# Introduction to cryptography

## 3. Hashing

Gilles Van Assche
Olivier Markowitch

INFO-F-405
Université Libre de Bruxelles
2020-2021

# Cryptographic hash functions

$$h : \{0,1\}^* \to \{0,1\}^n$$

Input message →  → Digest

- Applications
  - *Signatures*: $\text{sign}_{\text{RSA}}(h(M))$ instead of $\text{sign}_{\text{RSA}}(M)$
  - *Key derivation*: master key $K$ to derived keys $(K_i = h(K \| i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K \| M)$
  - ...

# Cryptographic hash functions

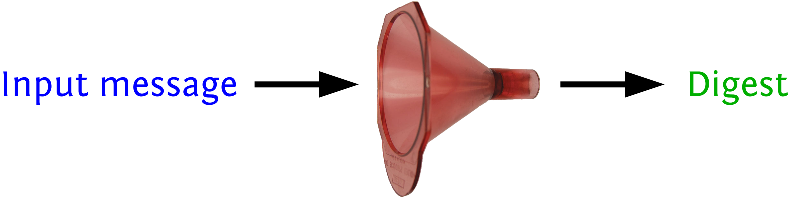
$$h : \{0,1\}^* \to \{0,1\}^n$$

Input message →  → Digest

- Applications
  - *Signatures*: $\mathrm{sign}_{\mathrm{RSA}}(h(M))$ instead of $\mathrm{sign}_{\mathrm{RSA}}(M)$
  - *Key derivation*: master key $K$ to derived keys $(K_i = h(K\|i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K\|M)$
  - ...

# Cryptographic hash functions

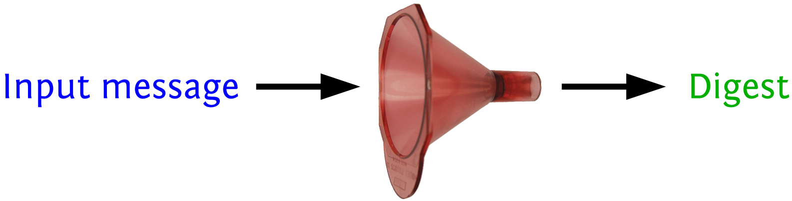
$$h \; : \; \{0,1\}^* \to \{0,1\}^n$$




- Applications
  - *Signatures*: $\mathrm{sign}_{\mathrm{RSA}}(h(M))$ instead of $\mathrm{sign}_{\mathrm{RSA}}(M)$
  - *Key derivation*: master key $K$ to derived keys $(K_i = h(K\|i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K\|M)$
  - ...

# Cryptographic hash functions

$$h \ : \ \{0,1\}^* \to \{0,1\}^n$$




- Applications
  - *Signatures*: $\mathrm{sign}_{\mathrm{RSA}}(h(M))$ instead of $\mathrm{sign}_{\mathrm{RSA}}(M)$
  - *Key derivation*: master key $K$ to derived keys $(K_i = h(K \| i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K \| M)$
  - ...

# Generalized: extendable output function (XOF)

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

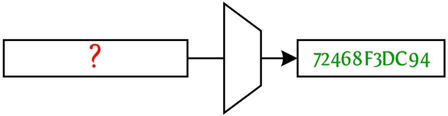
"XOF: a function in which the output can be extended to any length."

[Ray Perlner, SHA 3 workshop 2014]

- Applications
  - *Signatures*: full-domain hashing, mask generating function
  - *Key derivation*: as many/long derived keys as needed
  - *Stream cipher*: $C = P \oplus h(K \| \text{nonce})$

# Preimage resistance

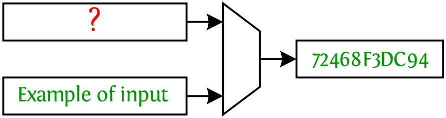
- Given $y \in \mathbf{Z}_2^n$, find $x \in \mathbf{Z}_2^*$ such that $h(x) = y$




- If $h$ is a random function, about $2^n$ attempts are needed
- **Example**: given derived key $K_1 = h(K \| 1)$, find master key $K$

# Second preimage resistance

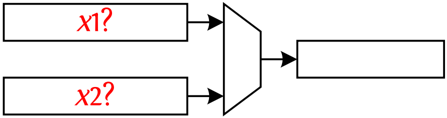
- Given $x \in \mathbf{Z}_2^*$, find $x' \neq x$ such that $h(x') = h(x)$




- If $h$ is a random function, about $2^n$ attempts are needed
- **Example**: signature forging
  - Given $M$ and $\mathrm{sign}(h(M))$, find $M' \neq M$ with equal signature

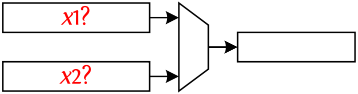
# Collision resistance

- Find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$




- If $h$ is a random function, about $2^{n/2}$ attempts are needed
  - Birthday paradox: among 23 people, probably two have same birthday
  - Scales as $\sqrt{|\text{range}|} = 2^{n/2}$

# Collision resistance (continued)




- **Example**: “secretary” signature forging
  - Set of good messages $\{M_i^{\text{good}}\}$
  - Set of bad messages $\{M_i^{\text{bad}}\}$
  - Find $h(M_i^{\text{good}}) = h(M_j^{\text{bad}})$
  - Boss signs $M_i^{\text{good}}$, but valid also for $M_j^{\text{bad}}$

[Yuval, 1979]

# Other requirements

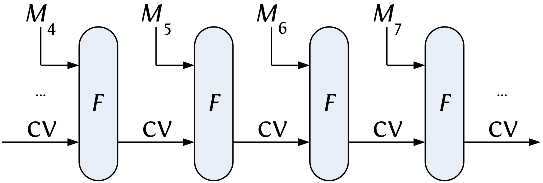
- Security claims by listing desired properties
  - Collision resistant
  - (Second-) preimage resistant
    - Multi-target preimage resistance
    - Chosen-target forced-prefix preimage resistance
  - Correlation-free
  - Resistant against length-extension attacks
  - ...
- But ever-growing list of desired properties
- A good hash function should behave like a **random mapping**...

