05

SHA3

Security strength of the SHA-3 function

- ✓ Characteristics of the hash function:

Variable Size ⊕ **Collision Resistant** ⊕ **Preimage Resistant** ⊕ **Second Preimage Resistant** ⊕ **Pseudo-Randomness**

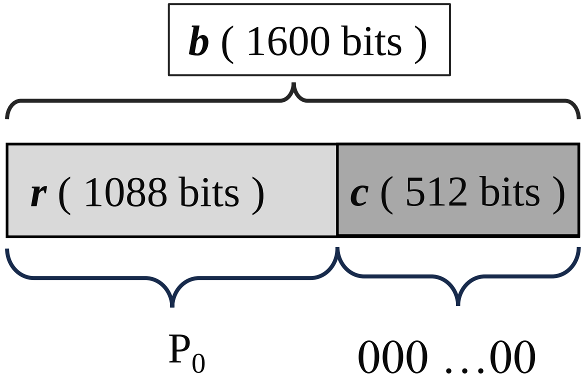
- ✓ Security strengths of the hash function:

Function	SHA3 Primitive	Output Size	Security Strengths in Bits		
			Collision	Preimage	2nd Preimage
Cryptographic Hash Function	SHA3-224	224	112	224	224
	SHA3-256	256	128	256	256
	SHA3-384	384	192	384	384
	SHA3-512	512	256	512	512
Extendable-Output Function	SHAKE128	d	min(d/2, 128)	≥ min(d, 128)	min(d, 128)
	SHAKE256	d	min(d/2, 256)	≥ min(d, 256)	min(d, 256)

Sponge Construction

- ✓ Sponge Construction: This structure is composed of absorbing phases and squeezing phases
- ✓ Padding: The padding algorithm (pad10*1)
- ✓ Function: The internal function used to process each input block include θ, ρ, π, χ, and ι

Ex: SHAKE-256



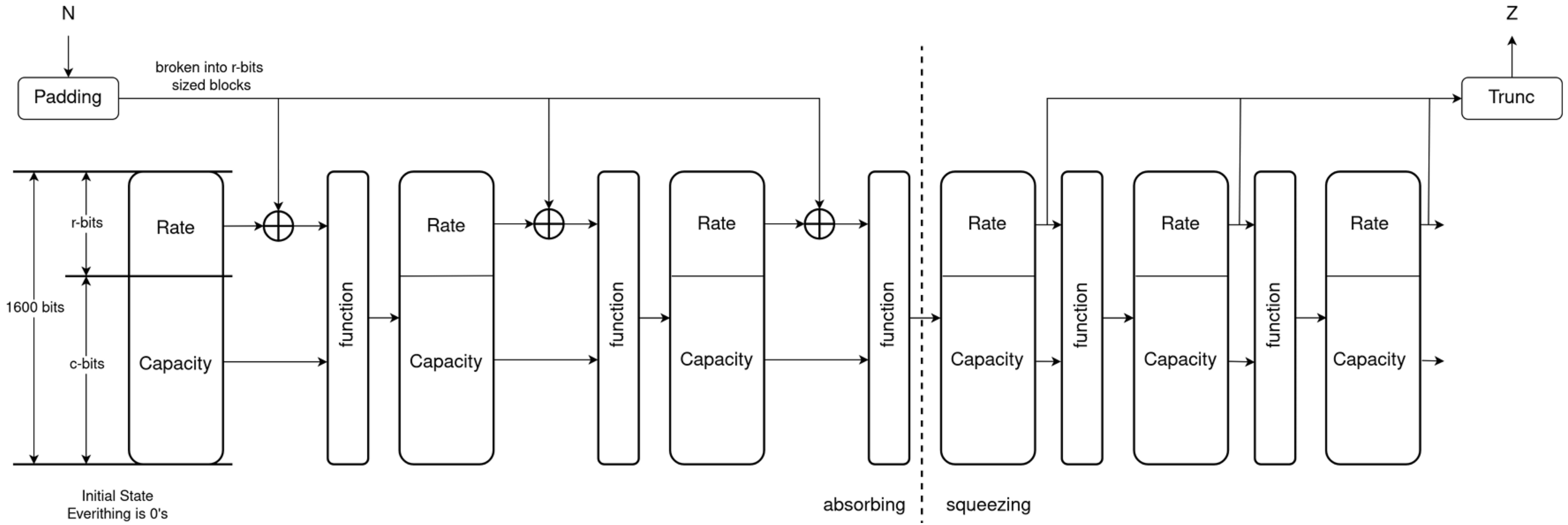
b (1600 bits)

r (1088 bits)

c (512 bits)

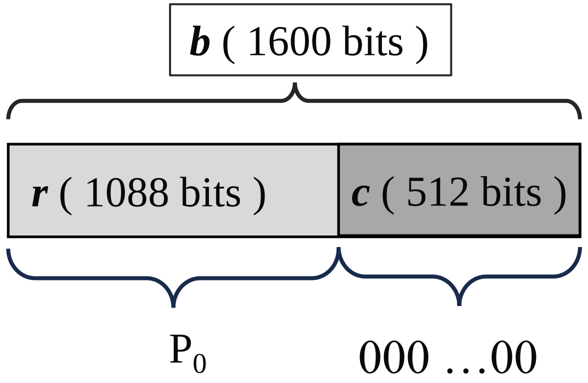
P_0

000 …00



KECCAK-p

Ex: SHAKE-256



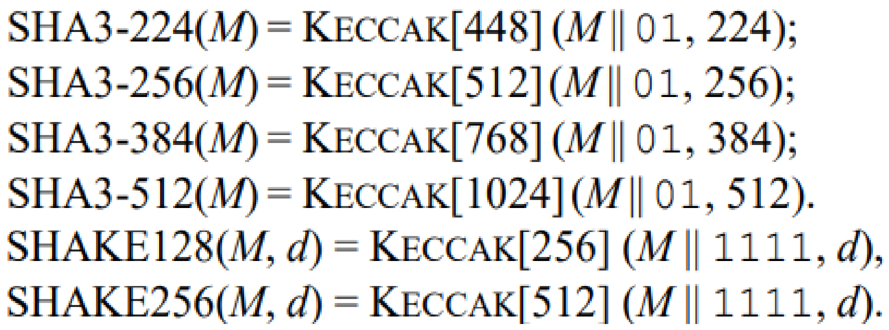
- ✓ For *b*=1600, the KECCAK family is referred to as KECCAK[c]:

  KECCAK[c]=SPONGE[KECCAK−p[1600, 24], pad10∗1, 1600−c]

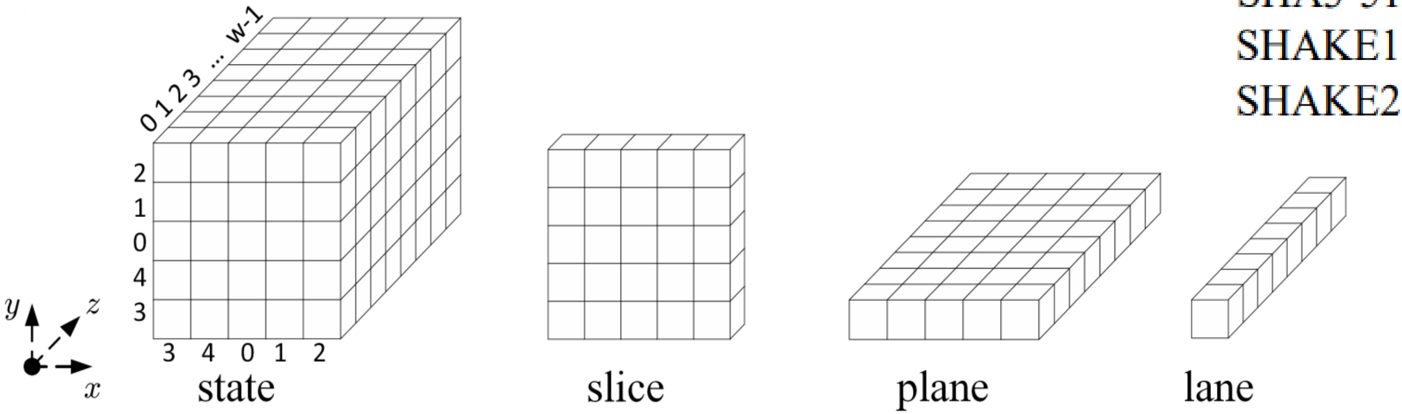
- ✓ Given an input bit string N and an output length d:

  KECCAK[c](M, d)=SPONGE[KECCAK−p[1600, 24], pad10∗1, 1600−c](M, d)

b	**25**	**50**	**100**	**200**	**400**	**800**	**1600**
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6



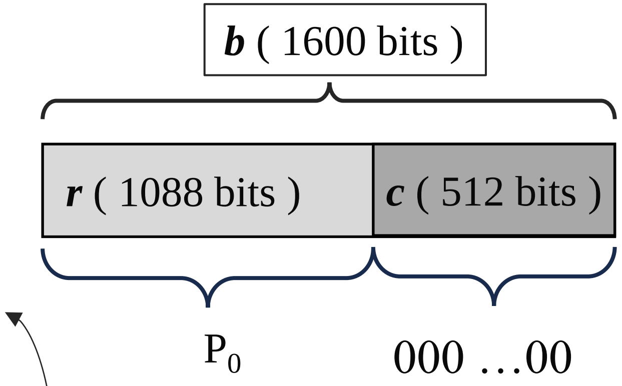
SHA3-224(*M*) = KECCAK[448] (*M* || 01, 224);
SHA3-256(*M*) = KECCAK[512] (*M* || 01, 256);
SHA3-384(*M*) = KECCAK[768] (*M* || 01, 384);
SHA3-512(*M*) = KECCAK[1024] (*M* || 01, 512).
SHAKE128(*M*, *d*) = KECCAK[256] (*M* || 1111, *d*),
SHAKE256(*M*, *d*) = KECCAK[512] (*M* || 1111, *d*).



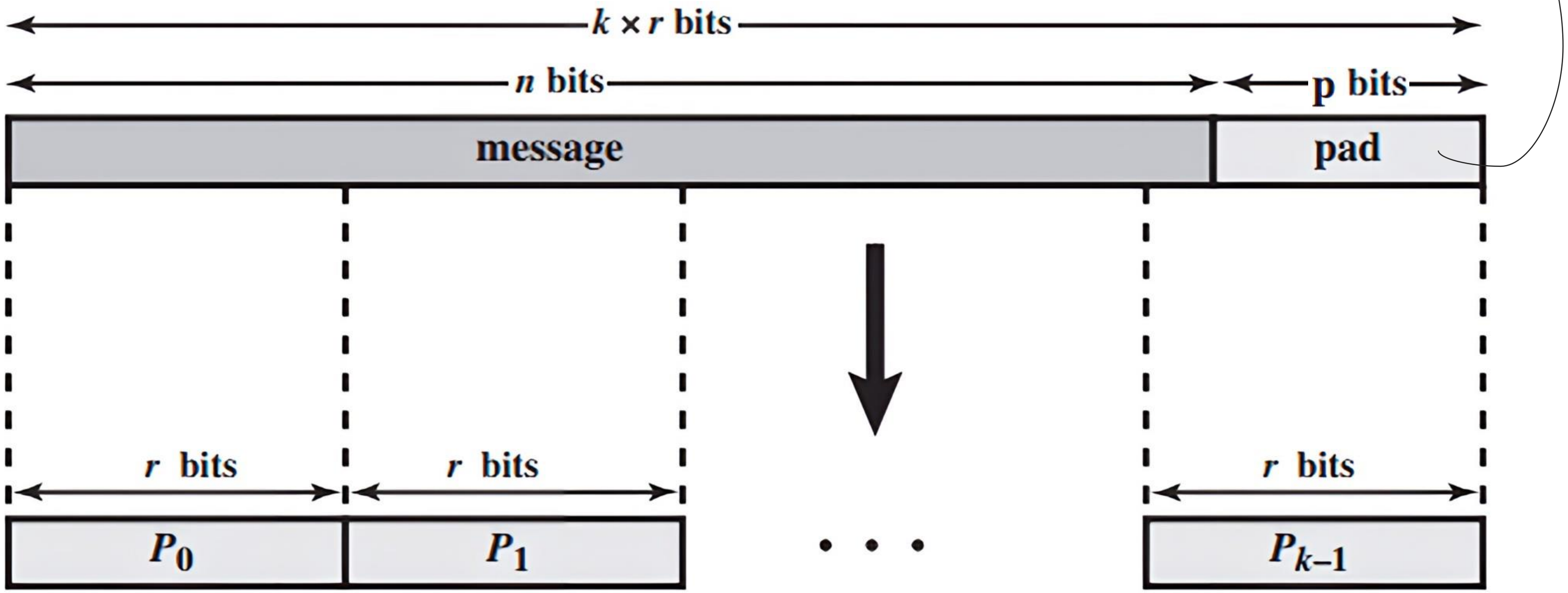
Padding

- ✓ **Simple padding :** The filling method is to pad10(0*)
- ✓ **Multi-rate padding :** The filling method is to pad10(0*)1
- ✓ The padding rule for KECCAK uses **multi-rate padding**

