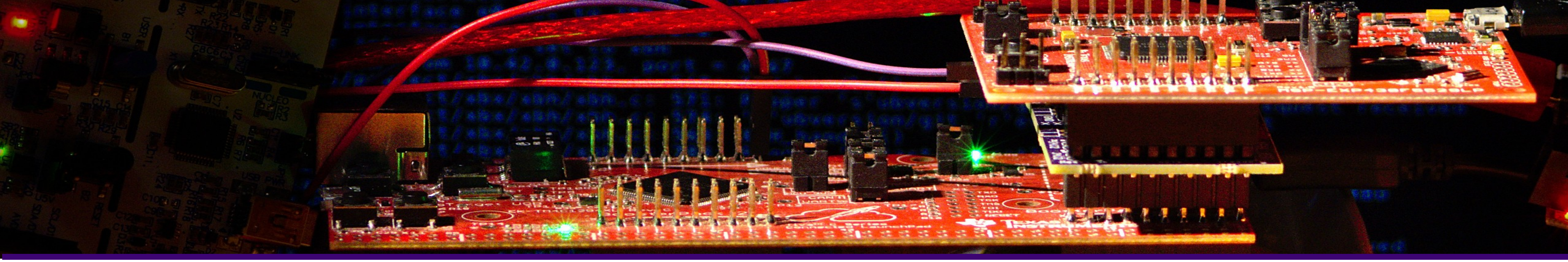
Feasibility and Performance of PQC Algorithms on Microcontrollers

Brian Hession, Jens-Peter Kaps

Second PQC Standardization Conference
August 2019

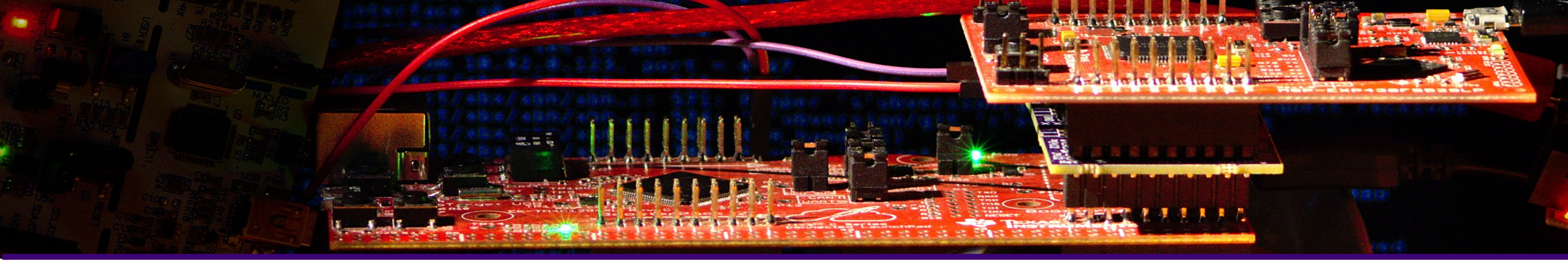
Cryptographic Engineering Research Group

cryptography.gmu.edu



Overview

- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results



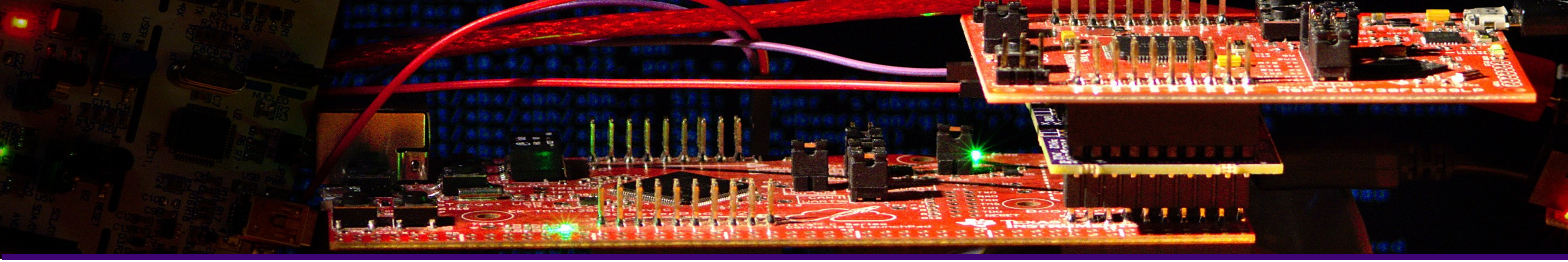
Overview

- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results

Motivation

- The move to the Internet of Things (IoT) leads to formerly “dumb” devices being connected to the Internet.
- Quantum Computers will make breaking many public key algorithms possible.
- Even IoT devices will need to run PQC algorithms.
- By 2025, over 75 billion devices will be connected to the Internet.
- 32-bit microcontrollers are projected to take lead over 8/16-bit.
- 51% of all 32-bit microcontrollers were ARM based in 2012.

PQC algorithms have to be benchmarked on microcontrollers



Overview

- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results

SUPERCOP

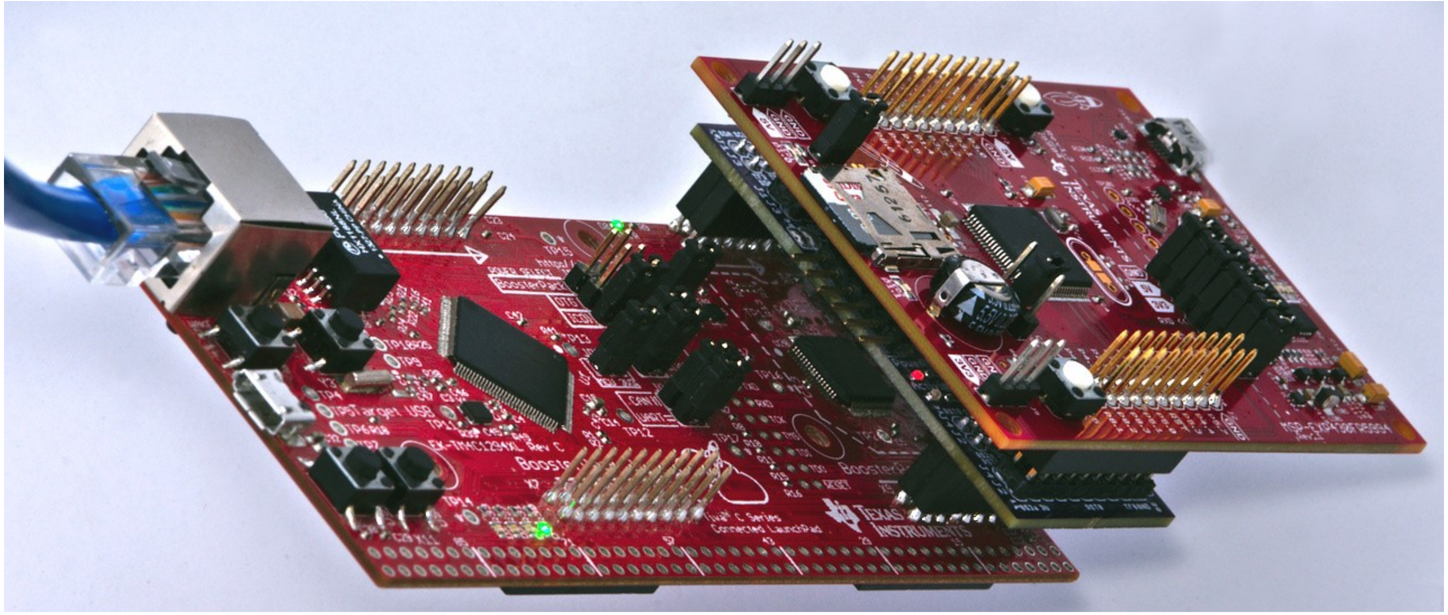
- System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives.
- Benchmarks many implementations of many primitives across multiple operations on multiple hardware platforms.
- Supports environments capable of running Linux and hosting a compiler.
- Series of shell scripts and C test harnesses, and comprehensive collection of algorithm primitive implementations.
- Verifies correct execution of implementations and times cycles required per byte processed.
- Does not measure ROM and RAM usage or power consumption.
- http://bench.cr.yp.to/supercop.html

