SIMD Instruction Set Extensions for KECCAK with Applications to SHA-3, Keyak and Ketje

Hemendra K. Rawat and Patrick Schaumont

Virginia tech, Blacksburg, USA

{hrawat, schaum}@vt.edu

Motivation

- Secure systems and protocols rely on a suite of cryptographic applications
  - Hashing, MAC, Encryption, PRNG, AEAD etc...
- Traditionally, different algorithms: SHA-1, SHA-2, AES, GMAC, HMAC
- Different Instruction set extensions: AES-NI, SHA and Carry-less multiplication
  - Vector extensions in Intel, ARM, SPARC and PowerPC
- KECCAK is the SHA-3
  - Successor of SHA-2

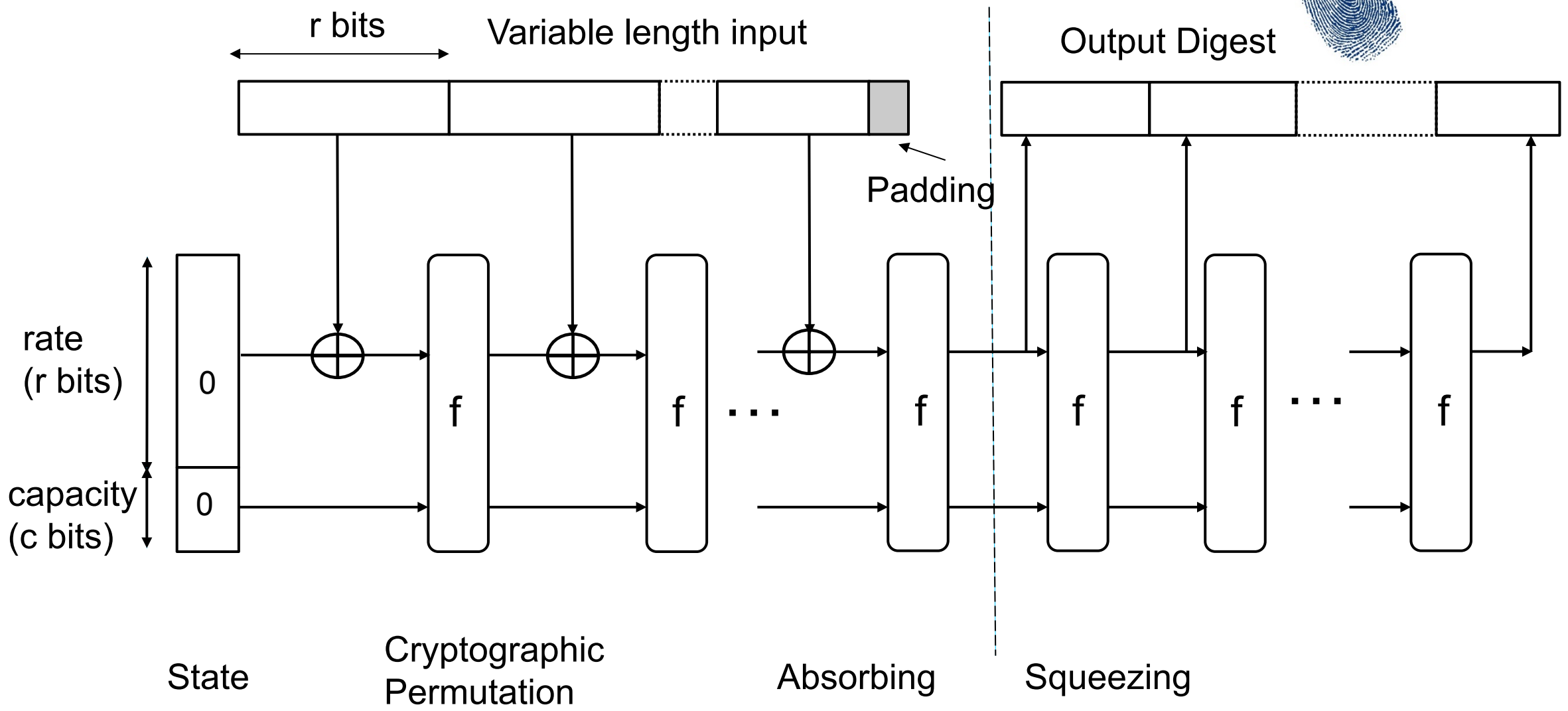
- Subset of the cryptographic primitive family KECCAK SPONGE
- Not just a Hash
- A Flexible Sponge for all symmetric cryptography

"A flexible SIMD Instruction set, for flexible KECCAK Sponge"

Outline

- ✈ KECCAK Background
- ❑ Novelty Claims
- ❑ Design Principles
- ❑ Proposed Instruction Set Extensions
- ❑ Results
  - ❑ Performance
  - ❑ Hardware Overhead
- ❑ Conclusion

KECCAK HASH (SHA-3)