# Security requirements summarized

- Hash or XOF $h$ with $n$-bit output
- Modern security requirements
  - $h$ behaves like a random mapping
  - ... up to security strength $s$
- Classical security requirements, derived from it

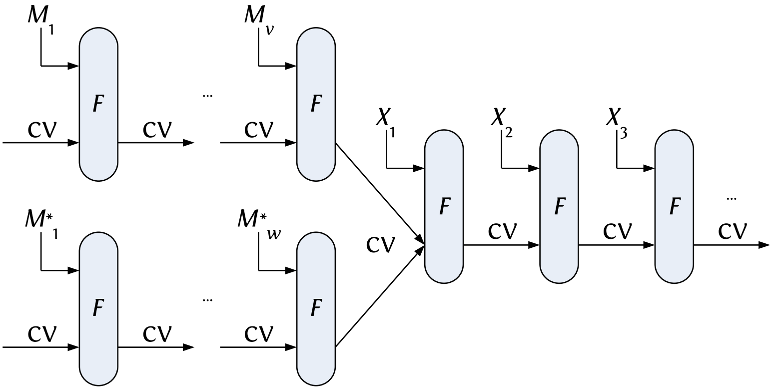
| | |
|---|---|
| Preimage resistance | $2^{\min(n,s)}$ |
| Second-preimage resistance | $2^{\min(n,s)}$ |
| Collision resistance | $2^{\min(n/2,s)}$ |

# Iterated functions




- All practical hash functions are iterated
  - Message $M$ cut into blocks $M_1, \ldots, M_l$
  - $q$-bit chaining value
- Output is function of final chaining value

# Internal collisions!




- Different inputs $M$ and $M^*$ giving the same chaining value
- Messages $M \| X$ and $M^* \| X$ always collide for any string $X$

Does not occur in a random mapping!

# Examples of hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - SHA-224, SHA-256, SHA-384 and SHA-512
- SHA-3: based on KECCAK
  - Designed by Bertoni, Daemen, Peeters and VA in 2008
  - Standardized by NIST in 2015
  - SHA3-{224, 256, 384, 512}, SHAKE{128, 256}, ParallelHash{128, 256}, ...
- Other SHA-3 finalists
  - Blake (Aumasson et al.), Grøstl (Gauravaram et al.), JH (Wu), Skein (Ferguson et al.)

# Examples of hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - SHA-224, SHA-256, SHA-384 and SHA-512
- SHA-3: based on KECCAK
  - Designed by Bertoni, Daemen, Peeters and VA in 2008
  - Standardized by NIST in 2015
  - SHA3-{224, 256, 384, 512}, SHAKE{128, 256}, ParallelHash{128, 256}, ...
- Other SHA-3 finalists
  - Blake (Aumasson et al.), Grøstl (Gauravaram et al.), JH (Wu), Skein (Ferguson et al.)

# Examples of hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - SHA-224, SHA-256, SHA-384 and SHA-512
- SHA-3: based on KECCAK
  - Designed by Bertoni, Daemen, Peeters and VA in 2008
  - Standardized by NIST in 2015
  - SHA3-{224, 256, 384, 512}, SHAKE{128, 256}, ParallelHash{128, 256}, ...
- Other SHA-3 finalists
  - Blake (Aumasson et al.), Grøstl (Gauravaram et al.), JH (Wu), Skein (Ferguson et al.)

# Examples of hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - SHA-224, SHA-256, SHA-384 and SHA-512
- SHA-3: based on KECCAK
  - Designed by Bertoni, Daemen, Peeters and VA in 2008
  - Standardized by NIST in 2015
  - SHA3-{224, 256, 384, 512}, SHAKE{128, 256}, ParallelHash{128, 256}, ...
- Other SHA-3 finalists
  - Blake (Aumasson et al.), Grøstl (Gauravaram et al.), JH (Wu), Skein (Ferguson et al.)

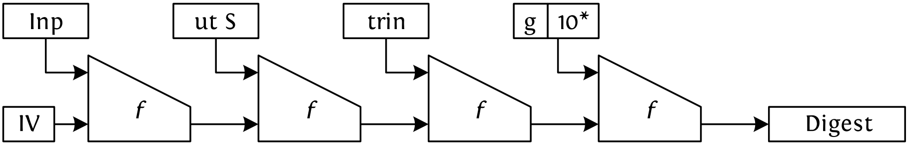
# Examples of hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - SHA-224, SHA-256, SHA-384 and SHA-512
- SHA-3: based on KECCAK
  - Designed by Bertoni, Daemen, Peeters and VA in 2008
  - Standardized by NIST in 2015
  - SHA3-{224, 256, 384, 512}, SHAKE{128, 256}, ParallelHash{128, 256}, ...
- Other SHA-3 finalists
  - Blake (Aumasson et al.), Grøstl (Gauravaram et al.), JH (Wu), Skein (Ferguson et al.)

# Attacks on MD5, SHA-0 and SHA-1

- 2004: SHA-0 broken (Joux et al.)
- 2004: MD5 broken (Wang et al.)
- 2005: practical attack on MD5 (Lenstra et al., and Klima)
- 2005: SHA-1 theoretically broken (Wang et al.)
- 2006: SHA-1 broken further (De Cannière and Rechberger)
- 2016: freestart collision on SHA-1 (Stevens, Karpman and Peyrin)
- 2017: actual collision on SHA-1 (Stevens, Bursztein, Karpman, Albertini and Markov)

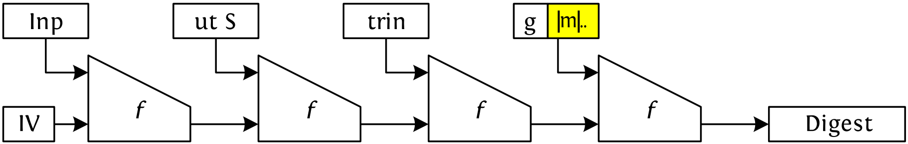
# Merkle-Damgård




- Uses a compression function from $n + m$ bits to $n$ bits
- Instances: MD5, SHA-1, SHA-2 ...
- Merkle-Damgård strengthening

[Merkle, CRYPTO'89], [Damgård, CRYPTO'89]