XBX

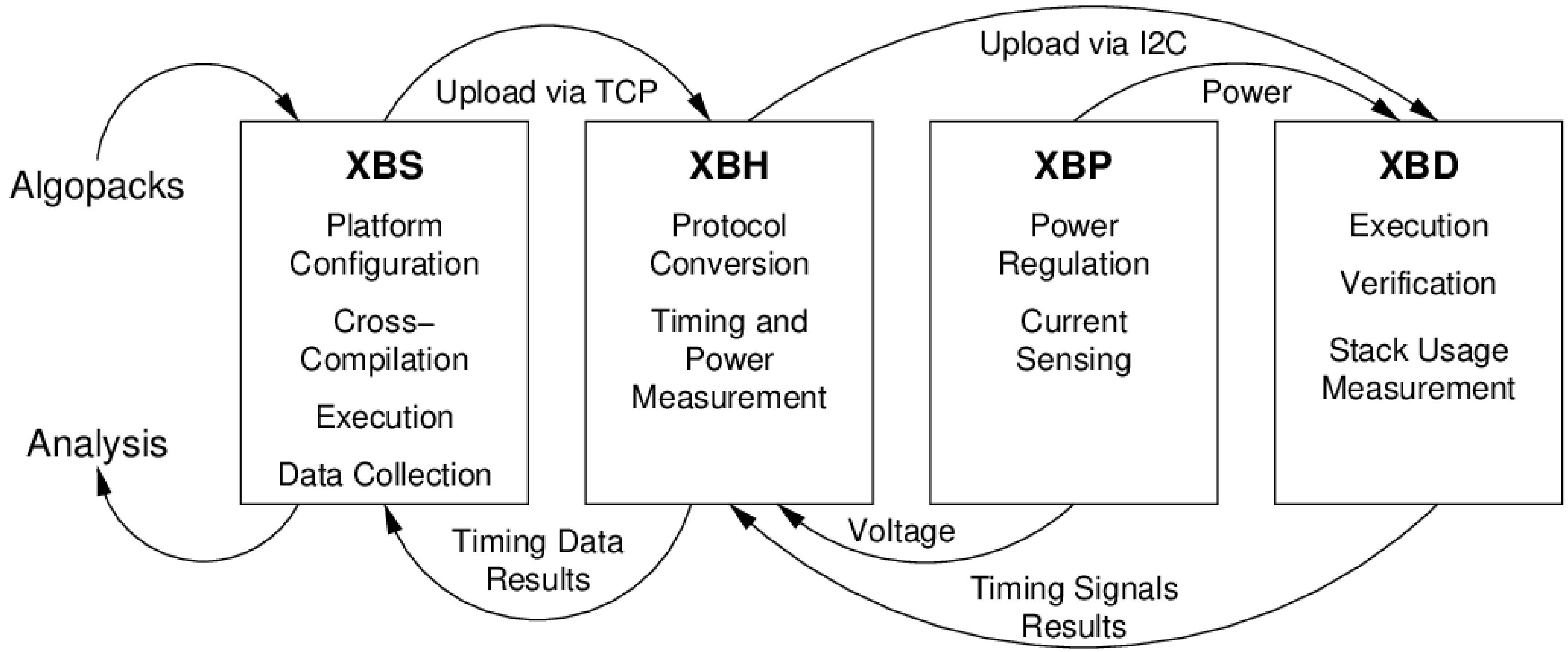
- eXternal Benchmarking eXtension – extends SUPERCOP
- Automated testing on real microcontrollers
- Compatibility with SUPERCOP algorithm collection (“algopacks”) and output format
- Low cost hardware and software
- Our contribution to original XBX was to port it to the MSP430 platform and provide results for SHA-3 finalists.
- Measures ROM and RAM usage. Does not measure power consumption.

XXBX

- eXtended eXternal Benchmarking eXtension
- Extends XBX to
  - Support AEAD, now also PQC
  - Adds power measurement
  - Rewritten in Python 3, results in SQLite database