Hashing using KECCAK Sponge Construction

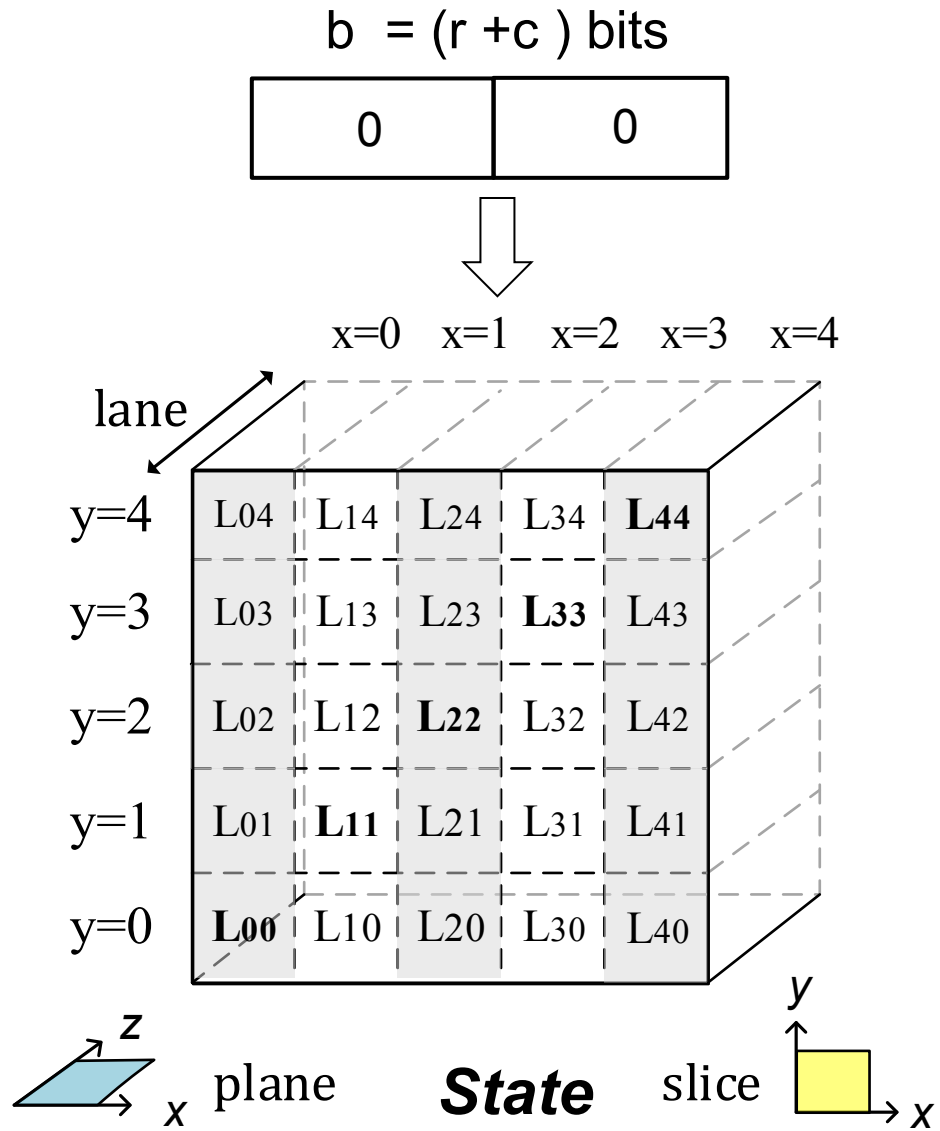
KECCAK-*f* in Nutshell

f

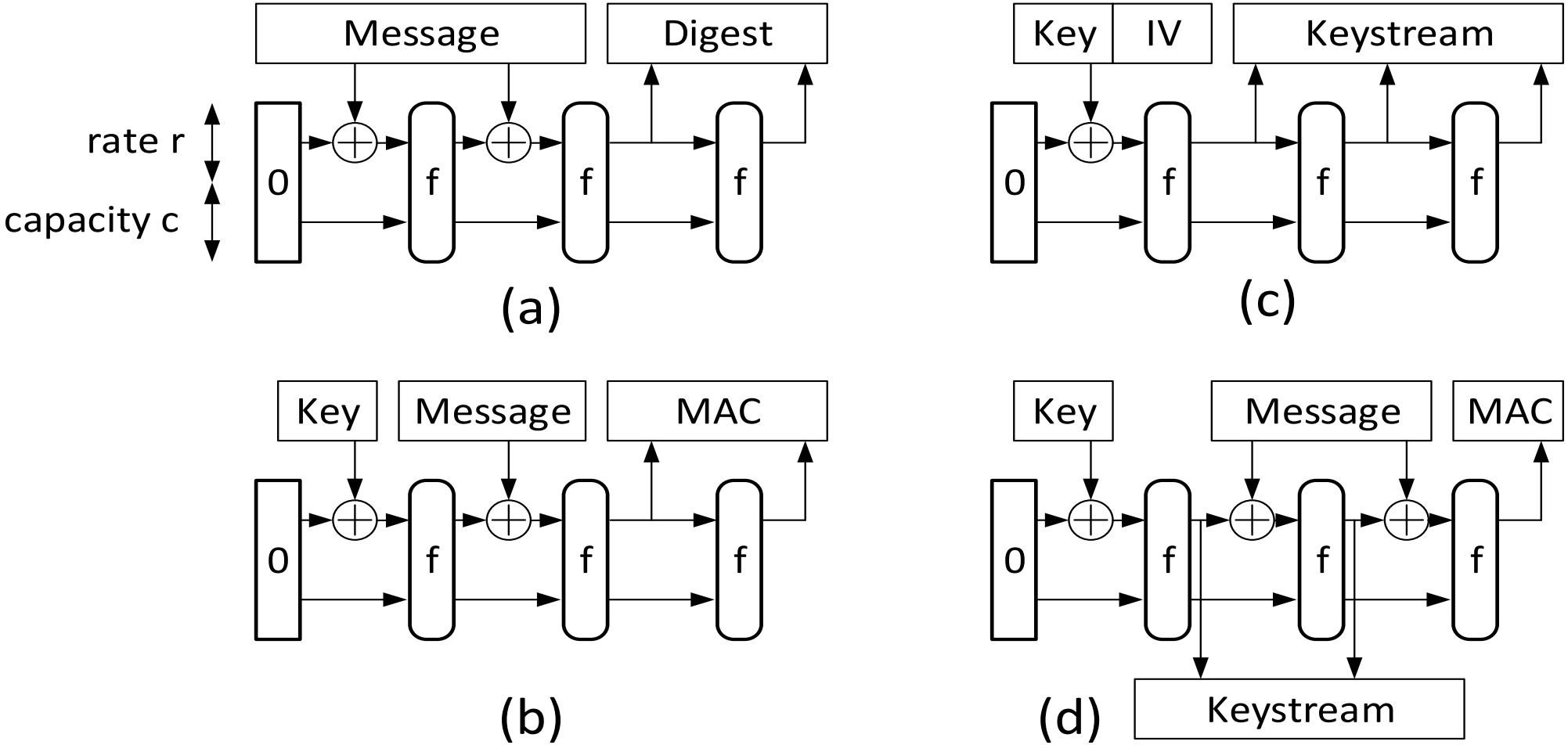
KECCAK-*f[b]* Permutation

```
state [0 : b]
def keccakf[b]:
        for numRounds in range(0, maxRounds):
            theta(state)          # Add column parities
            rho(state)            # Rotate lanes
            pi(state)             # Transpose lanes
            chi(state)            #  Add non-linearity
            iota(state)           # L00 ⊕ Kround
```