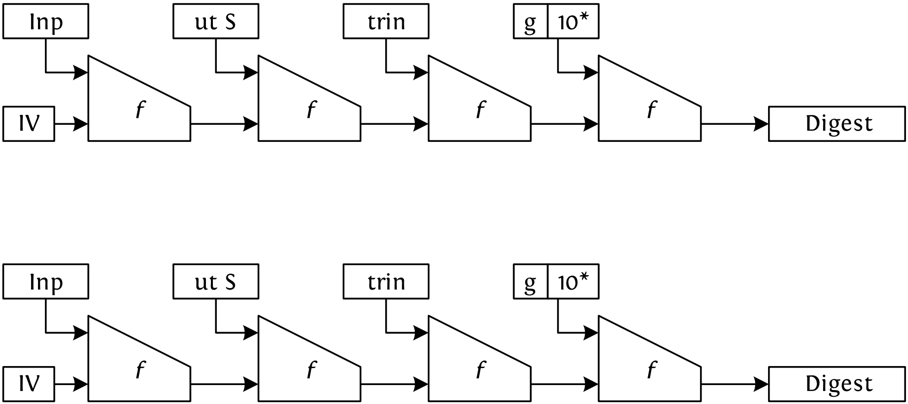
# Merkle-Damgård



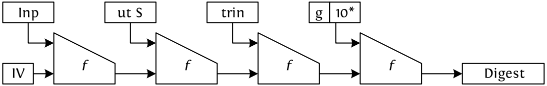

- Uses a compression function from $n + m$ bits to $n$ bits
- Instances: MD5, SHA-1, SHA-2 ...
- Merkle-Damgård strengthening

[Merkle, CRYPTO'89], [Damgård, CRYPTO'89]

# Merkle-Damgård: preserving collision resistance



# Merkle-Damgård: length extension




Recurrence (modulo the padding):

- $h(M_1) = f(\mathrm{IV}, M_1) = \mathrm{CV}_1$
- $h(M_1 \| \dots \| M_i) = f(\mathrm{CV}_{i-1}, M_i) = \mathrm{CV}_i$

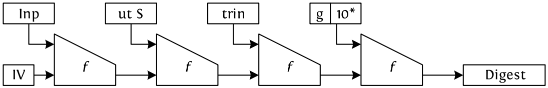
Forgery on naïve message authentication code (MAC):

- $\mathrm{MAC}(M) = h(\mathrm{Key} \| M) = \mathrm{CV}$
- $\mathrm{MAC}(M \| \mathrm{suffix}) = f(\mathrm{CV} \| \mathrm{suffix})$

Solution: **HMAC**

$$\mathrm{HMAC}(M) = h(\mathrm{Key}_{\mathrm{out}} \| h(\mathrm{Key}_{\mathrm{in}} \| M))$$

# Merkle-Damgård: length extension




Recurrence (modulo the padding):

- $h(M_1) = f(\mathrm{IV}, M_1) = \mathrm{CV}_1$
- $h(M_1 \| \dots \| M_i) = f(\mathrm{CV}_{i-1}, M_i) = \mathrm{CV}_i$

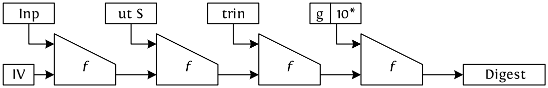
Forgery on naïve message authentication code (MAC):

- $\mathrm{MAC}(M) = h(\mathrm{Key} \| M) = \mathrm{CV}$
- $\mathrm{MAC}(M \| \mathrm{suffix}) = f(\mathrm{CV} \| \mathrm{suffix})$

Solution: **HMAC**

$$\mathrm{HMAC}(M) = h(\mathrm{Key}_{\mathrm{out}} \| h(\mathrm{Key}_{\mathrm{in}} \| M))$$

# Merkle-Damgård: length extension




Recurrence (modulo the padding):

- $h(M_1) = f(\mathrm{IV}, M_1) = \mathrm{CV}_1$
- $h(M_1 \| \dots \| M_i) = f(\mathrm{CV}_{i-1}, M_i) = \mathrm{CV}_i$

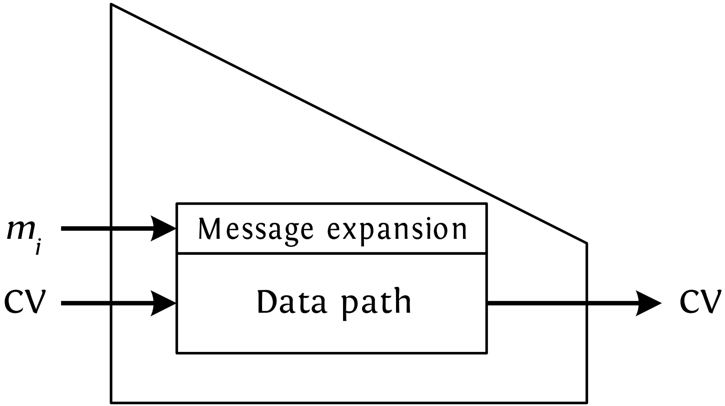
Forgery on naïve message authentication code (MAC):

- $\mathrm{MAC}(M) = h(\mathrm{Key} \| M) = \mathrm{CV}$
- $\mathrm{MAC}(M \| \mathrm{suffix}) = f(\mathrm{CV} \| \mathrm{suffix})$

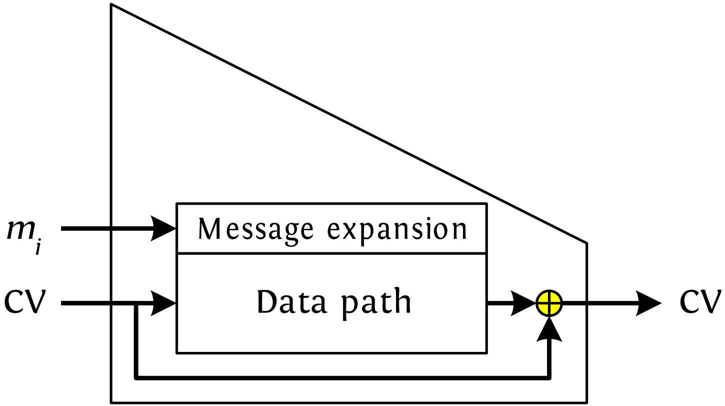
Solution: **HMAC**

$$\mathrm{HMAC}(M) = h(\mathrm{Key}_{\mathrm{out}} \| h(\mathrm{Key}_{\mathrm{in}} \| M))$$

