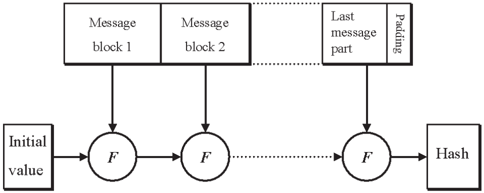
Cryptographic Hash Functions



Cryptographic Hash Functions

- A hash function provides message integrity and authentication
- It is used to compute a short “fingerprint” of the data
- If the data is modified, the fingerprint will not be valid
- $h(\cdot)$ is the hash function, and x is the data:
- The fingerprint is defined as $y = h(x)$
- The fingerprint y is also called “message digest”
- The data can be very long, but the hash is short: 128 to 512 bits

Cryptographic Hash Functions

- Assume that y is stored in a safe place, but x is publicly accessible
- If x is changed to x', we hope that $h(x')$ is different from the original y
- Therefore, instead of comparing x and x' which are very long messages, and also x may no longer be available, we can detect a change in x by re-computing the message digest of x' and checking if $h(x) \neq h(x')$
- Message digest functions are also used in digital signatures: The message and signature pair $[m, s]$ is obtained using a publicly available unkeyed hash function $h(\cdot)$, and the private key encryption function $E_d(\cdot)$ as

$$h = h(m) \quad \rightarrow \quad s = E_d(h) \quad \rightarrow \quad [m, s]$$

Keyed Hash Functions

- Hash functions can also be used with a secret key, which are useful for message and origin authentication
- User A & User B are sharing a secret key K, used as an index in the keyed hash function to compute the hash value of x as $y = h(K, x)$
- User A sends the pair (x, y) to User B which verifies the authenticity $y = h(K, x)$ and becomes confident that x and y are not changed, as long as the hash function is secure

Keyed versus Unkeyed

The assurance of data integrity are different:

- **Unkeyed hash functions:** the message digest value y must be securely stored so that it cannot be changed by an unauthorized party
- **Keyed hash function:** the key K must be kept secret, but x & y can be sent of over on insecure channel

Hash-Based Message Authentication Code

- A hash-based message authentication code (HMAC) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function
- The general form of a keyed hash function (HMAC) with the key K and the message m as input is

$$\mathrm{HMAC}(K, m) = H(K \oplus opad \,||\, H((K \oplus ipad) \,||\, m))$$

  where *opad* and *ipad* are the paddings, and $\oplus$ is the bitwise XOR and $||$ is the concatenation operations
- There are also other forms of message authentication codes, for example, some MACs are based on block ciphers (OMAC and CBC-MAC)

Security of Hash Functions

- Given a hash function $y = h(x)$, the following security requirements are desired in their applications in cryptographic protocols
  - One-way or Preimage Resistance
  - Second Preimage Resistance
  - Collision Resistance

Preimage Resistance

- Given a hash function $h(\cdot)$ and a message digest y, the **problem of preimage computation** is the computation of x such that

$$y = h(x)$$

- A hash function for which a preimage problem cannot be efficiently solved is called a **one-way function** or **preimage resistant function**

$$y,\ h(x) \quad \xrightarrow{\text{hard}} \quad x \quad \text{such that} \quad h(x) = y$$

Second Preimage Resistance

- Given a hash function $h(\cdot)$ and a message x, the **problem of second preimage computation** is the computation of x' such that

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

  but $x' \neq x$

- A hash function for which the second preimage computation cannot be efficiently done is called **second preimage resistant function**

$$x,\ h(x) \quad \xrightarrow{\text{hard}} \quad x' \quad \text{such that} \quad h(x') = h(x)$$

Collision Resistance

- Given a hash function $h(\cdot)$, the **problem of collision computation** is the computation of a pair of x & x' (which are not equal) such that

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

- If such a valid pair is found, then we have detected a collision: (x, y) is valid pair so is (x', y)
- A hash function for which the collision problem cannot be efficiently solved is called **collision resistant function**

$$h(x) \quad \xrightarrow{\text{hard}} \quad \text{any} \quad x \,\&\, x' \quad \text{such that} \quad h(x') = h(x')$$