b = 1600, 800, 400, 200, 100, 50, 25 bits

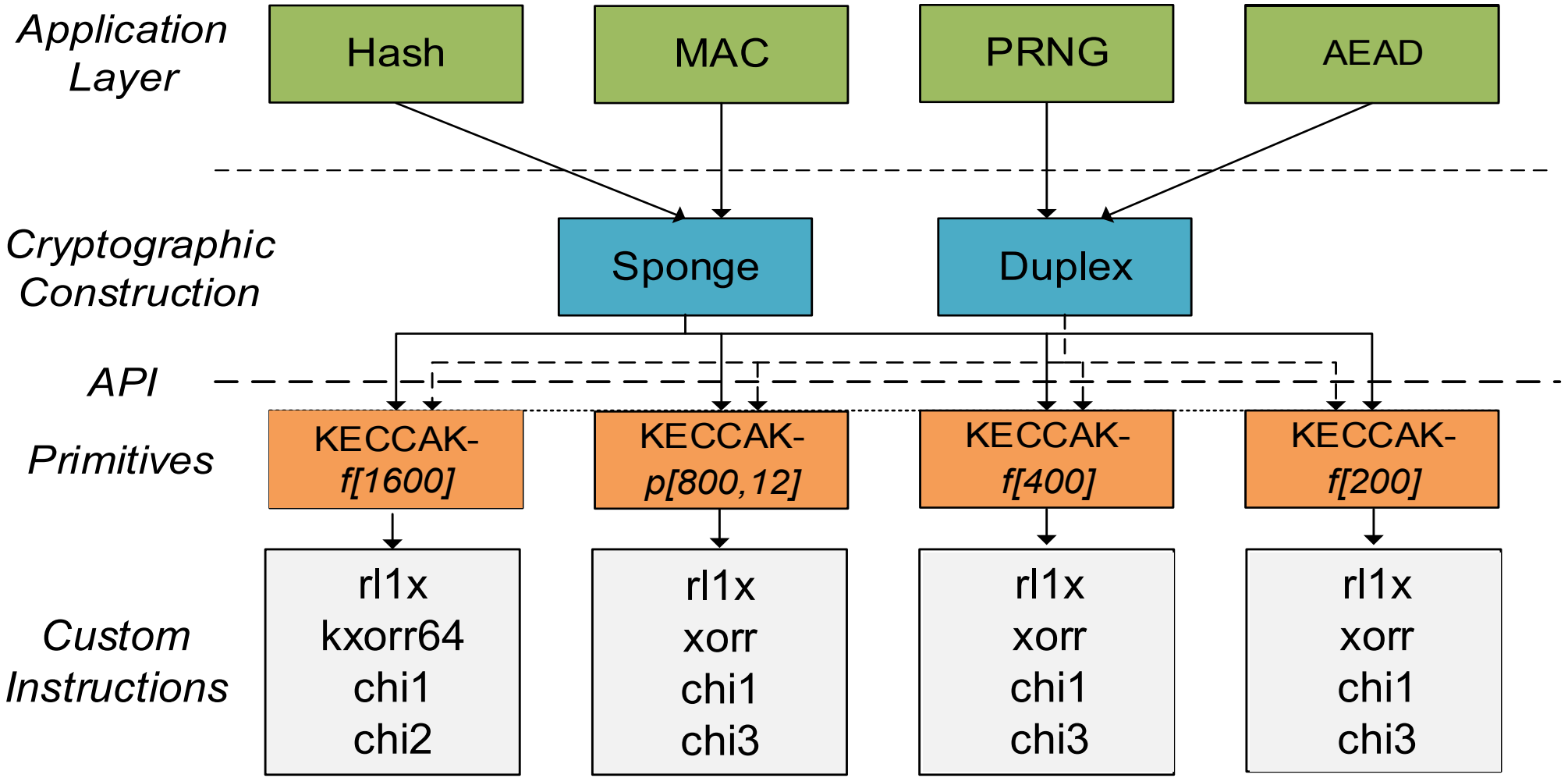


KECCAK Applications



Constructions with sponge: (a) Hash, (b) Message Authentication Code, (c) Keystream Generation. Construction with duplex: (d) Authenticated Encryption

Application Stack



Novelty Claims

Previous work

- Custom-instruction designs for a 16 bit micro-controller for SHA-3 finalists, Constantin et al.
- Integration of 64 bit KECCAK(SHA-3) data path into a 32 bit LEON3, Wang et al.

This work

- Six new custom instructions for 128 bit SIMD unit.
- Multipurpose KECCAK:1600, 800, 400 and 200 bits.
- Five KECCAK algorithms: SHA3-512 hash, LakeKEYAK, RiverKEYAK, KetjeSR and KetjeJR authenticated ciphers.
- Compatible with ARM NEON Instruction set

Design Principles

Portability	▪ Easy integration into different processor architectures ▪ RISC-like instruction format (2 input,1 output operand) ▪ No non-standard architectural features

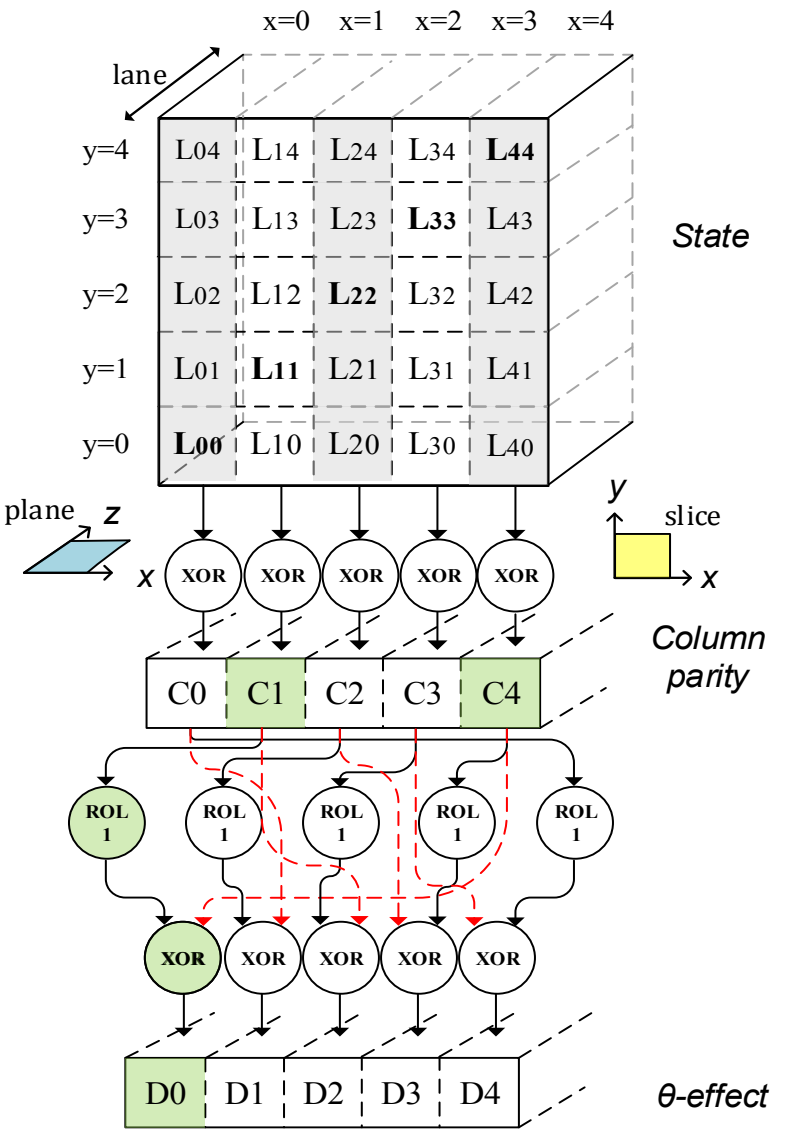
Flexibility	▪ Support for multiple symmetric cryptographic applications
	  ▪ Hashing
	  ▪ MACing
	  ▪ Stream Ciphers
	  ▪ AEAD
	  ▪ PRNG

Simplicity	▪ Small set of instructions ▪ Low hardware Overhead ▪ Simple operations like XOR, AND, NOT and rotations ▪ Short critical path

