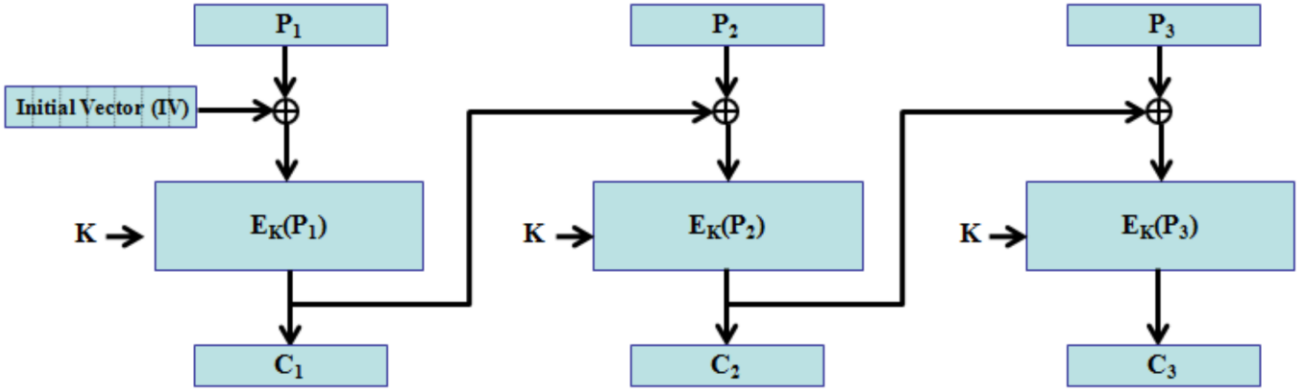
Permutation-based Crypto

2021.04.21

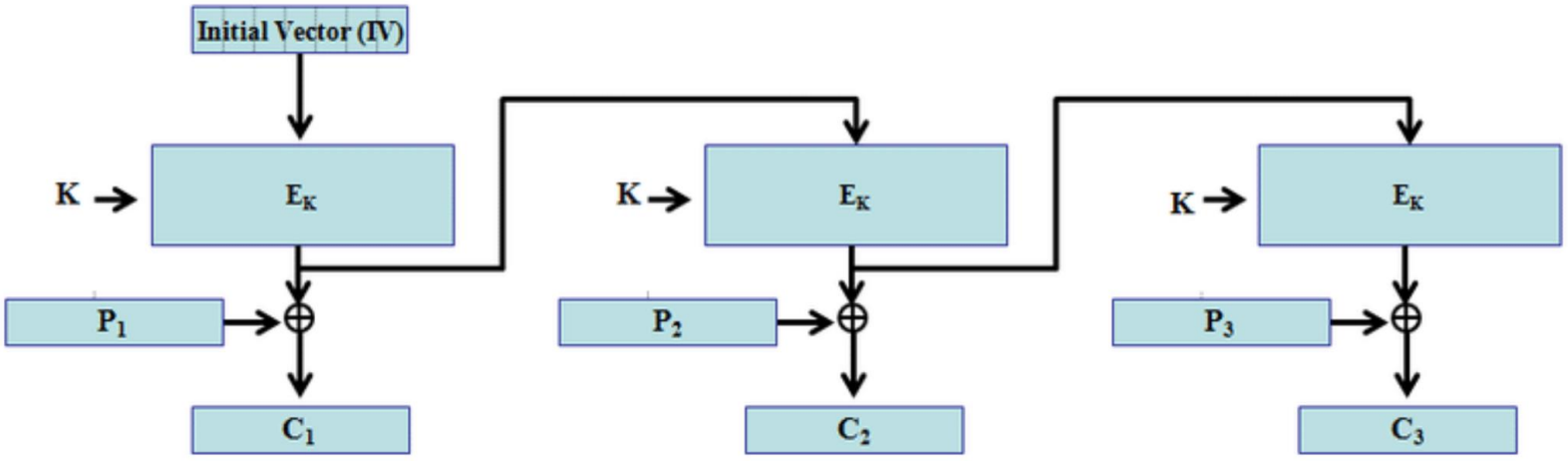
Symmetric cryptographic primitives

- Block encryption: CBC, …



Symmetric cryptographic primitives

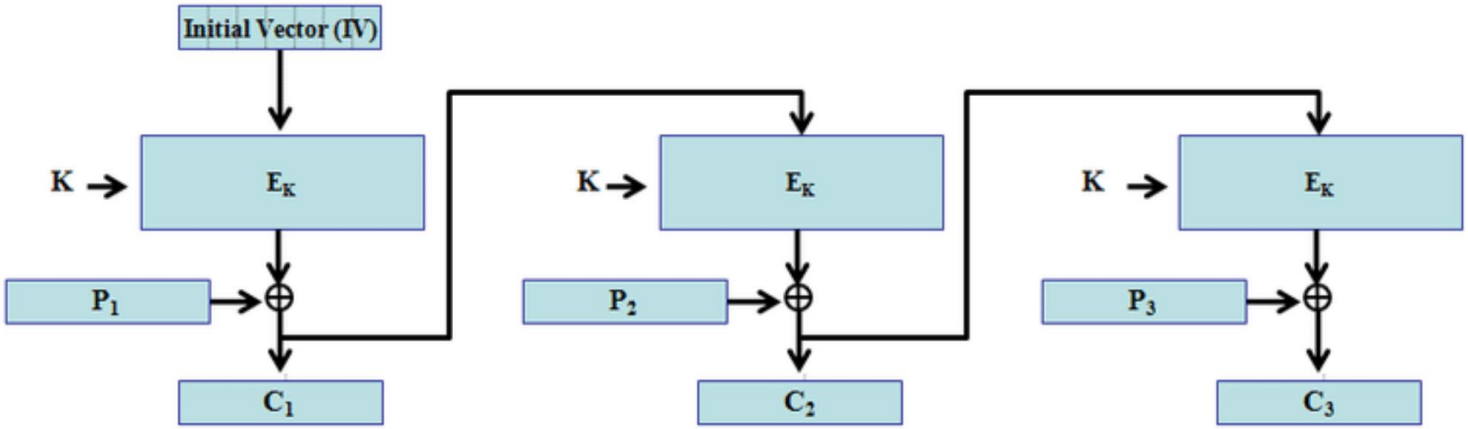
- Block encryption: CBC, …
- Stream cipher
  - Synchronous: OFB,…



The key stream is independent of the Plaintext/Ciphertext.

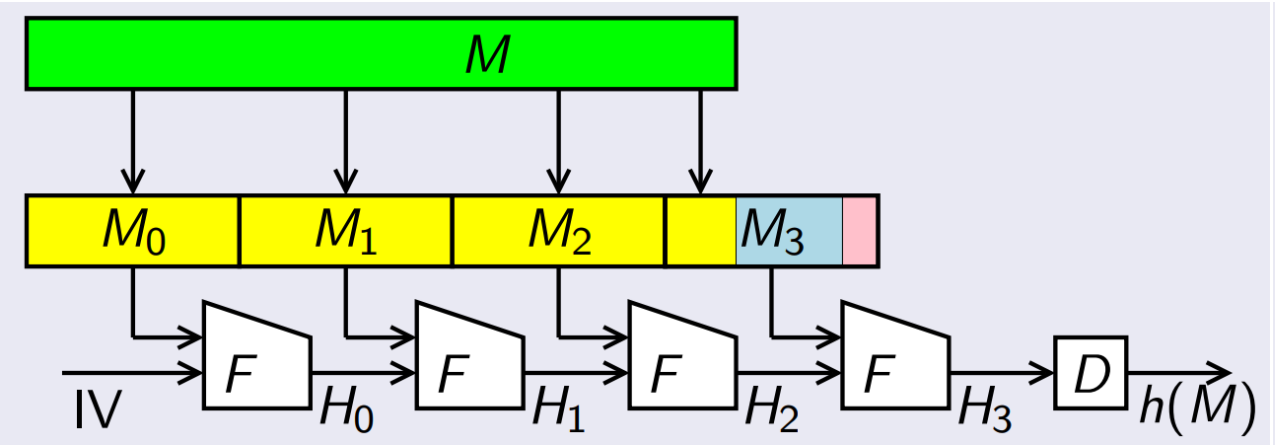
Symmetric cryptographic primitives

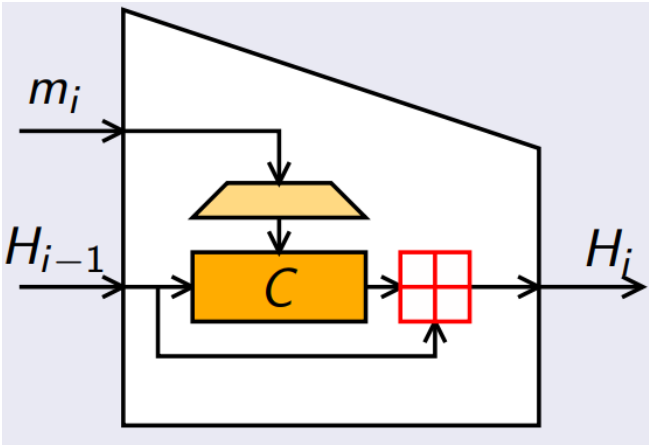
- Block encryption: CBC, …
- Stream cipher
  - Synchronous: OFB,…
  - Self-synchronizing: CFB, …



The key stream is dependent of the Plaintext/Ciphertext.

Symmetric cryptographic primitives

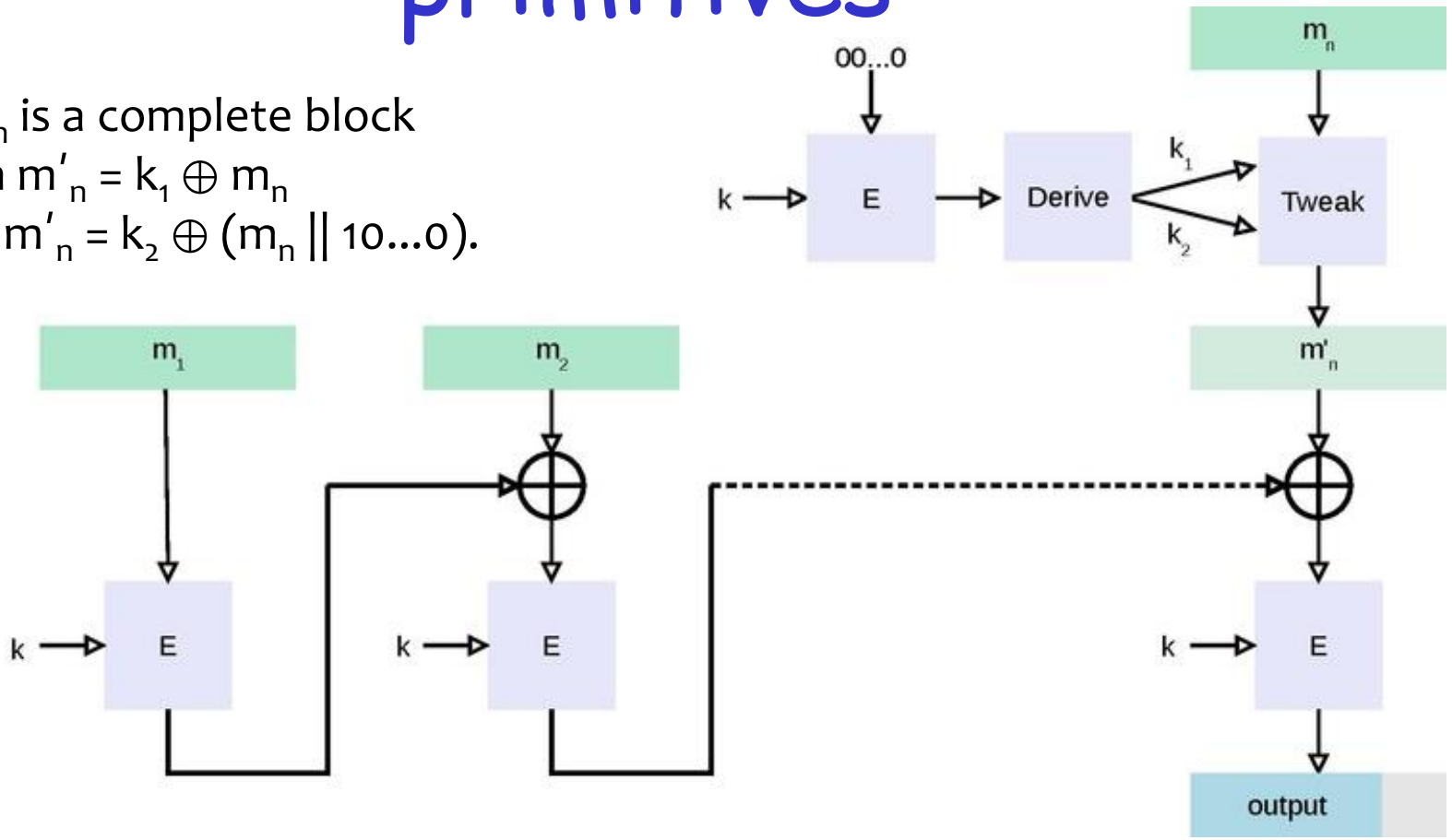




- Hash functions makes use of block ciphers
  - SHA-1, SHA-2, Whirlpool, RipeMD,…
  - HMAC, MGF1, …

