Towards Efficient Post-Quantum Cryptography

From Standards to Implementations

Chaoyun Li

SCCS, University of Surrey

January 27, 2026

† Slides developed in collaboration with Harry Hart and Suparna Kundu

Outline

Outline

The quantum threat

Classical public key cryptography

- Discrete log problem over groups
  - Diffie-Hellman problems
  - Elliptic curve cryptography
- Integer factorization
  - RSA (Rivest, Shamir and Adleman, 1978)

All broken by quantum computers!

Quantum computers

1981 A new computational paradigm based on quantum mechanics: exponential parallelism by superposition and quantum entanglement [Feynman and Manin]

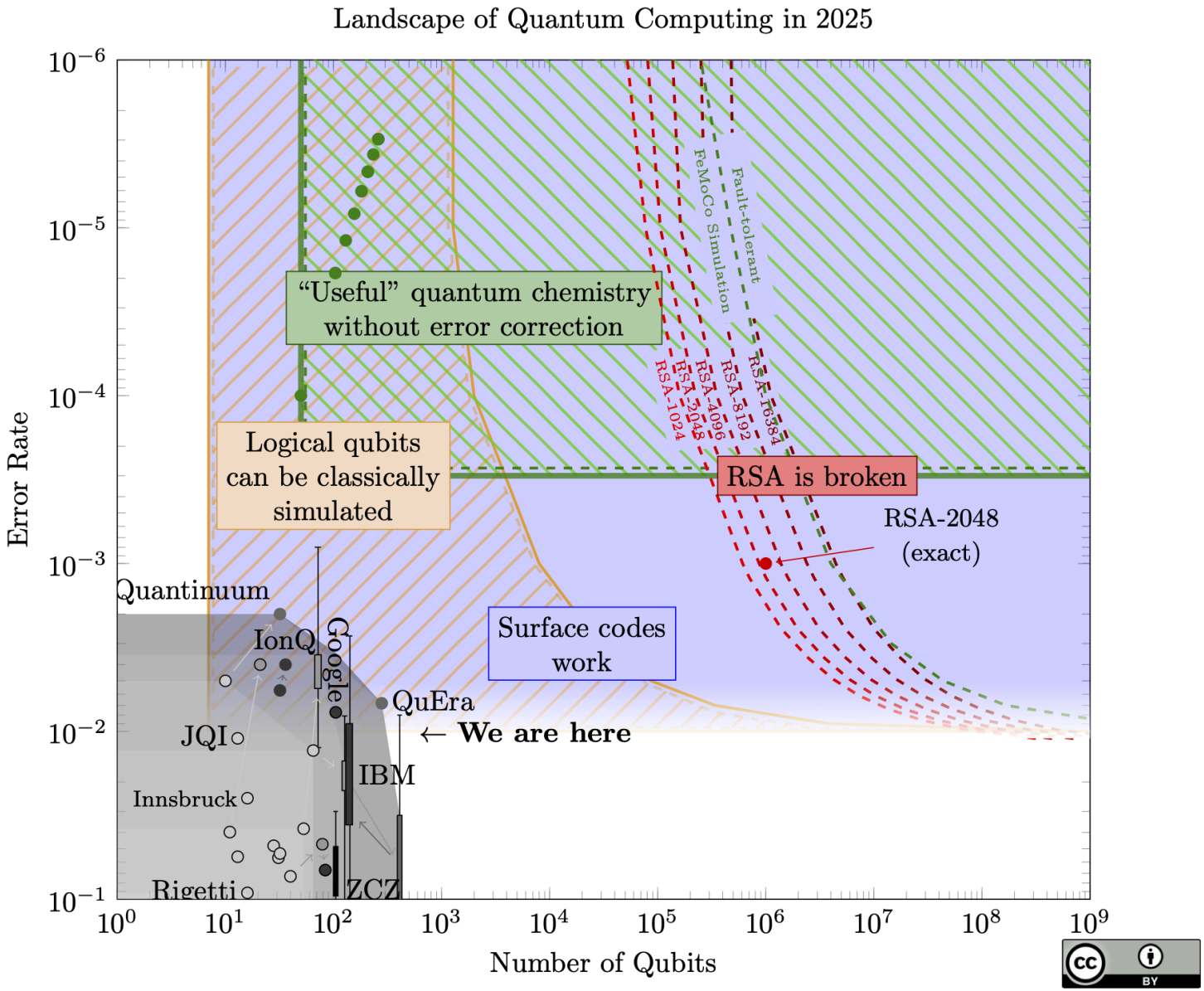
1994 Shor's algorithm

2001 Experimental realization of Shor's algorithm factors $15 = 3 \times 5$ using 7 qubits [Vandersypen *et al.*]

2016 NIST PQC competition launched

2019 Google 53-qubit Sycamore

2024 Google 105-qubit Willow chip



Source: Samuel Jaques

Quantum-safe cryptography

The hype?

- Quantum factorization records seem questionable[a]

[a] https://eprint.iacr.org/2025/1237.pdf

Quantum-safe cryptography

The hype?

- Quantum factorization records seem questionable[a]

[a] https://eprint.iacr.org/2025/1237.pdf

Hope for the best but prepare for the worst

Transition to quantum-safe cryptography

- Transition needs time
- Harvest now, decrypt later

QKD + OTP?

QKD + OTP?

ChatGPT 5.2: *QKD still needs authentication. "QKD + OTP" is not a self-contained magic shield. It's part of a larger system that must already be quantum-safe on the classical side.*
OTP is a bandwidth monster. ... long-distance QKD key rates can be orders of magnitude below what OTP would need for bulk traffic.
Use PQC + symmetric crypto...

Post-Quantum Cryptography (PQC)

- Design cryptosystems that can run on today's (classical) computers while being secure against quantum attacks
- An ongoing revolution of public key cryptography
- NIST PQC standardization process (2016-present)

Main PQC constructions

Approach	Key Size	Ciphertext/ Signature size	Speed	Maturity
Lattice	Medium(1–2 KB)	Medium(1–3 KB)	Fast	High
Code	Large(100+KB)	Medium	Slow	High
Hash (sig)	Small	Large	Medium	High
Multivariate(sig)	Large(100+ KB)	Small	Fast (sign) Slow (verify)	?
Isogeny	Small	Small	Slow	?