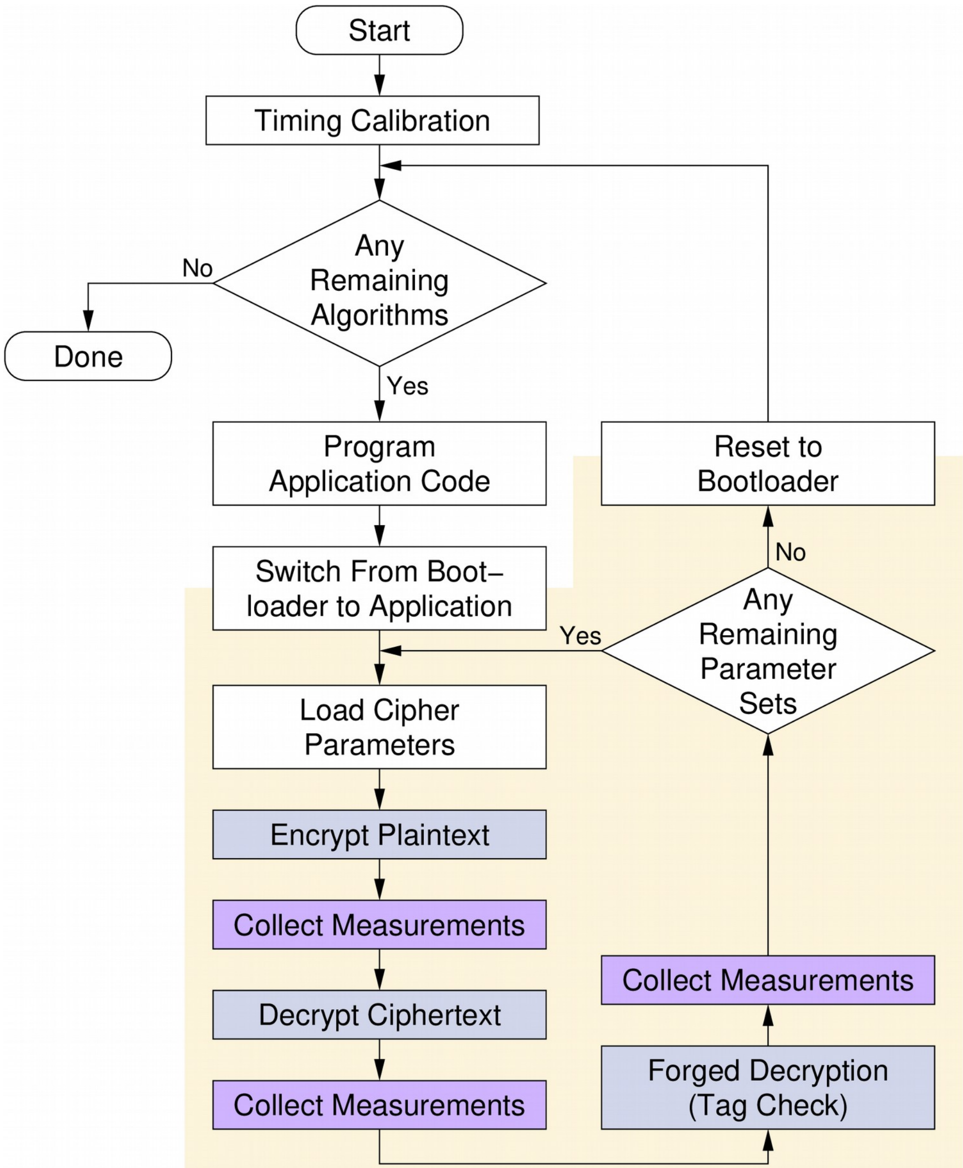


XXBX Components



- XBS – Benchmarking System
- XBH – Harness
- XBP – Power Shim
- XBD – Device under Test

Benchmarking Flow

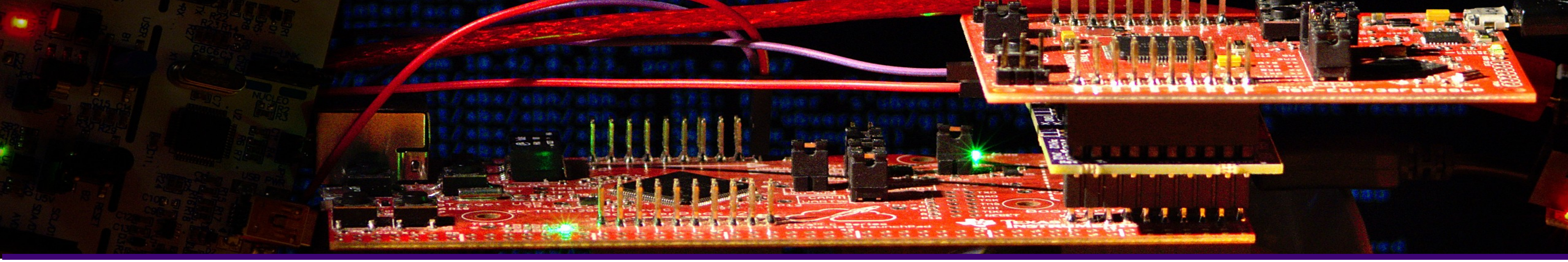


- Bootloader on XBD
  - Writes application into ROM
  - Runs timing calibration
  - Switches to application
- Application
  - Loads parameters
  - Paints stack
  - Sends execution start signal
  - Executes crypto algorithm
  - Sends execution end signal
  - Sends results incl stack usage
  - Returns to bootloader

Supported XBDs

Board	by	CPU	ISA	Bus [bit]	f [MHz]	ROM [kByte]	RAM [kByte]
MSP-EXP430F5229	TI	MSP430F	MSP430X	16	25	12	10
MSP-EXP430FR5994	TI	MSP430FR	MSP430X	16	16	256	8
MSP-EXP432P401R	TI	ARM Cortex M4F	ARMv7E-M	32	48	256	64
EK-TM4C123GXL	TI	ARM Cortex M4F	ARMv7E-M	32	80	256	32
EK-TM4C129EXL	TI	ARM Cortex M4F	ARMv7E-M	32	120	1024	256
NUCLEO-F091RC	STM	ARM Cortex M0	ARMv6-M	32	48	256	32
NUCLEO-F103RB	STM	ARM Cortex M3	ARMv7-M	32	72	128	20

- All boards are inexpensive: < $30
- RAM is very limited, below key sizes of several PQC algorithms

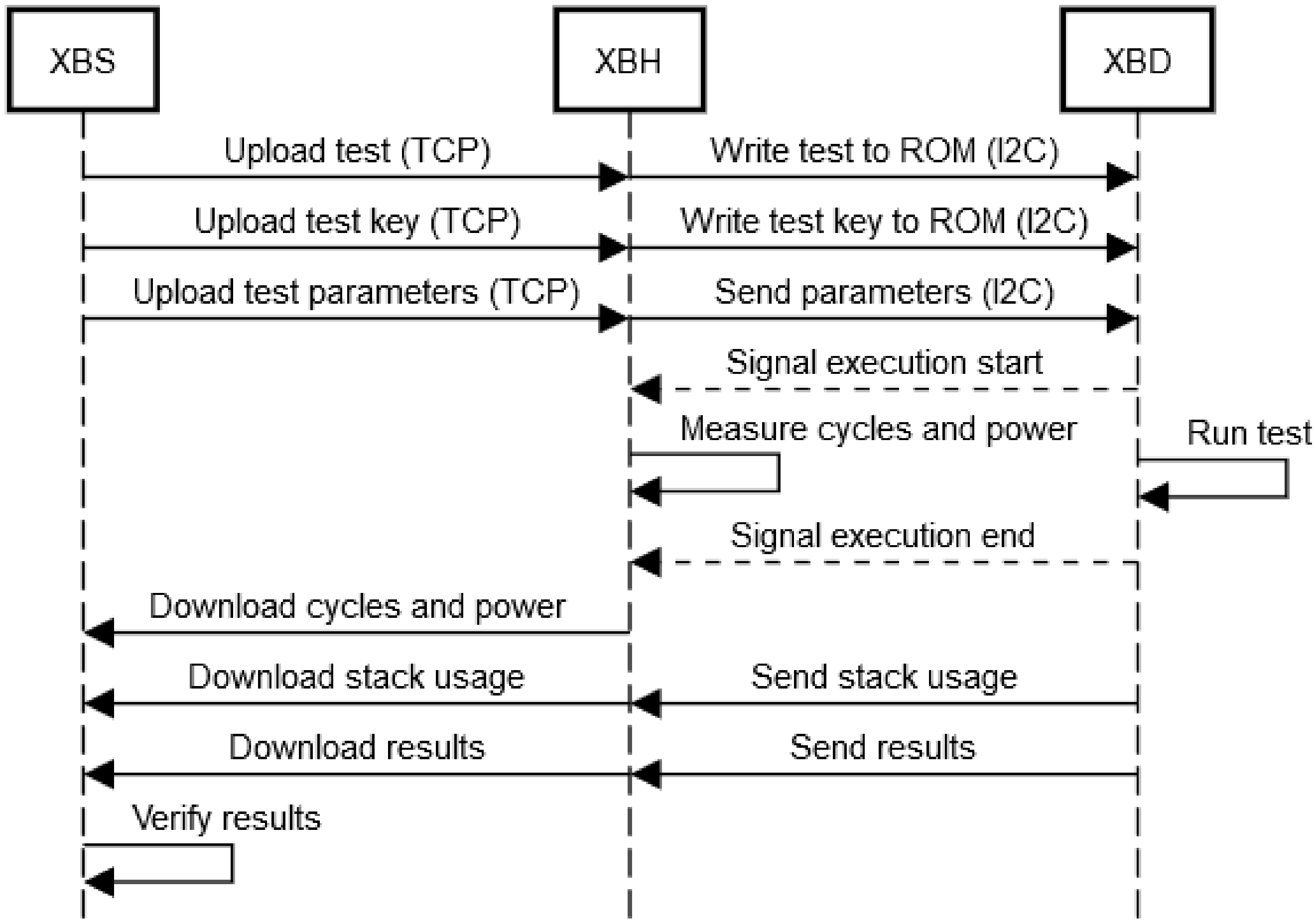


Overview

- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results

Getting XXBX Ready for PQC

- Save RAM
  - Moved test key (public / private) to ROM
  - Opens up stack



PQC Implementation (KEM)