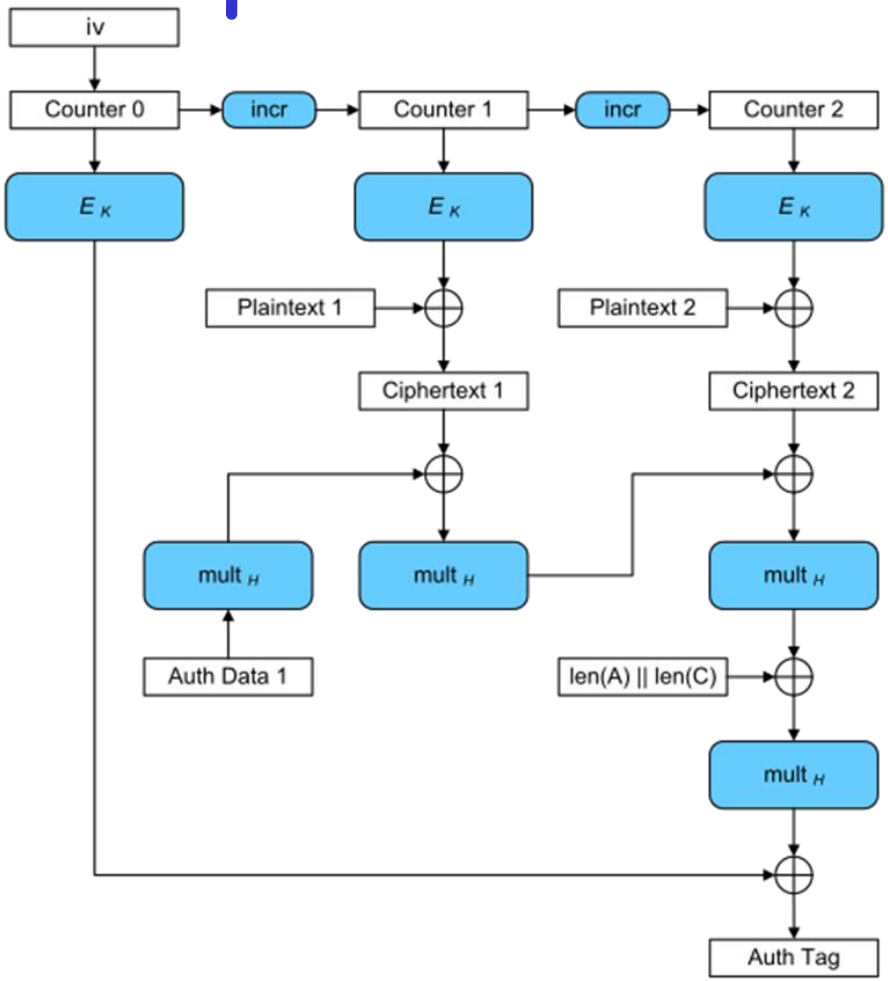
Symmetric cryptographic primitives

If m_n is a complete block
then $m'_n = k_1 \oplus m_n$
else $m'_n = k_2 \oplus (m_n \,||\, 10\ldots0)$.



- MAC: CMAC, ...

Symmetric cryptographic primitives

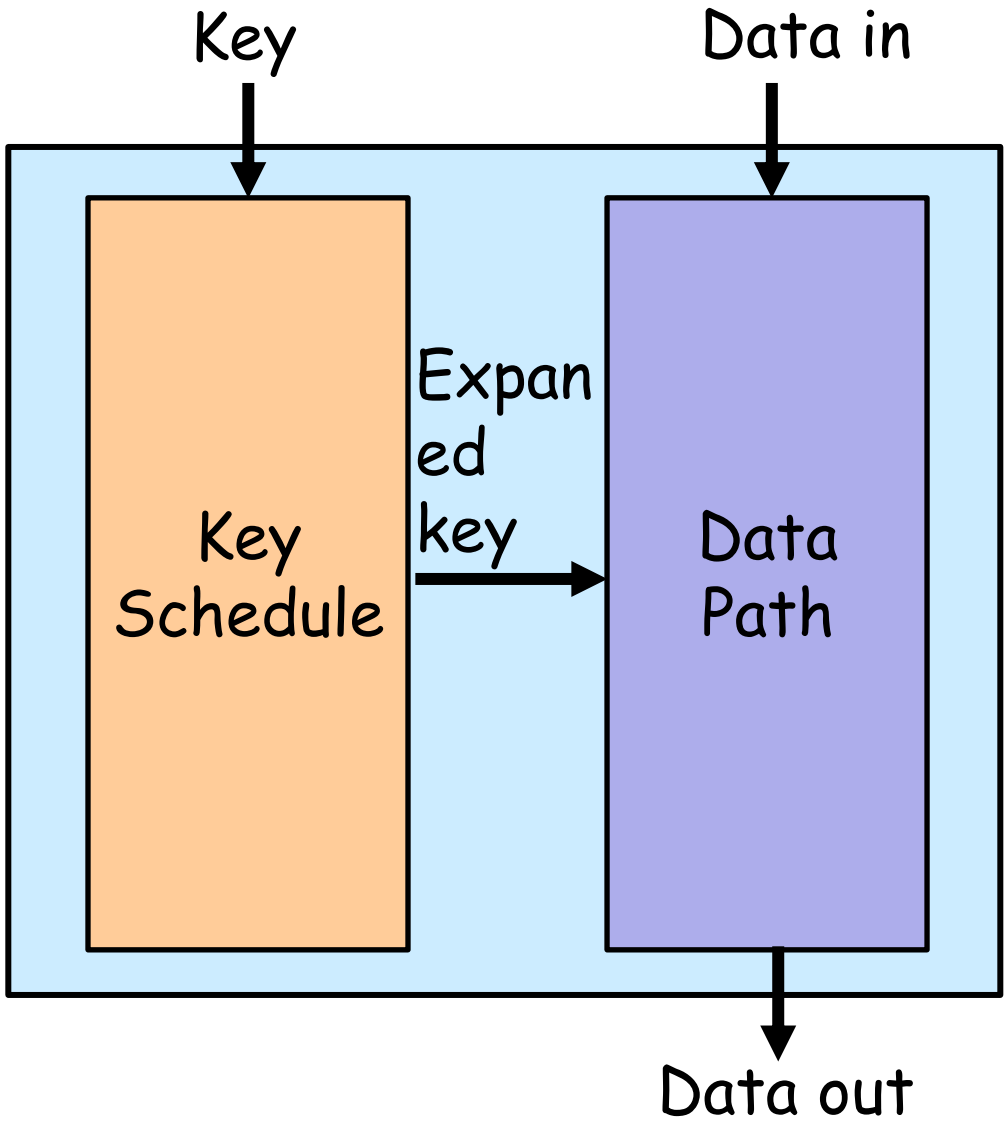


- Authenticated encryption: GCM

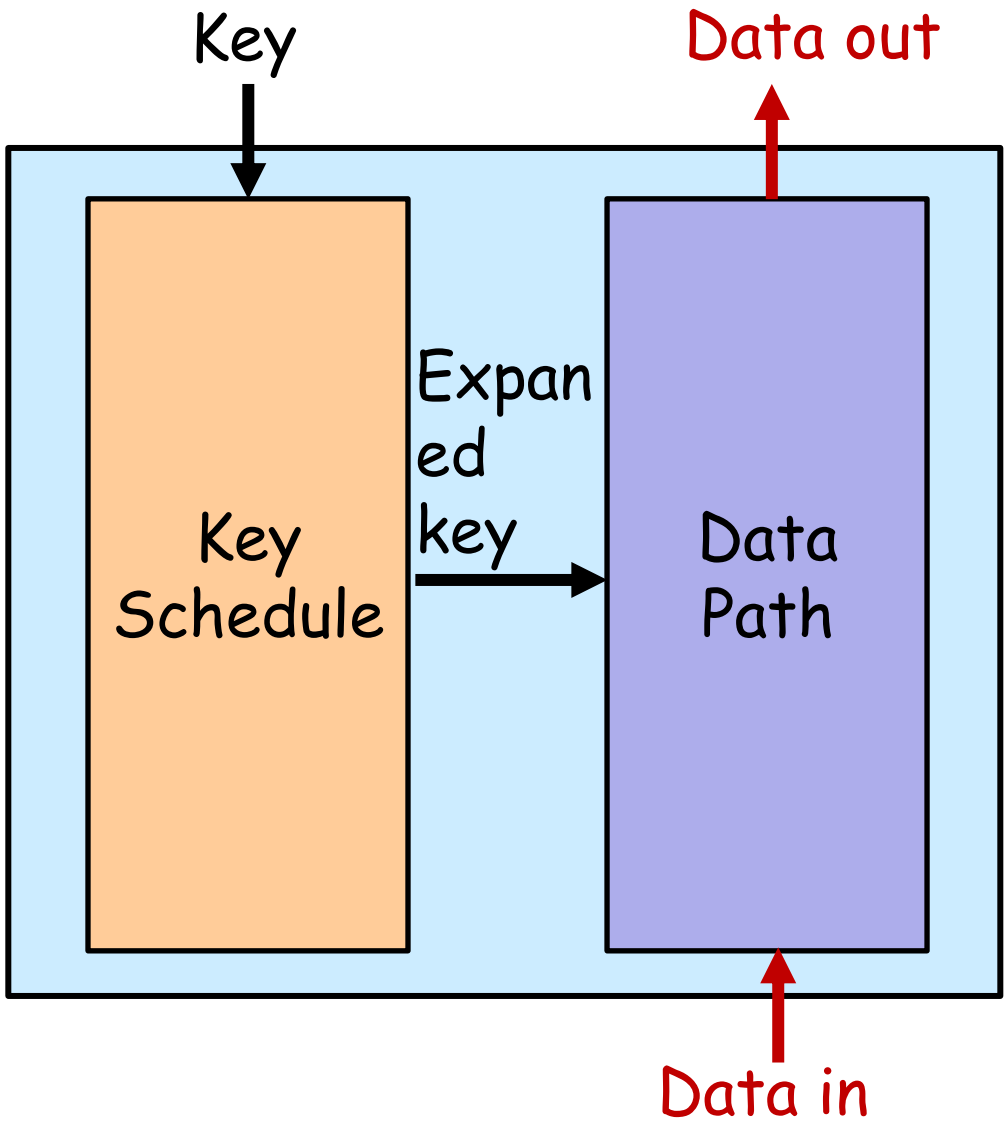
Modern-day cryptography is block-cipher centric

- Block encryption: CBC, …
- Stream cipher
  - Synchronous: OFB,…
  - Self-synchronizing: CFB, …
- Hash functions makes use of block ciphers
  - SHA-1, SHA-2, Whirlpool, RipeMD,…
  - HMAC, MGF1, …
- MAC: CMAC
- Authenticated encryption: GCM, OCB, …

Structure of a block cipher



Structure of a block cipher (inverse)



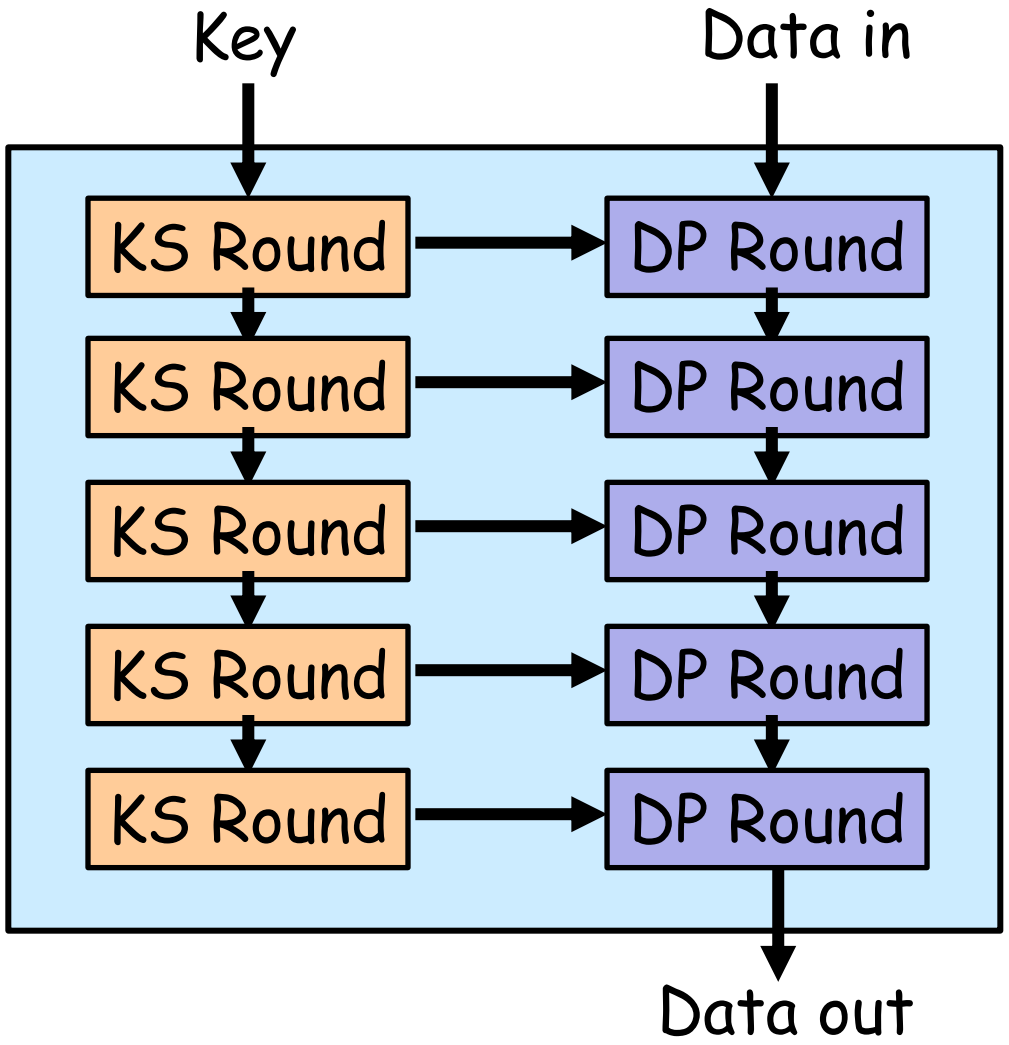
When is the inverse block cipher needed?