Current PQC standards

Organization	KEM	Digitial Signatures
NIST	Kyber, HQC	Dilithium, SPHINCS+, Falcon
BSI	FrodoKEM, Classic McEliece, Kyber	LMS/HSS, XMSS/XMSS^MT, Dilithium, SPHINCS+
ANSSI	FrodoKEM, Kyber	Dilithium, Falcon, XMSS/LMS, SPHINCS+
S. Korea	NTRU+, SMAUG-T	AIMer, HAETAE
China	upcoming	

Current PQC standards

Organization	KEM	Digitial Signatures
NIST	Kyber, HQC	Dilithium, SPHINCS+, Falcon
BSI	FrodoKEM, Classic McEliece, Kyber	LMS/HSS, XMSS/XMSS^MT, Dilithium, SPHINCS+
ANSSI	FrodoKEM, Kyber	Dilithium, Falcon, XMSS/LMS, SPHINCS+
S. Korea	NTRU+, SMAUG-T	AIMer, HAETAE
China	upcoming	

† NIST renames the schemes, *e.g.* Kyber $\rightarrow$ ML-KEM

Current PQC standards

Organization	KEM	Digitial Signatures
NIST	Kyber, HQC	Dilithium, SPHINCS+, Falcon
BSI	FrodoKEM, Classic McEliece, Kyber	LMS/HSS, XMSS/XMSS^MT, Dilithium, SPHINCS+
ANSSI	FrodoKEM, Kyber	Dilithium, Falcon, XMSS/LMS, SPHINCS+
S. Korea	NTRU+, SMAUG-T	AIMer, HAETAE
China	upcoming	

† NIST renames the schemes, *e.g.* Kyber $\rightarrow$ ML-KEM

What next?

Digital Signatures Standards: FIPS 186

1994-v0 DSA only

1998-v1 Includes RSA

2000-v2 Introduces ECDSA using P-256 with smaller keys than RSA/DSA

2009-v3 Deprecated weak key sizes ($<$112 bits of security)

2023-v5 EdDSA using Ed25519 and Ed448

`https://csrc.nist.rip/projects/digital-signatures`

The cryptographic life cycle

1. **Theoretical design**
   Hard problems and security proof
2. **Standardization**
   Public evaluation (e.g. NIST competition)
3. **Adoption**
   Implementations in TLS, OS, and chips etc.
4. **Weakening and Retirement**
   - Advancements in cryptanalysis and in computational power
   - Broken and weak algorithms eliminated

Post-quantum migration

Real-world deployment of PQC

- Optimize implementations of given schemes such as standards
  - Awareness of platforms
  - Resistance to implementation attacks
- New designs minimizing implementation costs
  - Explore the corners of design space
  - Domain-specific vs. general-purpose design

Lightweight cryptography

Symmetric cryptographic standards

AES, SHA3, SHAKE,...

Optimized for general-purpose platforms

Lightweight cryptographic standards

- Lightweight block cipher: PRESENT (ISO)
- Lightweight authenticated encryption and hash: ASCON (NIST)

Suitable for resource-constrained devices

Towards Efficient PQC: Costs

- Area
- Latency
- Power
- Energy

Towards Efficient PQC: Costs

- **Area** $\leftarrow$ This talk
- Latency
- Power
- Energy

Area

- ASIC = Application Specific Integrated Circuit
  - Gate Equivalents (GE): NAND gate
- FPGA = Field Programmable Gate Array
  - Look-up tables, FF, Block RAM
- Microcontrollers
  - Memory = code size + data size

Outline

NIST Additional Digital Signature Schemes: Round Two

"...additional general-purpose signature schemes based on a security assumption that did not use structured lattices as well as signature schemes with short signatures and fast verification."

- Lattice: HAWK
- Code: CROSS , LESS
- MPC-in-the-head: Mirath, MQOM, PERK, RYDE, SDitH
- Multivariate: MAYO, QR-UOV, SNOVA, UOV
- Supersingular elliptic curve isogeny: SQIsign
- Symmetric: FAEST

`https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`

NIST Additional Digital Signature Schemes: Round Two

"...additional general-purpose signature schemes based on a security assumption that did not use structured lattices as well as signature schemes with short signatures and fast verification."

- Lattice: HAWK
- Code: **CROSS**, LESS
- MPC-in-the-head: Mirath, MQOM, PERK, RYDE, SDitH
- Multivariate: MAYO, QR-UOV, SNOVA, UOV
- Supersingular elliptic curve isogeny: SQIsign
- Symmetric: FAEST

`https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`

CROSS: Introduction

- Fiat-Shamir transformation of ZK protocol

Restricted Syndrome Decoding Problem (RSDP)

Let $\mathbb{F}_p$ be a finite field containing $g \in \mathbb{F}_p^*$ of order z. Then given:

- A parity check matrix $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$
- A syndrome $\mathbf{s} \in \mathbb{F}_p^{n-k}$
- A restricted set $\mathbb{E} = \langle g \rangle = \{g^i \mid i \in \{1, \ldots, z\}\}$

Find a vector $\mathbf{e} \in \mathbb{E}^n$ such that

$$\mathbf{s} = \mathbf{e}\mathbf{H}^\top$$

RSDPG asks for a solution in a subgroup $G \leq \mathbb{E}^n$

CROSS ZK protocol

Private key: restricted vector $\mathbf{e} \in G$

Public key: group $G \leq \mathbb{E}^n$, parity-check matrix $\mathbf{H}$, syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$

PROVER		VERIFIER
Sample $\mathtt{Seed} \xleftarrow{\$} \{0;1\}^\lambda$, $(\mathbf{u}', \mathbf{e}') \xleftarrow{\mathtt{Seed}} \mathbb{F}_q^n \times G$ \\Randomness		
Compute $\mathbf{d} \in G$ such that $\mathbf{d} \star \mathbf{e}' = \mathbf{e}$ \\$\mathbf{d}$ is uniformly random over G		
Set $\mathbf{u} = \mathbf{d} \star \mathbf{u}'$ and $\widetilde{\mathbf{s}} = \mathbf{u}\mathbf{H}^\top$		
Set $c_0 = \mathsf{Hash}(\widetilde{\mathbf{s}}, \mathbf{d})$, $c_1 = \mathsf{Hash}(\mathbf{u}', \mathbf{e}')$ \\Commitments		
	$\xrightarrow{(c_0, c_1)}$	
	$\xleftarrow{\beta}$	Sample $\beta \xleftarrow{\$} \mathbb{F}_q^*$
Compute $\mathbf{y} = \mathbf{u}' + \beta\mathbf{e}'$ \\Uniformly random over $\mathbb{F}_q$		
Set $h = \mathsf{Hash}(\mathbf{y})$ \\First response		
	$\xrightarrow{h}$	
		Sample $b \xleftarrow{\$} \{0, 1\}$
	$\xleftarrow{b}$	
If $b = 0$, set $\mathtt{rsp} = (\mathbf{y}, \mathbf{d})$ \\Second response (the larger one)		
If $b = 1$, set $\mathtt{rsp} = \mathtt{Seed}$ \\Second response (the shorter one)		
	$\xrightarrow{\mathtt{rsp}}$	
		Verify c_b using $\mathtt{rsp}$

- The signature consists of the transcripts of each of the rounds
- Main building blocks: matrix multiplication, hash trees