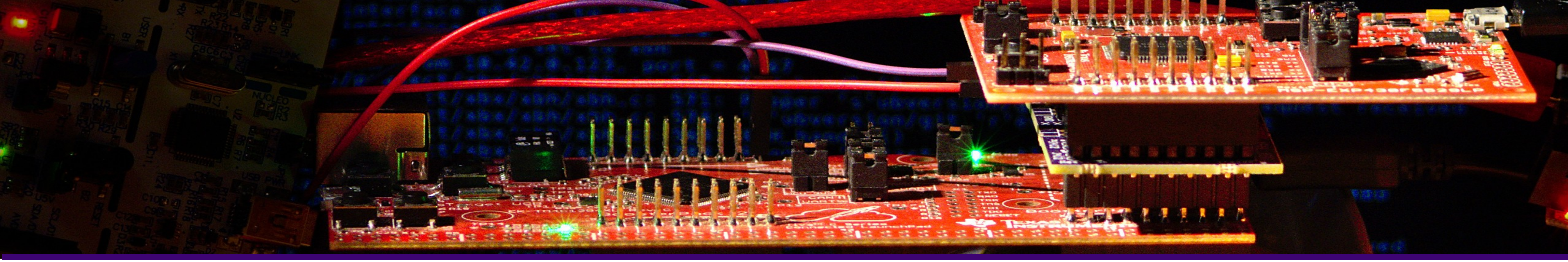
- Started with SUPERCOP's 20190110 implementations of PQ KEM algorithms.
- Added pqm4 implementations for Cortex-M4 optimizations.
  - https://github.com/mupq/pqm4
- Extended pqm4 to include some Level I and Level V versions of algorithms.
- Less than 2 weeks ago updated to SUPERCOP 20190811 which contained many more PQC algorithms.
- More than 100 variants (algorithm – parameter set combinations) are available.
- Many have more than one implementation.

PQC Signature Operations

- ek-tm4c123gxl is limited to 32kB worth of RAM.
  - Most of the Signature algorithms' implementations exceed this memory constraint
  - Signature algorithms need to be reworked to free stack space

Algorithms	Sign (B)	Verify (B)
dilithium	86,720	54,904
qTesla-I	29,336	23,096
qTesla-III-size	58,116	45,732
qTesla-III-speed	58,112	45,712

- Ways forward:
  - Limit the life of temporary variables
  - Recycle or repurpose allocated structures



Overview

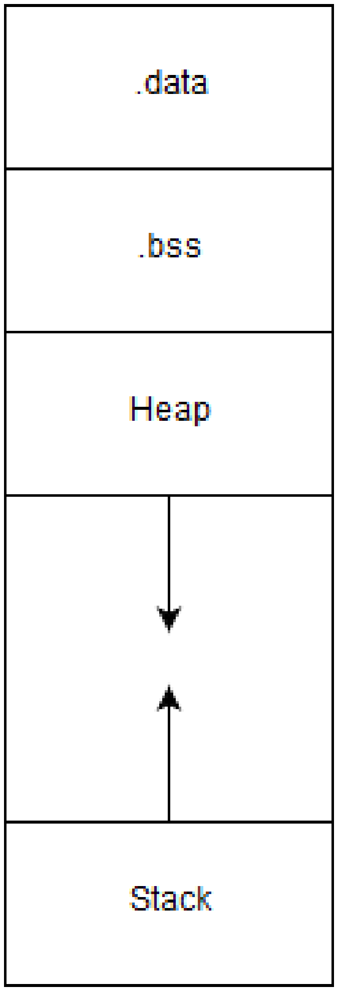
- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results

Memory Constraints

- Small amount of memory available puts constraints on which algorithms can run
- Key sizes too large for RAM and/or ROM
  - E.g.: Classic McEliece, FrodoKEM, HQC, LEDAcrypt, NTS-KEM, Round5 with non-ring parameter set

Typical Problems

- Algorithms don’t compile
  - Dealing with dynamic memory allocation on system without OS.
    - Solution: Implemented simplified _sbrk() system call
  - Algorithm depends on other libraries
    - E.g.: SHA2, Keccak, etc.
- Requires more RAM than available
  - Stack grows into .bss and .data sections
  - E.g.: Kyber 90s, Ledakem12, ntruhps4096821, firesaber



Algorithm Selection (Levels 1 & 2)

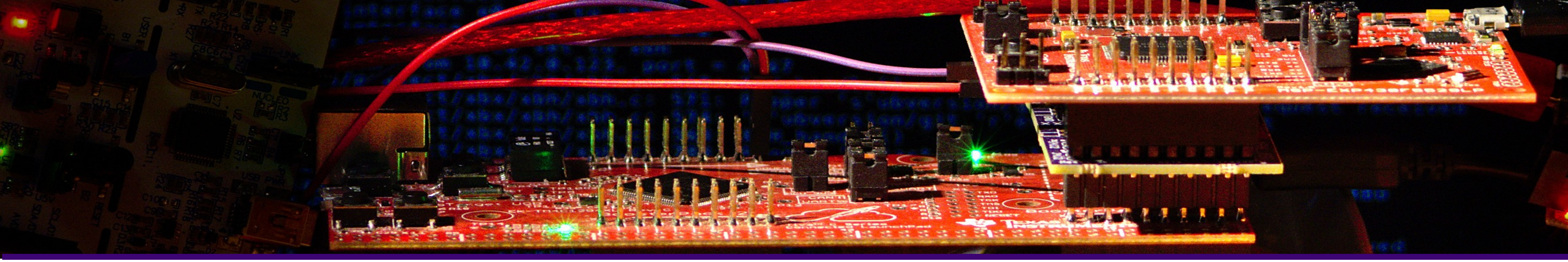
Algorithm	Implementation	Security Level	Type	Public Key	Secret Key	Session Key	Ciphertext
babybear	xbdref	2	Lattice	804	40	32	1,307
kyber512	m4v2	1	Lattice	800	1,632	32	736
lightsaber	m4v2	1	Lattice	672	1,568	32	1,088
newhope512cca	ref	1	Lattice	928	1,888	32	1,120
ntruhps2048509	m4v2	1	Lattice	699	935	32	930
r5nd1kemcca0d	m4v2	1	Lattice	634	666	16	698
r5nd1kemcca0dsneik	m4v2	1	Lattice	634	666	16	565
r5nd1kemcca5d	m4v2	1	Lattice	445	477	16	565
r5nd1kemcca5dsneik	m4v2	1	Lattice	445	477	16	1,005
sikep503	xbdref	2	Isogeny	378	434	16	402

Algorithm Selection (Levels 3 & 4)

Algorithm	Implemen-tation	Security Level	Type	Public Key	Secret Key	Session Key	Cipher-text
kyber768	m4v2	3	Lattice	1,184	2,400	32	1,088
ntruhps2048677	m4v2	3	Lattice	930	1,234	32	1,230
ntruhrss701	m4v2	3	Lattice	1,138	1,418	32	1,025
mamabear	xbdref	4	Lattice	1,194	40	32	1,697
r5nd3kemcca0d	m4v2	3	Lattice	909	957	24	1,005
r5nd3kemcca0dsneik	m4v2	3	Lattice	909	957	24	883
r5nd3kemcca5d	m4v2	3	Lattice	780	828	24	883
r5nd3kemcca5dsneik	m4v2	3	Lattice	780	828	24	1,306
saber	m4v2	3	Lattice	992	2,304	32	1,088