Indicated in red:

- Block encryption: CBC, …
- Stream cipher
  - Synchronous: OFB,…
  - Self-synchronizing: CFB, …
- Hash functions makes use of block ciphers
  - SHA-1, SHA-2, Whirlpool, RipeMD,…
  - HMAC, MGF1, …
- MAC: CMAC
- Authenticated encryption: GCM, OCB, …

So a block cipher without inverse can do a lot!

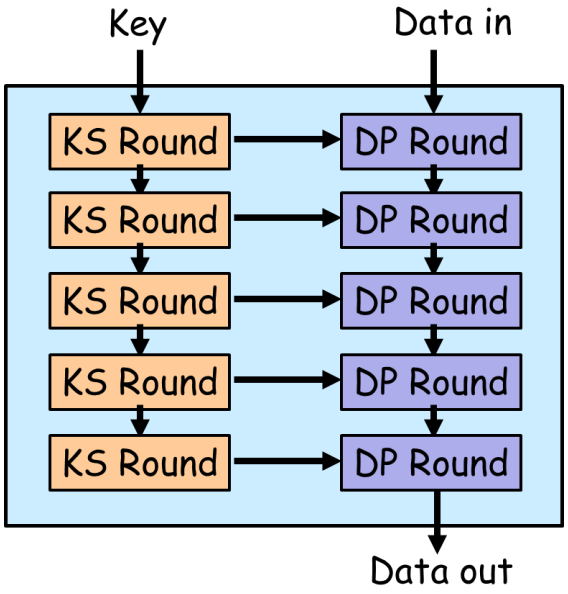
Look deep


Key
Data in
KS Round
KS Round
KS Round
KS Round
KS Round
DP Round
DP Round
DP Round
DP Round
DP Round
Data out

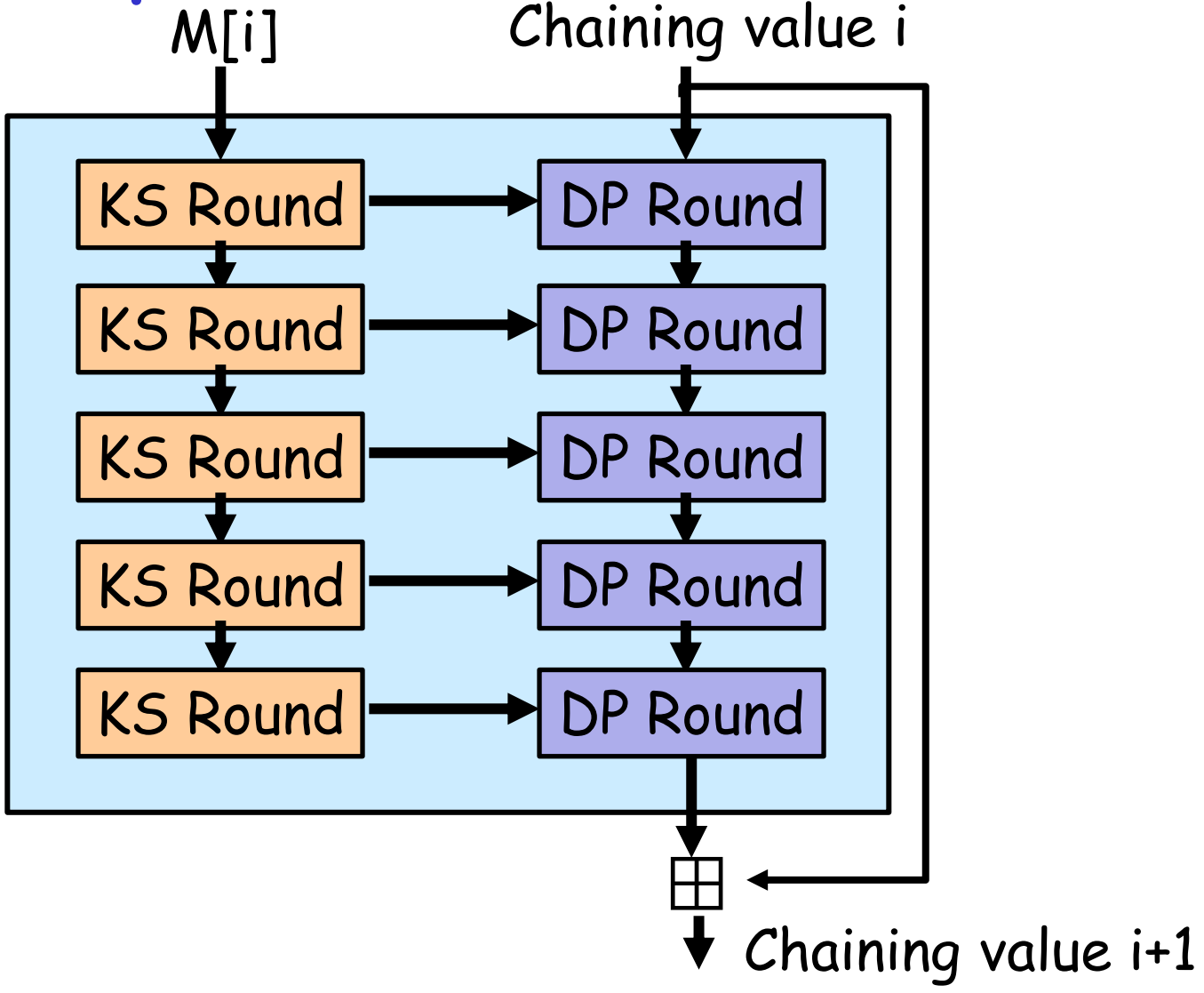
Look deep: n-bit block cipher with k-bit key

b-bit permutation with b = n+k

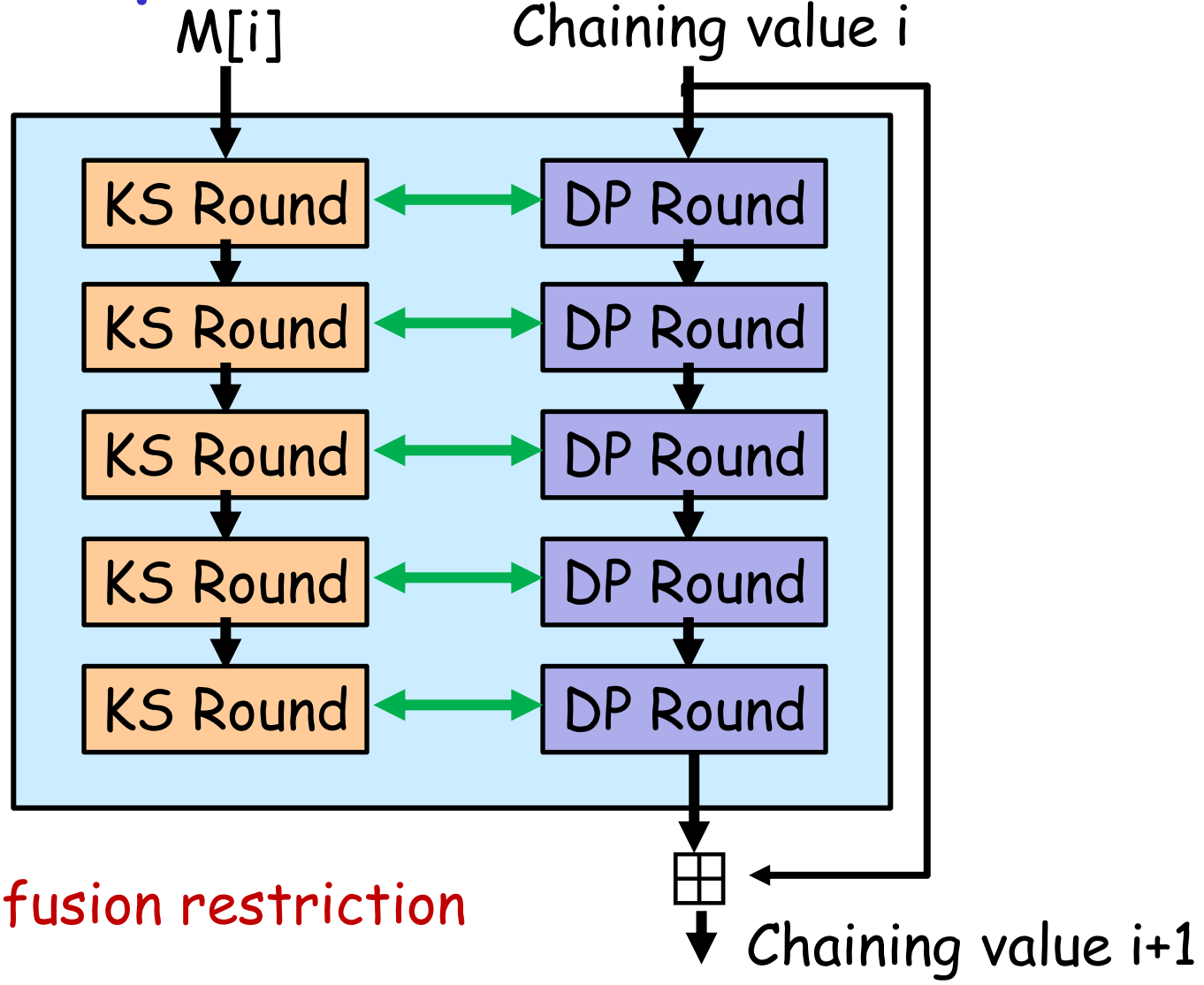
- Iterate an invertible round function
- Efficient inverse
- No diffusion from data part to key part