Generic Attacks to Hash Functions

- Assume that the hash function the k-bit message x to an n-bit hash value y such that k is generally much larger than n
- The generic attacks depend only on the bit size n of the hash value y, and are independent of the specific properties of the hash function
- Also, it is generally assumed that the hash function approximates a random function, otherwise these attacks will be even more successful
- There are essentially two types of generic attacks:
  - Random Second Preimage Attack
  - Birthday Attack

Random Second Preimage Attack

- Consider a message x and a hash value $y = h(x)$
- The attacker selects a random message x' and hopes that the given hash value is hit: $h(x') = y$
- The probability of success is 2^{-n} if the bit size of y is n
- The attack can be carried out off-line and in parallel
- We need to perform up to 2^n trials in order to find a second preimage
- Therefore, the bit size n should be sufficiently large to circumvent this attack: 64, 80, 128, 160, 256, etc

Birthday Attack

- Birthday attack attempts to find a collision
- This attack attempts to find any two x and x' such that $x \neq x'$ but their hash values are equal $y = h(x) = h(x')$
- This problem is related to finding two people with the same birthday (any year; for example, two people born on March 9 albeit different years)
- The probability of success is much higher than $1/365$
- Because the probability looks (at first sight) counter-intuitive, it is sometimes named “birthday paradox”

Birthday Attack Probability

- Assume we have r people in the room
- Since there are only 365 days in a year, if $r \geq 365$, it is guaranteed ($p = 1$) that we have at 2 least people with the same birthday, due to the pigeonhole principle
- Assume $r < 365$: We are interested in finding the probability p of having two people in the room with the same birthday (the probability of a collision)
- We will first calculate the probability p' of r people with all different birthdays (the probability of no collisions)
- We have the probability of collision as $p = 1 - p'$

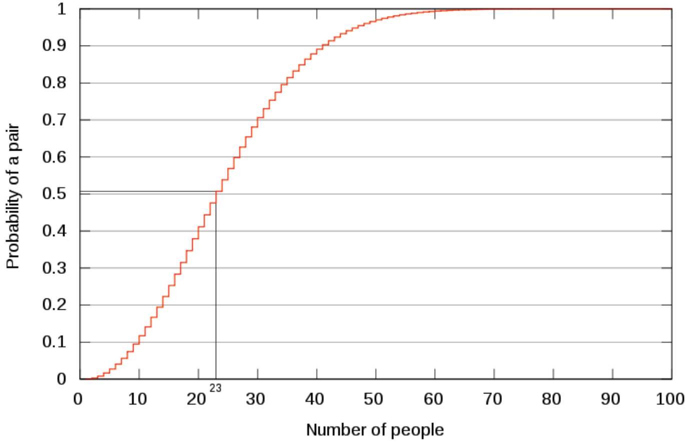
Birthday Attack Probability

- The probability that the birthday of the first person in a specific day of the year is equal to $1/365$ (for example, the probability that the first person was born on April 9)
- The probability that the birthday of the second person is not the same as the first person is $(1 - 1/365)$
- If the birthdays of the first two people are different, which implies that 2 days of the year are occupied, the probability that the birthday of the third person is different from the first two people is $(1 - 2/365)$
- The probability of collisions is found by the product rule

$$p'(t) = \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \left(1 - \frac{r-1}{365}\right)$$

Birthday Attack Probability

- We plot the probability of having two people with the same birthday in a room of t people $p(t) = 1 - p'(t)$ as follows



Birthday Attack Probability

- For example, if there are 10 people in the room, we find $p = 0.117$
- If there are $23 > \sqrt{365} \approx 19.1$ people, the probability of having two people with the same birthday is found to be $p = 0.507 > 50\%$
- If there are 41 people, the probability of having two people with the same birthday is found to be $p = 0.903 > 90\%$
- Intuitively, one expects lower probability: however, for 23 people, there are $(23 \times 22)/2 = 253$ pairs of people, and 365^2 pairs of the days
- Every pair of people gets $365^2/253 \approx 527$ pairs of days, among which there are $527/365 \approx 1.44$ same day pairs; since we have 253 pairs of people, the probability of at least one them having a same day pair is not low