Algorithm Selection (Level 5)

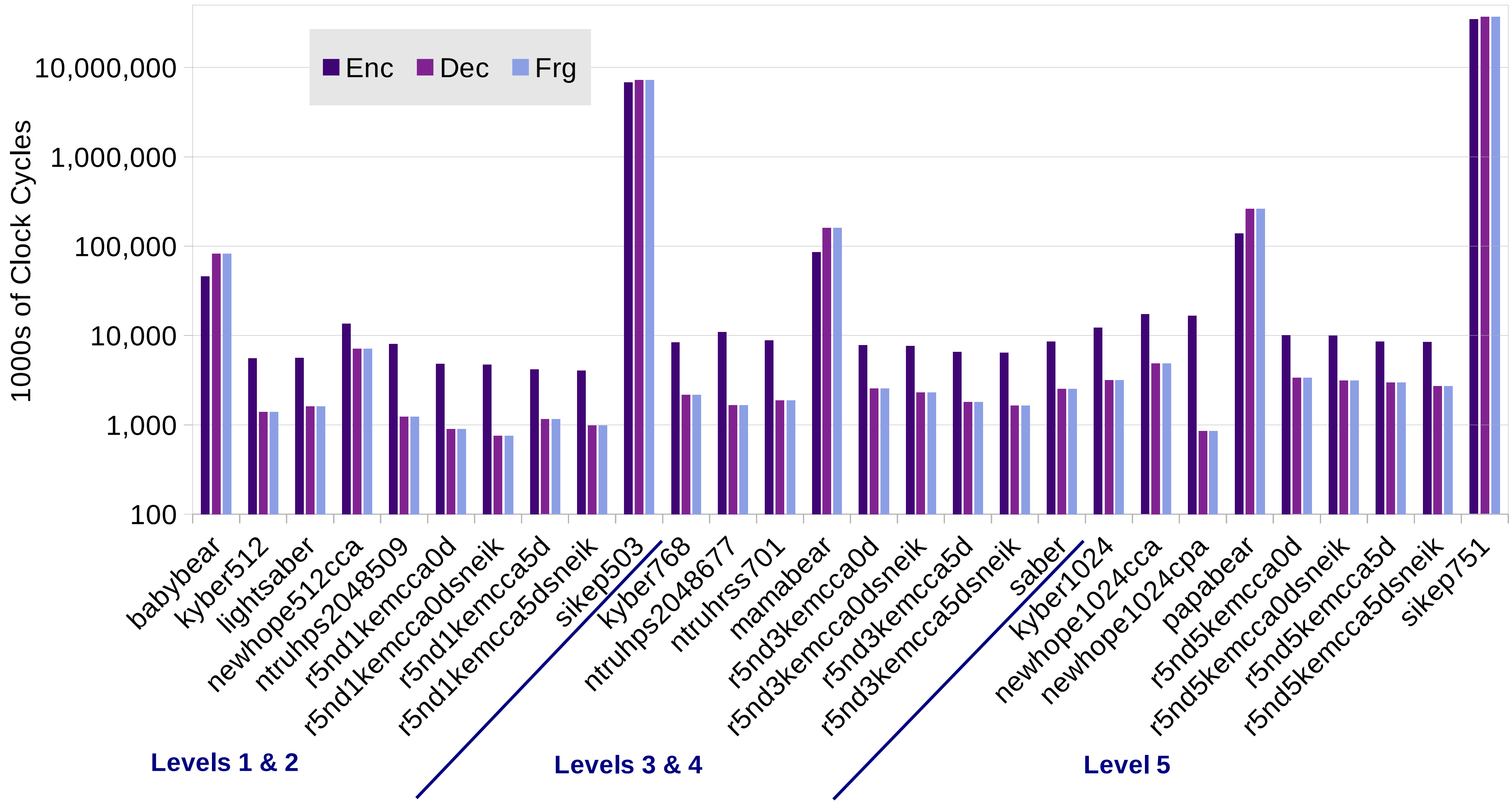
Algorithm	Implementation	Security Level	Type	Public Key	Secret Key	Session Key	Ciphertext
kyber1024	m4v2	5	Lattice	1,568	3,168	32	1,568
newhope1024cca	m4v2	5	Lattice	1,824	3,680	32	2,176
newhope1024cpa	m4v2	5	Lattice	1,824	1,792	32	1,120
papabear	xbdref	5	Lattice	1,584	40	32	1,697
r5nd5kemcca0d	m4v2	5	Lattice	1,178	1,242	32	1,306
r5nd5kemcca0dsneik	m4v2	5	Lattice	1,178	1,242	32	1,095
r5nd5kemcca5d	m4v2	5	Lattice	972	1,036	32	1,095
r5nd5kemcca5dsneik	m4v2	5	Lattice	972	1,036	32	1,690
sikep751	pqm4ref	5	Isogeny	564	644	24	917



Overview

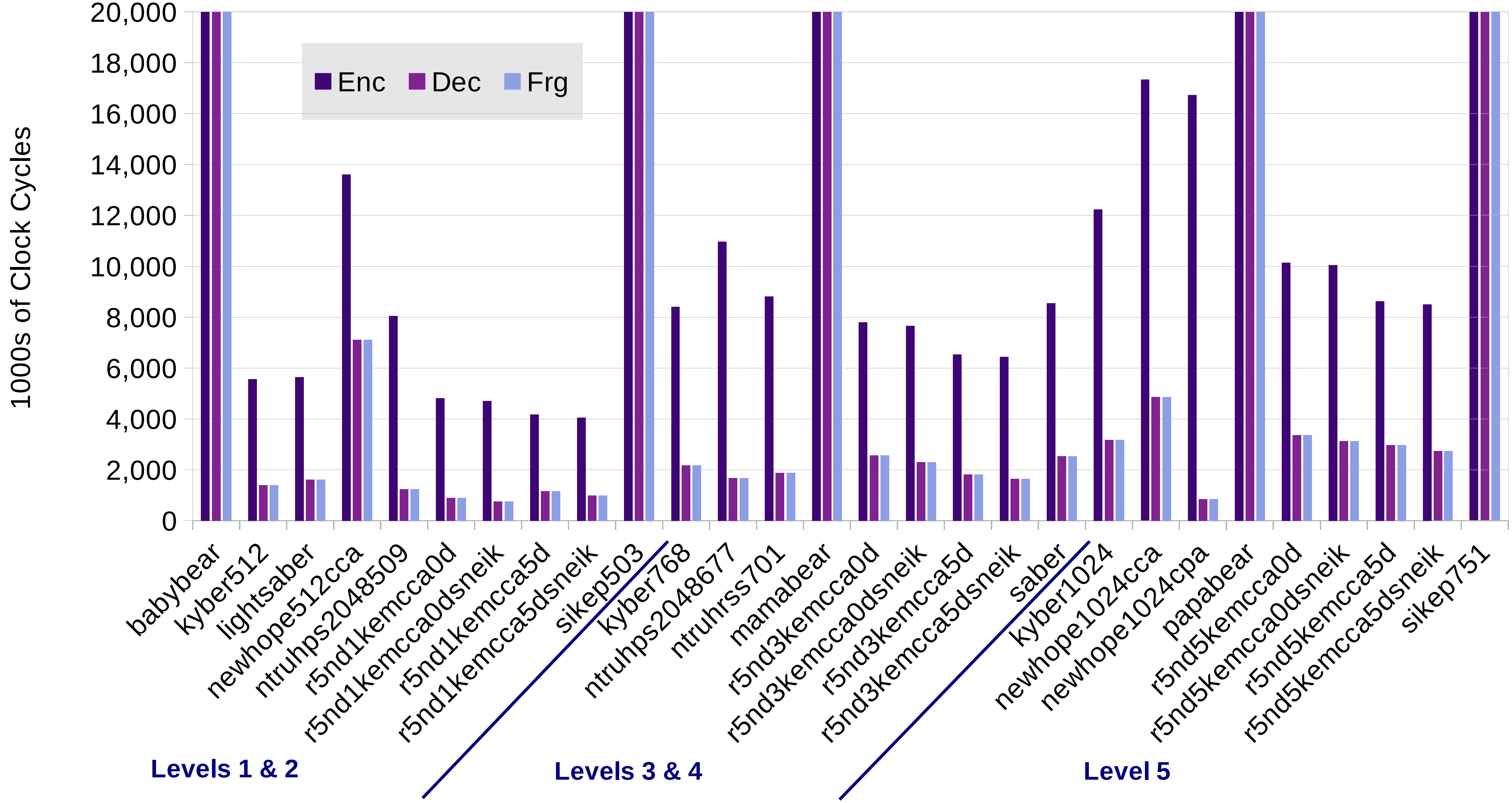
- Motivation
- Introduction to XXBX
- Getting XXBX ready for PQC
- Benchmarking
- Results