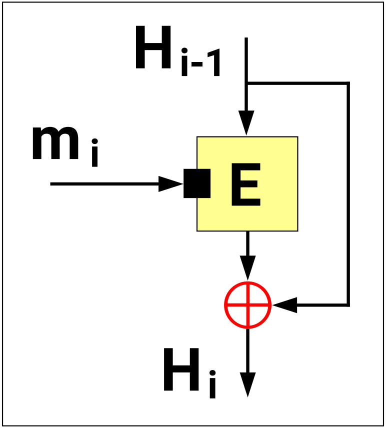
# Davies-Meyer



# Davies-Meyer



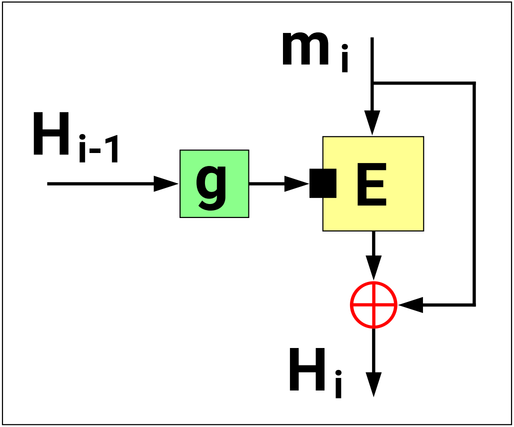
# Other constructions using block ciphers




Davies-Meyer

[Matyas et al., IBM Tech. D. B., 1985], [Quisquater et al., Eurocrypt'89]

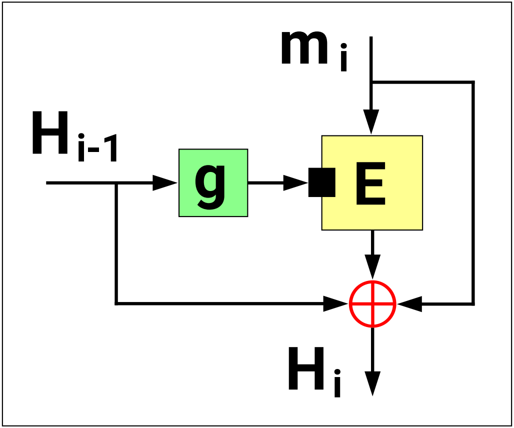
# Other constructions using block ciphers




Matyas-Meyer-Oseas

[Matyas et al., IBM Tech. D. B., 1985]

# Other constructions using block ciphers




Miyaguchi-Preneel

[Miyaguchi et al., NTT Rev., 1990], [Preneel, PhD th., 1993]

# Inside SHA-1

- Uses Davies-Meyer with
  - data path $n = 160 = 5 \times 32$
  - message expansion $m = 512 = 16 \times 32$
- State initialized with $(A, B, C, D, E) =$

$$(\texttt{67452301}, \texttt{EFCDAB89}, \texttt{98BADCFE}, \texttt{10325476}, \texttt{C3D2E1F0})$$

- Message block $(w_0, \dots, w_{15})$ expanded as

$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lll 1 \quad (16 \le t \le 79)$$

- Data path with 80 steps...

# Inside SHA-1

- Uses Davies-Meyer with
  - data path $n = 160 = 5 \times 32$
  - message expansion $m = 512 = 16 \times 32$
- State initialized with $(A, B, C, D, E) =$

$$(\texttt{67452301}, \texttt{EFCDAB89}, \texttt{98BADCFE}, \texttt{10325476}, \texttt{C3D2E1F0})$$

- Message block $(w_0, \ldots, w_{15})$ expanded as

$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lll 1 \quad (16 \leq t \leq 79)$$

- Data path with 80 steps...

# Inside SHA-1

- Uses Davies-Meyer with
  - data path $n = 160 = 5 \times 32$
  - message expansion $m = 512 = 16 \times 32$
- State initialized with $(A, B, C, D, E) =$

$$(\texttt{67452301}, \texttt{EFCDAB89}, \texttt{98BADCFE}, \texttt{10325476}, \texttt{C3D2E1F0})$$

- Message block $(w_0, \ldots, w_{15})$ expanded as

$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lll 1 \quad (16 \leq t \leq 79)$$

- Data path with 80 steps...

# Inside SHA-1

- Uses Davies-Meyer with
  - data path $n = 160 = 5 \times 32$
  - message expansion $m = 512 = 16 \times 32$
- State initialized with $(A, B, C, D, E) =$

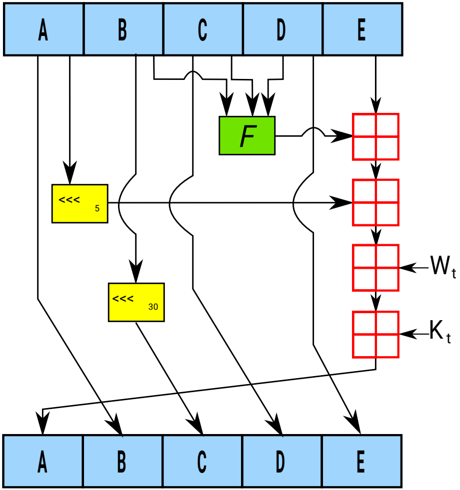
$$(\texttt{67452301}, \texttt{EFCDAB89}, \texttt{98BADCFE}, \texttt{10325476}, \texttt{C3D2E1F0})$$

- Message block $(w_0, \ldots, w_{15})$ expanded as

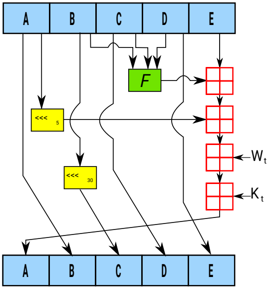
$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lll 1 \quad (16 \leq t \leq 79)$$

- Data path with 80 steps...