Parameter set of all variants of CROSS

Algorithm and security category	Dependent on opt. corner					Independent of opt. corner	
	p	z	n	k	m	t (fast/balanced/small)	w (fast/balanced/small)
RSDP 1	127	7	127	76	-	157/256/520	82/215/488
RSDP 3			187	111		239/384/580	125/321/527
RSDP 5			251	150		321/512/832	167/427/762
RSDPG 1	509	127	55	36	25	147/256/512	76/220/484
RSDPG 3			79	48	40	224/268/512	119/196/463
RSDPG 5			106	69	48	300/356/642	153/258/575

Memory issue for embedded systems

- NIST requests PQC algorithms to test and benchmark on the embedded platforms ARM Cortex-M4[1]
- A commonly used testing board such as STM32-L4R5ZI has 640KB of RAM and 2MiB of flash memory
- CROSS reference implementation fails to run on board for larger parameters due to large memory footprint

Memory Usage by 'Sign' Measured at Runtime Using the pqm4 Library (KB)						
Security Level	**1**		**3**		**5**	
Optimisation Corner/Hard Problem	**RSDP**	**RSDPG**	**RSDP**	**RSDPG**	**RSDP**	**RSDPG**
fast	116	78	258	170	463	301
balanced	210	157	463	241	826	423
small	421	310	695	453	1332	755

Additionally, even more restricted embedded applications may have as little as 32KB of RAM.

[1] `https://github.com/mupq/pqm4`

Low-memory implementation of CROSS

- Minimize the runtime memory requirements
  - GGM and hash trees
  - Matrix multiplications
  - Reuse and recalculation of variables
- Maintain similar speed as reference
  - Use Cortex-M4's built-in Digital Signal Processing (DSP) instruction set to boost arithmetic operations
- Do not introduce any timing side-channel attacks

Optimizing tree algorithms for ZK proofs

Merkle tree and GGM tree are the largest consumers of memory

- GGM tree generates random round seeds from a single seed
- Merkle tree computes commitments in the ZK proofs

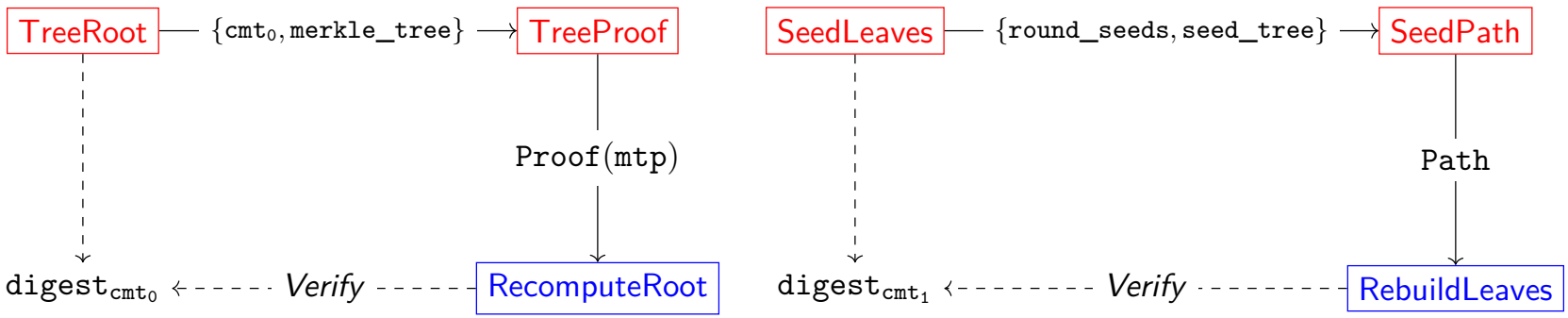


Figure 1: The functions in red are used in CROSS_Sign while those in blue are part of verification.