Take hashing as an example: compression function

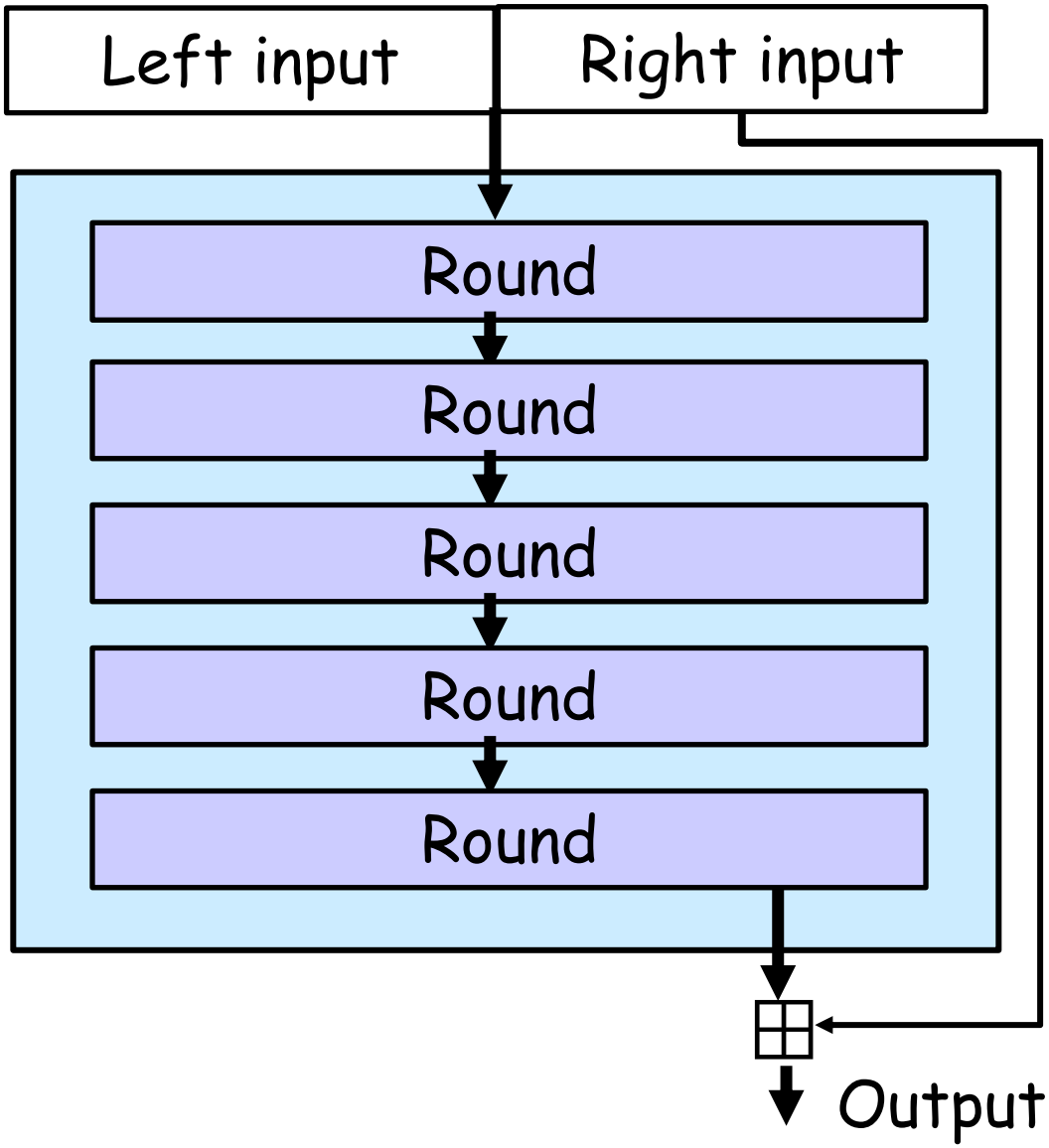


Take hashing as an example: compression function



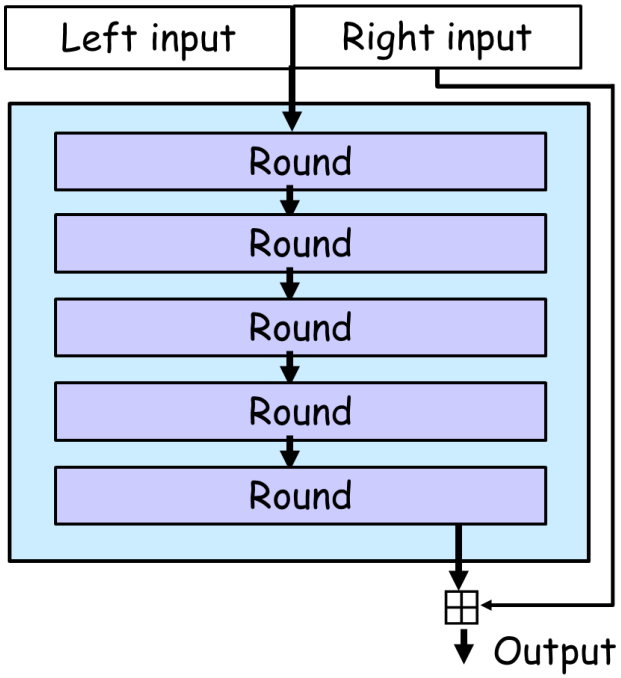
Removing diffusion restriction

Simplify the view: iterated perm



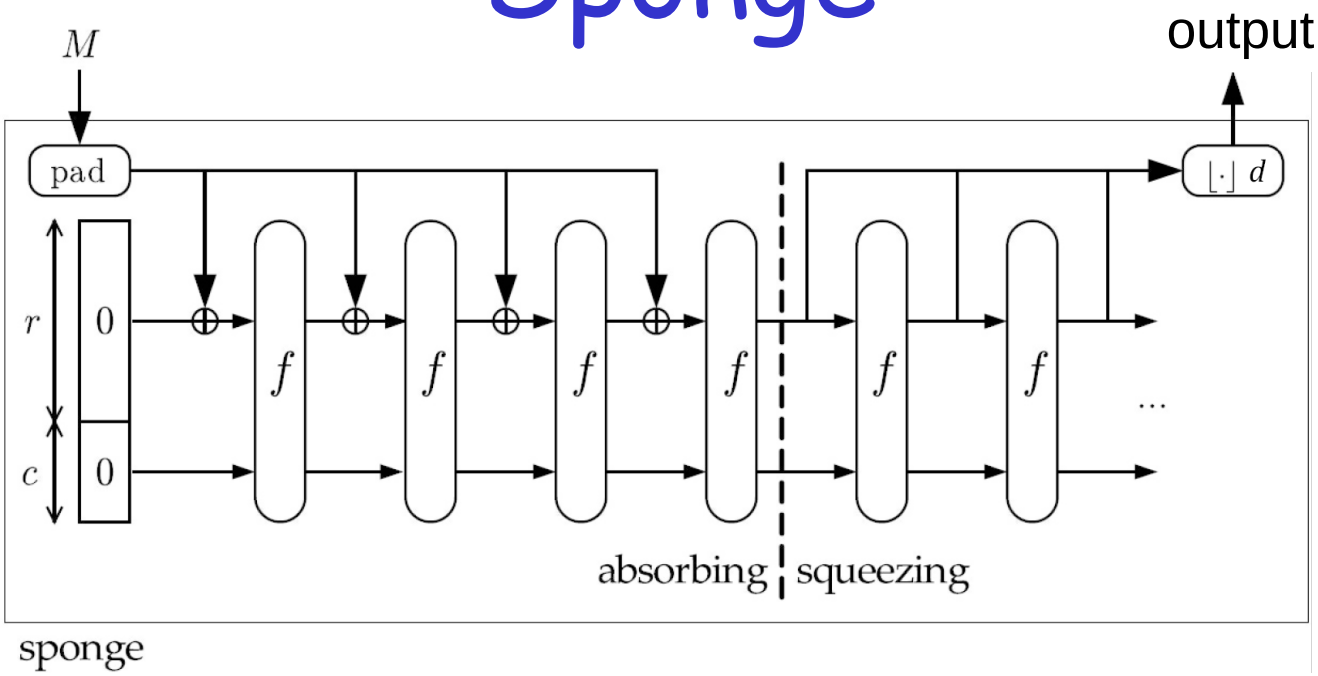
BC without inverse: wide perm

- Applies to all modes where inverse is not needed
- Do not need a separate key schedule
- n-bit block cipher -> b-bit permutation
  - b=n+k
- Permutation as a generalization of a block cipher



Permutation-based Crypto

Permutation-based construction: Sponge

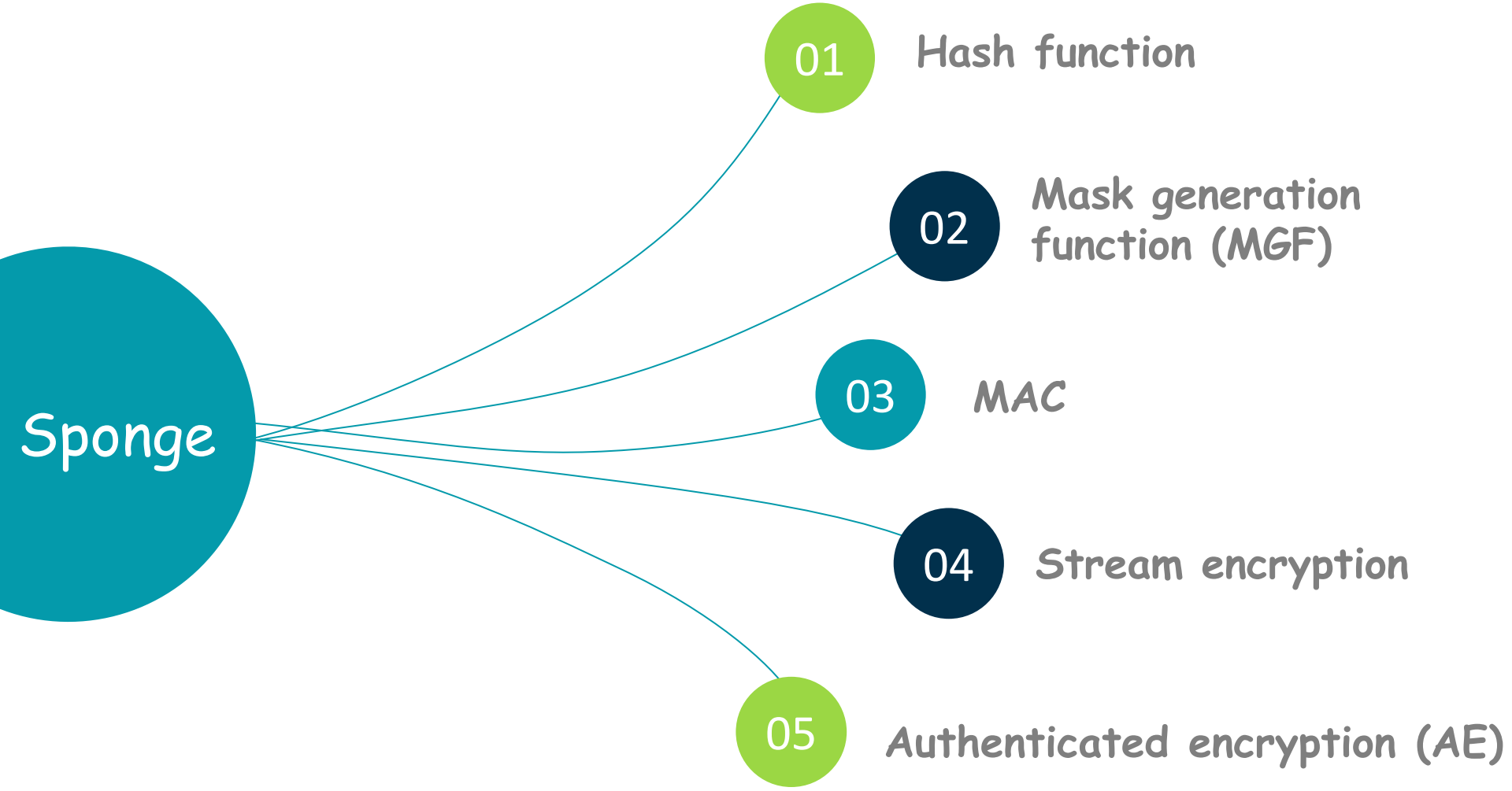


- f: a b-bit permutation with b = r + c
  - Efficiency: process an r-bit block per call to f
  - Security: provably resists generic attacks up to $2^{c/2}$
- Trade-off between r and c can be made.