# Inside SHA-1: data path



# Inside SHA-1: data path details




| | | |
|---|---|---|
| $0 \leq t \leq 19$ | $f(B, C, D) = (B \odot C) \oplus (\bar{B} \odot D)$ | $K_t = 5A827999$ |
| $20 \leq t \leq 39$ | $f(B, C, D) = B \oplus C \oplus D$ | $K_t = 6ED9EBA1$ |
| $40 \leq t \leq 59$ | $f(B, C, D) = (B \odot C) \oplus (B \odot D) \oplus (C \odot D)$ | $K_t = 8F1BBCDC$ |
| $60 \leq t \leq 79$ | $f(B, C, D) = B \oplus C \oplus D$ | $K_t = CA62C1D6$ |

# Collision in SHA-1

- February 23, 2017: first collision on SHA-1 published
- Estimated complexity: $2^{63} \ll 2^{80}$

[Stevens, Bursztein, Karpman, Albertini and Markov]

# Collision in SHA-1

$$\text{SHA-1}(P \| M_1^{(1)} \| M_2^{(1)} \| S) = \text{SHA-1}(P \| M_1^{(2)} \| M_2^{(2)} \| S)$$

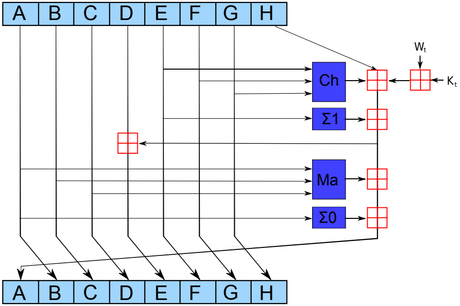
| | |
|---|---|
| $CV_0$ | 4e a9 62 69 7c 87 6e 26 74 d1 07 f0 fe c6 79 84 14 f5 bf 45 |
| $M_1^{(1)}$ | 7f 46 dc 93 a6 b6 7e 01 3b 02 9a aa 1d b2 56 0b |
| | 45 ca 67 d6 88 c7 f8 4b 8c 4c 79 1f e0 2b 3d f6 |
| | 14 f8 6d b1 69 09 01 c5 6b 45 c1 53 0a fe df b7 |
| | 60 38 e9 72 72 2f e7 ad 72 8f 0e 49 04 e0 46 c2 |
| $CV_1^{(1)}$ | 8d 64 d6 17 ff ed 53 52 eb c8 59 15 5e c7 eb 34 f3 8a 5a 7b |
| $M_2^{(1)}$ | 30 57 0f e9 d4 13 98 ab e1 2e f5 bc 94 2b e3 35 |
| | 42 a4 80 2d 98 b5 d7 0f 2a 33 2e c3 7f ac 35 14 |
| | e7 4d dc 0f 2c c1 a8 74 cd 0c 78 30 5a 21 56 64 |
| | 61 30 97 89 60 6b d0 bf 3f 98 cd a8 04 46 29 a1 |
| $CV_2$ | 1e ac b2 5e d5 97 0d 10 f1 73 69 63 57 71 bc 3a 17 b4 8a c5 |

| | |
|---|---|
| $CV_0$ | 4e a9 62 69 7c 87 6e 26 74 d1 07 f0 fe c6 79 84 14 f5 bf 45 |
| $M_1^{(2)}$ | 73 46 dc 91 66 b6 7e 11 8f 02 9a b6 21 b2 56 0f |
| | f9 ca 67 cc a8 c7 f8 5b a8 4c 79 03 0c 2b 3d e2 |
| | 18 f8 6d b3 a9 09 01 d5 df 45 c1 4f 26 fe df b3 |
| | dc 38 e9 6a c2 2f e7 bd 72 8f 0e 45 bc e0 46 d2 |
| $CV_1^{(2)}$ | 8d 64 c8 21 ff ed 52 e2 eb c8 59 15 5e c7 eb 36 73 8a 5a 7b |
| $M_2^{(2)}$ | 3c 57 0f eb 14 13 98 bb 55 2e f5 a0 a8 2b e3 31 |
| | fe a4 80 37 b8 b5 d7 1f 0e 33 2e df 93 ac 35 00 |
| | eb 4d dc 0d ec c1 a8 64 79 0c 78 2c 76 21 56 60 |
| | dd 30 97 91 d0 6b d0 af 3f 98 cd a4 bc 46 29 b1 |
| $CV_2$ | 1e ac b2 5e d5 97 0d 10 f1 73 69 63 57 71 bc 3a 17 b4 8a c5 |

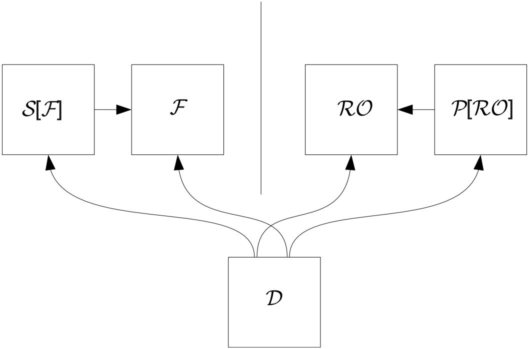
# From SHA-1 to SHA-2

Changes from SHA-1 to SHA-2:

- Two compression functions
  - SHA-{224, 256}: $n = 256 = 8 \times 32$ and $m = 512 = 16 \times 32$
  - SHA-{384, 512}: $n = 512 = 8 \times 64$ and $m = 1024 = 16 \times 64$
- Non-linear message expansion
- Stronger data path mixing



# Generic security: indifferentiability [Maurer et al. (2004)]




Applied to hash functions in [Coron et al. (2005)]

- distinguishing mode-of-use from ideal function ($\mathcal{RO}$)
- covers adversary with access to primitive $\mathcal{F}$ at left
- additional interface, covered by a *simulator* at right