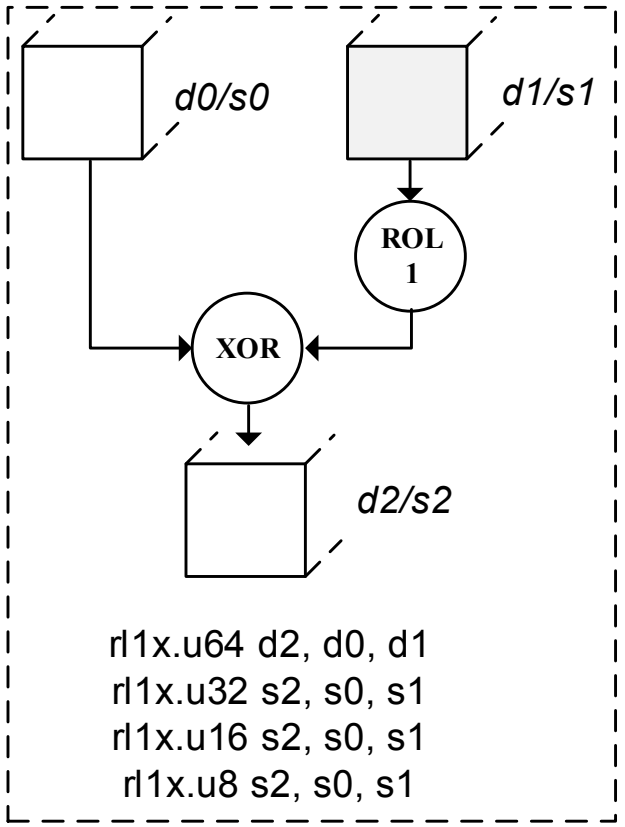
Design Principles

Performance	▪ **Optimize Computation intensive part – KECCAK-*{f,p}* primitives**
	▪ Partition dataflow graph into SIMD-like instruction patterns
	  ▪ 2x128 bit input and 128 bit output
	▪ Minimize the schedule length of the graph
	▪ Optimize instruction shapes and functionality for
	  ▪ High ILP (instruction-level parallelism)
	  ▪ Minimum register spills
	  ▪ Minimum MOV/VEXT for register rearrangements
	  ▪ In-place round computation

Mapping KECCAK to Instructions



Data dependencies of θ-effect



Instruction rl1x

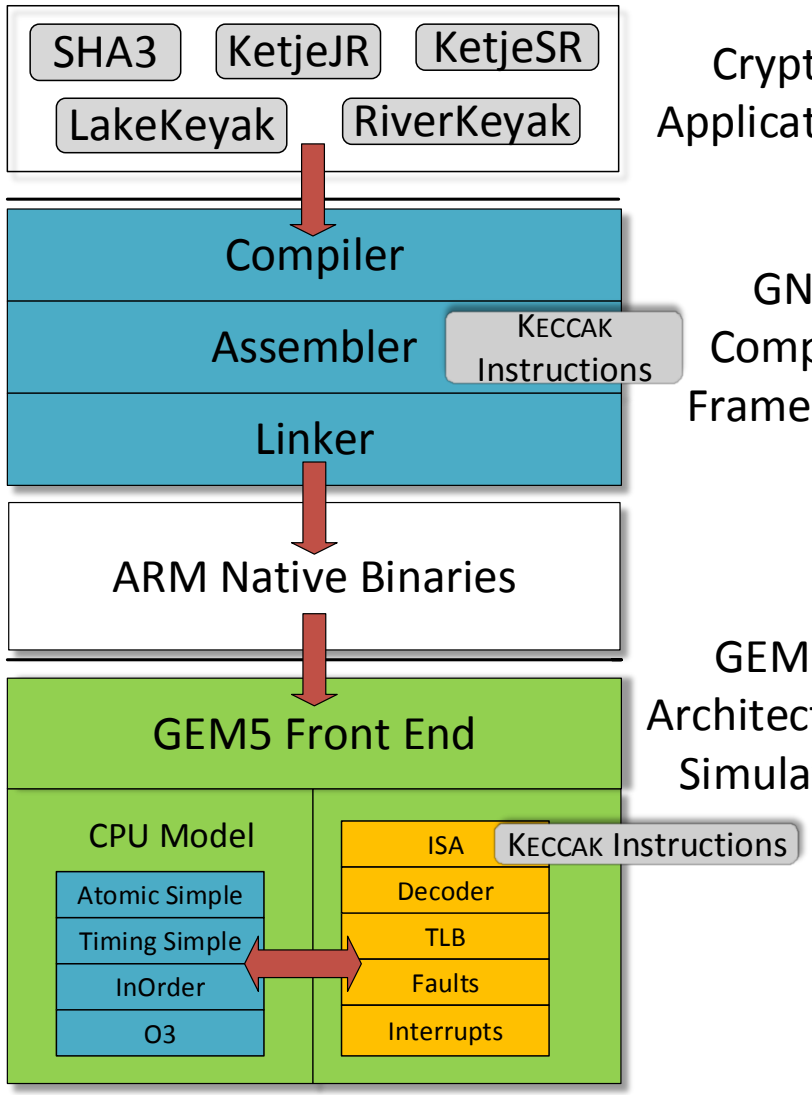
NEON Register Naming Convention
Qi = Quad word (128 bit)
Di = Double word (64 bit)
Si = Single word (32 bit)

NEON Instruction specifier
VADD.I32 q1, q2, q3 specifies
q1, q2 and q3 have 4x32-bit integer