Merkle tree in CROSS

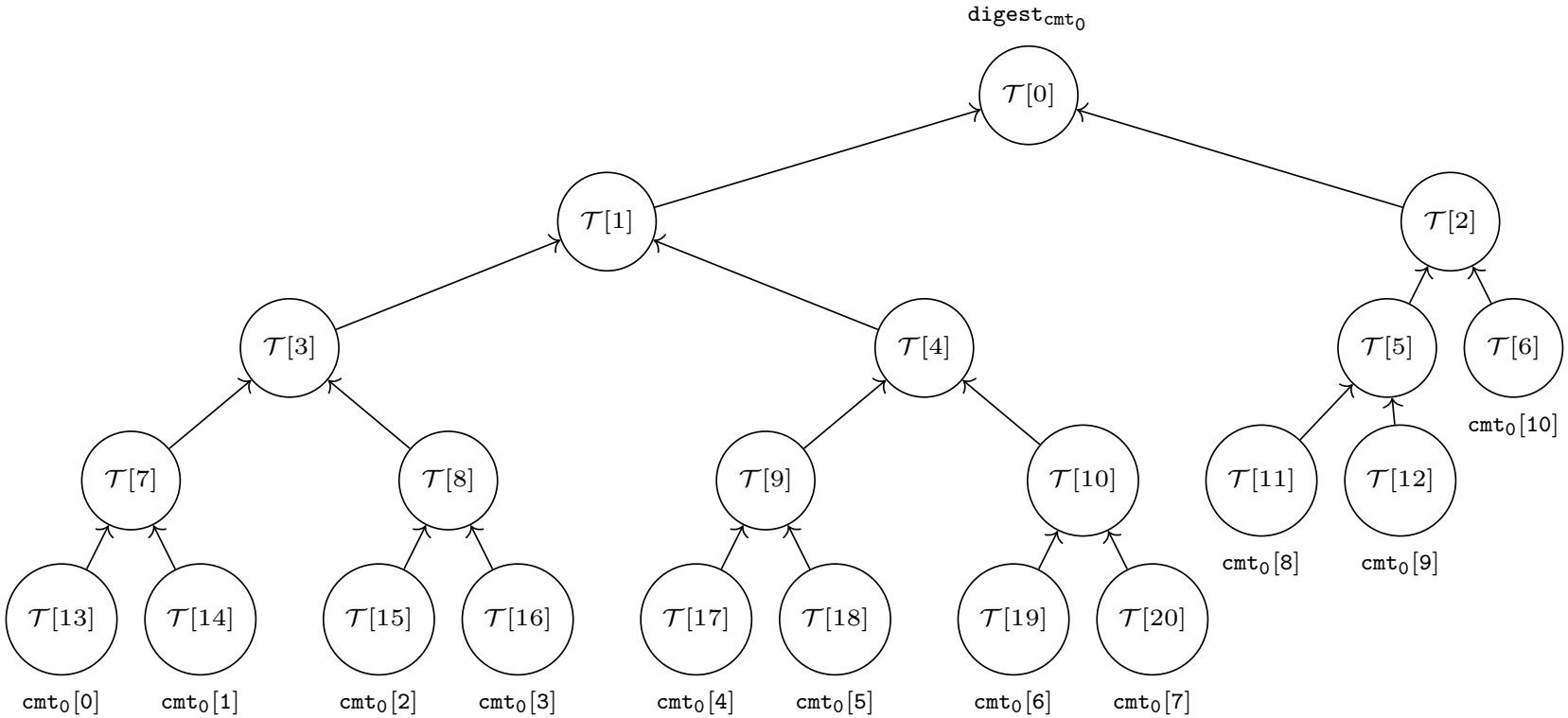


Figure 2: Example Merkle tree for $t = 11$

Lightening CROSS: on-the-fly TreeRoot

Key ideas

- To compute the root, it is not necessary to store the full hash tree
- If we visit the leaves in order (either right-to-left or left-to-right) any time we are holding two nodes n_1, n_2 from the same level, we can calculate their parents by Hash $(n_1 \mid n_2)$, and discard the children
- The intermediate nodes can be stored in a static array

Results

We reduce total memory complexity from $(4t-2)\lambda$ to $2\log_2(t)\lambda + \log_2(t)$. For instance, for RSDP-1 small, from over over 250KB to less than 2.5KB

Merkle tree in hash based signatures

A classical algorithm for Merkle signature scheme

Algorithm 2.1 Treehash

Input: Height $H \geq 2$
Output: Root of the Merkle tree

1. **for** $j = 0, \ldots, 2^H - 1$ **do**
   a) Compute the jth leaf: $\text{NODE}_1 \leftarrow \text{LEAFCALC}(j)$
   b) **While** NODE_1 has the same height as the top node on STACK **do**
      i. Pop the top node from the stack: $\text{NODE}_2 \leftarrow \text{STACK.pop}()$
      ii. Compute their parent node: $\text{NODE}_1 \leftarrow g(\text{NODE}_2 \| \text{NODE}_1)$
   c) Push the parent node on the stack: $\text{STACK.push}(\text{NODE}_1)$
2. Let R be the single node stored on the stack: $R \leftarrow \text{STACK.pop}()$
3. **Return** R

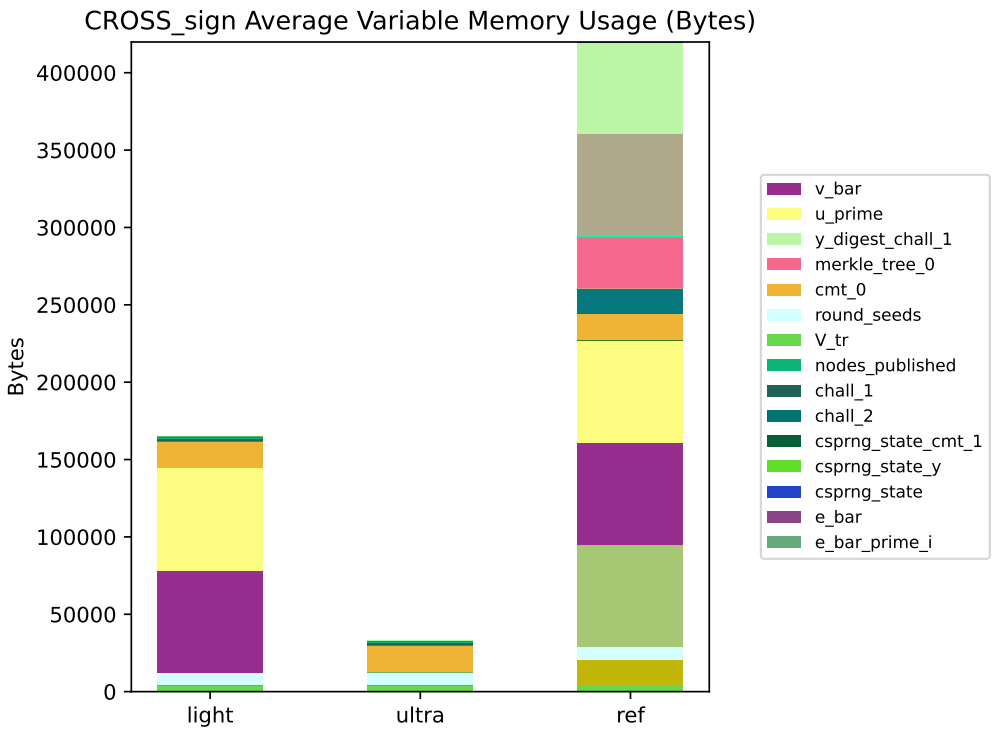
What's new?

- Do not introduce *Node* and *Stack* but leverage existing helper arrays
- We devise a static array that encodes the depth of a node structurally instead of storing it

Light and Ultralight implementations

Reuse and recalculation of variables

- If $x = y + z$, then only store x, y, compute z if necessary
- By execution flow analysis, discard some intermediate values
- Generate big matrices on-the-fly every time we need them



Implementation results

Our implementation reduces the memory footprint of Key Generation, Signature Generation, and Verification of the CROSS reference code by as much as 95%, 92%, and 85%, respectively

Table 1: Improvements over reference implementation of the CROSS RSDP-1 small variant on ARM CortexM4.

CROSS-RSDP-I small variant	Steps	Memory (bytes)			Performance (kcycles)		
		KG (diff [%])	Sign (diff [%])	Verify (diff [%])	KG (diff [%])	Sign (diff [%])	Verify (diff [%])
Reference [1]	-	8,112	421,392	223,832	400	76,710	36,495
Our optimizations	OPT_KEYGEN	1,120 (86.19)	421,392 (0)	223,896 (-0.03)	518 (-29.5)	78,764 (-2.68)	36,242 (0.69)
	OPT_MERKLE	1,120 (0)	404,744 (3.95)	207,384 (7.37)	518 (0)	76,299 (3.13)	36,413 (-0.47)
	OPT_HASH_CMT1	1,120 (0)	388,584 (3.99)	190,984 (7.91)	518 (0)	76,862 (-0.74)	36,505 (-0.25)
	OPT_HASH_Y	1,120 (0)	264,608 (31.9)	67,048 (64.89)	518 (0)	79,447 (-3.36)	36,326 (0.49)
	OPT_V_BAR	1,120 (0)	198,696 (24.91)	67,048 (0)	518 (0)	79,353 (0.12)	36,325 (0)
	OPT_E_BAR_PRIME	1,120 (0)	198,696 (0)	67,048 (0)	518 (0)	77,142 (2.79)	36,327 (-0.01)
	OPT_OTF_MERKLE	1,120 (0)	183,080 (7.86)	50,424 (24.79)	518 (0)	81,752 (-5.98)	36,699 (-1.02)
	OPT_GGM	1,120 (0)	168,352 (8.04)	50,424 (0)	518 (0)	86,206 (-5.45)	36,700 (0)
	OPT_RECOMPUTE_ROOT	1,120 (0)	168,352 (0)	35,340 (29.91)	518 (0)	86,229 (-0.03)	36,422 (0.76)
	OPT_DSP	1,120 (0)	168,360 (0)	35,252 (0.25)	518 (0)	68,236 (20.87)	35,514 (2.49)
	OPT_Y_U_OVERLAP	1,120 (0)	168,216 (0.09)	35,252 (0)	518 (0)	68,117 (0.17)	35,511 (0.01)
	OPT_KEYGEN_BLOCKS	1,296 (-15.71)	168,216 (0)	35,252 (0)	438 (15.44)	68,069 (0.07)	35,512 (0)
	OPT_U_PRIME_EPH	1,296 (0)	36,752 (78.15)	35,252 (0)	438 (0)	85,421 (-25.49)	35,511 (0)
	OPT_U_V_VERIFY	1,296 (0)	36,752 (0)	35,132 (0.34)	438 (0)	85,438 (-0.02)	35,514 (-0.01)