Ex: SHAKE-256



100…0001



Ex: SHAKE-256

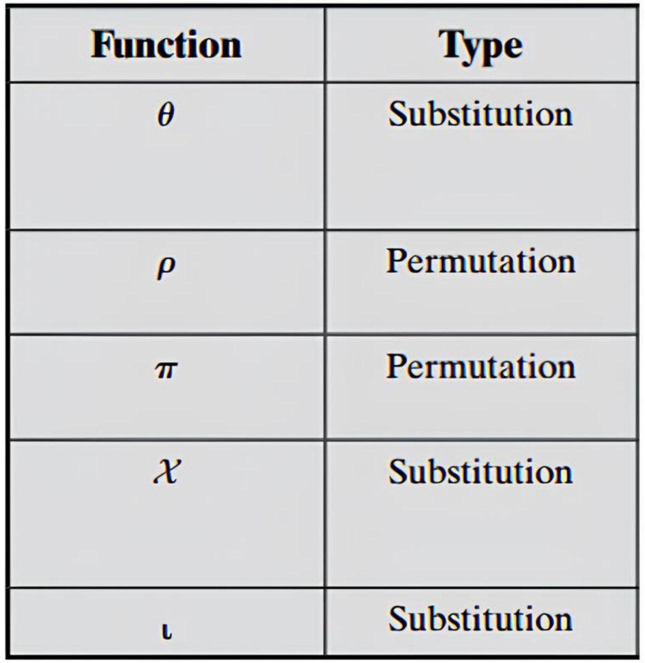
r = 1088 bits

n = 10496 bits

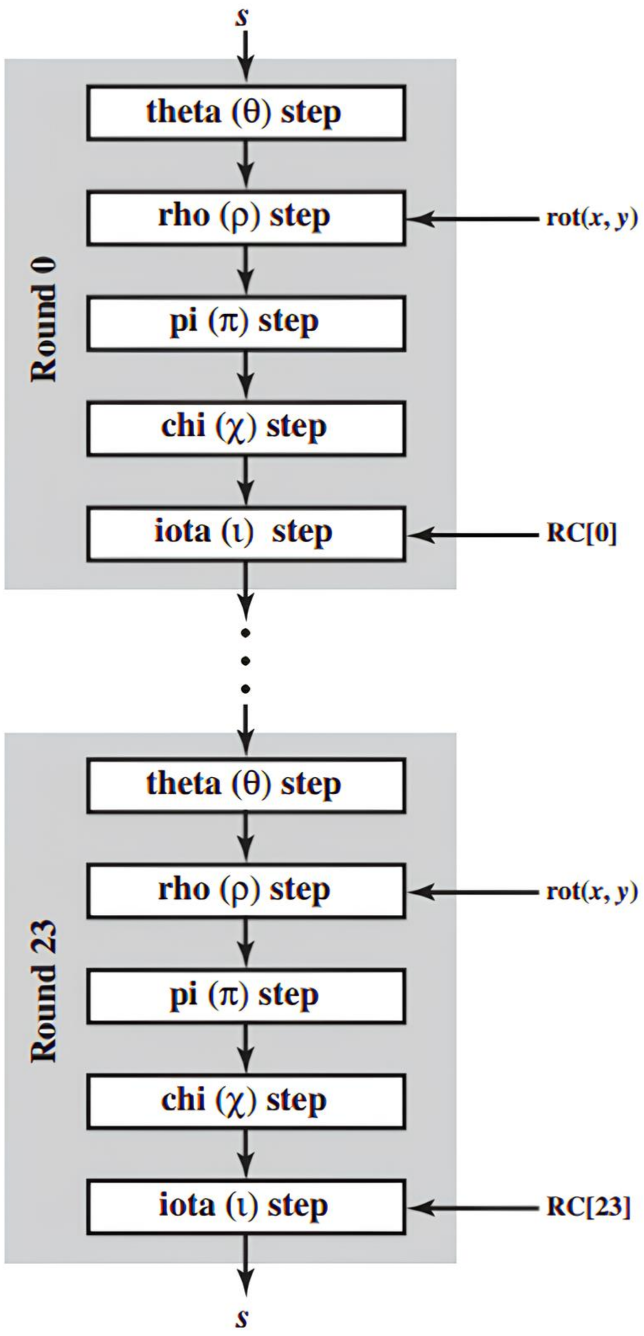
k = $\lceil 10496 / 1088 \rceil$ = 10

p = 10496 mod 1088 = 704

Function

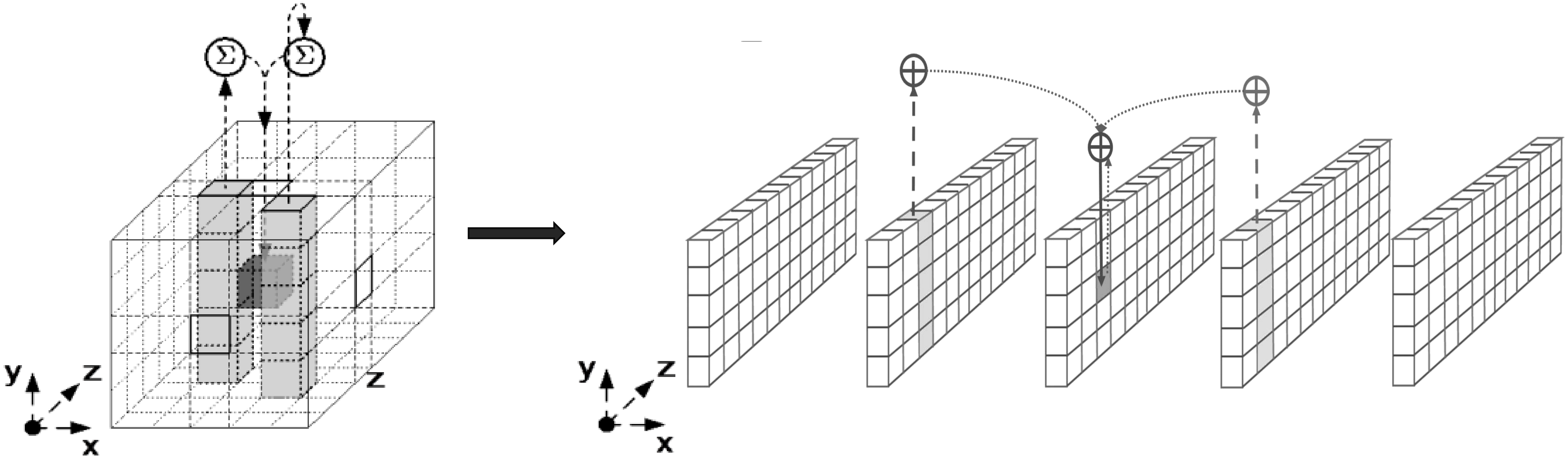


Function	Type
θ	Substitution
ρ	Permutation
π	Permutation
χ	Substitution
ι	Substitution



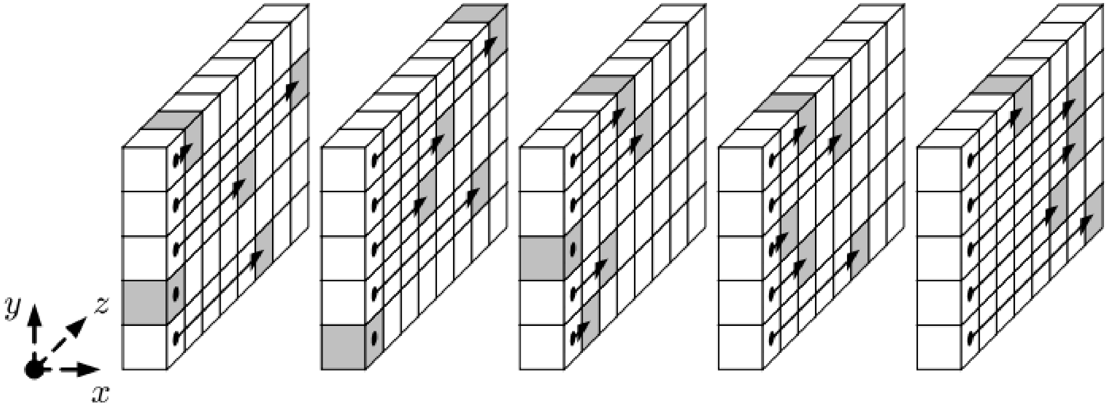
Function - Theta **θ**

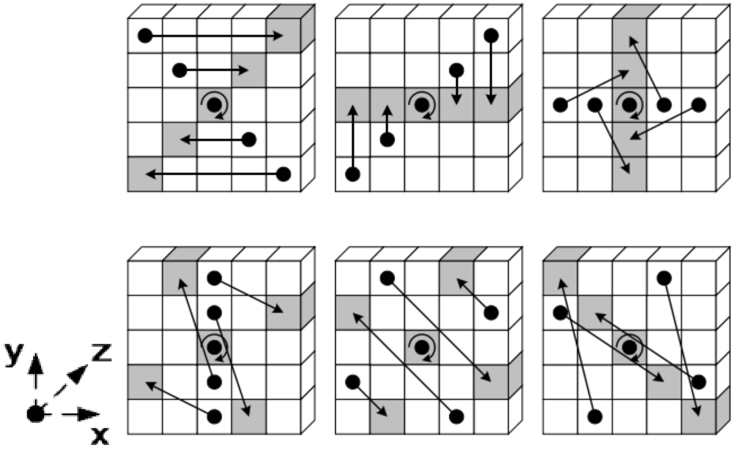
- ✓ $C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4] \quad \forall\ x \text{ in } 0\ldots4$
- ✓ $D[x] = C[x - 1] \oplus \mathrm{ROT}(C[x + 1], 1) \quad \forall\ x \text{ in } 0\ldots4$
- ✓ $A[x, y] = A[x, y] \oplus D[x] \quad \forall\ (x, y) \text{ in } (0\ldots4, 0\ldots4)$



Function - Rho $\boldsymbol{\rho}$ and Pi $\boldsymbol{\pi}$

✓ B[y, 2x + 3y] = ROT(A[x, y], r[x, y]) ∀ (x, y) in (0...4, 0...4)

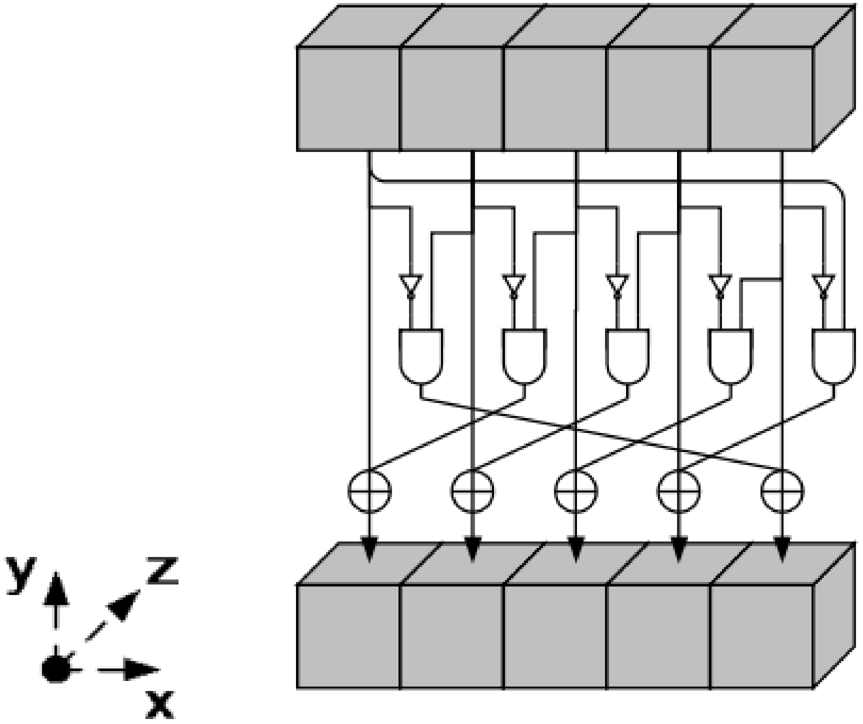




	x = 3	x = 4	x = 0	x = 1	x = 2
y = 2	25	39	3	10	43
y = 1	55	20	36	44	6
y = 0	28	27	0	1	62
y = 4	56	14	18	2	61
y = 3	21	8	41	45	15