KECCAK Custom Instructions

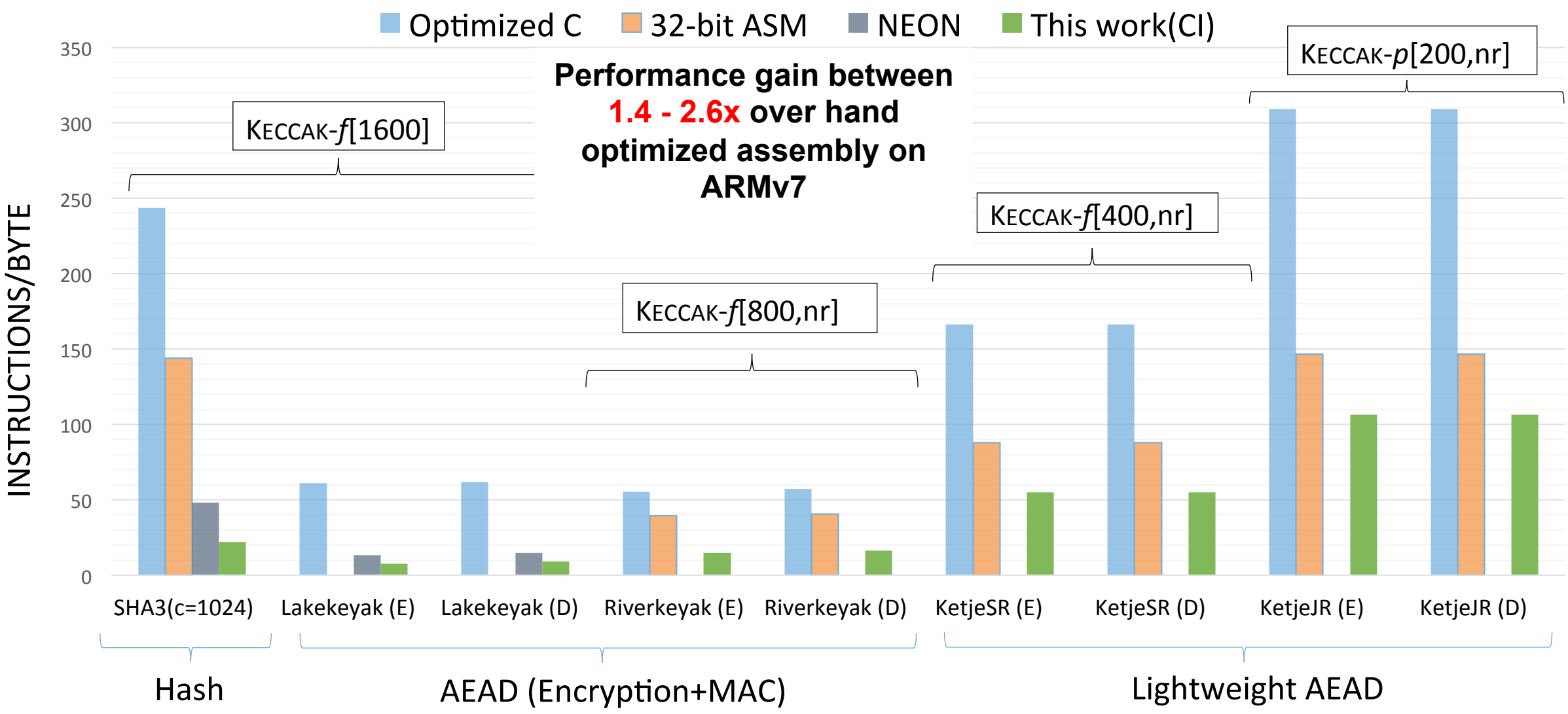
Instruction	Step	Description	Target Primitive	Syntax
rl1x	theta	Rotate Left 1 and XOR	1600, 800, 400, 200	rl1x.u64 d2, d0, d1 rl1x.u32 s2, s0, s1 rl1x.u16 s2, s0, s1 rl1x.u8 s2, s0, s1
kxorr64	theta, rho & pi	XOR Rotate & Assign	1600	kxorr64 d2, d0, d1, *#i*
xorr	theta, rho & pi	XOR Rotate & Assign	800, 400, 200	xorr.u32 s2, s0, s1, *#i* xorr.u16 s2, s0, s1, *#i* xorr.u8 s2, s0, s1, *#i*
chi1	chi	chi step	1600, 800, 400, 200	chi1.u32 q2, q0, q1 chi1.u64 q2, q0, q1
chi2	chi	chi step (last lane)	1600	chi2.u64 d4, q0, q1
chi3	chi	chi step (last lane)	800, 400, 200	chi3.u32 s4, d0, d1

Implementation & Validation



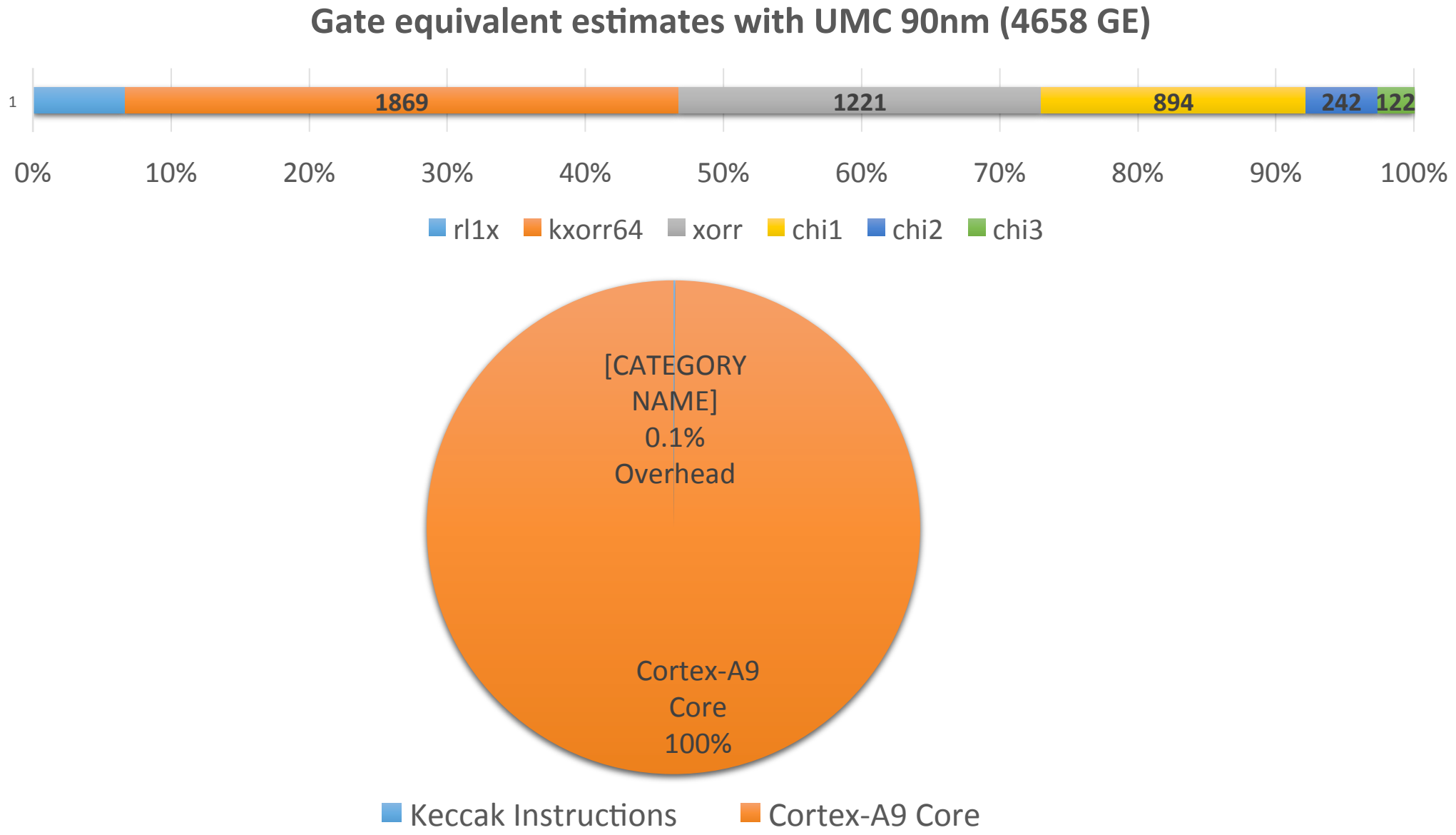
- ❑ Reference Implementations
  - KeccakCodePackge : http://keccak.noekeon.org/files.html
- ❑ Hand Optimized Custom Instruction implementations
  - KECCAK-*f,p* {1600,800,400, 200}
- ❑ Cross compiled GCC with KECCAK Instruction support
- ❑ Timing simple CPU model in GEM5
- ❑ Instructions in GEM5's ISA description language
- ❑ Simulation
  - Single core ARM CPU @ 1 GHz
  - 32KB L1 I and D cache
  - L2 cache of 2MB