Clock Cycles



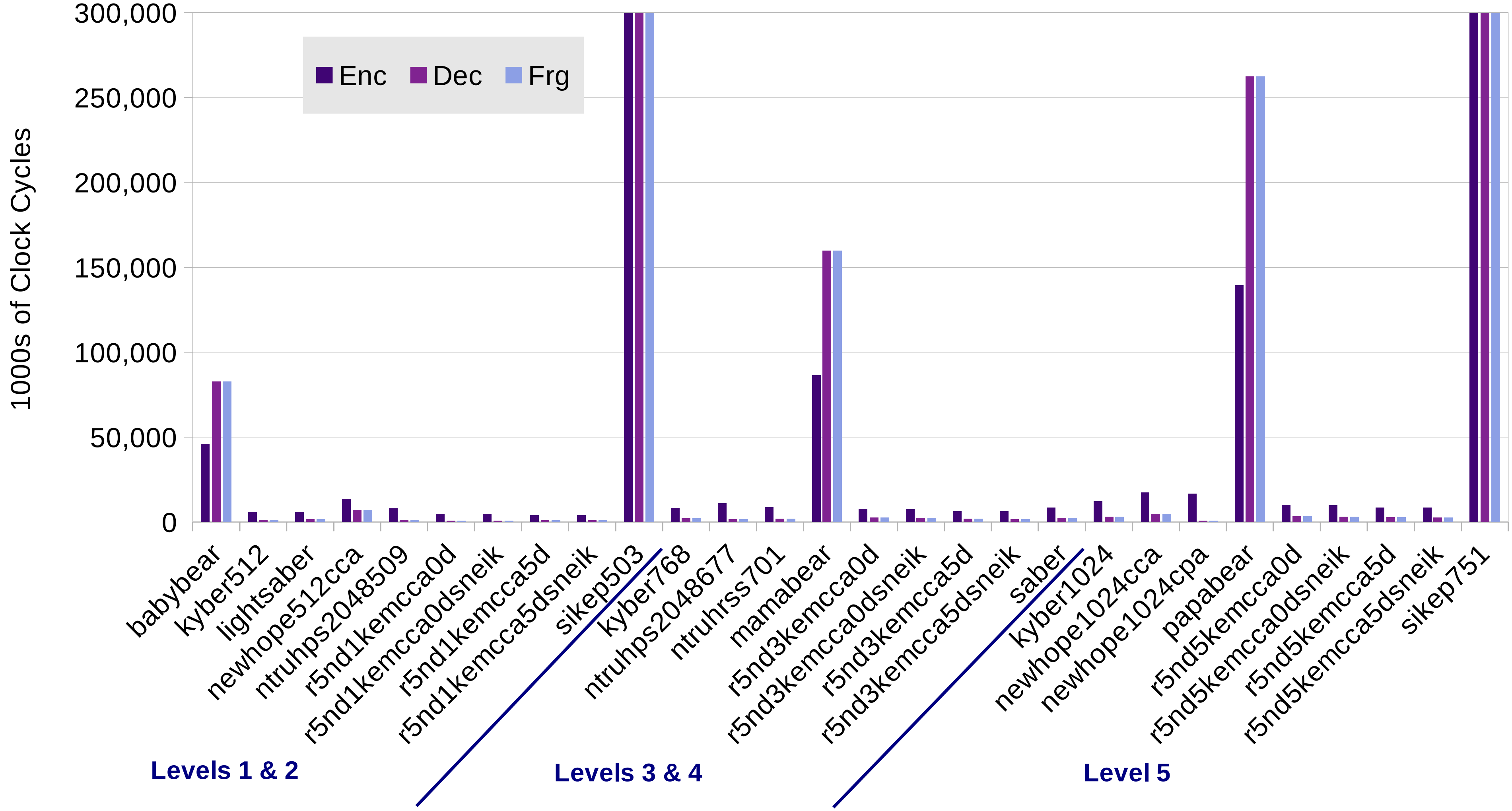
- Logarithmic Scale!

