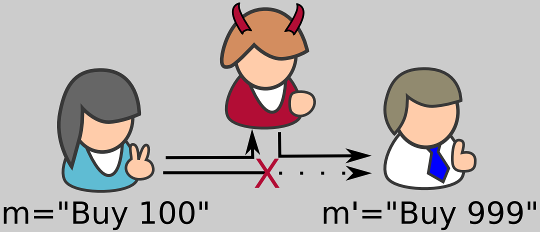
Data Integrity

Data Manipulation attacks



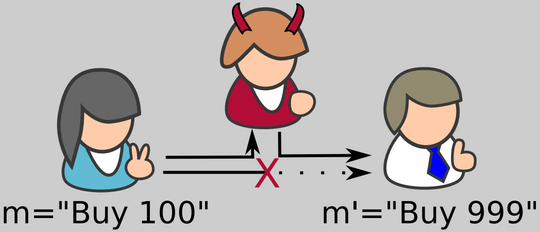
- Alice sends Bob a message:
  - "Hi Bob, I'm Alice, please buy 100 stocks of Company A"
- Alice sends the message in plaintext
- Attacker Even wants to manipulate Alice's stock trade.
  - Eve can jam, eavesdrop and insert
- What kind of attacks are possible here?

Data Manipulation attacks



- Attack example: Attacker eavesdrops, jams, spoofs similar message:
  - "Hi Bob, I'm Alice, please buy 999 stocks of Company B"
- Bob assumes the message is from Alice, buys stocks for her
- What is the problem here?

Data Manipulation attacks



- Attack example: Attacker eavesdrops, jams, spoofs similar message:
  - "Hi Bob, I'm Alice, please buy 999 stocks of Company B"
- Bob assumes the message is from Alice, buys stocks for her
- What is the problem here?

- Secure authentication and integrity of the message

How to protect the message?

- Obvious idea: encrypt the message (e.g., using OTP)

Example (Using OTP to encrypt "buy100")

```
- "buy100"= 0x6275793130300a
- Key     = 0xA29C7B1E0E3AEE
- Result  = 0xC0E9022F3E0AE4
```

How to protect the message?

- Obvious idea: encrypt the message (e.g., using OTP)

Example (Using OTP to encrypt "buy100")

```
- "buy100"= 0x6275793130300a
- Key     = 0xA29C7B1E0E3AEE
- Result  = 0xC0E9022F3E0AE4
```

- Can an eavesdropper break the confidentiality of the message?
- Can an eavesdropping and injecting attacker change the content?

Does symmetric encryption protect data integrity?

No! Confidentiality does not imply integrity

Example (OTP and "buy100")

```
- "buy100"= 0x6275793130300a
- Key     = 0xA29C7B1E0E3AEE
- Result  = 0xC0E9022F3E0AE4
- mask    = 0x00000008090900 <- "buy100" ^ "buy999"
- Result  = 0xC0E90227303E4
- Plaintex= 0x6275793939390a = "buy999"
```

- As integrity is not protected, authenticity is also not protected

How does this attack work?

- We assume the attacker knows the message m="buy100"
- Lets assume the attacker wants to change to m'="buy999"
- mask on the last slide is the binary XOR of both strings=$m \oplus m'$
- With $m \oplus k=c$, the attacker creates $c \oplus mask=c'$,
- Decrypting c' with k yields:

$c' \oplus k=((m \oplus k) \oplus (m \oplus m')) \oplus k=m'$

Other measures to protect integrity

- Block ciphers are not always enough
- We need a dedicated tool to validate message integrity

Checksums

Checksum basics

- Checksums are widely used to detect errors
  - Transmission errors in unreliable medium (e.g., wireless)
  - Storage errors
  - Can also be used to check if two files are identical
  - Example other uses: ISBN numbers, credit card numbers
- Main characteristics:
  - Checksums are computed when data is created
  - Compressing: checksums are smaller than the data processed
  - Any number of bit flips in data will lead to different checksum[1]