Probability of a Collision

- Given a hash function with u possible hash values, and by generating $r \geq 2$ random input messages, we have a chance of no collision as

$$p'(r,u) = \left(1 - \frac{1}{u}\right) \times \left(1 - \frac{2}{u}\right) \times \cdots \times \left(1 - \frac{r-1}{u}\right)$$

- Using the limit $\lim_{u\to\infty}(1 + \frac{x}{u})^u = e^x$, we can approximate

$$1 + \frac{x}{u} \approx e^{x/u}$$

and thus, write the above product as

$$p'(r,u) = \prod_{i=1}^{r-1}\left(1 + \frac{-i}{u}\right) \approx \prod_{i=1}^{r-1} e^{-\frac{i}{u}} = e^{-\frac{k(k-1)}{2u}}$$

Probability of a Collision

- Therefore, the probability of collision is

$$1 - e^{-\frac{k(k-1)}{2u}}$$

- If we want to have ϵ probability of collision, then we need to generate r messages, which as a function of the number of hashes u is

$$r \approx \sqrt{2u \log(1/(1-\epsilon))}$$

- For example, $\epsilon = 0.5 = 50\%$ probability gives the number of trials as

$$r = 1.1774 \cdot \sqrt{u}$$

- The birthday attack shows that an n-bit hash function can be broken with probability at least 50 % using $1.18 \cdot 2^{n/2}$ trials

Birthday Attack to Hash Functions

- Consider a hash function $h(x)$ of length n bits
- The attacker selects two random messages x and x' and hopes that the hash values are equal: $h(x) = h(x')$
- The probability of success is 50 % if we perform $1.182^{n/2}$ trials for a hash function of size n bits
- The birthday attack can also be carried out off-line and in parallel
- The birthday attack on hash functions is stronger than known plaintext attack on block ciphers, requiring $1.18 \cdot 2^{n/2}$ trials for success instead of 2^n trials
- Hash function size has to be at least twice the size of a block cipher
- The bit size n should be sufficiently large to circumvent this attack: 128, 160, 256, 512, etc

Merkle-Damgård Construction

- Merkle-Damgård model is the most widely accepted model of hash functions, based on iterative application of a compression function
- Compression function has fixed input size and process every block the same way
- The iterated hash function repeatedly uses the compression function in order to produce the final hash value
- The message is broken into equal size blocks, each of which is applied to the compression function (message is also padded)

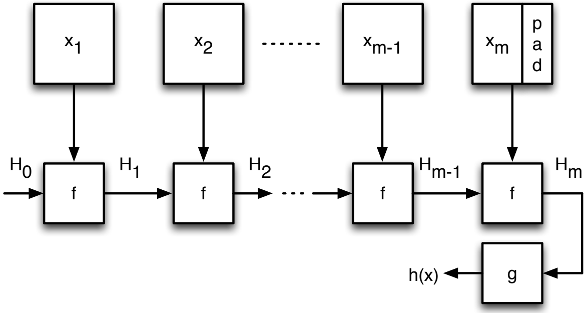
Merkle-Damgård Construction

- Let the block length be k bits, and the message x is broken to equal length m blocks, that is $x = x_1 x_2 \cdots x_m$, for $|x_i| = k$
- The final hash value and temporary hash values are of length n
- The last message block x_m is usually appended with a fixed padding and also the total length of the original message in bits
- The iterated hash construction is based on the compression function $h_i = H(x_i, H_{i-1})$ that takes one block of message of length n and the previous temporary hash value H_{i-1} of length n, and produces the next temporary hash value H_{i-1} of length n
- The initial temporary hash value H_0 is set to be a fixed value called IV, which is the same for all messages
- The final hash value $y = h(x)$ is produced using an output function $g(H_m)$ that takes the final temporary hash value H_m as input