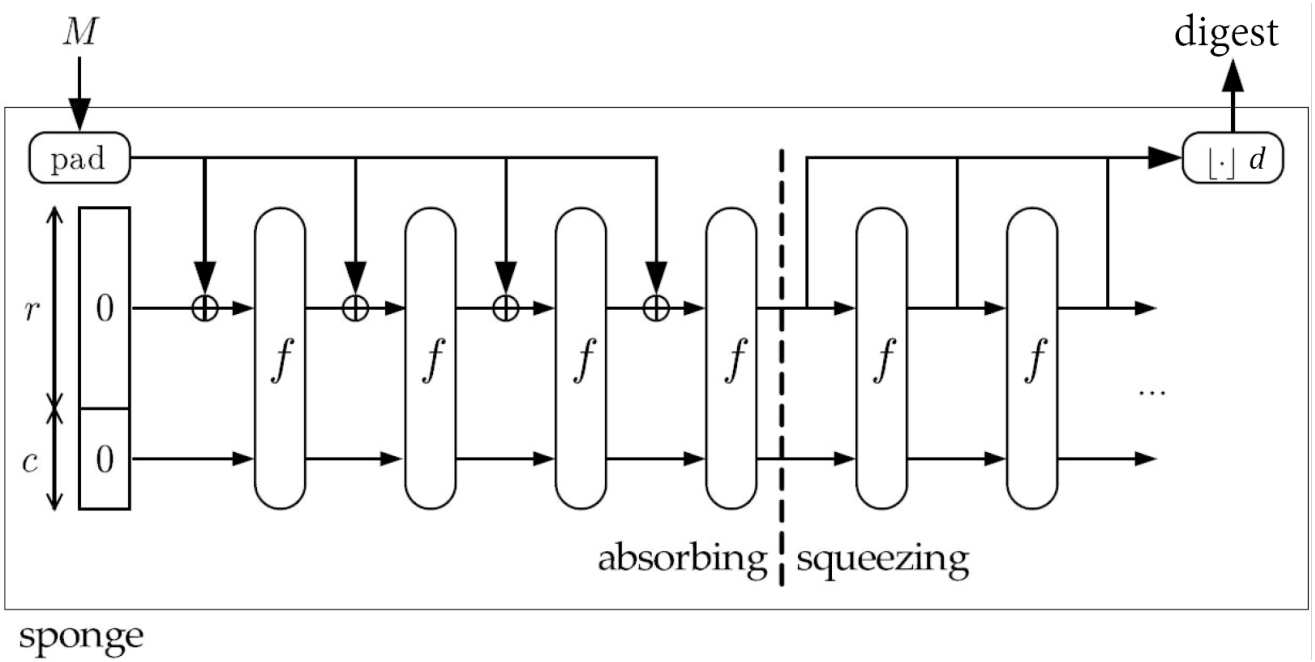
Security of Sponge

- Generic security:
  - Assume f is chosen randomly
  - Resist against generic attacks
  - Construction as sound as theoretically possible
- For a specific choice of f
  - Security proof is infeasible
  - Design with attacks in mind
  - Security based on absence of attacks despite public scrutiny

Usage of Sponge



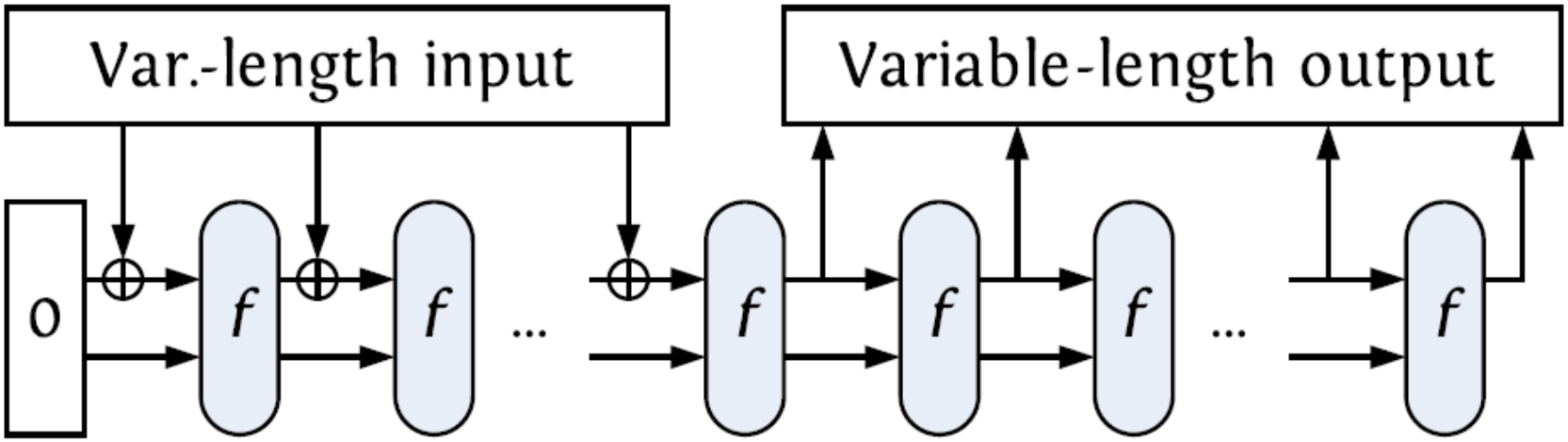
Usage of Sponge: Hashing



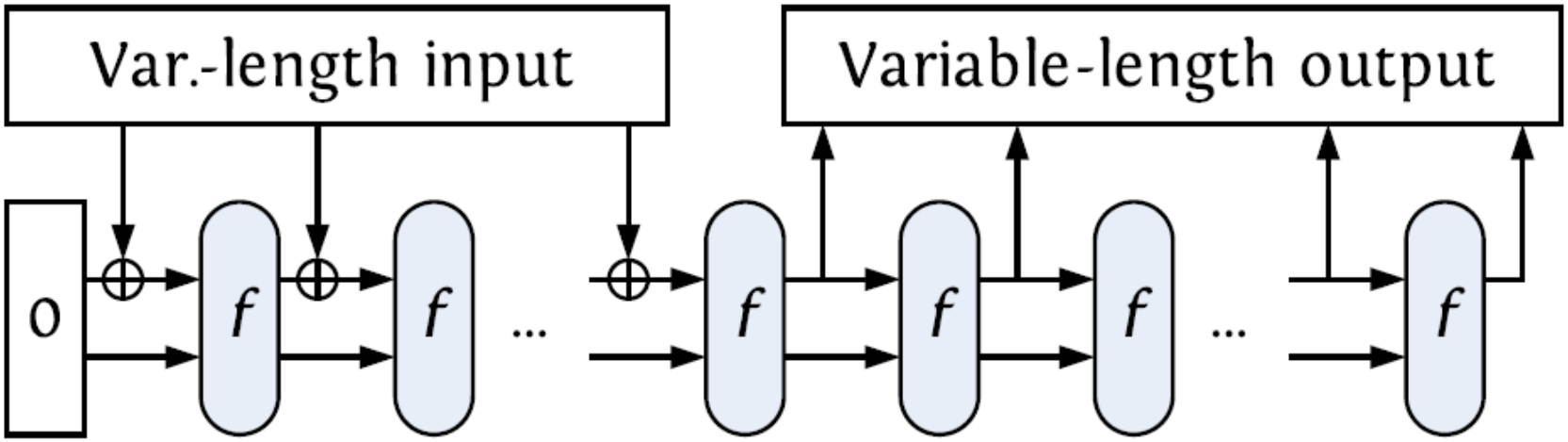
Security margin

- Collision: $\min(2^{c/2}, 2^{d/2})$
- Preimage: $\min(2^{c/2}, 2^{d})$
- Second Preimage: $\min(2^{c/2}, 2^{d})$

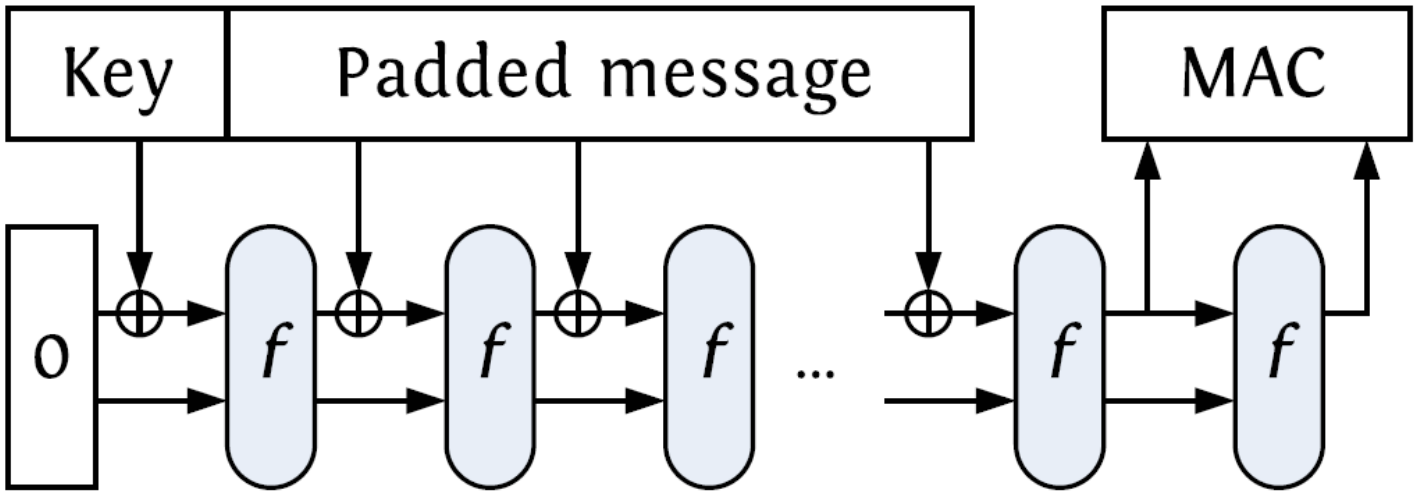
Usage of Sponge: Mask generating function



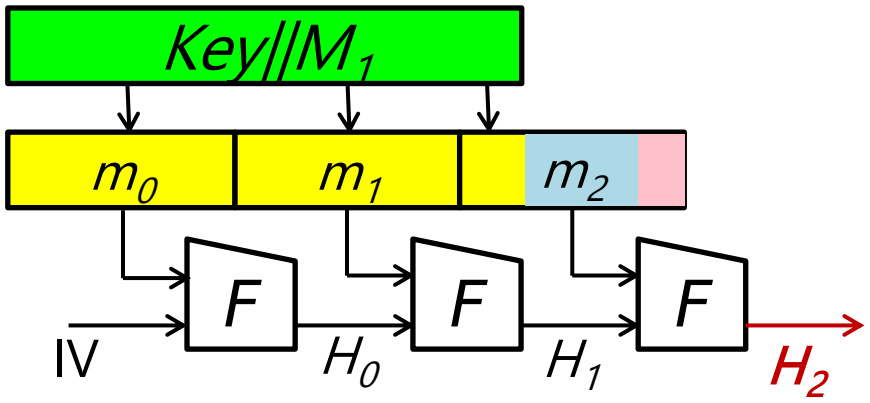
Usage of Sponge: Mask generating function



Usage of Sponge: MAC generation

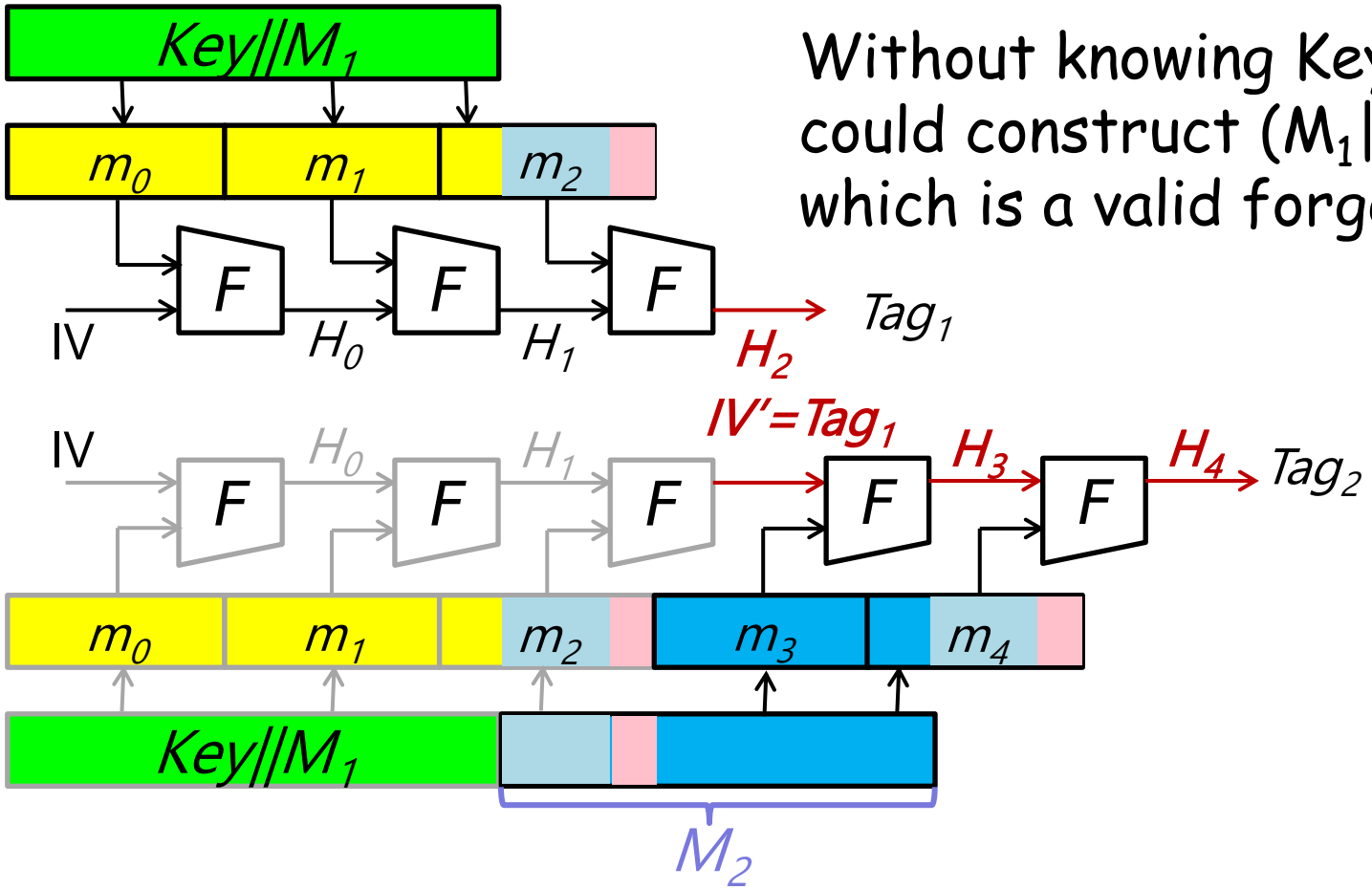


Compare with MD construction



Given (M_1, Tag_1)