[1] With small probability of fails for larger number or errors

Cyclic redundancy checks (CRC)

- CRCs are widely used (e.g., Ethernet) to detect errors
- For fixed length message blocks, they produce an n-bit code $H(m)$
- To transmit m using CRC:
  - Alice computes $H(m)$ based on m
  - Alice transmits m and $H(m)$ to Bob
  - Bob recomputes code $H(m')$ based on m', accepts m' if $H(m') = H(m)$
- Simple example:
  - Parity bits. $H(m) = 0$ if m contains an even number of ONEs.

CRC Security Analysis

- System: A sends m to be, together with CRC $H(m)$. m and $H(m)$ are encrypted with OTP
- Attacker: Can modify encrypted m, must be undetected
- Requirement: Changes to the data are detected by receiver
- Secure?

CRC Security Analysis

- System: A sends m to be, together with CRC $H(m)$. m and $H(m)$ are encrypted with OTP
- Attacker: Can modify encrypted m, must be undetected
- Requirement: Changes to the data are detected by receiver
- Secure?

- Problem: crc() is *linear*
  - crc(m$\oplus$ x) = crc(m)$\oplus$ crc(x)
  - Attacker can compute crc(x) and XOR it with crc(m) for correct crc
  - In particular: E(crc(m))$\oplus$ crc(x) = E(crc(m$\oplus$ x)) (stream cipher/OTP)

Extended Example on CRC

- Message m sent with concatenated CRC32 checksum H(m)
- Attacker wants to change m into m' with valid H(m')
- Attacker can compute mask $x = m \oplus m'$ to manipulate m
  - Even if not all of m is known, target area can be manipulated
- But how can the attacker know H(m) without knowing m?
  - Without H(m), how to compute mask $y = H(m) \oplus H(m')$?
- Assume $H(a \oplus b) = H(a) \oplus H(b)$ for CRC
  - CRC32 is linear. Why? We will discuss much later.
  - If you don't believe me, try this yourself[2]
- Attacker knows only $H(m) \oplus k$ (from eavesdropping)
  - To change that to $H(m') \oplus k$, Attacker has to $\oplus$ with H(x)
  - Why? Because $H(m) \oplus k \oplus H(x) = k \oplus (H(m) \oplus H(x))$
  - And (based on assumption) $k \oplus (H(m) \oplus H(x)) = k \oplus (H(m \oplus x))$
  - Which is $k \oplus (H(m \oplus x)) = k \oplus (H(m'))$

[2] Set initial state of H(b) to zero if you do so

Cryptographic Hash Functions

Cryptographic properties for functions

- In cryptography, *preimage* resistance means that given $y = f(x)$
  - it is *hard* to find the input x for f to produce y
- *Second pre-image* resistance means that given x and f
  - it is *hard* to find an input x' for f such that $f(x) = f(x')$
- *Collision* resistance means that given f
  - it is *hard* to find any two inputs x, x' for f such that $f(x) = f(x')$
- *Random oracle* property: A random oracle maps each unique input to random output with uniform distribution
  - Informally: for two correlated inputs m_1 and m_2, the output of f is completely uncorrelated
- CRCs have only preimage resistance

Design goals for hash functions

- Cryptographic hash functions are design to have all four properties
  - Preimage resistance
  - Second preimage resistance
  - Collision resistance
  - Random oracle property
- Using cryptographic hash functions, *message authentication codes* can be constructed
- We now discuss special algorithms, similar goals can be achieved with block ciphers
- Standard hash functions are not designed to have all of these properties

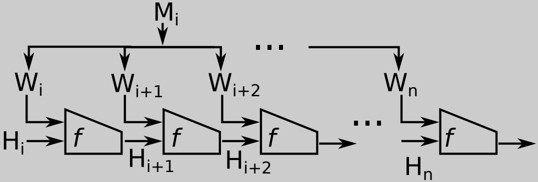
SHA-1

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

We will explain hash functions based on SHA-1. It has the following characteristics:

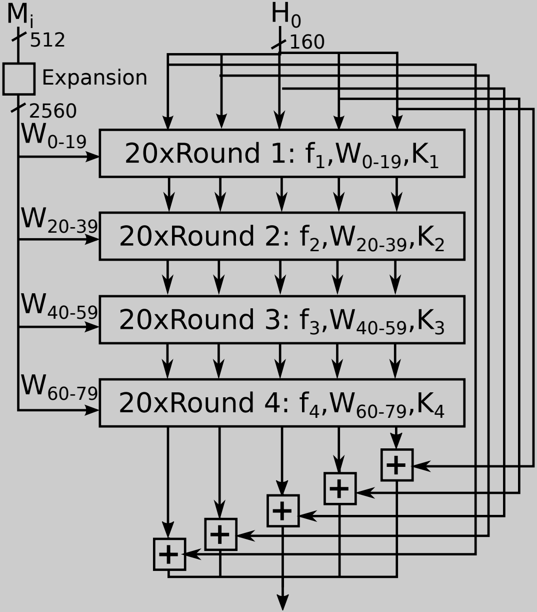
- Processes (1+) input blocks of 512 bit
- Pre-defined initial state of 160 bit
- Hash output is a 160 bit block
- Uses Merkle-Dåmgard construction
- 80 internal rounds in total

Merkle-Dåmgard construction

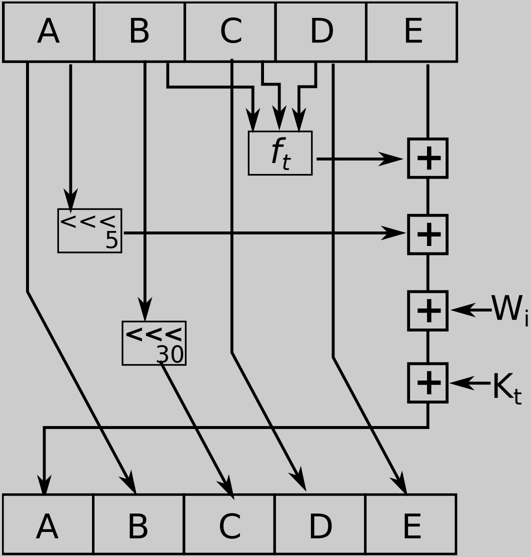


- Merkle-Dåmgard is a construction for cryptographic hashes:
  - Repeated application of a collision resistant compressing function
  - Each stage uses previous output and new chunk of input
- In SHA-1
  - SHA-1 has a constant (public) initial values in the MD chain
  - 512 bit input blocks are expanded into 2560 bit=80· 32 bit words
  - 4 stages, each stage has 20 rounds of compression
  - Each stage has different constants K_t and a non-linear function f_t

Overall SHA-1 operation



One round in SHA-1



Why 80 rounds?

- Increasing the number of rounds has several benefits:
  - It makes brute force attacks more expensive (each hashing takes longer)
  - It makes attacks relying on *differential cryptanalysis* harder
- The exact value for SHA-1 was most likely chosen as compromise between effort and security
- For SHA-2, 64 rounds are default. Attacks have been found for 52 round versions

Cryptanalysis of hash functions

- Two potential goals for attacker: find preimages or collisions
- Collisions are much easier to find, but less useful
- It has been shown that for MD, if f is collision resistant, then H is collision resistant
- Attacking the collision resistance of f is a first part of attack
  - Find two plaintexts that hash to the same value
  - What is the estimated effort for an n bit hash? 2^n?

Cryptanalysis of hash functions

- Two potential goals for attacker: find preimages or collisions
- Collisions are much easier to find, but less useful
- It has been shown that for MD, if f is collision resistant, then H is collision resistant
- Attacking the collision resistance of f is a first part of attack
  - Find two plaintexts that hash to the same value
  - What is the estimated effort for an n bit hash? 2^n?