# Consequences of indifferentiability

**Theorem 2.** *Let $\mathcal{H}$ be a hash function, built on underlying primitive $\pi$, and RO be a random oracle, where $\mathcal{H}$ and RO have the same domain and range space. Denote by $\boldsymbol{Adv}_{\mathcal{H}}^{\mathrm{pro}}(q)$ the advantage of distinguishing $(\mathcal{H}, \pi)$ from $(RO, S)$, for some simulator $S$, maximized over all distinguishers $\mathcal{D}$ making at most $q$ queries. Let* atk *be a security property of $\mathcal{H}$. Denote by $\boldsymbol{Adv}_{\mathcal{H}}^{\mathrm{atk}}(q)$ the advantage of breaking $\mathcal{H}$ under* atk*, maximized over all adversaries $\mathcal{A}$ making at most $q$ queries. Then:*

$$\boldsymbol{Adv}_{\mathcal{H}}^{\mathrm{atk}}(q) \leq \boldsymbol{Pr}_{RO}^{\mathrm{atk}}(q) + \boldsymbol{Adv}_{\mathcal{H}}^{\mathrm{pro}}(q), \tag{1}$$

*where $\boldsymbol{Pr}_{RO}^{\mathrm{atk}}(q)$ denotes the success probability of a generic attack against $\mathcal{H}$ under* atk*, after at most $q$ queries.*

[Andreeva, Mennink, Preneel, ISC 2010]

# Limitations of indifferentiability

- Only about the mode
  - No security proof with a concrete primitive
- Only about single-stage games [Ristenpart et al., Eurocrypt 2011]
  - Example: hash-based storage auditing

$$Z = h(\mathsf{File} \| C)$$

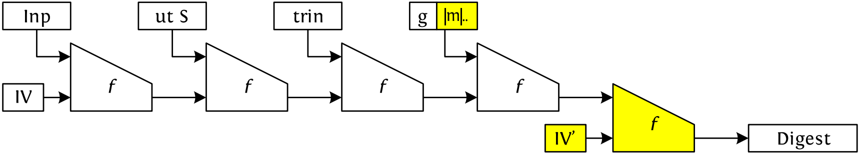
# Limitations of indifferentiability

- Only about the mode
  - No security proof with a concrete primitive
- Only about single-stage games [Ristenpart et al., Eurocrypt 2011]
  - Example: hash-based storage auditing

$$Z = h(\mathsf{File} \| C)$$

# Making Merkle-Damgård indifferentiable

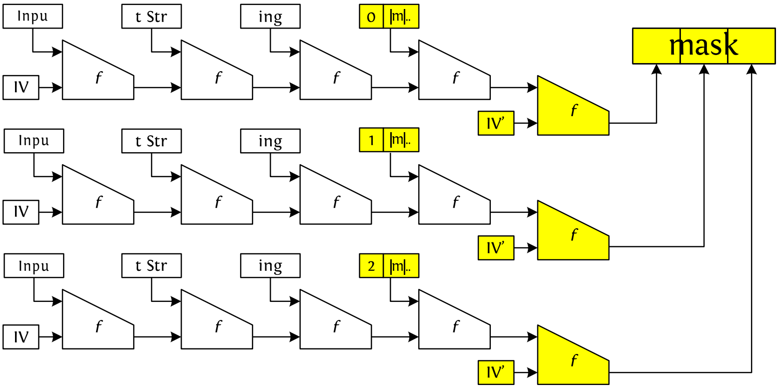
## Enveloped Merkle-Damgård



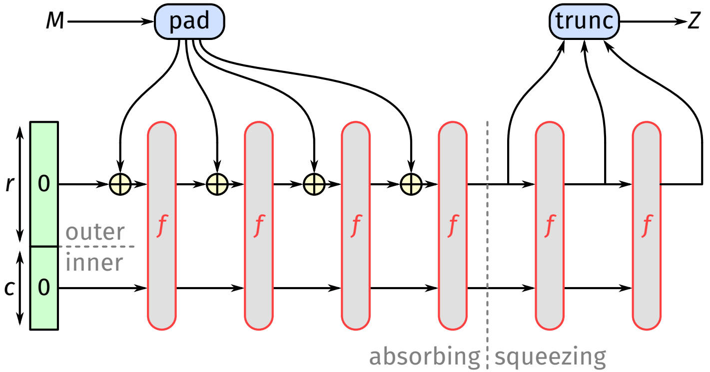

[Bellare and Ristenpart, Asiacrypt 2006]

# Making Merkle-Damgård suitable for XOFs

Mask generating function construction “MGF1”



# The sponge construction




- Calls a $b$-bit permutation $f$, with $b = r + c$
  - $r$ bits of *rate*
  - $c$ bits of *capacity* (security parameter)
- Natively implements a XOF

# Generic security of the sponge construction

## Theorem (Bound on the $\mathcal{RO}$-differentiating advantage of sponge)

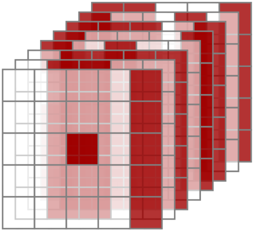
$$\text{Adv} \leq \frac{t^2}{2^{c+1}}$$

Adv: *differentiating advantage of random sponge from random oracle*
$t$: *time complexity (# calls to $f$)*    $c$: *capacity*    [Eurocrypt 2008]

| | |
|---|---|
| Preimage resistance | $2^{\min(n,c/2)}$ |
| Second-preimage resistance | $2^{\min(n,c/2)}$ |
| Collision resistance | $2^{\min(n/2,c/2)}$ |
| Any other attack | $2^{\min(\mathcal{RO},c/2)}$ (*) |

(*) This means the minimum between $2^{c/2}$ and the complexity of the attack on a random oracle.