Results: Performance



Performance in instructions/byte for various KECCAK modes

Results: Hardware Cost



Conclusion

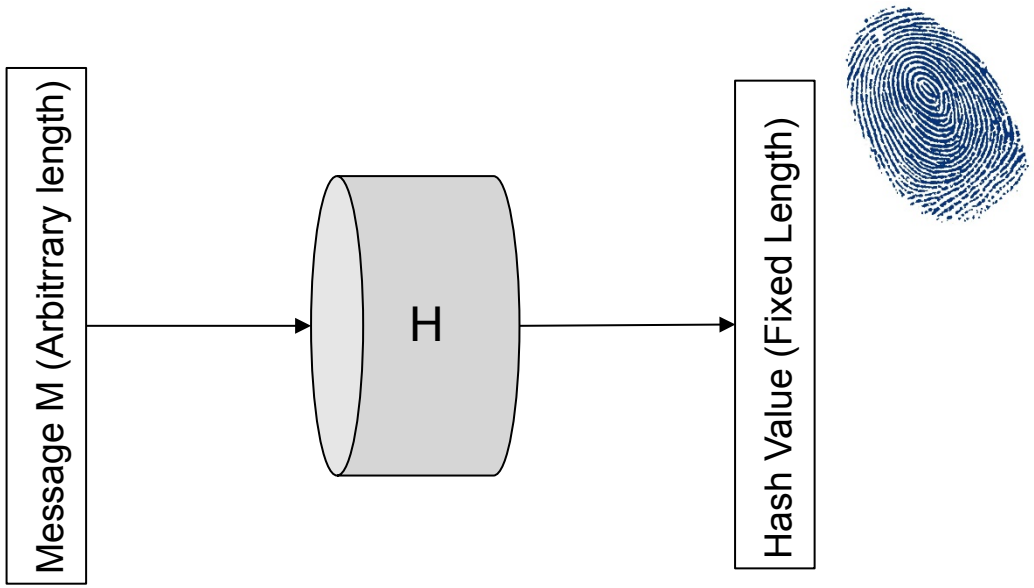
- Analysis of the instruction set design space for KECCAK primitives
- Six custom instructions based on NEON instruction set in ARMv7
- Five different KECCAK applications: SHA3, LakeKEYAK, RiverKEYAK, KetjeSR and KetjeJR
- Performance gain between 1.4 - 2.6x over hand optimized assembly on ARMv7 at a hardware overhead of just 4658 GEs.
- Portability Aspects, Intel AVX, Generic 64/32 bit architectures, in the paper….

Thank you ….

Questions

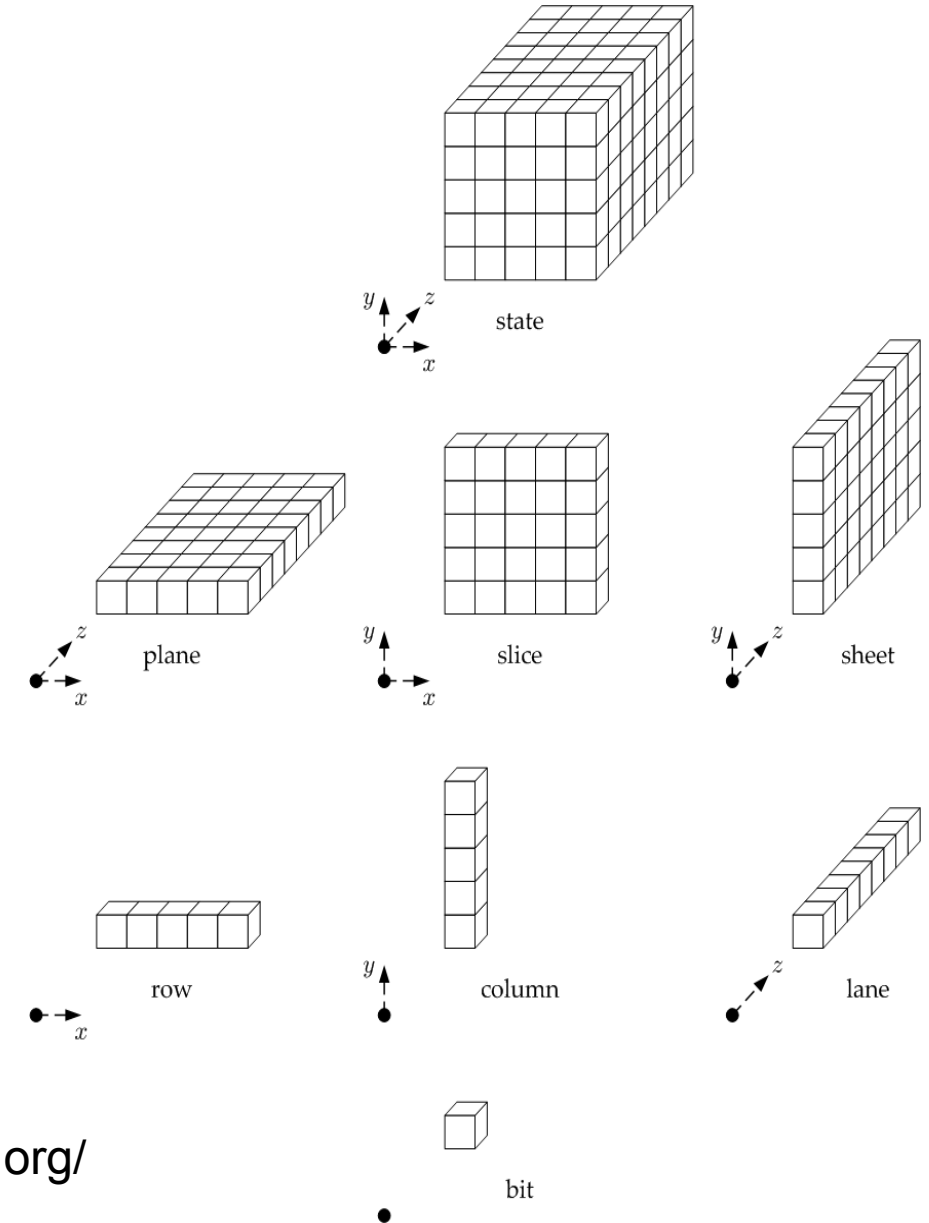
Appendix

KECCAK (SHA-3)