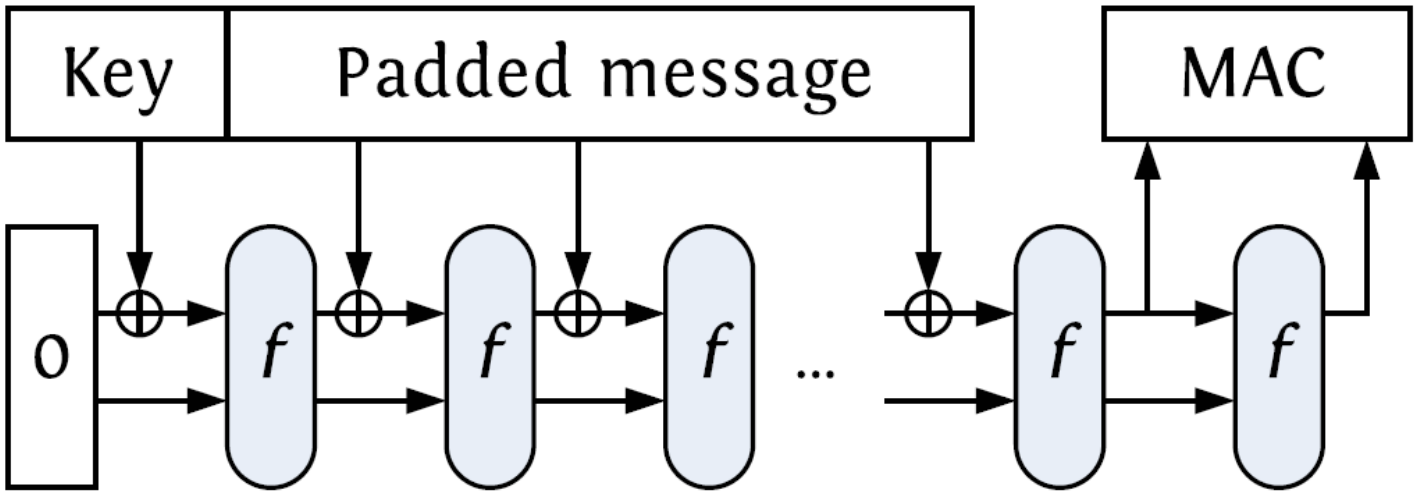
Compare with MD construction



Without knowing Key, we could construct ($M_1||M_2$, Tag_2) which is a valid forgery.

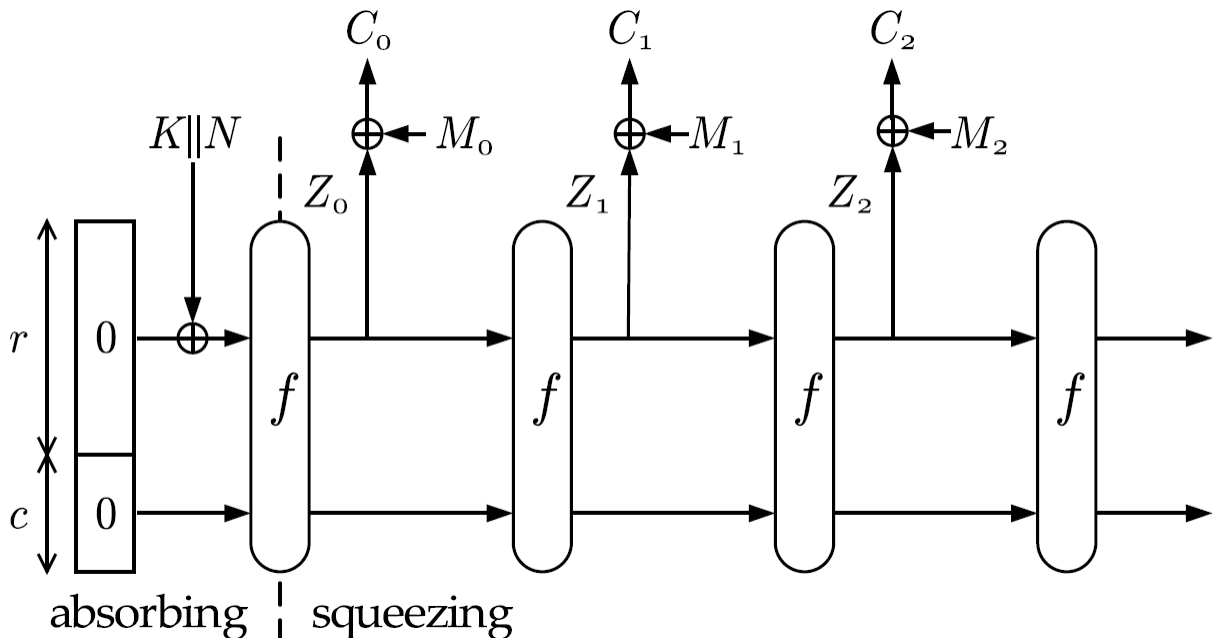
We **cannot** feed (K||M) directly into a MD hash for generating MAC.

Usage of Sponge: MAC generation



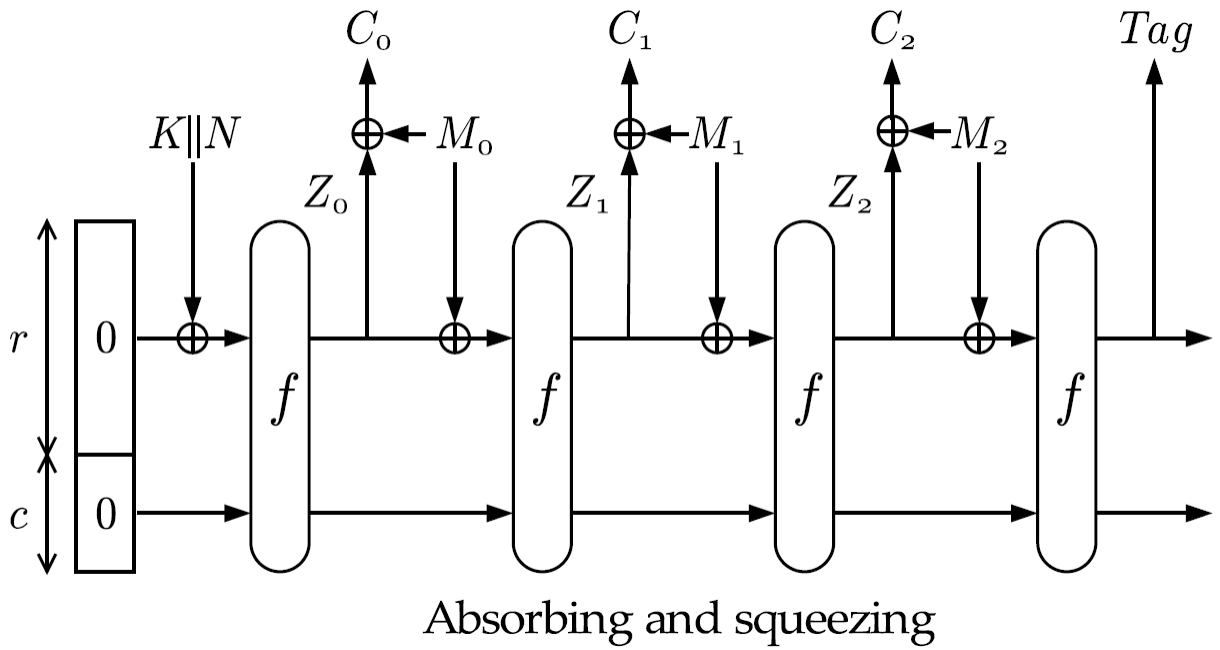
We **can** feed (K||M) directly into a Sponge hash for generating MAC. Why?

Usage of Sponge: encryption

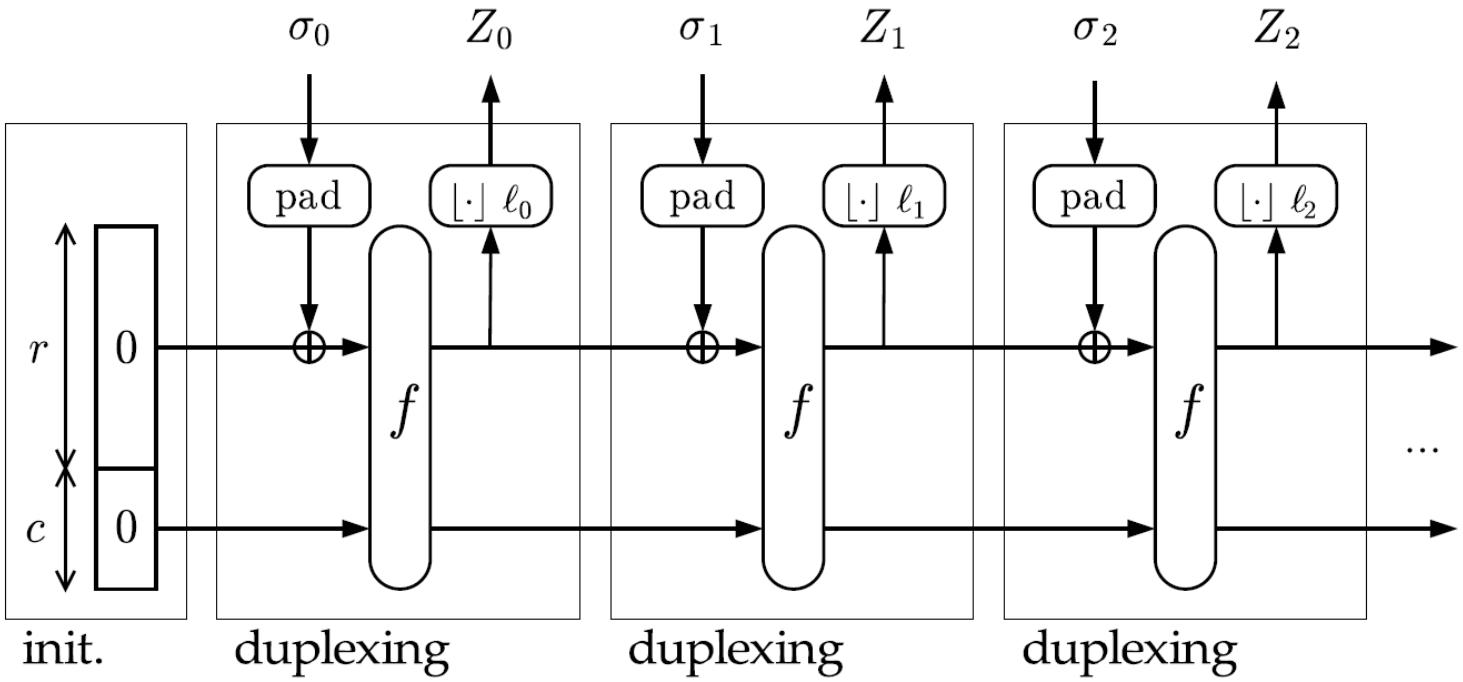


As a stream cipher

Usage of Sponge: authenticated encryption



A variant of Sponge for authenticated encryption



The Duplex construction

- Two additional parameters ℓ_0, ℓ_1
- Generic security equivalent to that of sponge

Permutation vs. Block ciphers

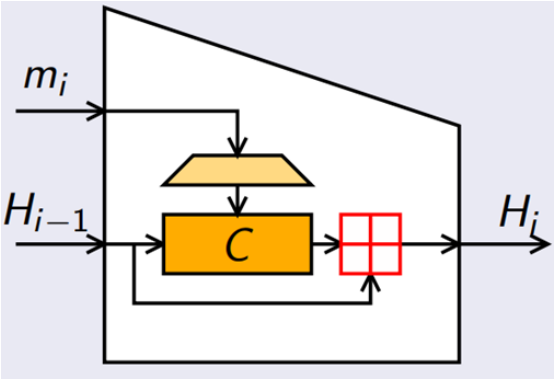
Hash functions based on Sponge

A selection of Sponge hash functions

Hash function	designers	ref	State size
Keccak	Bertoni, Daemen, Peeters, Van Assche	SHA-3 (2008)	**25, 50, 100, 200, 400, 800, 1600**
Quark	Aumasson, Henzen, Meier, Naya-Plasencia	CHEC 2010	**136, 176, 256**
Photon	Guo, Peyrin, Poschmann	Crypto 2011	**100, 144, 196, 256, 288**
Spongent	Bogdanov, Knezevic, Leander, Toz, Varici, Verbauwhede	CHES 2011	**88, 136, 176, 248, 320**

State sizes of hash functions

- Small state -> low area
- Target security strength c/2
- Block cipher-based
  - Block size n >= c
  - Message block length (key size) k >=n
  - Feedforward: n
  - Total state size **>= 3c**
- Sponge-based
  - Permutation width: b = c + r
  - r can be as small as 1 byte
  - Total state size **>= c + 8**



With the same state size, block cipher based schemes (MAC, AE) may have higher throughput.