# KECCAK-$f$



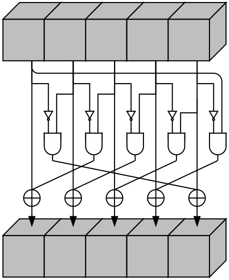
- The seven permutation army:
  - 25, 50, 100, 200, 400, 800, 1600 bits
  - toy, lightweight, fastest
  - standardized in [FIPS 202]
- Repetition of a simple round function
  - that operates on a 3D state
  - $(5 \times 5)$ lanes
  - up to 64-bit each

# KECCAK-$f$ in pseudo-code

```
KECCAK-F[b](A) {
  forall i in 0…nr-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  θ step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4], forall x in 0…4
  D[x] = C[x-1] xor rot(C[x+1],1),                           forall x in 0…4
  A[x,y] = A[x,y] xor D[x],                                  forall (x,y) in (0…4,0…4)

  ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]),                        forall (x,y) in (0…4,0…4)

  χ step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]),         forall (x,y) in (0…4,0…4)

  ι step
  A[0,0] = A[0,0] xor RC

  return A
}
```

https://keccak.team/keccak_specs_summary.html

# $\chi$, the nonlinear mapping in KECCAK-$f$



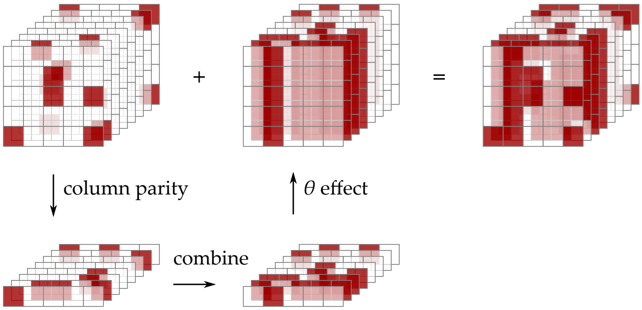
- "Flip bit if neighbors exhibit 01 pattern"
- Operates independently and in parallel on 5-bit rows
- Cheap: small number of operations per bit
- Algebraic degree 2, inverse has degree 3

# $\theta$, mixing bits

- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z-1}$$

- Cheap: two XORs per bit

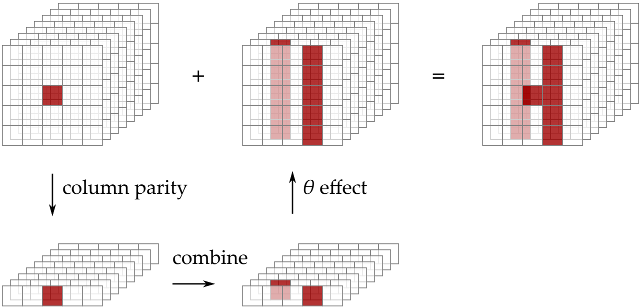


# $\theta$, mixing bits

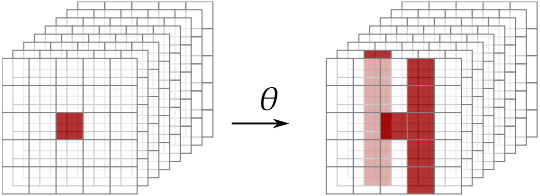
- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z-1}$$

- Cheap: two XORs per bit

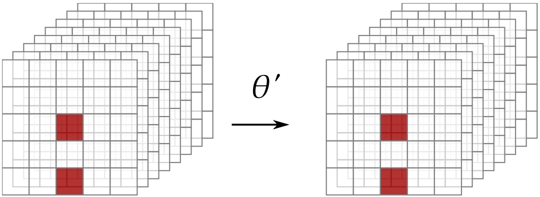


# Diffusion of $\theta$



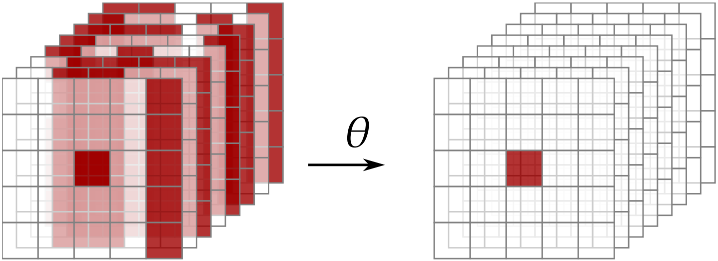
$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x + x^4 z\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

# Diffusion of $\theta$ (kernel)



$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x + x^4 z\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

# Diffusion of $\theta^{-1}$



$$1 + \left(1 + y + y^2 + y^3 + y^4\right) \mathbf{Q},$$

$$\text{with } \mathbf{Q} = 1 + \left(1 + x + x^4 z\right)^{-1} \bmod \left\langle 1 + x^5, 1 + z^w \right\rangle$$

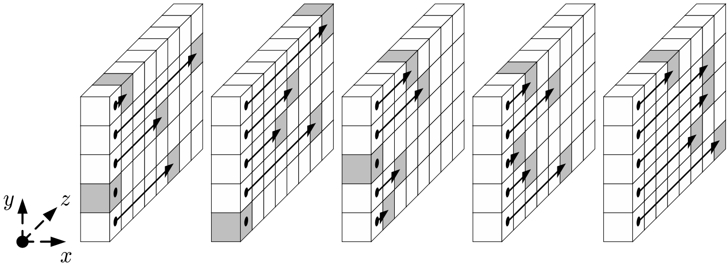
- **Q** is dense, so:
  - Diffusion from single-bit output to input very high
  - Increases resistance against LC/DC and algebraic attacks

# $\rho$ for inter-slice dispersion

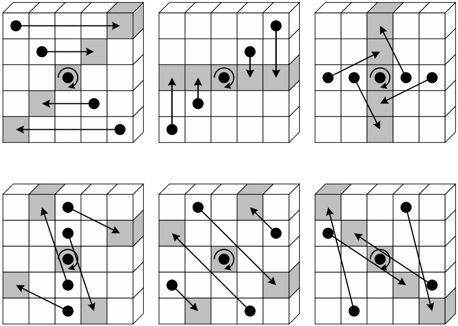
- We need diffusion between the slices ...
- $\rho$: cyclic shifts of lanes with offsets

$$i(i+1)/2 \bmod 2^{\ell}, \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{i-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- Offsets cycle through all values below $2^{\ell}$



# $\pi$ for disturbing horizontal/vertical alignment



$$a_{x,y} \leftarrow a_{x',y'} \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

# $\iota$ to break symmetry

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

# KECCAK-$f$ summary

- Round function:
$$\mathsf{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$
- Number of rounds: $12 + 2\ell$
  - KECCAK-$f[25]$ has 12 rounds
  - KECCAK-$f[1600]$ has 24 rounds