[1] https://github.com/secomms/RBG

Our work

Harry Hart, Puja Mondal, Suparna Kundu, Supriya Adhikary, Angshuman Karmakar, Chaoyun Li. *Lightening CROSS: Memory Optimized Implementations of CROSS*, to appear in CHES 2026 (2).

Paper available at `https://eprint.iacr.org/2024/1929.pdf`,

Artifacts will be online soon.

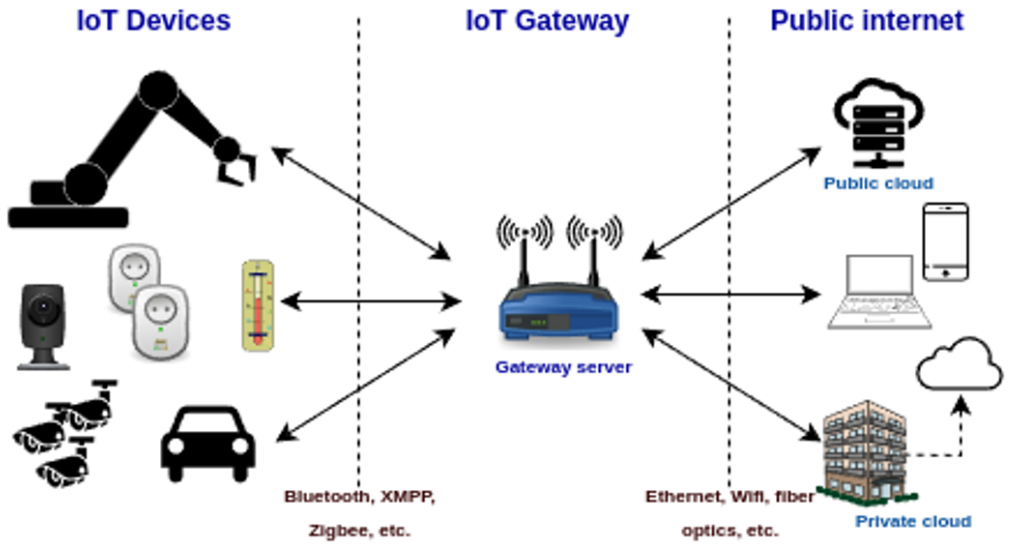
Follow-up works

- A new paper on memory-efficient implementation of CROSS[2]
- Apply our techniques to other NIST round two signature schemes
  - LESS
  - FAEST

[2] J. Schupp et al. Optimizing the Post Quantum Signature Scheme CROSS for Resource Constrained Devices. https://eprint.iacr.org/2025/1928

Outline

A use case of PQ KEMs



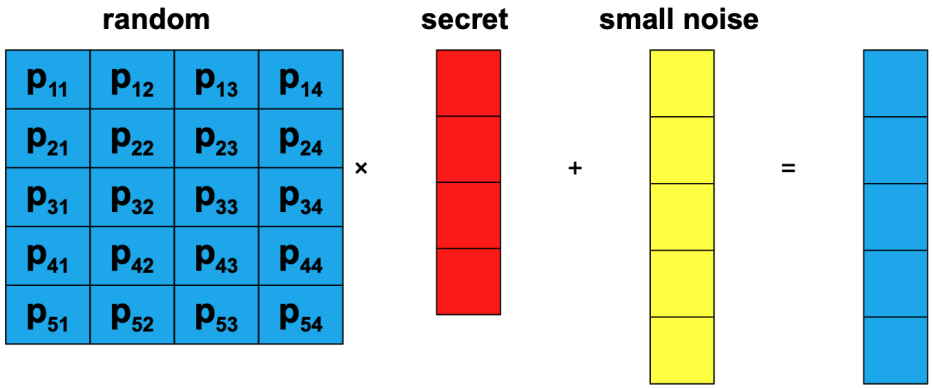
- IoT gateway connects IoT devices and Internet
- Gateway server runs authentication and authorization protocols serving numerous devices simultaneously
  - High throughput and low latency
- An IoT device connects to the gateway server occasionally
  - Low area and low power

Lightweight PQ KEM: Rudraksh

- Kyber-*esque*
- Post-quantum CCA-secure key-encapsulation mechanism
- Lightweight hardware implementation
  - Minimize area on FPGA

S. Kundu, A. Ghosh, A. Karmakar, S. Sen and I. Verbauwhede: *Rudraksh: A Compact and Lightweight Post-quantum Key-Encapsulation Mechanism*, CHES 2025.

The hard problem: Module-LWE



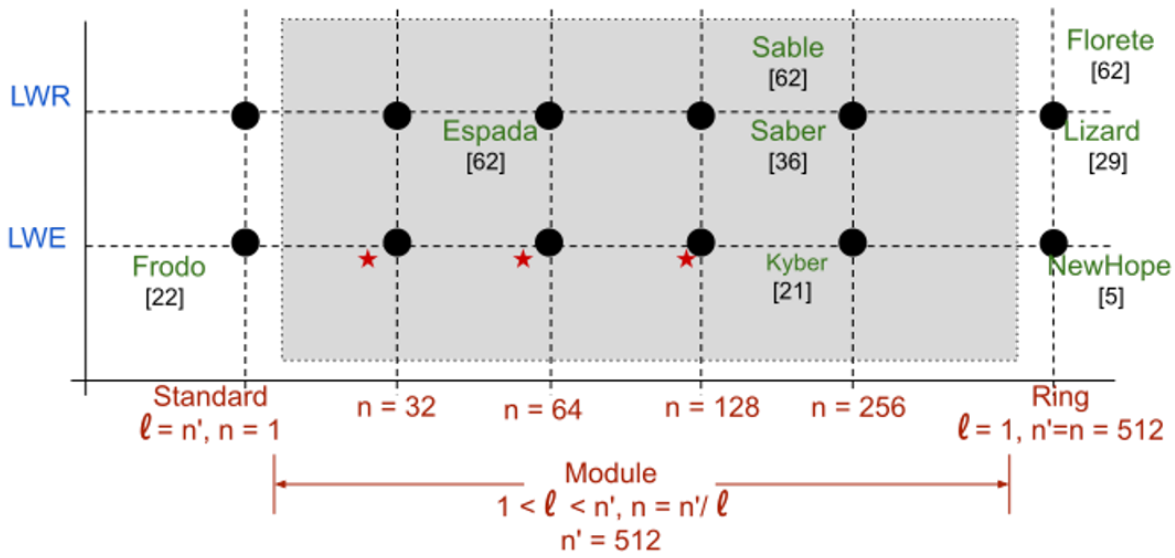
every matrix entry is a polynomial in $\mathbb{Z}_q[x]/(x^n + 1)$