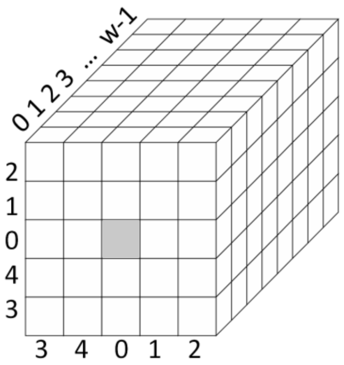
Function - Chi χ

- ✓ A[x, y] = B[x, y] ⊕ ((NOT B[x + 1, y]) AND B[x + 2, y]),   ∀ (x, y) in (0...4, 0...4)



Function - Iota ι

- ✓ The round constant for each round is generated by a linear feedback shift register (LFSR)
- ✓ $A[0, 0] = A[0, 0] \oplus RC$
- ✓ The bits corresponding to 63, 31, 15, 7, 3, 1, and 0 are extracted



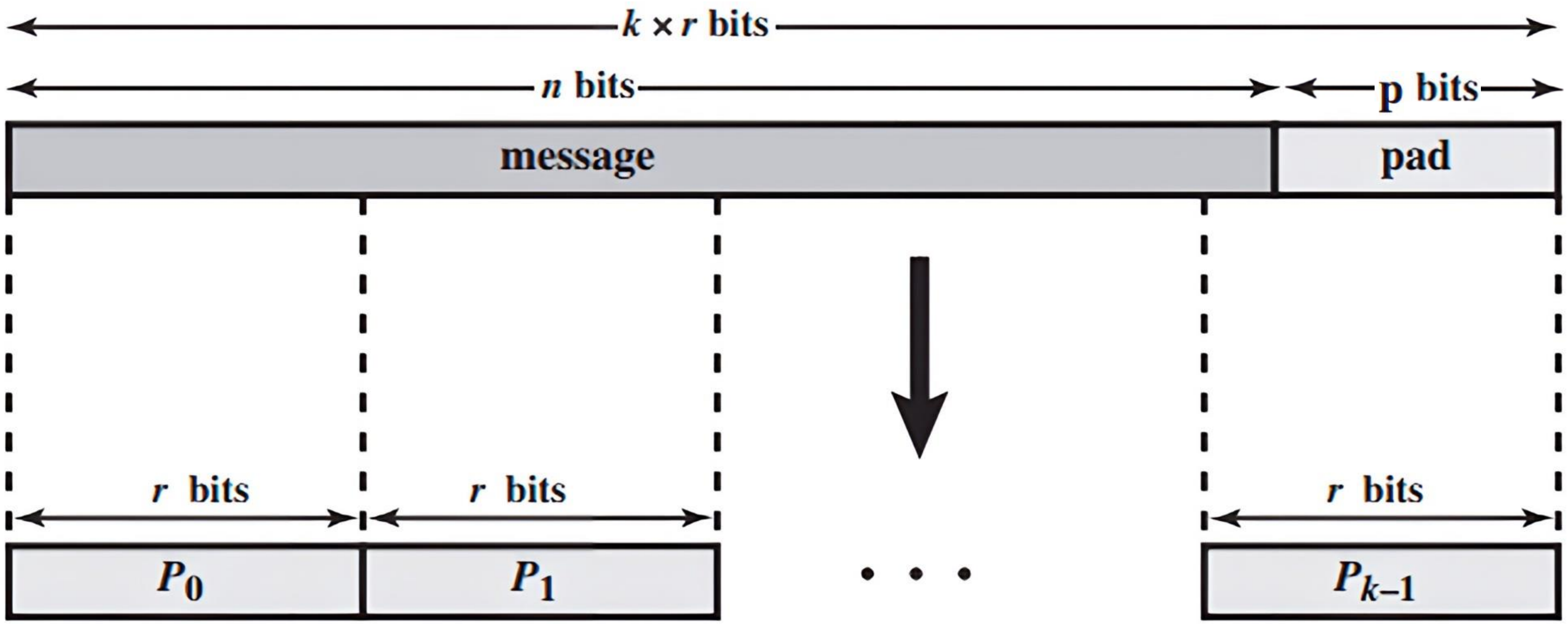
Round	Constant	Round	Constant
0	0000_0000_0000_0001	12	0000_0000_8000_808B
1	0000_0000_0000_8082	13	8000_0000_0000_008B
2	8000_0000_0000_808A	14	8000_0000_0000_8089
3	8000_0000_8000_0000	15	8000_0000_0000_8003
4	0000_0000_0000_808B	16	8000_0000_0000_8002
5	0000_0000_8000_0001	17	8000_0000_0000_0080
6	8000_0000_8000_8081	18	0000_0000_0000_800A
7	8000_0000_0000_8009	19	8000_0000_8000_000A
8	0000_0000_0000_008A	20	8000_0000_8000_8081
9	0000_0000_0000_0088	21	8000_0000_0000_8080
10	0000_0000_8000_8009	22	0000_0000_8000_0001
11	0000_0000_8000_000A	23	8000_0000_8000_8008

Ex.

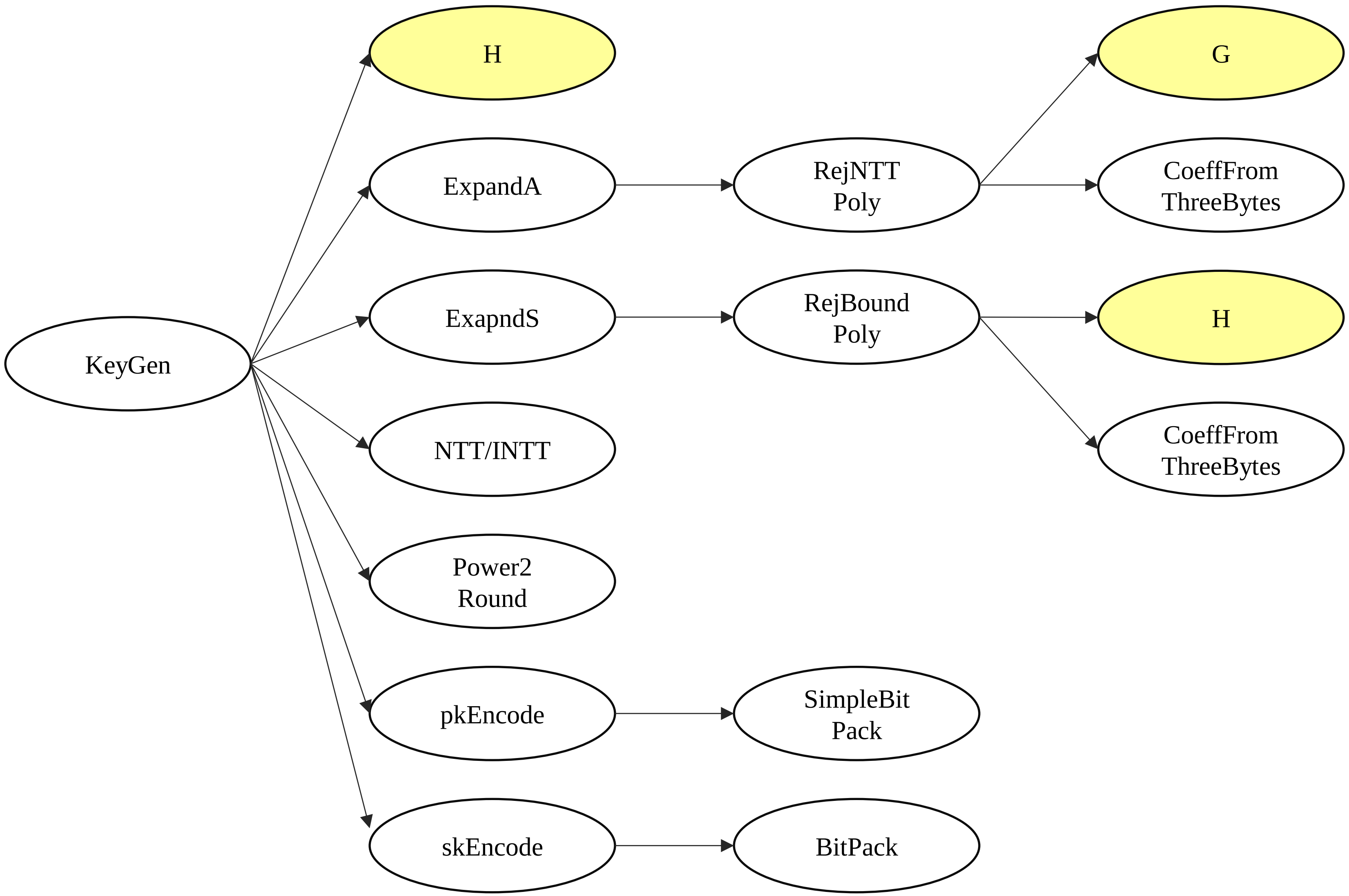
```
rc[0] = i[0] | i[4] | i[5] | i[6] | i[7] | i[10] |
        i[12] | i[13] | i[14] | i[15] | i[20] | i[22];
```