# NIST FIPS 202 (August 2015)

- Four drop-in replacements to SHA-2
- Two *extendable output functions* (XOF)

| **XOF** | **SHA-2 drop-in replacements** |
|---|---|
| $\textsc{Keccak}[c = 256](M\|11\|11)$ | |
| | first 224 bits of $\textsc{Keccak}[c = 448](M\|01)$ |
| $\textsc{Keccak}[c = 512](M\|11\|11)$ | first 256 bits of $\textsc{Keccak}[c = 512](M\|01)$ |
| | first 384 bits of $\textsc{Keccak}[c = 768](M\|01)$ |
| | first 512 bits of $\textsc{Keccak}[c = 1024](M\|01)$ |
| **SHAKE128** and **SHAKE256** | **SHA3-224** to **SHA3-512** |

- Toolbox for building other functions

# NIST SP 800-185 (December 2016)

Customized SHAKE (**cSHAKE**)

- $H(x) = \text{cSHAKE}(x, \text{name}, \text{customization string})$
- E.g., $\text{cSHAKE128}(x, N, S) = \textsc{Keccak}[c = 256](\text{encode}(N, S) \| x \| \texttt{00})$
- $\text{cSHAKE128}(x, N, S) \triangleq \text{SHAKE128}$ when $N = S = $ ""

**KMAC**: message authentication code (no need for HMAC-SHA-3!)

$$\text{KMAC}(K, x, S) = \text{cSHAKE}(\text{encode}(K) \| x, \text{"KMAC"}, S)$$

**TupleHash**: hashing a sequence of strings $\mathbf{x} = x_n \circ x_{n-1} \circ \cdots \circ x_1$

$$\text{TupleHash}(\mathbf{x}, S) = \text{cSHAKE}(\text{encode}(\mathbf{x}), \text{"TupleHash"}, S)$$

# NIST SP 800-185 (December 2016)

Customized SHAKE (**cSHAKE**)

- $H(x) = \text{cSHAKE}(x, \text{name}, \text{customization string})$
- E.g., $\text{cSHAKE128}(x, N, S) = \textsc{Keccak}[c = 256](\text{encode}(N, S) \| x \| \texttt{00})$
- $\text{cSHAKE128}(x, N, S) \triangleq \text{SHAKE128 when } N = S = \text{“”}$

**KMAC**: message authentication code (no need for HMAC-SHA-3!)

$$\text{KMAC}(K, x, S) = \text{cSHAKE}(\text{encode}(K) \| x, \text{"KMAC"}, S)$$

**TupleHash**: hashing a sequence of strings $\mathbf{x} = x_n \circ x_{n-1} \circ \cdots \circ x_1$

$$\text{TupleHash}(\mathbf{x}, S) = \text{cSHAKE}(\text{encode}(\mathbf{x}), \text{"TupleHash"}, S)$$

# NIST SP 800-185 (December 2016)

Customized SHAKE (**cSHAKE**)

- $H(x) = \text{cSHAKE}(x, \text{name}, \text{customization string})$
- E.g., $\text{cSHAKE128}(x, N, S) = \textsc{Keccak}[c = 256](\text{encode}(N, S) \| x \| \texttt{00})$
- $\text{cSHAKE128}(x, N, S) \triangleq \text{SHAKE128}$ when $N = S =$ “”

**KMAC**: message authentication code (no need for HMAC-SHA-3!)

$$\text{KMAC}(K, x, S) = \text{cSHAKE}(\text{encode}(K) \| x, \text{"KMAC"}, S)$$

**TupleHash**: hashing a sequence of strings $\mathbf{x} = x_n \circ x_{n-1} \circ \cdots \circ x_1$

$$\text{TupleHash}(\mathbf{x}, S) = \text{cSHAKE}(\text{encode}(\mathbf{x}), \text{"TupleHash"}, S)$$

# NIST SP 800-185 (December 2016)

Customized SHAKE (**cSHAKE**)

- $H(x) = \text{cSHAKE}(x, \text{name}, \text{customization string})$
- E.g., $\text{cSHAKE128}(x, N, S) = \text{KECCAK}[c = 256](\text{encode}(N, S) \| x \| \texttt{00})$
- $\text{cSHAKE128}(x, N, S) \triangleq \text{SHAKE128}$ when $N = S =$ “”

**KMAC**: message authentication code (no need for HMAC-SHA-3!)

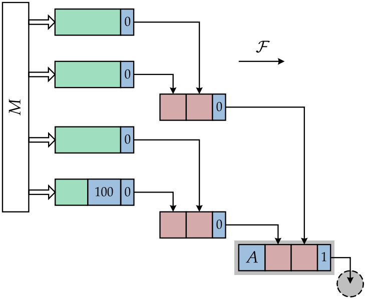
$$\text{KMAC}(K, x, S) = \text{cSHAKE}(\text{encode}(K) \| x, \text{"KMAC"}, S)$$

**TupleHash**: hashing a sequence of strings $\mathbf{x} = x_n \circ x_{n-1} \circ \cdots \circ x_1$

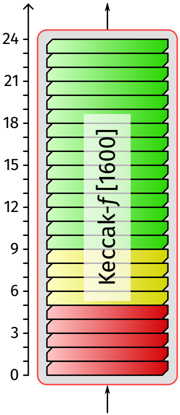
$$\text{TupleHash}(\mathbf{x}, S) = \text{cSHAKE}(\text{encode}(\mathbf{x}), \text{"TupleHash"}, S)$$

# NIST SP 800-185 (December 2016)

**ParallelHash**: faster hashing with parallelism



# Status of KECCAK cryptanalysis




- Collision attacks up to 5 rounds
  - Also up to 6 rounds, but for non-standard parameters ($c = 160$)

  [Song, Liao, Guo, CRYPTO 2017]
- Distinguishers
  - 7 rounds (practical time)
    [Huang et al., EUROCRYPT 2017]
  - 8 rounds ($2^{128}$ time)
    [Dinur et al., EUROCRYPT 2015]
  - 9 rounds ($2^{64}$ time)
    [Suryawanshi et al., AFRICACRYPT 2020]
- Lots of third-party cryptanalysis available at:
  `https://keccak.team/third_party.html`

# KangarooTwelve