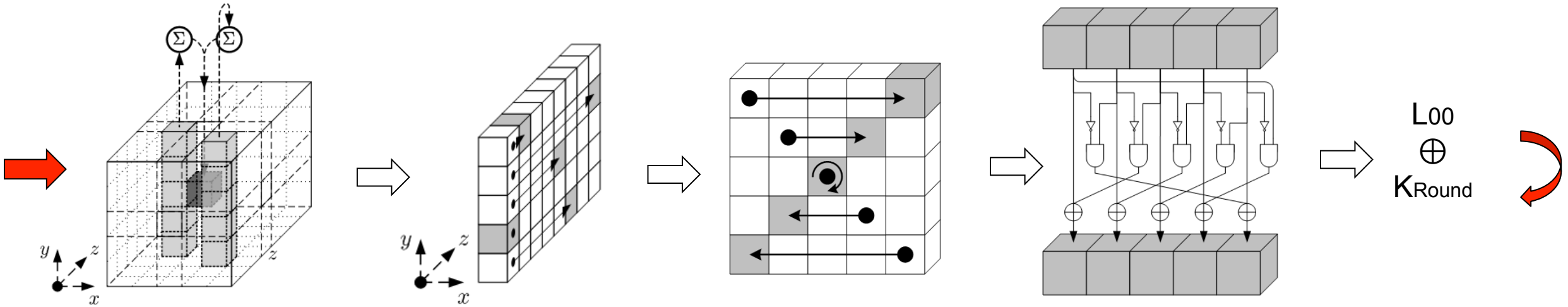
Instance	Definition
SHA3-224(M)	Keccak[448](M||01, 224)
SHA3-256(M)	Keccak[512](M||01, 256)
SHA3-384(M)	Keccak[768](M||01, 384)
SHA3-512(M)	Keccak[1024](M||01, 512)
SHAKE128(M, d)	Keccak[256](M||1111, d)
SHAKE256(M, d)	Keccak[512](M||1111, d)

Keccak



Source: http://keccak.noekeon.org/

KECCAK-*f* Permutation (2)

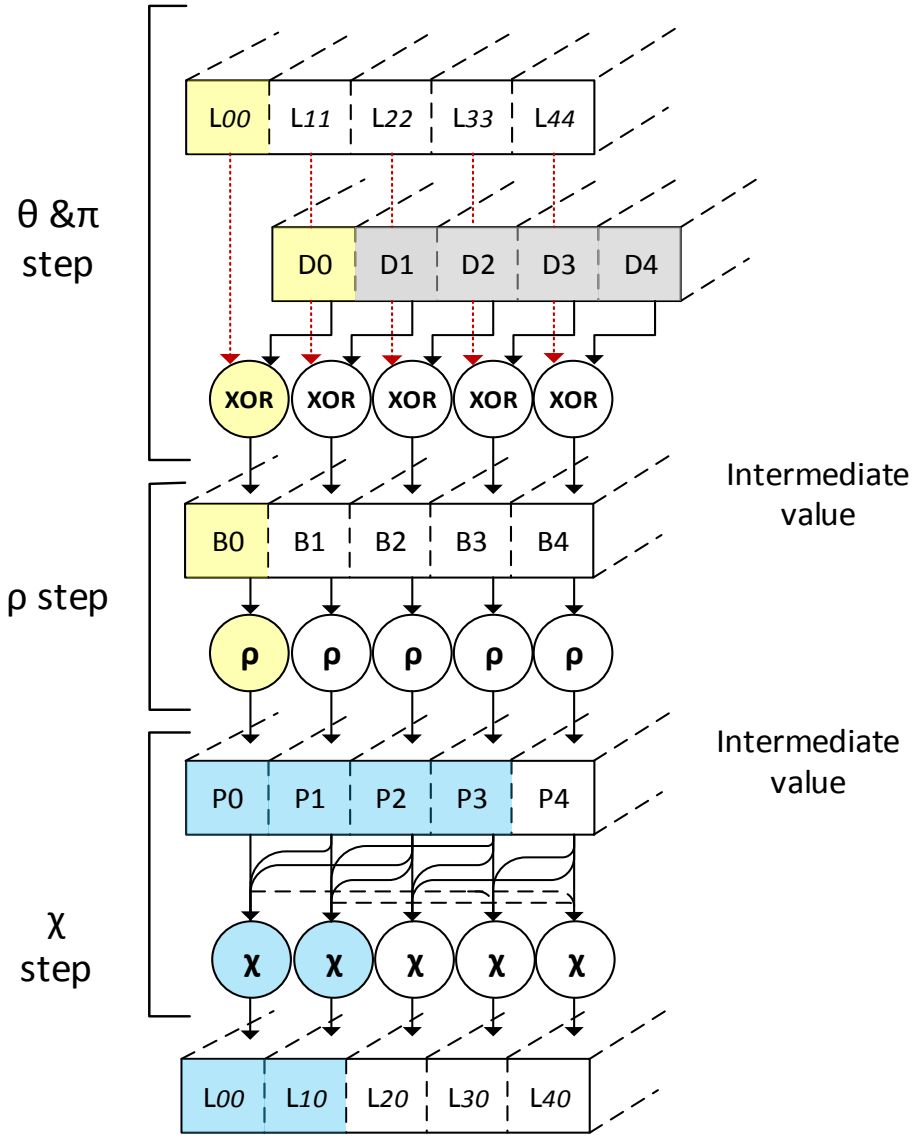


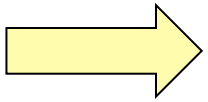
θ step ρ step π step χ step ι step

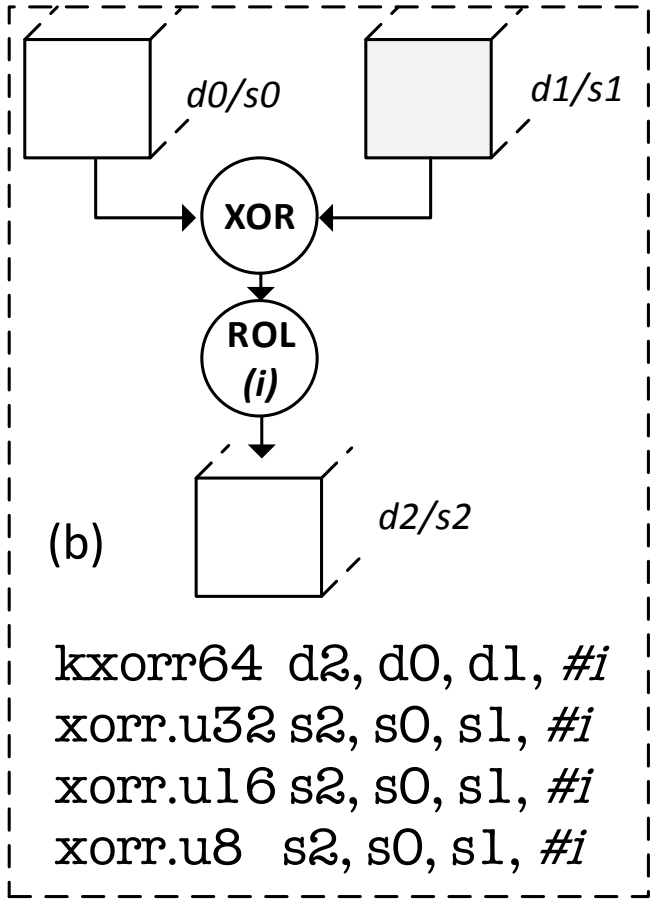
$$R = \theta \circ \rho \circ \pi \circ \chi \circ \iota$$