KECCAK-p PERMUTATIONS

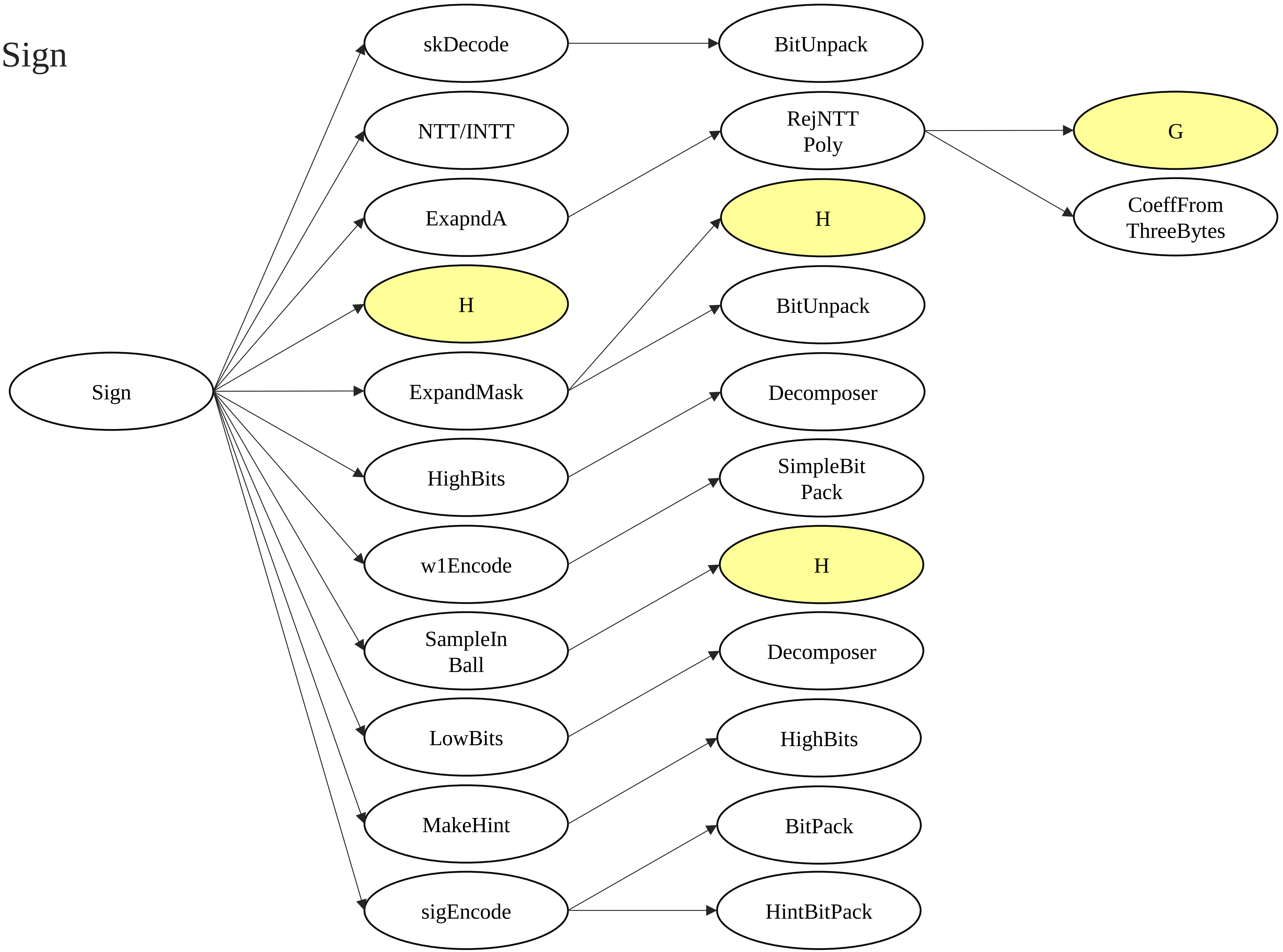
SHA3 primitives	MLDSA primitives	Padding	Size in bits		
			r	c	output length
SHAKE128	G	M ‖ 0x1f ‖ 0x00 … ‖ 0x80	1344	256	unlimited
SHAKE256	H	M ‖ 0x1f ‖ 0x00 … ‖ 0x80	1088	512	unlimited



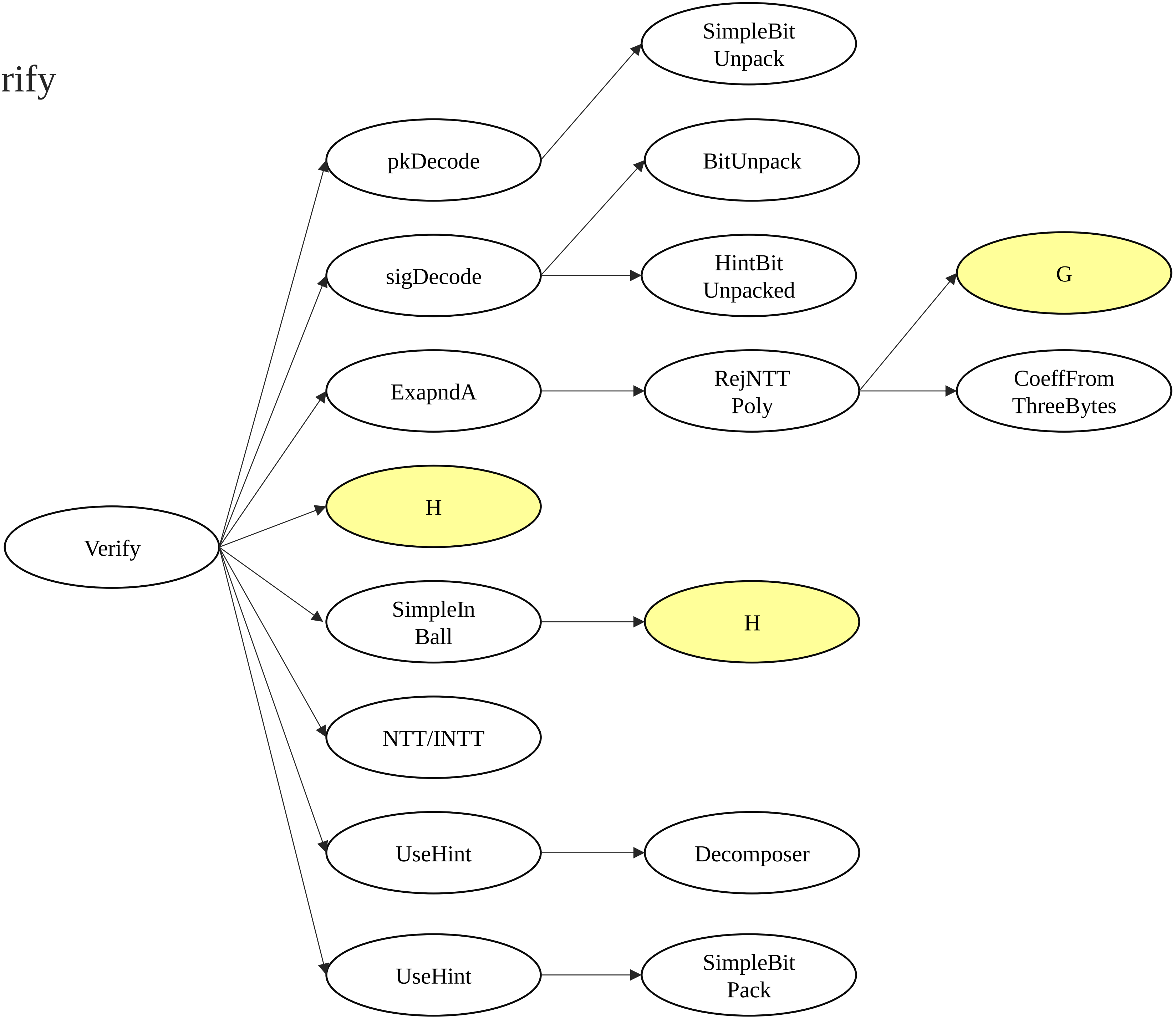
KeyGen


KeyGen
H
ExpandA
ExapndS
NTT/INTT
Power2 Round
pkEncode
skEncode
RejNTT Poly
RejBound Poly
SimpleBit Pack
BitPack
G
CoeffFrom ThreeBytes
H
CoeffFrom ThreeBytes