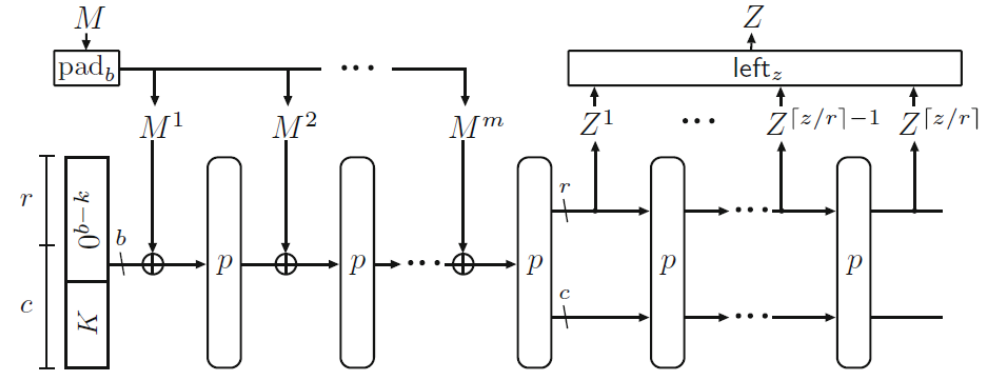
Other features

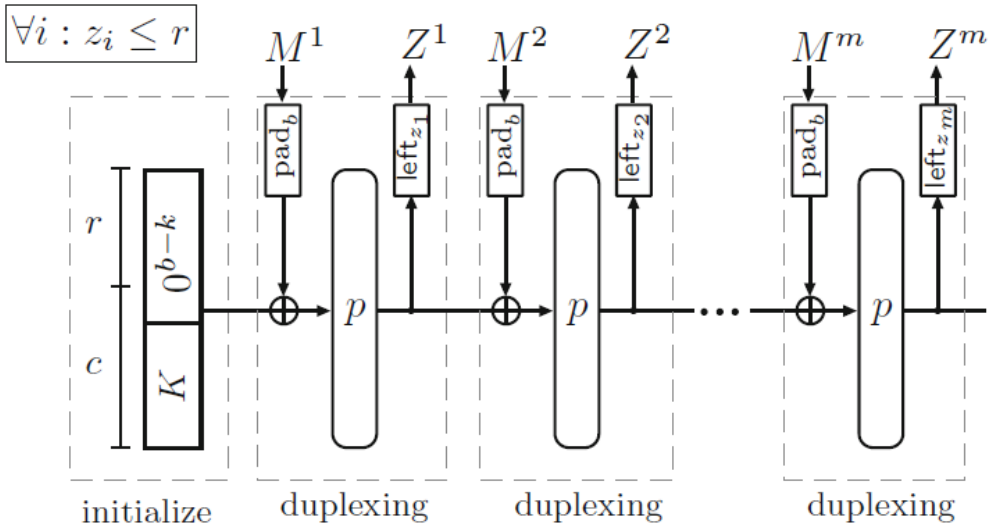
- Block cipher-based
  - Pre-computation of key schedule
    - Storing expanded key cost memory
    - May be prohibitive in resource-constrained devices
- Sponge-based
  - Diffusion across full state
  - Flexibility in choice of rate/capacity

Keyed Sponge

- Distinguishing vulnerability in keyed vs unkeyed modes
  - in keyed modes attacker has less power
  - allows decreasing number of rounds in permutation
- Rate/capacity trade-off
  - Allows full-state absorption
- Introducing dedicated variants
  - MAC computation
  - authenticated encryption

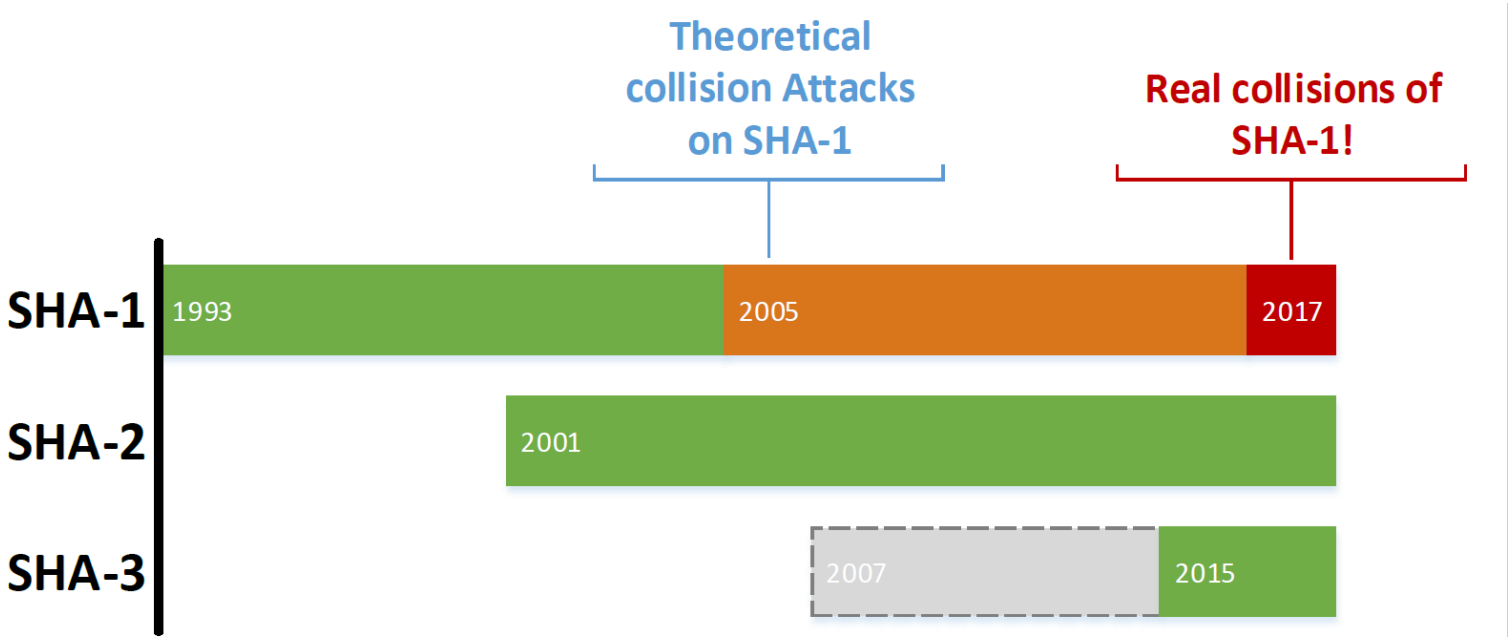
Full-state Keyed Sponge





KECCAK (SHA-3)

NIST standards of Secure Hash Algorithm



- The complexity of the 2017 real collision of SHA-1 remains the same as 2^{63} as for the 2005 breakthrough.

Out of the hash function crisis

- Trust in established hash functions was crumbling
  - Use of modular addition, rotation, xor
  - Adoption of MD construction
  - SHA-2 is based on the same principles of SHA-1
- 2007: NIST calls for SHA-3
  - Similar to AES contest

SHA-3 contest

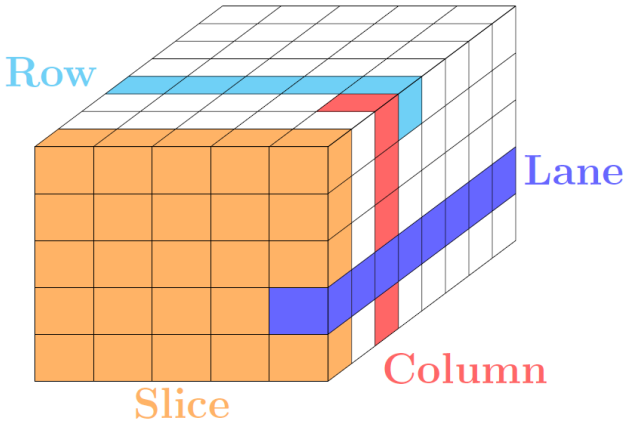
- Open competition organized by NIST
  - NIST provides a forum
  - scientific community contributes: designs, attacks, implementations, comparisons
  - NIST draws conclusions and decides
- Goal: replacement for the SHA-2 family
  - 224, 256, 384 and 512-bit output sizes
  - other output sizes are optional
- Requirements
  - security levels specified for traditional attacks
  - each submission must have
    - complete documentation, including design rationale
    - reference and optimized implementations in C
- The ongoing LWC competition follow the same way

KECCAK permutation: Keccak-f

- 1600 bits: seen as a 5×5 array of 64-bit lanes, $A[x, y], 0 \leq x, y < 5$
- 24 rounds
- each round R consists of five steps:

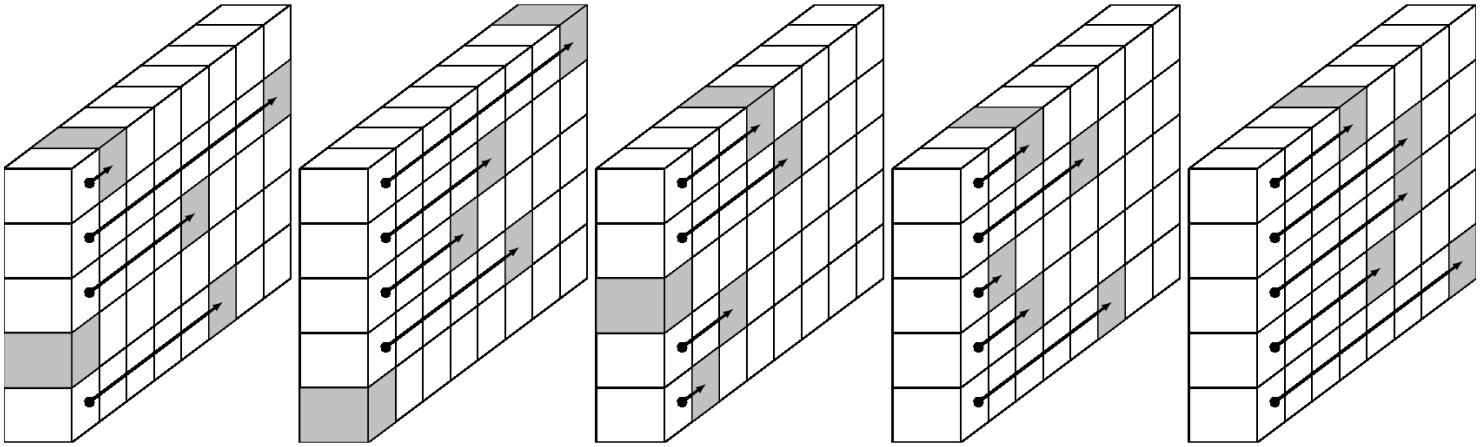
$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- χ : the only nonlinear operation, a 5-bit Sbox applies to each row.