Search Module-LWE problem: given **blue**, find **red**

- Trade-off between standard (FrodoKEM) and ring lattices (NewHope)
- Efficient polynomial multiplication by NTT
- Security reduction similar to Kyber

Figure from Douglas Stebila

Module space



n is polynomial size, ℓ vector size, and $n' = n\ell$ the dimension of lattice

- Smaller n, lower memory, but worse performance
- $n = 64$ yields the lowest-memory design

Scheme name	Module Parameter		Primary modulus		Compression modulus		CBD parameter		Encoding	Bit-security	Failure probability
	ℓ	n	q	$\lceil \log_2 q \rceil$	$\lceil \log_2 p \rceil$	$\lceil \log_2 t \rceil$	η_1	η_2	B	(Quantum, Classical)	
KEM-poly32	21	32	31873	15	12	3	2	2	4	(105, 116)	-113
KEM-poly64	9	64	7681	13	10	3	2	2	2	(104, 114)	-128
KEM-poly128	4	128	3329	12	10	2	2	2	1	(101, 111)	-179
Kyber [ABD+21]	2	256	3329	12	10	3	3	2	1	(107, 118)	-139
NewHope [ADPS16]	1	512	12289	14	14	2	4	4	1	(101, 112)	-213

The hash function

- Hash and extended output function (XOF) take 50% time and area in most cases
- Use a lightweight hash: ASCON, the winner of the NIST lightweight cryptography competition

Scheme	**Area** (KGE)	Frequency (MHz)	Throughput (Gbps)	Throughput per area (Mbps/KGE)
ASCON	**24.437**	**336.7**	**43.098**	**1763**
Keccak	289.92	93.897	126.20	435.28

Performance

Schemes	LUT	FF	Slice	BRAM	DSP	ENS[3]
Rudraksh	2896	1413	9	1.5	1	1112
Kyber[4]	4993	2765	1452	205	0	3191

- Area: Rudraksh provides a $3\times$ improvement to opt. Kyber
- Time-area-product: Rudraksh improves $2\times$ to SOTA Kyber

[3] ENS (equivalent number of slices) = Slice + DSP$\times$100 + BRAM$\times$196 + LUT/4

[4] S. He, H. Li, F. Li, and R. Ma. A lightweight hardware implementation of CRYSTALS-Kyber. Journal of Information and Intelligence, 2(2):167–176, 2024.

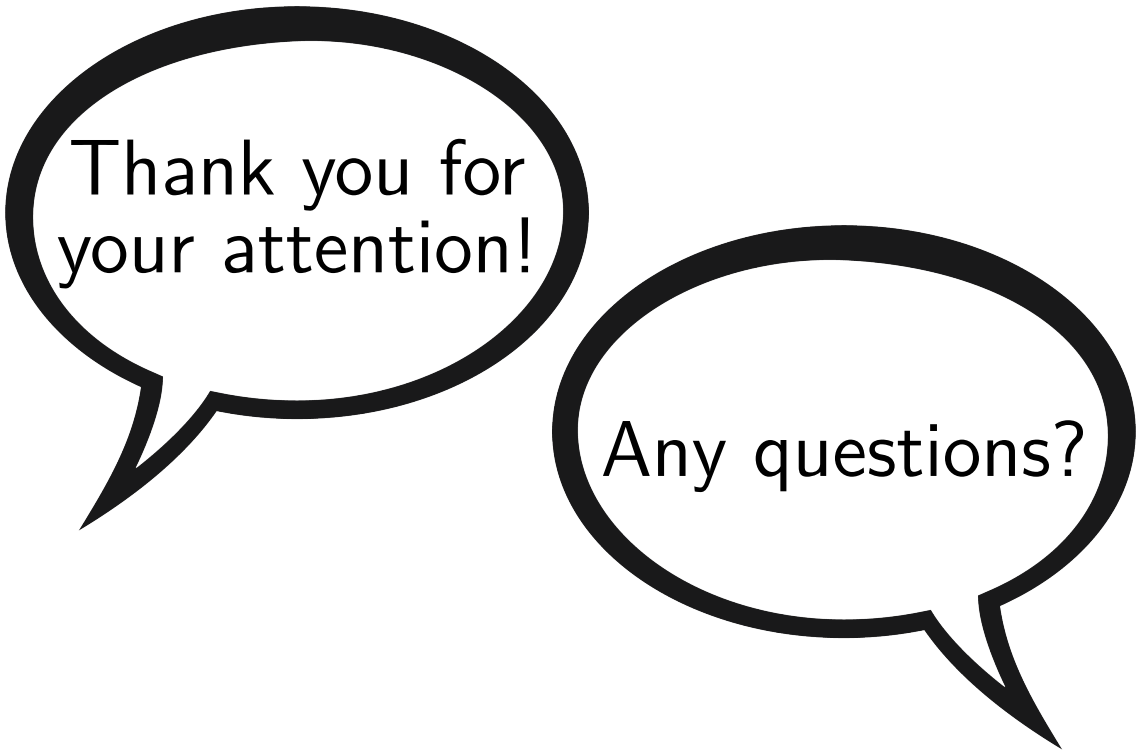
Towards Rudraksh 2.0

- Rudraksh is not the end but a demonstrator of the lightweight design strategies
- Make it smaller on microcontrollers
- Chinese PQC competition requires a different hash: the benefits of ASCON lost
- Lightweight Fault and SCA countermeasures