Sign


Sign
skDecode
BitUnpack
NTT/INTT
ExapndA
RejNTT Poly
G
CoeffFrom ThreeBytes
H
H
BitUnpack
ExpandMask
Decomposer
HighBits
SimpleBit Pack
w1Encode
H
SampleIn Ball
Decomposer
LowBits
HighBits
MakeHint
BitPack
sigEncode
HintBitPack

Verify


SimpleBit Unpack
pkDecode
BitUnpack
sigDecode
HintBit Unpacked
G
ExapndA
RejNTT Poly
CoeffFrom ThreeBytes
H
Verify
SimpleIn Ball
H
NTT/INTT
UseHint
Decomposer
UseHint
SimpleBit Pack

MLDSA SHA3

Main algorithm	SHA3 primitives	Input Size	Output Size
KeyGen	H_SeedGen	(32 + 2)*8 = 272	128*8 = 1024
	G_ExapndA	34*8 = 272	894*8 = 7152
	H_ExpandS	(64 + 2) * 8 = 528	481 * 8 =3848
	H_tr	1312 * 8 = 10496	64 * 8 = 512
Sign	G_ExapndA	34*8 = 272	894*8 = 7152
	H_μ	(64 +(2+n+m))*8	64 * 8 =512
	H_ρ''	(32 + 32 + 64) *8 = 1024	64 * 8 =512
	H_ExpandMask	(64 + 2)*8 = 528	32 * 18 * 8 = 4608
	H_C_tilde	(64 + 768)*8 = 6656	128 / 4 * 8 = 256
	H_SampleInball	128 / 4 * 8 = 256	221 * 8 = 1768
Verify	G_ExapndA	34*8 = 272	894*8 = 7152
	H_tr	1312 * 8 = 10496	64 * 8 = 512
	H_μ	(64 + 283)*8 = 2776	64 * 8 =512
	H_SampleInball	128 / 4 * 8 = 256	221 * 8 = 1768

MLDSA SHA3

SHA3 primitives	Input Size	Output Size	mode	is_last cycle $\lfloor \text{Input Size}/64 \rfloor + 1$	byte_num $\frac{\text{Input Size mod 64}}{8}$	Squeeze time $\left\lceil \frac{Output\ Size}{mode\ size} \right\rceil - 1$
H_SeedGen	272	1024	1	5	2	0
G_ExapndA	272	7152	0	5	2	6
H_ExpandS	528	3848	1	9	2	4
H_tr	10496	512	1	164	0	0
H_μ	2776	512	1	44	3	0
H_ρ''	1024	512	1	17	0	0
H_ExpandMask	528	4608	1	9	2	4
H_C_tilde	6656	256	1	105	0	0
H_SampleInball	256	1768	0	5	0	1

MLDSA SHA3

SHA3 primitives	Primitives sign	MLDSA_mode	Stage	mode	Input side		Output side		
H(ξ+x04+x04)	0	KeyGen	1	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM
Gen(s1)	1	KeyGen	2	H	Rho_prime MEM	s1_gen_index	s1 MEM		
Gen(s2)	2	KeyGen	3	H	Rho_prime MEM	s2_gen_index	s2 MEM		
Gen(A)	3	KeyGen	4	G	Rho MEM	A_gen_index	A MEM		
H($\rho \parallel$ t1) = tr	4	KeyGen	7	H	Rho MEM	t1 MEM	tr MEM		
H($tr \parallel M'$)_1	5	SignGen	1	H	MLDSA_in_1		u MEM		
H($K \parallel rnd \parallel$ u)	6	SignGen	2	H	MLDSA_in_1		p" MEM		
Gen(y)	7	SignGen	3	H	p" MEM	y_gen_index	y MEM		
Gen(A)	3	SignGen	4,5	G	Rho MEM		A MEM		
H($\mu \parallel \mathbf{w}1$)	8	SignGen	9	H	u MEM	w MEM	c_tilde MEM		
Gen(c)_1	9	SignGen	10	H	c_tilde MEM		c MEM		
Gen(y)	7	SignGen	15	H	p" MEM	y_gen_index	y MEM		
Gen(c)_2	10	SignVer	1	H	MLDSA_in_1		c MEM		
Gen(A)	3	SignVer	2,3	G	Rho MEM	A_gen_index	A MEM		
H(pk)	11	SignVer	4	H	MLDSA_in_1		tr MEM		
H($tr \parallel M'$)_2	12	SignVer	5	H	tr MEM	MLDSA_in_1	u MEM		
H($\mu \parallel \mathbf{w}1$)	8	SignVer	8	H	u MEM	w MEM	c_tilde MEM		

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
H(ξ+x04+x04)	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen(s1)	1	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen(s2)	2	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen(A)	3	G	Rho MEM	A_gen_index	A MEM			010	5
H(pk)_1	4	H	Rho MEM	t1 MEM	tr MEM			000	165
H($tr \parallel M'$)_1	5	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$
H($K \parallel rnd \parallel$ u)	6	H	MLDSA_in_1		p" MEM			000	17
Gen(y)	7	H	p" MEM	y_gen_index	y MEM			010	9
H($\mu \parallel \mathbf{w}1$)	8	H	u MEM	w1 MEM	c_tilde MEM			000	105
Gen(c)_1	9	H	c_tilde MEM		c MEM			000	5
Gen(c)_2	10	H	MLDSA_in_1		c MEM			000	5
H(pk)_2	11	H	MLDSA_in_1		tr MEM			000	165
H($tr \parallel M'$)_2	12	H	tr MEM	MLDSA_in_1	u MEM			011	44

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
H(ξ+x04+x04)	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen(A)	1	G	Rho MEM	A_gen_index	A MEM			010	5
Gen(s1)	2	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen(s2)	3	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen(y)	4	H	p" MEM	y_gen_index	y MEM			010	9
H($\mu \| \mathbf{w}1$)	5	H	u MEM	w MEM	c_tilde MEM			000	105
H(pk)_1	6	H	Rho MEM	t1 MEM	tr MEM			000	165
H(pk)_2	7	H	MLDSA_in_1		tr MEM			000	165
Gen(c)_1	8	H	c_tilde MEM		c MEM			000	5
Gen(c)_2	9	H	MLDSA_in_1		c MEM			000	5
H($K \| rnd \|$ u)	10	H	MLDSA_in_1		p" MEM			000	17
H($tr \| M'$)_1	11	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil\frac{64+2+n+m}{8}\right\rceil$
H($tr \| M'$)_2	12	H	tr MEM	MLDSA_in_1	u MEM			(64+2+n+m) mod 8	$\left\lceil\frac{64+2+n+m}{8}\right\rceil$

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
Gen_Seed	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen_A	1	G	Rho MEM	A_gen_index	A MEM			010	5
Gen_s1	2	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen_s2	3	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen_y	4	H	p" MEM	y_gen_index	y MEM			010	9
H_u_w1	5	H	u MEM	w MEM	c_tilde MEM			000	105
H_pk_1	6	H	Rho MEM	t1 MEM	tr MEM			000	165
H_pk_2	7	H	MLDSA_in_1		tr MEM			000	165
Gen_c_1	8	H	c_tilde MEM		c MEM			000	5
Gen_c_2	9	H	MLDSA_in_1		c MEM			000	5
H_K_rnd_u	10	H	MLDSA_in_1		p" MEM			000	17
H_tr_M_1	11	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$
H_tr_M_2	12	H	tr MEM	MLDSA_in_1	u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$