Merkle-Damgård Construction

- The final hash value $y = h(x)$ is produced using an output function $g(H_m)$ that takes the final temporary hash value H_m as input

$$\begin{aligned} H_0 &= IV \\ H_i &= f(x_i, H_{i-1}) \quad \text{for} \quad i = 1, 2, \ldots, m \\ h(x) &= g(H_m) \end{aligned}$$



Some Security Considerations

- The choice of IV is important
- The IV should be defined as part of the description of the hash function
- The choice of padding rule is important
- The padding rule should be unambiguous
- At the end, one should append the length of the message
- Deviations from these rules will make the hash function less secure

Attacks on Iterated Hash Functions

- Meet-in-the-middle attack: A variation of the birthday attack
- Compare intermediate chaining variables instead of the final hash
- More advanced versions of the attack was also developed: p-fold iterated schemes
- Fixed point attack: Try to look for intermediate values such that

$$f(x_i, H_{i-1}) = H_{i-1}$$

- If a fixed point exists: it is possible to insert an arbitrary number of data blocks without modifying the final hash

Hash Functions: MD4 & MD5

- MD5 was proposed by Rivest 1992, as part of the RSA Security PKCS (Public-Key Cryptpgraphy Standards)
- It was a replacement for an earlier version (MD4) proposed in 1990
- MD4 was a 128-bit hash function
- However, several weaknesses were found in MD4: collisions can be found and a preimage attack also succeeds
- MD5 also is a 128-bit message digest function
- Several serious flaws were found in MD5 in 2004, and later on
- MD5 is found to be not collision resistant: It is possible to create two files with the same the MD5 hash
- MD5 is considered broken and unsuitable for further use

Hash Functions: SHA-i Family

- Secure Hash Algorithms (SHA-i) were standardized by NIST
- 4 families of hash functions: SHA-0, SHA-1, SHA-2, and SHA-3
- SHA-1 is the same as SHA-1, which corrected a small error in SHA-0
- SHA-1 is is based on the same principles as MD5
- SHA-2 family has 4 functions in it, with different hash lengths: 224, 256, 384 and 512 bits
- SHA-2 family algorithms were designed to replace SHA-1, as some attacks on SHA-1 exist, but it is not yet broken
- SHA-3 family contains one algorithm called Keccak, which was selected through an international competition organized by the NIST

SHA-1

- Secure Hash Algorithm-1 (SHA-1) was standardized by the NIST in 1991, together with the DSA (Digital Signature Algorithm)
- SHA-1 produces a 160-bit message digest based on principles in the design of the MD4 and MD5 message digest algorithms, but has a more conservative design
- SHA-1 is the most widely used of the existing SHA hash functions, and is used in several applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec
- Cryptographers have produced collision pairs for SHA-0 and have found algorithms that should produce SHA-1 collisions in far fewer than the expected 2^{80} evaluations
- Concerns that these attacks will get more efficient led the NIST to design and introduce the SHA-2 family

Collisions found in SHA-1

- Researchers from the Dutch CWI and Google announced the first SHA-1 collision in Feb 23, 2017
- As a proof of their success they published two different PDFs, that hash to the same value, see: `https://shattered.it`
- The work has been going on for 2 years
- Based on the analytical work of Marc Stevens, the Google team organized the search effort
- The total computational effort was equivalent to 2^{63} SHA-1 compressions, and took 6,500 CPU and 100 GPU years
- The search code was run on a heterogeneous CPU and GPU clusters hosted by Google, spread over 8 physical locations

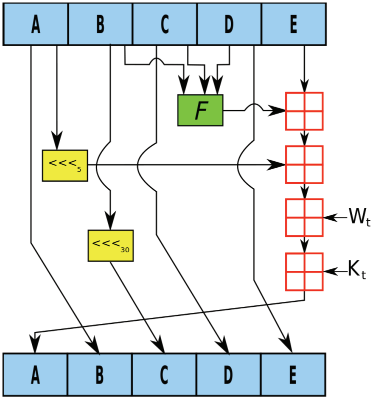
SHA-1 & SHA-2

- SHA-1 is based on a compression function that operates on 5 registers of width 32 bits
- SHA-2 family's first two members, SHA-224 and SHA-256, are also based on 32-bit registers
- SHA-384 and SHA-512 are fundamentally different, requiring 64-bit registers and 1024-bit block size