- Actually, it is only $2^{n/2}$. Why?

Birthday paradox:

- What is the probability, that in a group of n people, two have the same birthday?
- Variant: for which group size, the probability approaches 0.5?

Birthday paradox:

- What is the probability, that in a group of *n* people, two have the same birthday?
- Variant: for which group size, the probability approaches 0.5?

- for 23 people, the probability is 50%
- for 70 people, the probability is 99.9%

Birthday paradox:

- What is the probability, that in a group of *n* people, two have the same birthday?
- Variant: for which group size, the probability approaches 0.5?

- for 23 people, the probability is 50%
- for 70 people, the probability is 99.9%

- This is related to collisions for hashes: each additional plain text could match *n* hashes
- *n* inputs result in $\approx n^2/2$ pairings
- As result, a minimum hash length of 160 bits is usually suggested
- A 160 bit has relates to 2^{80} effort to find collision (considered infeasible today)

How to use this for an attack

- Collisions can be directly be used to attack
  - Commitment schemes
  - Digital signature schemes
  - TLS certificates (more on them later, breaks TLS)
- Anything where the plaintext is under direct control of attacker
- Attacks have been demonstrated for MD5, a precursor similar to SHA-1
  - Keyword: "MD5 Collisions Inc"
- Birthday paradoxon does not help for second preimage finding
  - Our message authentication system can use SHA-1 safely

Cryptanalysis of SHA-1

- In Feb 2005, researchers found the following:
  - Collisions can be found with effort 2^{69} steps (instead of 2^{80}, factor 2048)
  - In 2009, that result was claimed to be improved to 2^{52} steps (but found to be incorrect)
  - If assuming 2^{60} tries required, and 2^{14} ops per SHA-1 [3]
  - Nowadays, breaking SHA-1 would probably still cost >1M$ per hash
    - In 2015, $700k
    - In 2018, $173k
    - In 2021, $43k...

[3] schneier.com/blog/archives/2012/10/when_will_we_se.html

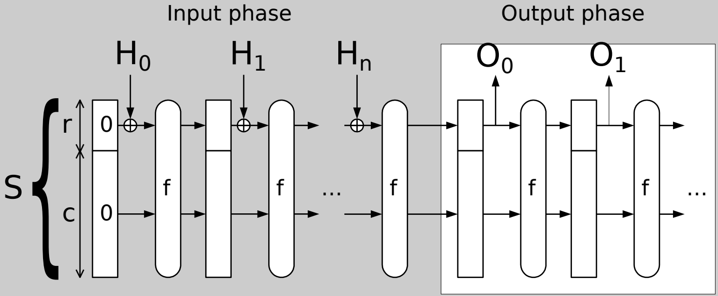
SHA-1, SHA-2, SHA-3

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

- SHA-2 was designed by NSA (like SHA-1), and published in 2001
- US National Institute of Standards and Technology (NIST) "promotes" security standards
- Successor of SHA-2 was chosen in a semi-public process
- In Oct 2012, Keccak was selected as SHA-3 algorithm
  - Focus on security and implementation speed
- SHA-1 appears to have weaknesses as discussed
  - SHA-2 shares a lot of the structure
- SHA-3 should be considered for high-security projects

SHA-3

- SHA-3 (Keccak) is fundamentally different to SHA-1/SHA-2
- It uses a "sponge" construction instead of MD
- *r* bits of message are "fed" into *S* per round
- *r* bits of output per round can be taken out afterwards



Conclusion

- Message integrity is not preserved by stream ciphers
- Many other ciphers also do not guarantee integrity
- Secure Hash functions are designed to allow integrity validation
  - In particular, second preimage resistance helps here
- MD5 is practically broken
- SHA-1's collision resistance is questionable
- Long term, SHA-3 is best choice