MLDSA SHA3

Data source	keccak_in_sel	mem_sel_1	mem_sel_2	index_sel	in_seed_sel
Rho MEM	0	0	X	X	X
Rho_prime MEM	0	1	X	X	X
Rho_prime_prime MEM	0	2	X	X	X
u MEM	0	3	x	X	X
w MEM	1	X	0	X	X
t1 MEM	1	X	1	X	X
c_tilde MEM	1	X	2	X	X
tr MEM	1	X	3	X	X
A_gen_index	2	X	X	0	X
s1_gen_index	2	X	X	1	X
s2_gen_index	2	X	X	2	X
y_gen_index	2	X	X	3	X
MLDSA_in_1	3	X	X	X	0
seed_gen_index	3	X	X	X	1

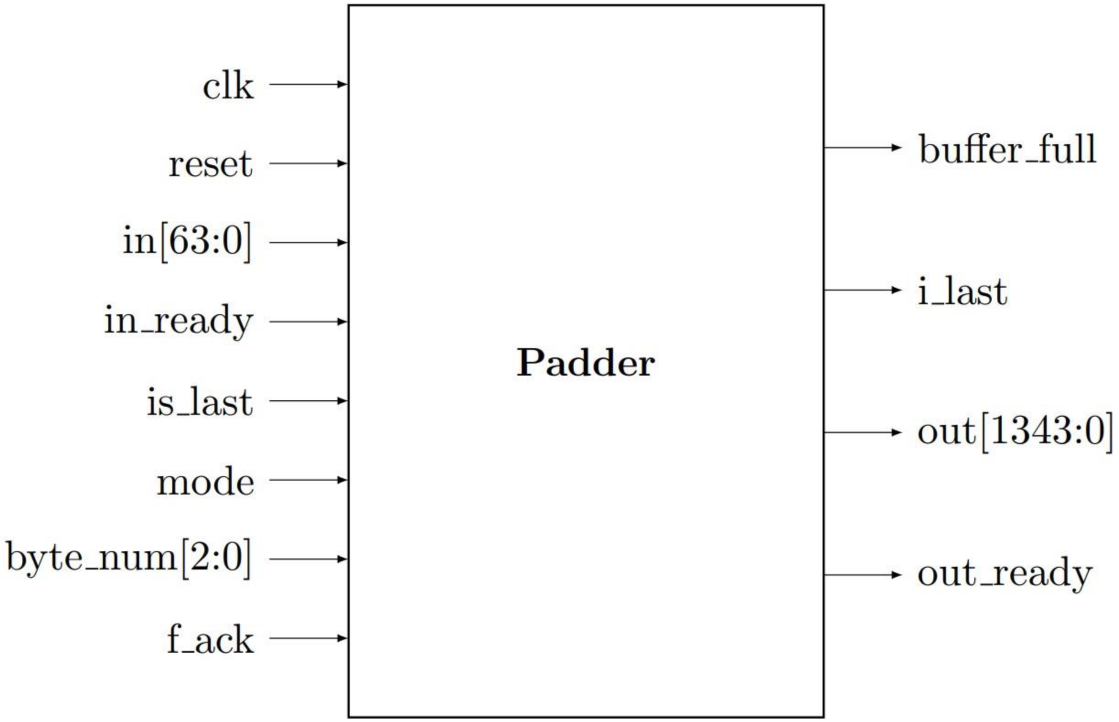
MLDSA SHA3

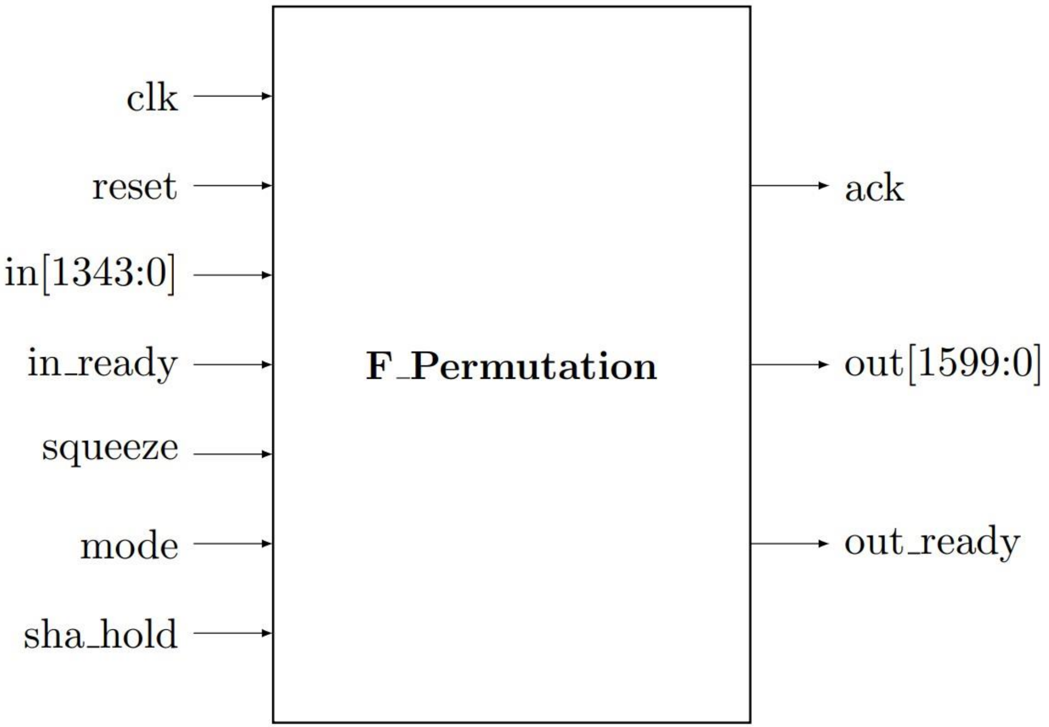
Data source	keccak_in_sel	mem_sel_1	mem_sel_2	index_sel	in_seed_sel
Seed MEM	0	0	X	X	X
u MEM	0	1	x	X	X
w MEM	1	X	0	X	X
t1 MEM	1	X	1	X	X
c_tilde MEM	1	X	2	X	X
tr MEM	1	X	3	X	X
A_gen_index	2	X	X	0	X
s1_gen_index	2	X	X	1	X
s2_gen_index	2	X	X	2	X
y_gen_index	2	X	X	3	X
MLDSA_in_1	3	X	X	X	0
seed_gen_index	3	X	X	X	1