Clock Cycles till 20,000,000



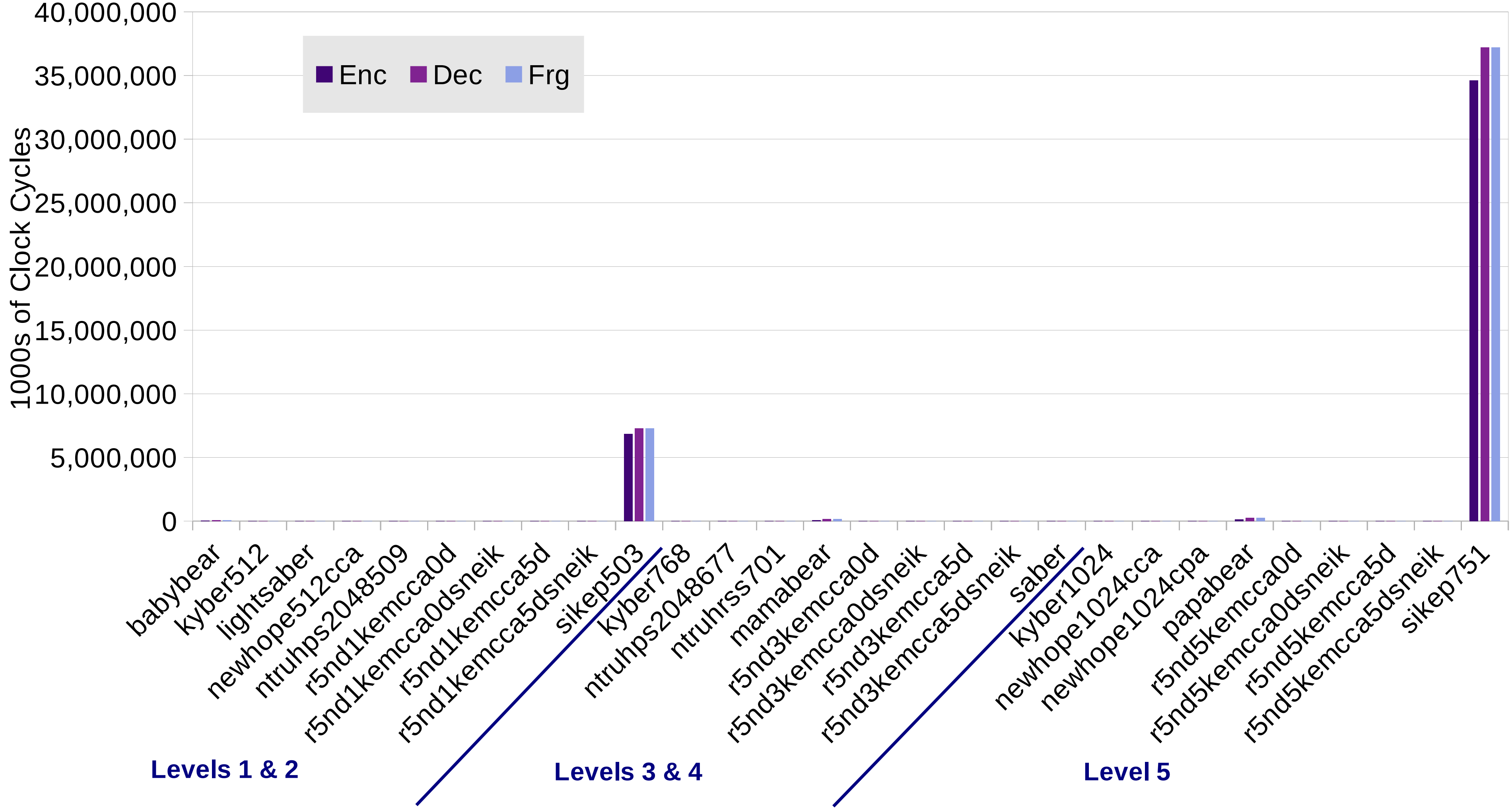
- Kyber, NewHope, NTRU, Round5, and Saber complete
- Encapsulation significantly slower than decapsulation

Clock Cycles till 300,000,000



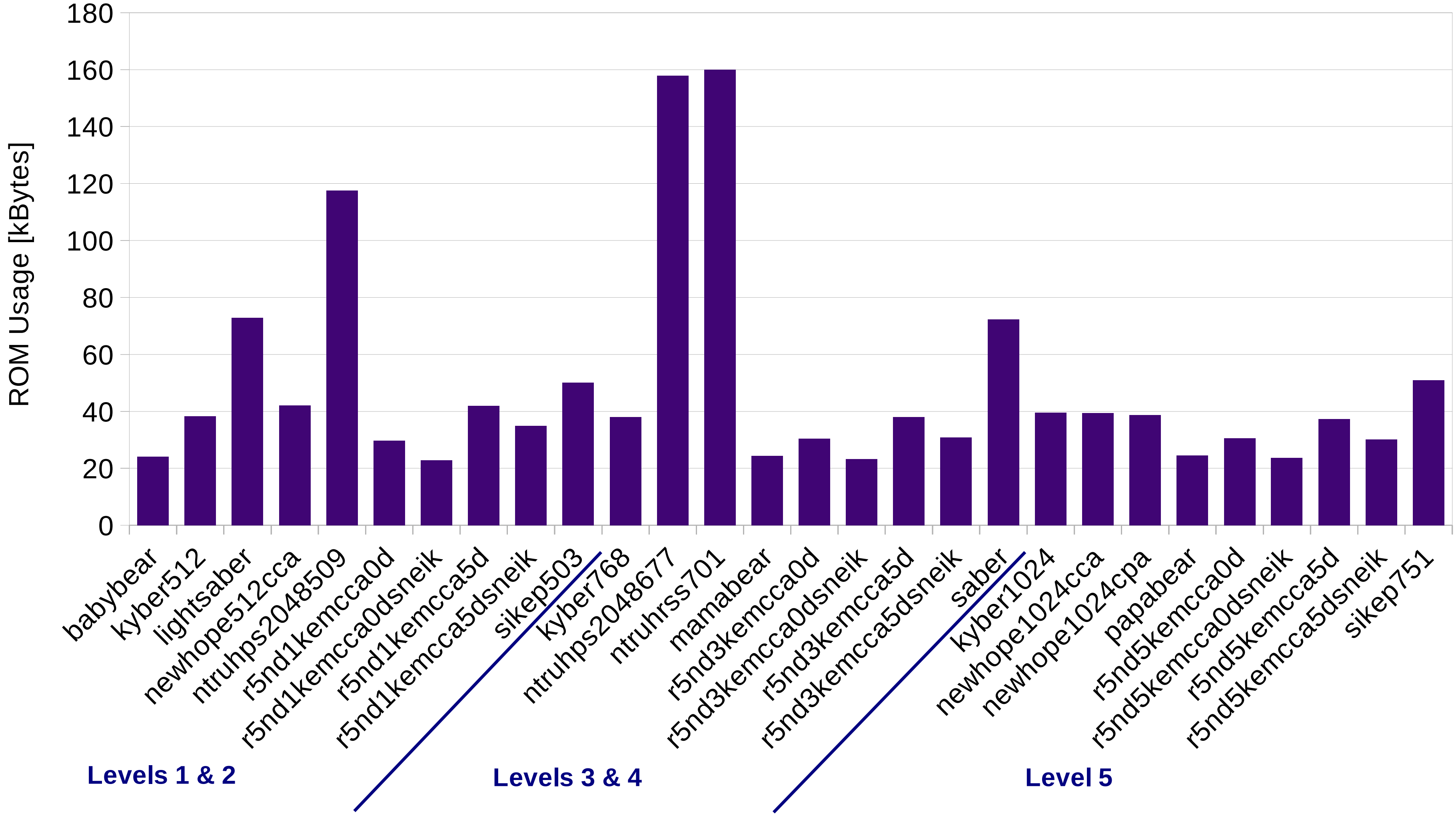
- Three Bears complete
- Reference implementation