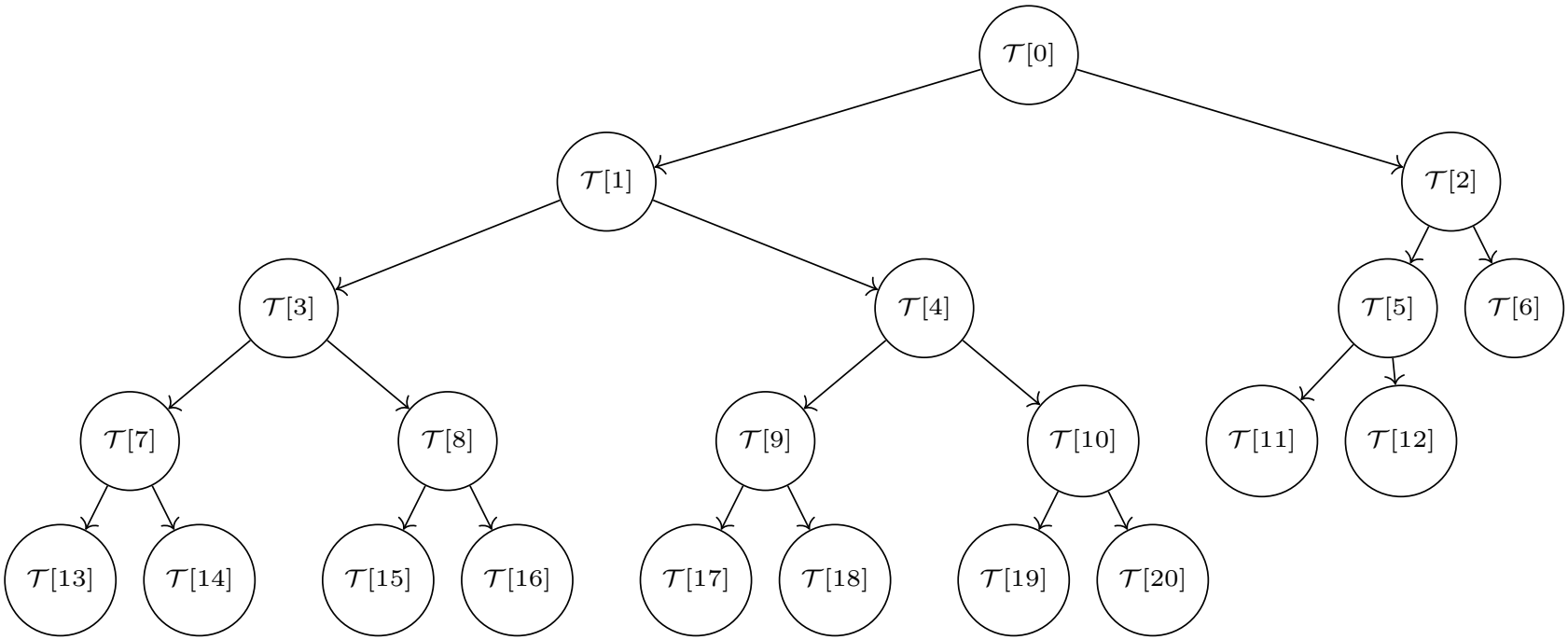
Outline

Summary

- Post-quantum migration calls for efficient PQC for real-world applications
- We propose techniques for low-memory implementations of signature schemes
- We present a case study of a lightweight KEM for hardware platforms
- Achieving lightweight PQC requires us to think beyond current standards


Thank you for your attention!
Any questions?

Lightening CROSS: GGM Tree



A GGM tree which generates 11 rounds from one seed $\mathcal{T}[0]$

Lightening CROSS: Variable Benchmark

The breakdown of variables' average memory usage over the lifetime of the algorithm

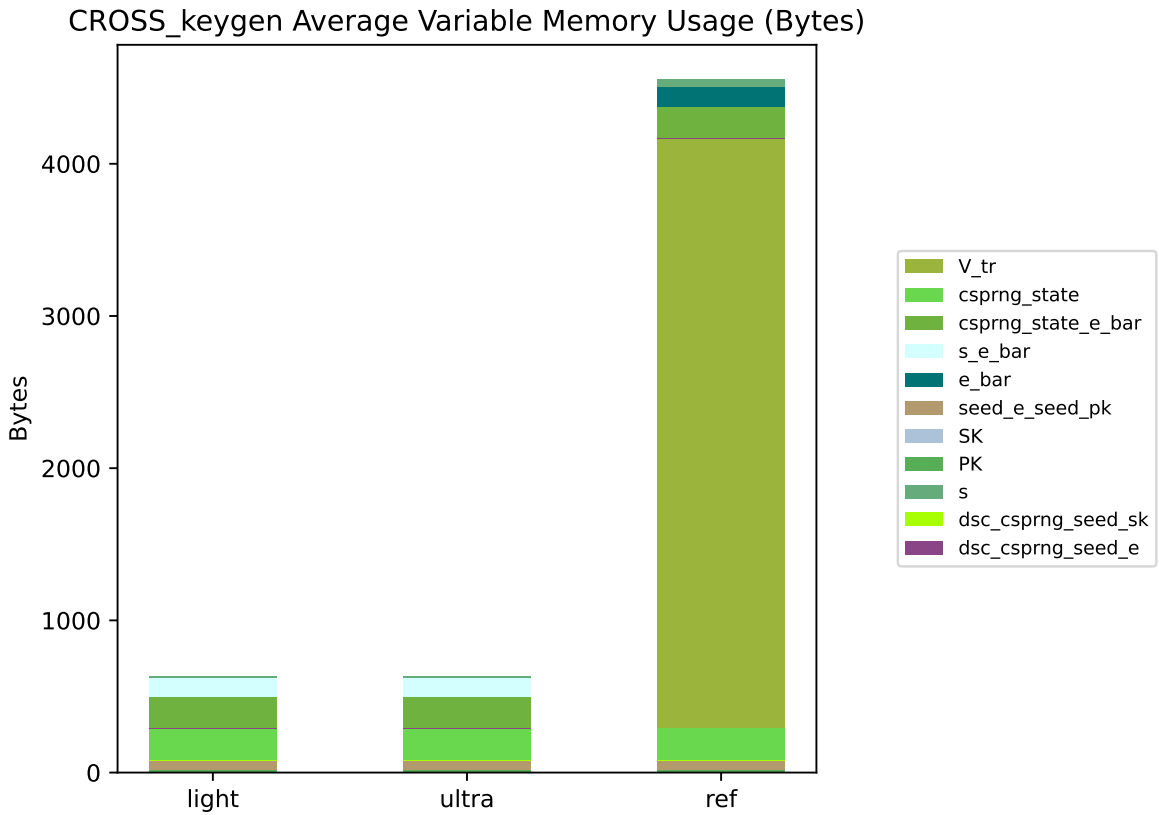


Figure 3: Top 15 memory consumers for the RSDP-1-small variant

Lightening CROSS: Variable Benchmark

The breakdown of variables' average memory usage over the lifetime of the algorithm