MLDSA SHA3

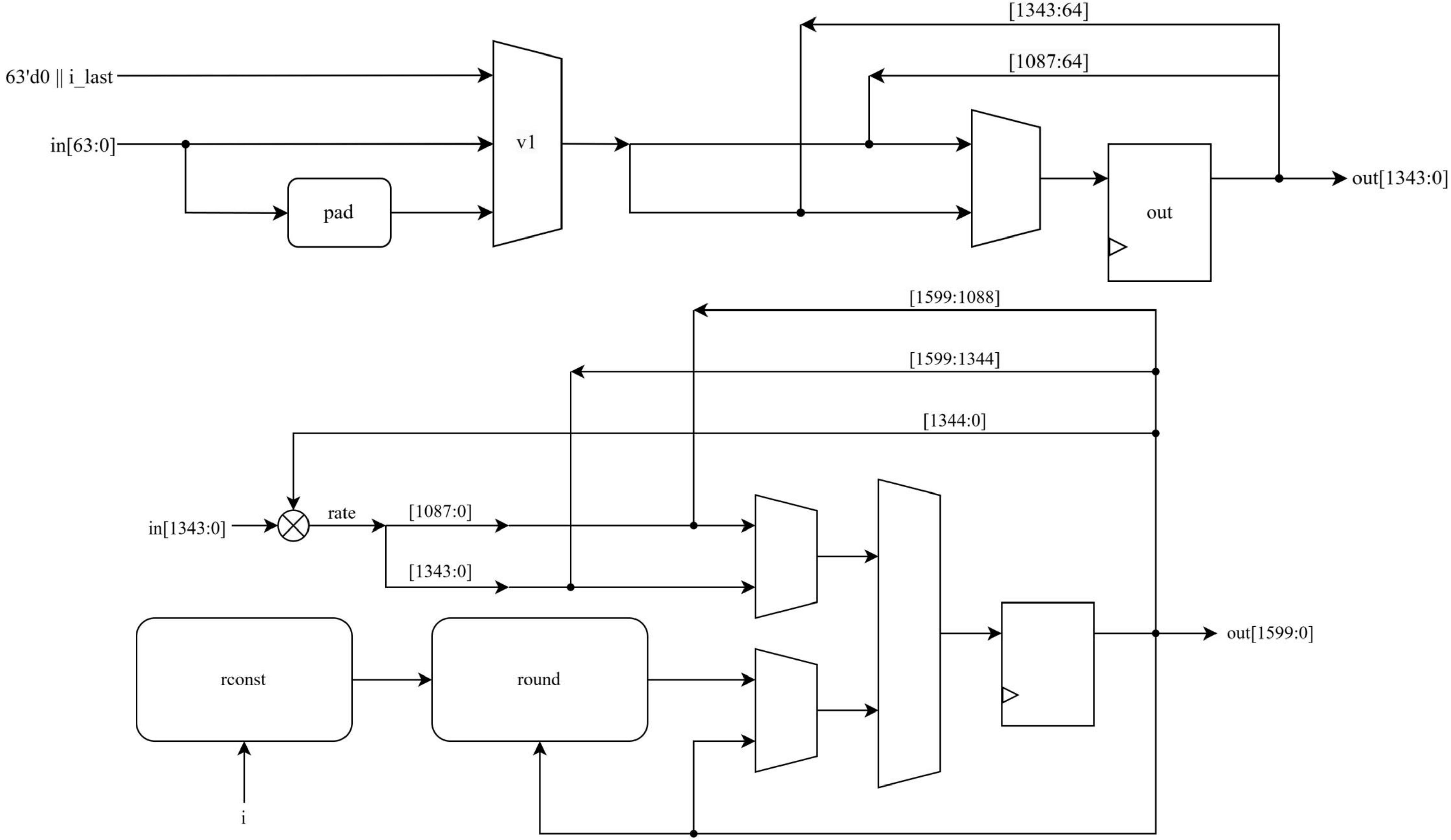
Data source	keccak_in_sel	kk_sub_sel_1	kk_sub_sel_2	kk_sub_sel_3
Seed MEM	0	0	X	X
u MEM	0	1	x	X
w MEM	0	2	X	X
t1 MEM	0	3	X	X
c_tilde MEM	1	X	0	X
tr MEM	1	X	1	X
zero_index	1	X	2	X
seed_gen_index	1	X	3	X
A_gen_index	2	X	X	0
s1_gen_index	2	X	X	1
s2_gen_index	2	X	X	2
y_gen_index	2	X	X	3
MLDSA_in_1	3	X	X	X

MLDSA SHA3 Module

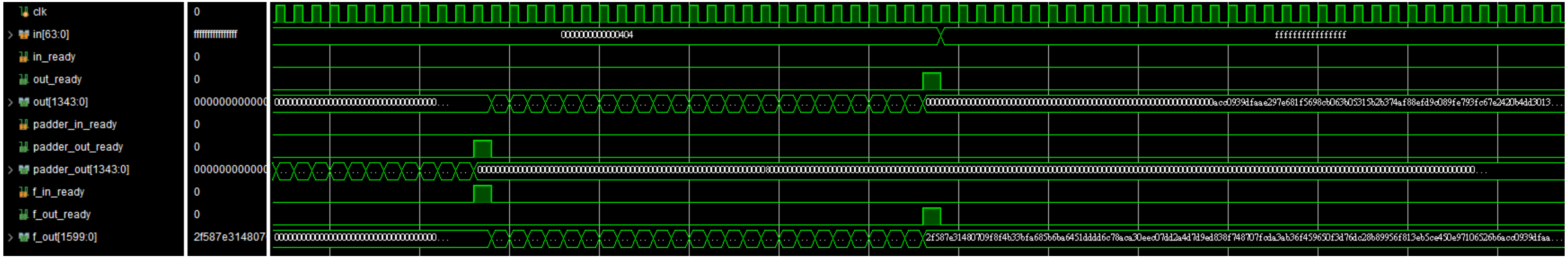


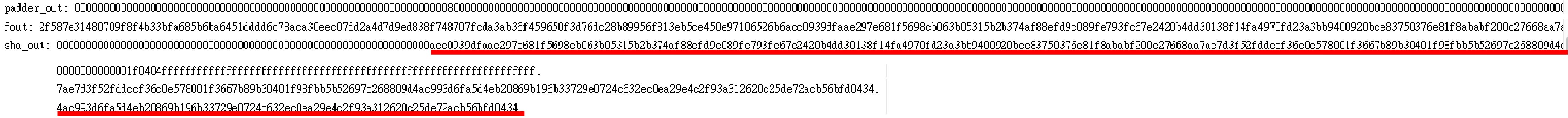


MLDSA SHA3 Block Diagram



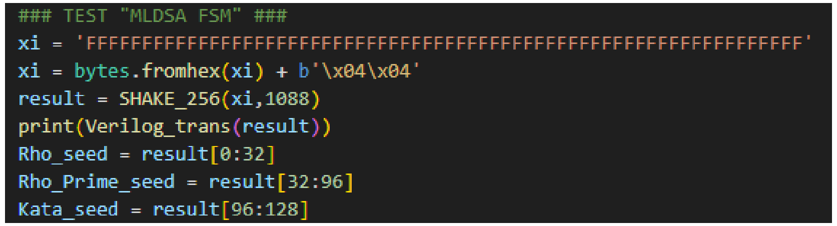
MLDSA SHA3 Block Diagram



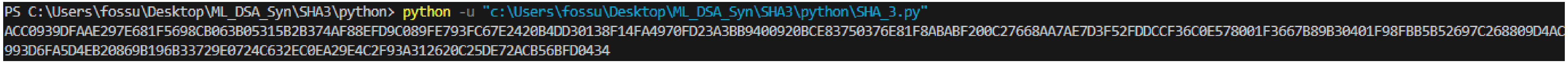


```
padder_out: 00000000000000000000000000000000000000000000000000000000000000008000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
fout: 2f587e31480709f8f4b33bfa685b6ba6451dddd6c78aca30eec07dd2a4d7d9ed838f748707fcda3ab36f459650f3d76dc28b89956f813eb5ce450e97106526b6acc0939dfaae297e681f5698cb063b05315b2b374af88efd9c089fe793fc67e2420b4dd30138f14fa4970fd23a3bb9400920bce83750376e81f8ababf200c27668aa7
sha_out: 0000000000000000000000000000000000000000000000000000000000000000acc0939dfaae297e681f5698cb063b05315b2b374af88efd9c089fe793fc67e2420b4dd30138f14fa4970fd23a3bb9400920bce83750376e81f8ababf200c27668aa7ae7d3f52fddccf36c0e578001f3667b89b30401f98fbb5b52697c268809d4

         0000000000001f0404ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff.
         7ae7d3f52fddccf36c0e578001f3667b89b30401f98fbb5b52697c268809d4ac993d6fa5d4eb20869b196b33729e0724c632ec0ea29e4c2f93a312620c25de72acb56bfd0434.
         4ac993d6fa5d4eb20869b196b33729e0724c632ec0ea29e4c2f93a312620c25de72acb56bfd0434.
```



```python
### TEST "MLDSA FSM" ###
xi = 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'
xi = bytes.fromhex(xi) + b'\x04\x04'
result = SHAKE_256(xi,1088)
print(Verilog_trans(result))
Rho_seed = result[0:32]
Rho_Prime_seed = result[32:96]
Kata_seed = result[96:128]
```



```
PS C:\Users\fossu\Desktop\ML_DSA_Syn\SHA3\python> python -u "c:\Users\fossu\Desktop\ML_DSA_Syn\SHA3\python\SHA_3.py"
ACC0939DFAAE297E681F5698CB063B05315B2B374AF88EFD9C089FE793FC67E2420B4DD30138F14FA4970FD23A3BB9400920BCE83750376E81F8ABABF200C27668AA7AE7D3F52FDDCCF36C0E578001F3667B89B30401F98FBB5B52697C268809D4AC
993D6FA5D4EB20869B196B33729E0724C632EC0EA29E4C2F93A312620C25DE72ACB56BFD0434
```