	Output	Block	Message	Register	Rounds
SHA-1	160	512	2^{64}	32	80
SHA-224	224	512	2^{64}	32	64
SHA-256	256	512	2^{64}	32	64
SHA-384	384	1024	2^{128}	64	80
SHA-512	512	1024	2^{128}	64	80

SHA-1 & SHA-2

- The basic operations of the SHA-1 and SHA-2 family functions are add, and, or, xor, rotate, mod, and shift operations
- Their construction is based on the classical Merkle-Damgård model



SHA-3 Competition

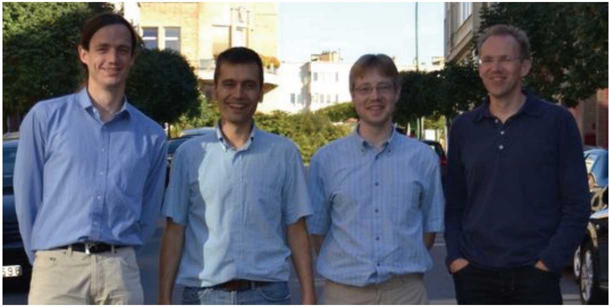
- The competition was held by the NIST, to develop a new hash function called SHA-3 to complement the older SHA-1 and SHA-2
- It started in 2007 and ended in Oct 2, 2012 when the NIST announced that one of the competing algorithms called Keccak is the new SHA-3 algorithm
- The first submissions were due October 31, 2008 and the list of 51 candidates accepted for the first round was published on Dec 9, 2008 by NIST: First Round Candidates
- The first round candidates were presented the First SHA-3 Candidate Conference at Katholieke Universiteit Leuven, Belgium on Feb 25-28, 2009
- It included our (UCSB) submission: Spectral Hash, designed by Koç and a group CCS CS & Math students

SHA-3 Competition

- Only 14 of them survived to the Second Round
- Our Spectral Hash was broken by one of Rivest's students :(
- NIST hosted a Second SHA-3 Candidate Conference at UCSB on Aug 23-24, 2010 to discuss the security and performance analyses of the second-round candidates
- NIST announced five SHA-3 finalists: BLAKE, Grøstl, JH, Keccak, and Skein on Dec 9, 2010
- NIST hosted a Third SHA-3 Candidate Conference in Washington, D.C. on Mar 22-23, 2012 to discuss the security and performance analyses of the finalists
- The winner was announced to be Keccak on Oct 2, 2012

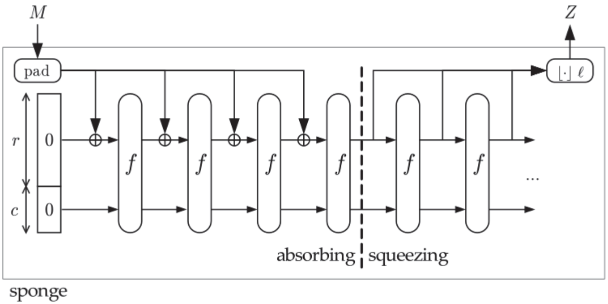
SHA-3 Finalist: Keccak

- Keccak is designed by four European researchers: Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche
- Joan Daemen is the designer of Rijndael/AES (with Vincent Rijmen)



SHA-3 Finalist: Keccak

- Keccak uses the sponge construction: message blocks are XORed into the initial bits of the state, which is then invertibly permuted
- In the version used in SHA-3, the state consists of a 5×5 array of 64-bit words, 1600 bits total


M
Z
pad
$\lfloor\cdot\rfloor_\ell$
r
c
0
0
f
...
absorbing
squeezing
sponge