Clock Cycles till 40,000,000,000



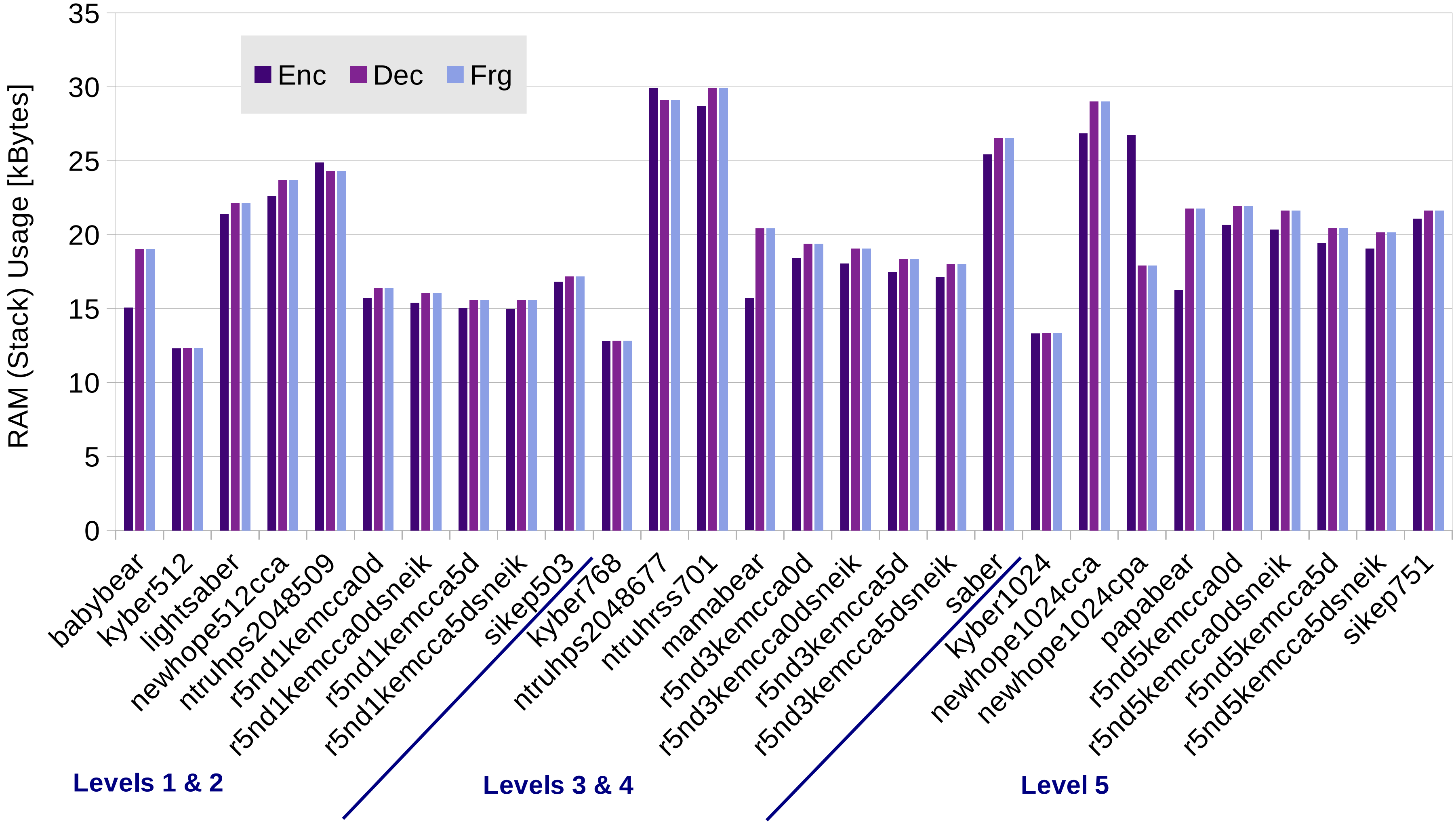
- Sike completes
- At 16 MHz, sikep503 takes 7 min, sikep751 takes 36 minutes

ROM Usage



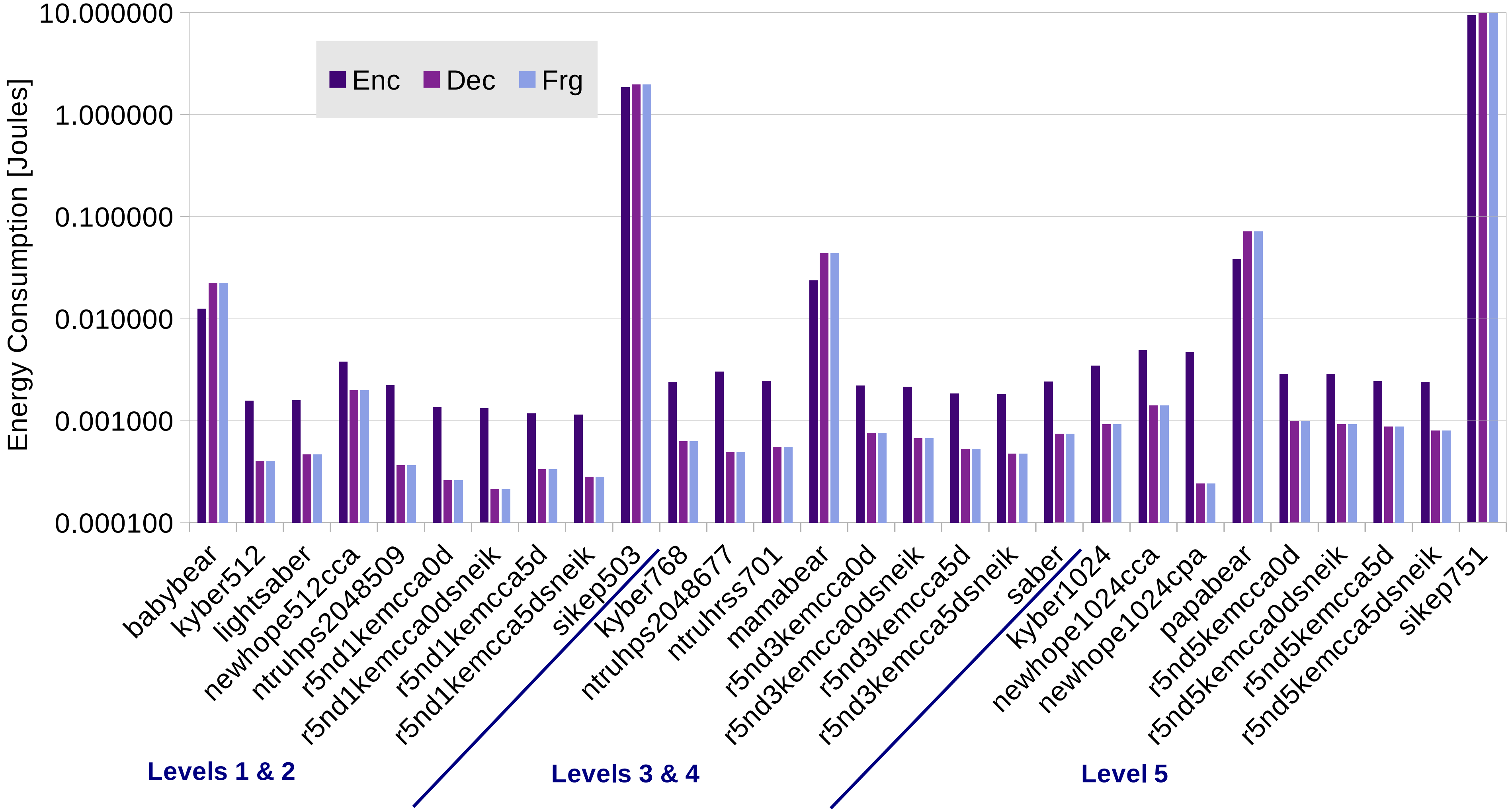
- Our Platform only has 256 kByte of ROM
- Most algorithms are significantly smaller

Stack Usage



- Our Platform only has 32 kByte of RAM
- This causes several algorithms to crash, e.g., firesaber

Energy Usage



- Logarithmic scale (again)
- Mostly dependent on # of clock cycles used, power variations are minimal

XXBX Future Work

- Bigger XBD → more RAM → more algorithms
- Trying to get more algorithms to compile
  - Removing malloc
  - Removing library dependencies
- Benchmarking signature algorithms

Questions

Thank you