Source: http://keccak.noekeon.org/

Proposed Instruction Set Extensions



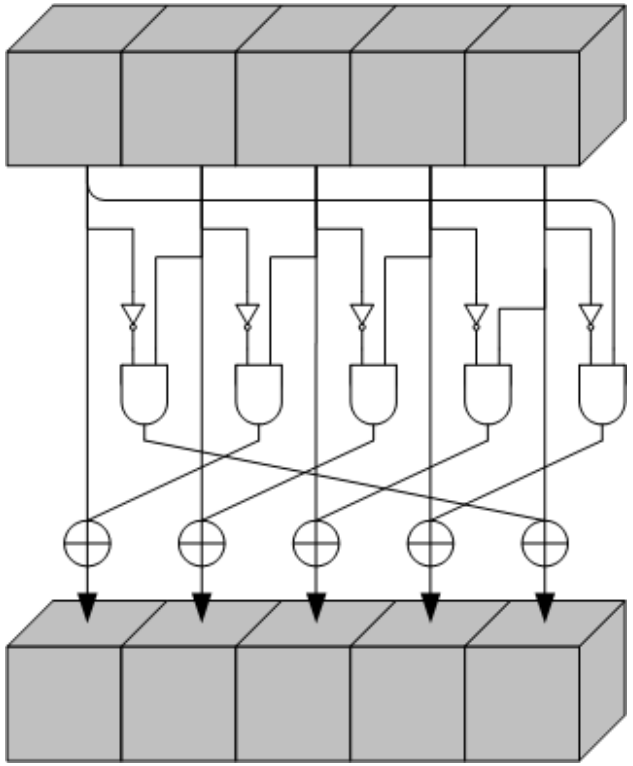




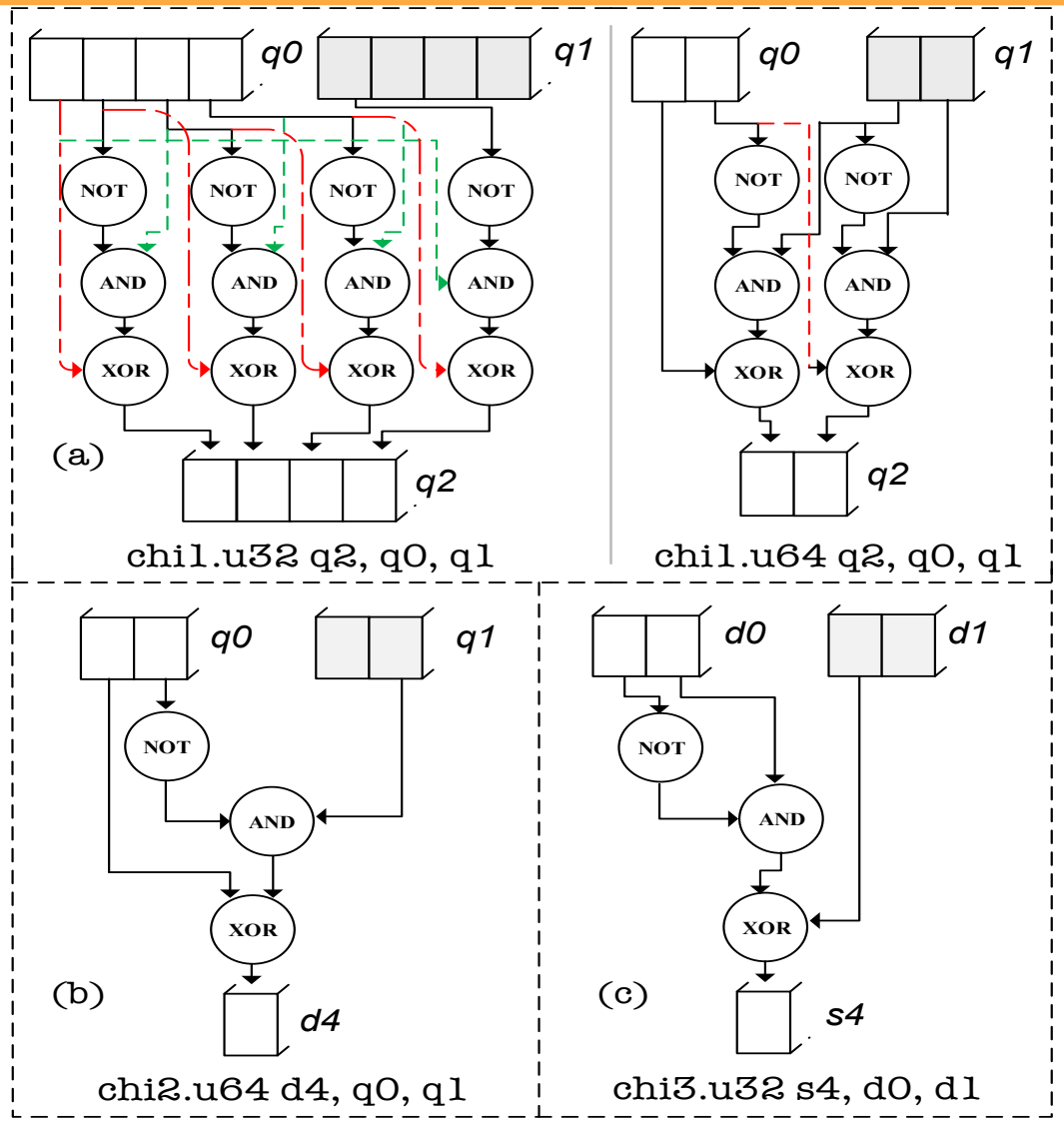
Instruction kxorr64/xorr

Data dependencies of θ,ρ ,π ,χ, ι (one plane)

Proposed Instruction Set Extensions



Data dependencies X step (one plane)



Instructions chi1,chi2 and chi3

http://keccak.noekeon.org/

Portability Aspects

- Based on instruction format, register width and the instruction encoding width.

Table 2: Feasibility of the proposed instructions on different platforms

Instruction	Intel AVX	64 bit Arch	32 bit Arch
rl1x	✓	✓*	✓*
kxorr64	✓	✓	X
xorr	✓	✓	✓
chi1	✓	X	X
chi2	✓	X	X
chi3	✓	✓	X

* Not all variants supported

Results: Hardware Cost

Table 1: Area, GE and Transistor count estimates for the proposed custom instructions

Instruction	μ^2	Gate equivalent	Transistor Equivalent
rl1x	1238	310	1238
kxorr64	7474	1869	7474
xorr	4884	1221	4884
chi1	3576	894	3576
chi2	966	242	966
chi3	486	122	486
Total	**18624**	**4658**	**18624**

Typical Cortex-A9 CPU 3.8 Million Gates
KECCAK Instructions have 0.1 % overhead

Motivation

Cryptographic Instruction-Set ⇨ Intel AES-NI, SHA-1,2, Carry-less multiplication extensions
ARMv8 Crypto Instructions

Processor Architecture ⇨ ARM (Soft IP)
RISC-V (Open Architecture from UC, Berkeley)

Symmetric Cryptography Applications ⇨ Hash, MAC, Encryption/Decryption, AEAD, PRNG

Universal Sponge Construction ⇨ **Hash**: SHA-3 (Winner of NIST SHA-3 Competition)
AEAD: LakeKeyak, Ketje AEAD
PRNG, Stream Ciphers etc.…