Keccak
The NIST Standard SHA3 Hash Functions

Hash Functions

- condenses arbitrary message to fixed size

  `h = H(M)`

- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

Requirements for Hash Functions

1. can be applied to any sized message `M`
2. produces fixed-length output `h`
3. is easy to compute `h=H(M)` for any message `M`
4. given `h` is infeasible to find `x` s.t. `H(x)=h`
    - one-way property
5. given `x` is infeasible to find `y` s.t. `H(y)=H(x)`
    - weak collision resistance
6. is infeasible to find any `x,y` s.t. `H(y)=H(x)`
    - strong collision resistance

Choosing the length of Hash outputs

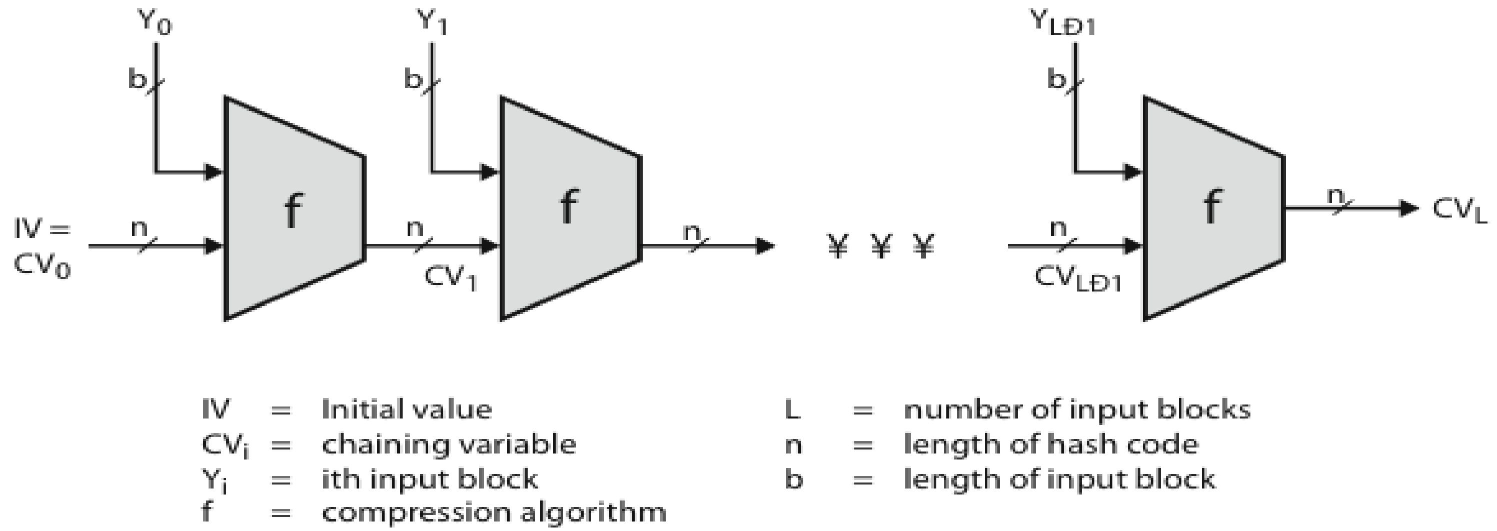
- For birthday attack, the length of hash outputs in general should double the key length of block ciphers
  - SHA-224 matches the 112-bit strength of triple-DES (encryption 3 times using DES)
  - SHA-256, SHA-384, SHA-512 match the new key lengths (128,192,256) in AES

Well Known Hash Functions

- MD5
  - output 128 bits
  - collision resistance completely broken by researchers in China in 2004
- SHA1
  - output 160 bits
  - no collision found yet, but method exist to find collisions in less than 2^80
  - 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
  - No real security concerns yet