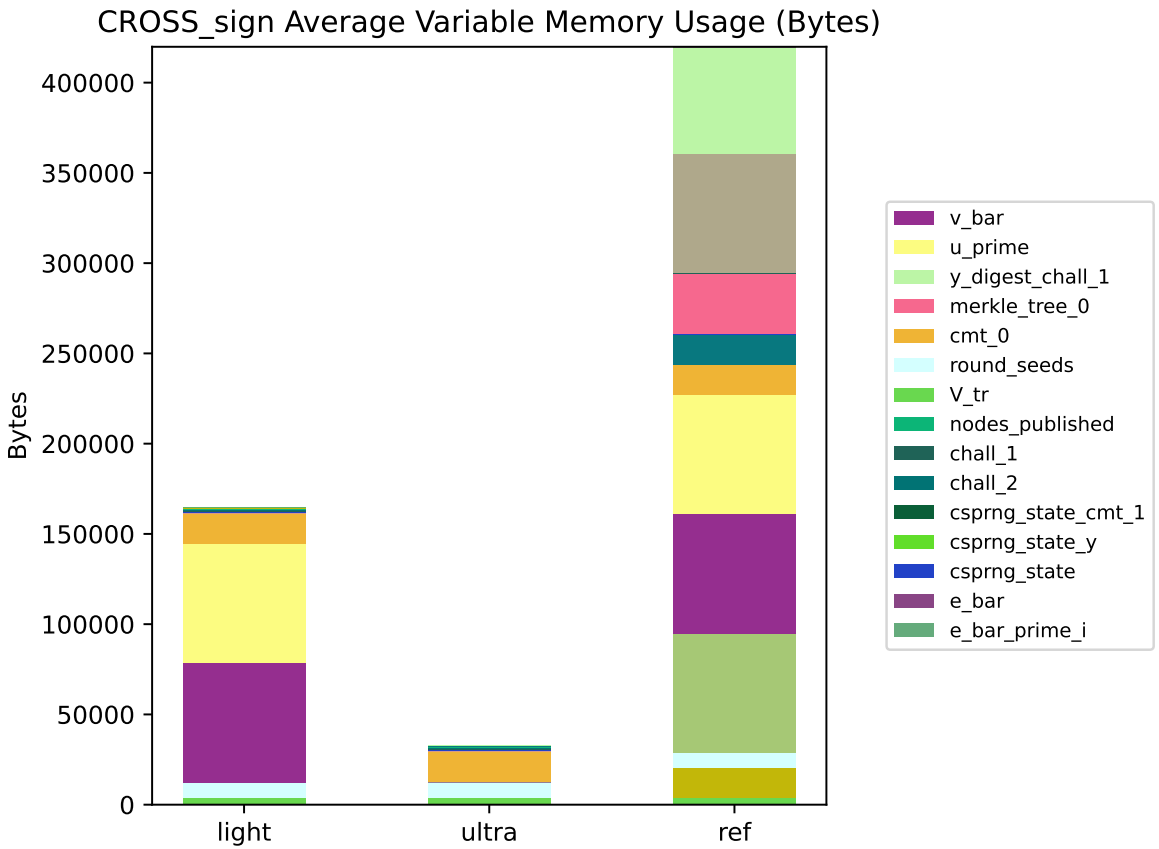


Figure 3: Top 15 memory consumers for the RSDP-1-small variant

Lightening CROSS: Variable Benchmark

The breakdown of variables' average memory usage over the lifetime of the algorithm

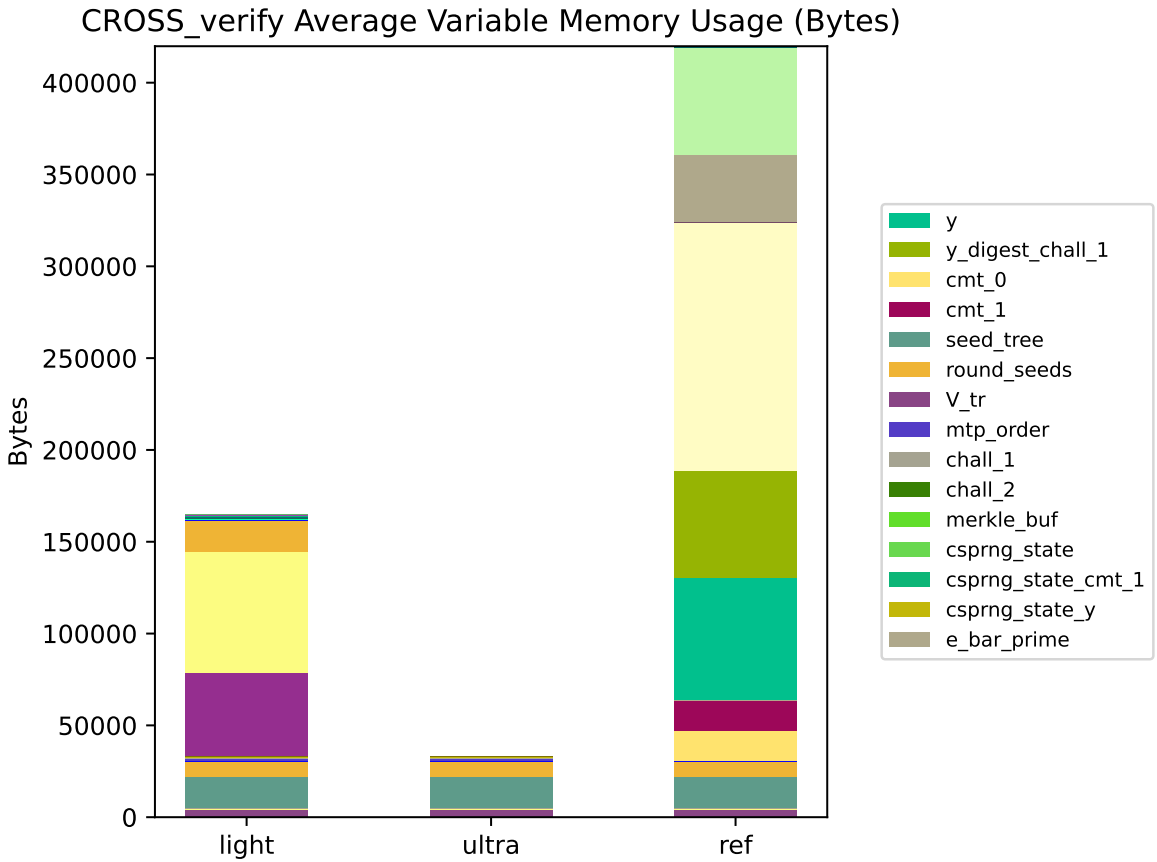


Figure 3: Top 15 memory consumers for the RSDP-1-small variant

The hard lattice problems

- FrodoKEM: LWE
  - Expensive multiplication
  - Large keys
- NewHope: Ring-LWE
  - Fast polynomial multiplication
  - Smaller keys
  - Weaker due to ideal structure?
- Kyber: Module-LWE