ρ

- Lane level rotations

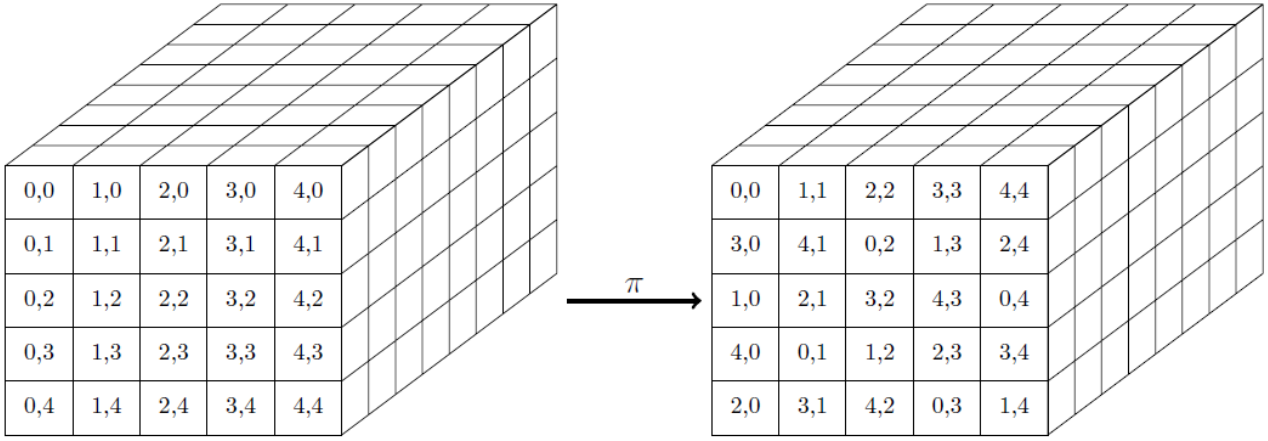


Rotation offsets $r[x, y]$

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$
$y = 0$	0	1	62	28	27
$y = 1$	36	44	6	55	20
$y = 2$	3	10	43	25	39
$y = 3$	41	45	15	21	8
$y = 4$	18	2	61	56	14

π

- Permutation on lanes

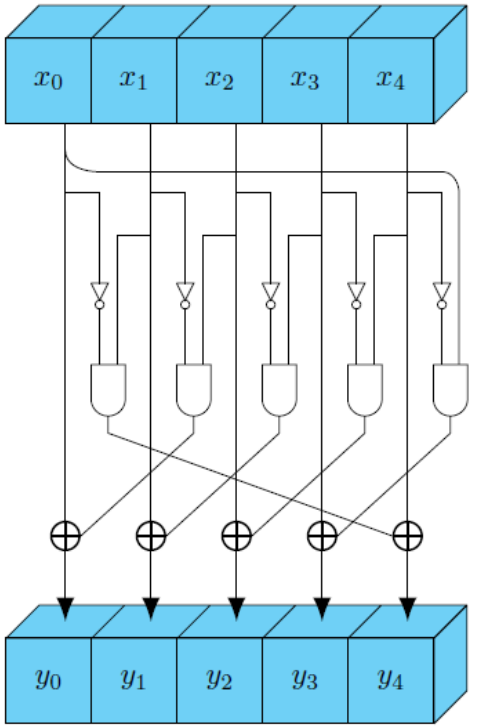


$$A[y, 2 * x + 3 * y] = A[x, y]$$

χ

- 5-bit S-boxes, nonlinear operation on rows

$$
\begin{aligned}
y_0 &= x_0 + (x_1 + 1) \cdot x_2, \\
y_1 &= x_1 + (x_2 + 1) \cdot x_3, \\
y_2 &= x_2 + (x_3 + 1) \cdot x_4, \\
y_3 &= x_3 + (x_4 + 1) \cdot x_0, \\
y_4 &= x_4 + (x_0 + 1) \cdot x_1.
\end{aligned}
$$



ι

- adding a round constant to $A[0,0]$, to destroy the symmetry.

Keccak-f

Internal state A: a 5 × 5 array of 64-bit lanes

θ step $C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4]$

$D[x] = C[x-1] \oplus (C[x+1] \lll 1)$

$A[x,y] = A[x,y] \oplus D[x]$

ρ step $A[x,y] = A[x,y] \lll r[x,y]$

- The constants $r[x,y]$ are the rotation offsets.

π step $A[y, 2*x+3*y] = A[x,y]$

χ step $A[x,y] = A[x,y] \oplus ((A[x+1,y]) \& A[x+2,y])$

ι step $A[0,0] = A[0,0] \oplus RC$

- $RC[i]$ are the round constants.

KECCAK instances

- KECCAK versions
  - KECCAK-*n*, *n* =224/256/384/512 and *c* = 2*n*, *d* = *n*.
- SHA-3 versions
  - SHA3-*n*, *n* =224/256/384/512 and *c* = 2*n*, *d* = *n*.
  - SHAKE*n* (eXtendable Output Functions, XOFs)
    - (SHAKE = SHA + KEccak)
    - *n* =128/256, *c* = 2*n*, *d* ≤ 2*n*.

Reasons for choosing Keccak by NIST

- Simple and elegant design
- Flexibility in choosing parameters
- Good performance in software (not as good as SHA-2)
- Excellent performance in hardware (better than SHA-2!)
- Built-in authenticated-encryption mode
- Different design than SHA-2

Other Permutation-based Crypto

Other schemes related to KECCAK

- The KECCAK-p permutations are derived from the KECCAK-f permutations and have a tunable number of rounds.

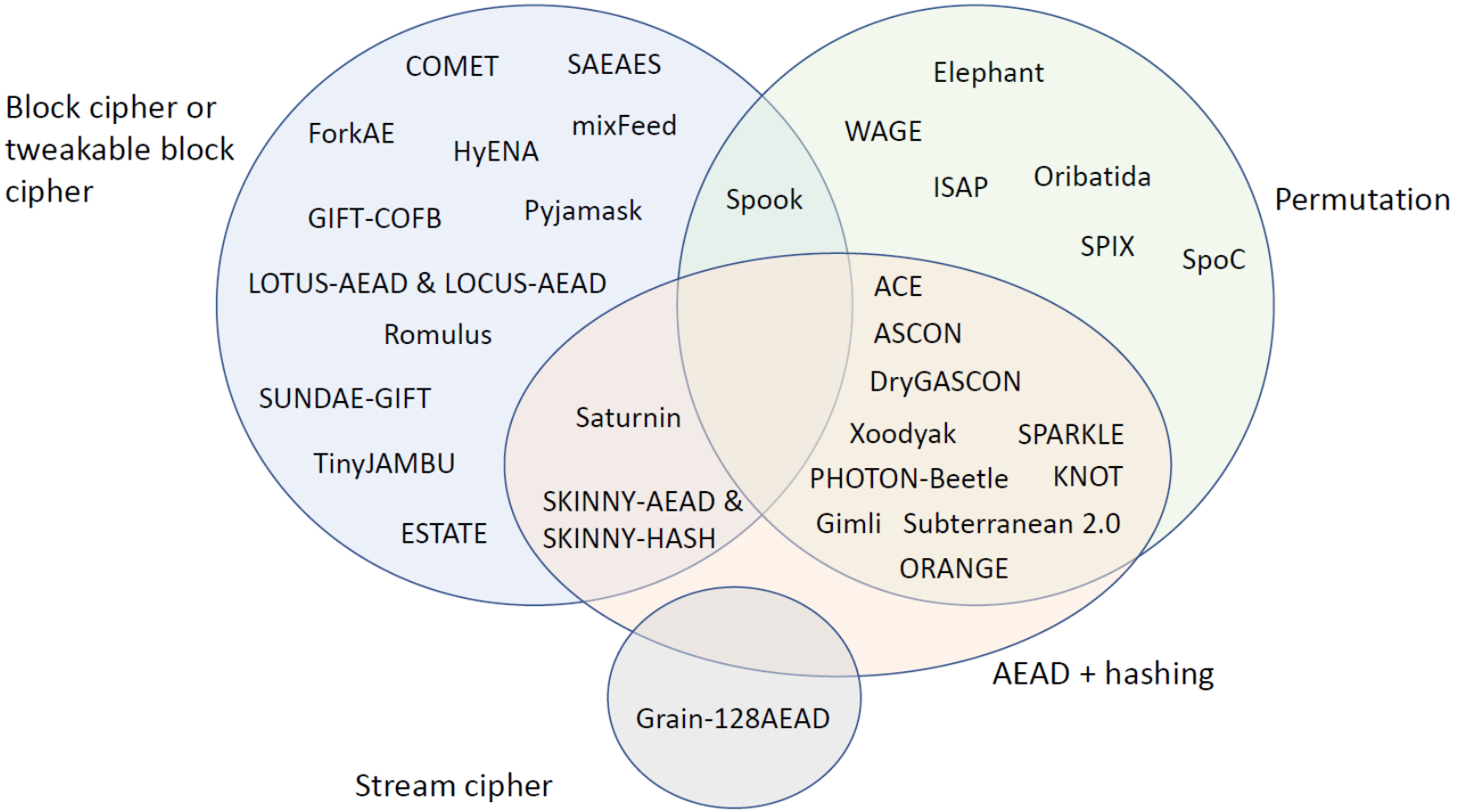
Underlying permutation	Structure	b	n_r	Schemes	Category
KECCAK-p	Sponge	1600	24	SHA-3 (KECCAK),cSHAKE, TupleHash, ParallelHash, KECCAK challenge instances	Hash functions
		1600	12	KangarooTwelve	
		1600	24	KMAC	MAC
	Duplex	400/200	20/18	Ketje	AE
		1600/800	12	Keyak	
	Farfalle	1600	6	Kravatte	PRF

NIST Lightweight Cryptography (LWC) Project

- Initiated in 2013
- To address growing industry need for security in resource constrained devices
  - Applications: Health tracking, Asset tracking (RFID), autonomous cars etc.
- To find new cryptographic primitives for constrained devices
  - To gather industry feedback on suitability of current crypto standards for constrained devices
  - To create standards for the use of Lightweight cryptography

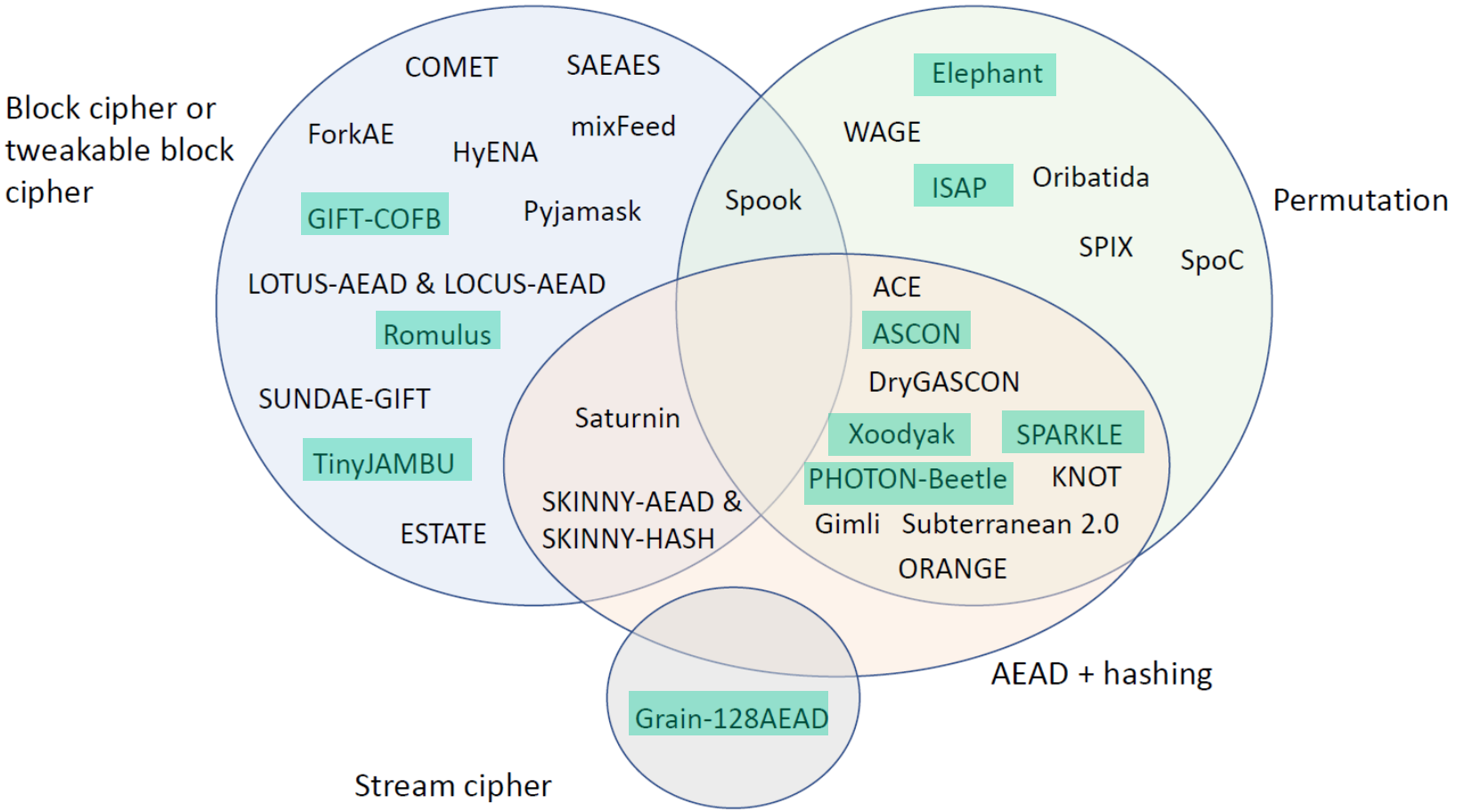
✓ Authenticated encryption with associated data (AEAD)
✓ Hash functions (option)

Round 2 candidates of LWC



Half of them are perm-based

Round 3 candidates of LWC



6 out of 10 are perm-based

Summary

- Permutations
  - New primitive
  - More flexible modes than with block ciphers
- Permutation-based keyed modes
  - Efficiency can be boosted
    - Bigger rate
    - Fewer rounds
- Trends
  - Design various permutations with different goal in mind