Merkle-Damgard Construction forHash Functions

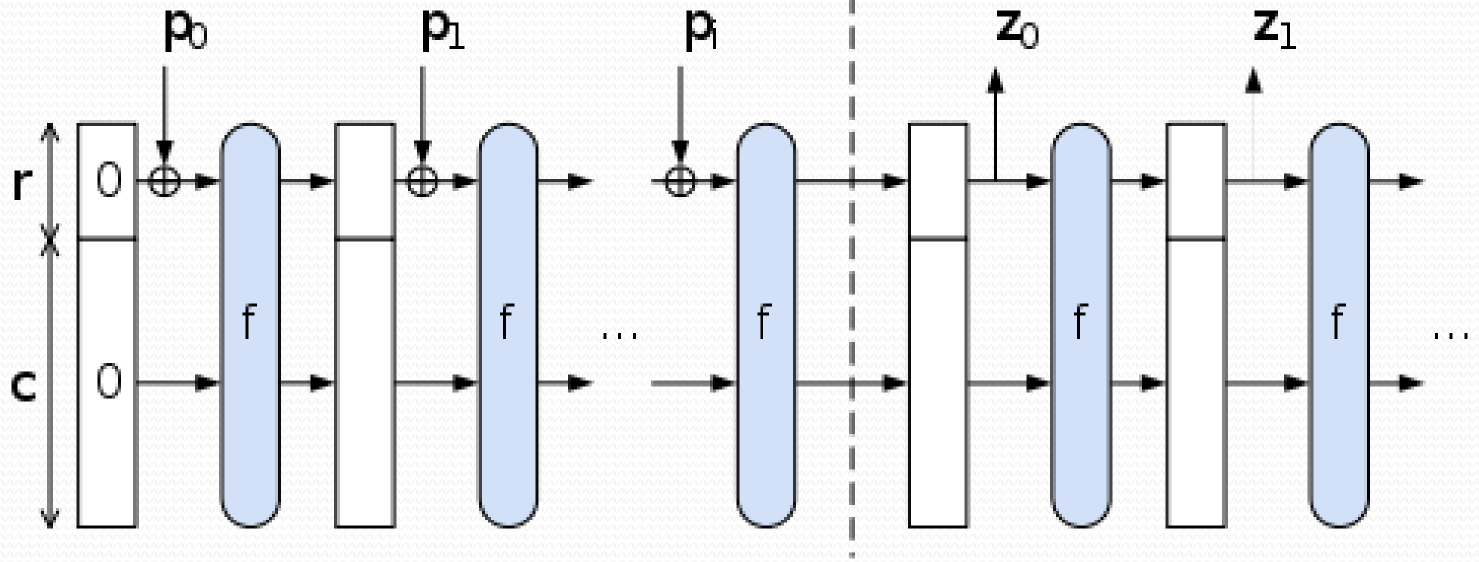
- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value



NIST SHA-3 Competition

- NIST is having an ongoing competition for SHA-3, the next generation of standard hash algorithms
- 2007: Request for submissions of new hash functions
- 2008: Submissions deadline. Received 64 entries. Announced first-round selections of 51 candidates.
- 2009: After First SHA-3 candidate conference in Feb, announced 14 Second Round Candidates in July.
- 2010: After one year public review of the algorithms, hold second SHA-3 candidate conference in Aug. Announced 5 Third-round candidates in Dec.
- 2011: Public comment for final round
- 2012: October 2, NIST selected SHA3
  - Keccak (pronounced "catch-ack") created by Guido Bertoni, Joan Daemen and Gilles Van Assche, Michaël Peeters

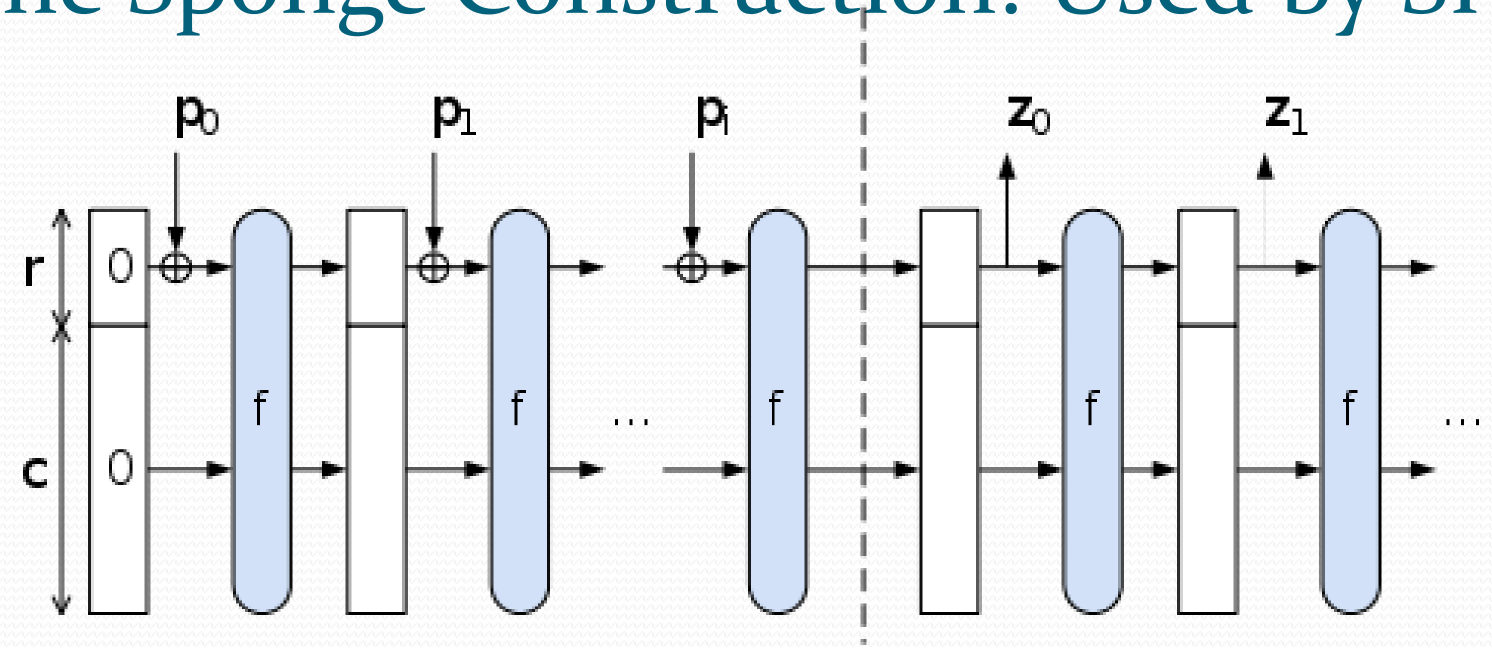
The Sponge Construction: Used by SHA-3



Absorbing Phase Squeezing Phase

- Absorbing Phase
  - the first message block is XORed (using the bitwise XOR) with the outer part of the initial 0-state and then the permutation f is applied to the resulting state. Then the next r-bit message block is XORed with the outer part of the state and f is applied again. This process continues interleaving the XOR operations with applications of the permutation f until all the input is consumed

The Sponge Construction: Used by SHA-3



Absorbing Phase Squeezing Phase

- Squeezing Phase
  - takes the first r bits from the state and then applying the permutation f. If the squeezed r bits are less than the requested n bits, the process continues by applying f to the state and then extracting the first r bits of the resulting state. The process is continued interleaving the extraction of the first r bits of the state with the application of f until the total number n, and the final output is obtained by truncating the squeezed bits to the first n bits.

Keccak

- Instantiation of a sponge function
- the permutation Keccak-f
  - 7 permutations: b → {25, 50, 100, 200, 400, 800, 1600}
  - Security-speed trade-offs using the same permutation, e.g., SHA-3 instance: r = 1088 and c = 512
    - permutation width: 1600
    - security strength 256: post-quantum sufficient
- Lightweight instance: r = 40 and c = 160
  - permutation width: 200
  - security strength 80: same as SHA-1

Keccak State

The state: an array of $5 \times 5 \times 2^\ell$ bits

- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

x Nonlinear Mapping in Keccak-f

- "Flip bit if neighbors exhibit 01 pattern"
- Operates independently and in parallel on 5-bit rows
- Algebraic degree 2, inverse has degree 3
- LC/DC propagation properties easy to describe and analyze

θ' first attempt at Mixing bits

- Compute parity cx,z of each column
- Add to each cell parity of neighboring columns:

  bx,y,z = ax,y,z + cx–1,z + cx+1,z

- Diffusion is performed

ρ for inter-slice dispersion

- We need diffusion between the slices … ι:
- cyclic shifts of lanes with offsets $i(i+1)/2 \bmod 2^\ell$
- Offsets cycle through all values below 2^ℓ

χ the nonlinear mapping in Keccak-f

- "Flip bit if neighbors exhibit 01 pattern"
- Operates independently and in parallel on 5-bit rows
- Algebraic degree 2, inverse has degree 3
- LC/DC propagation properties easy to describe and analyze

l to break symmetry in Keccak-f

- XOR of round-dependent constant to lane in origin
- Without L, the round mapping would be symmetric invariant to translation in the z-direction
- Without L, all rounds would be the same susceptibility to slide attacks defective cycle structure
- Without L, we get simple fixed points (000 and 111)

References

- https://csrc.nist.gov/csrc/media/projects/hash-functions/documents/keccak-slides-at-nist.pdf
- https://summerschool-croatia.cs.ru.nl/2015/SHA3.pdf

- https://keccak.team/files/Keccak-reference-3.0.pdf
- https://keccak.team/third_party.html