LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4

12th International Conference on Cryptology, Africacrypt 2020

Fabio Campos[1] Tim Kohlstadt[1] Steffen Reith[1] Marc Stöttinger[2]

July 19, 2020

[1]RheinMain University of Applied Sciences, Germany

[2]Continental AG, Germany

Motivation

Assumptions

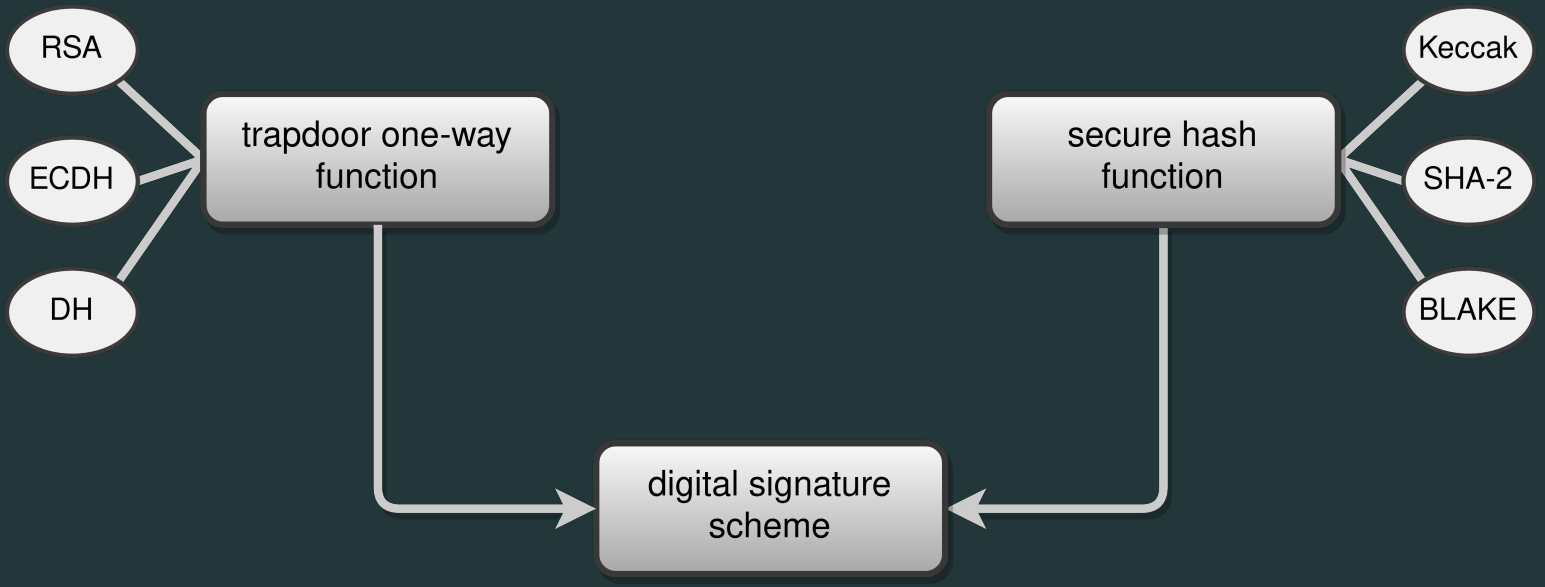

Fig.1: Assumptions of schemes used in practice today.

Quantum impact

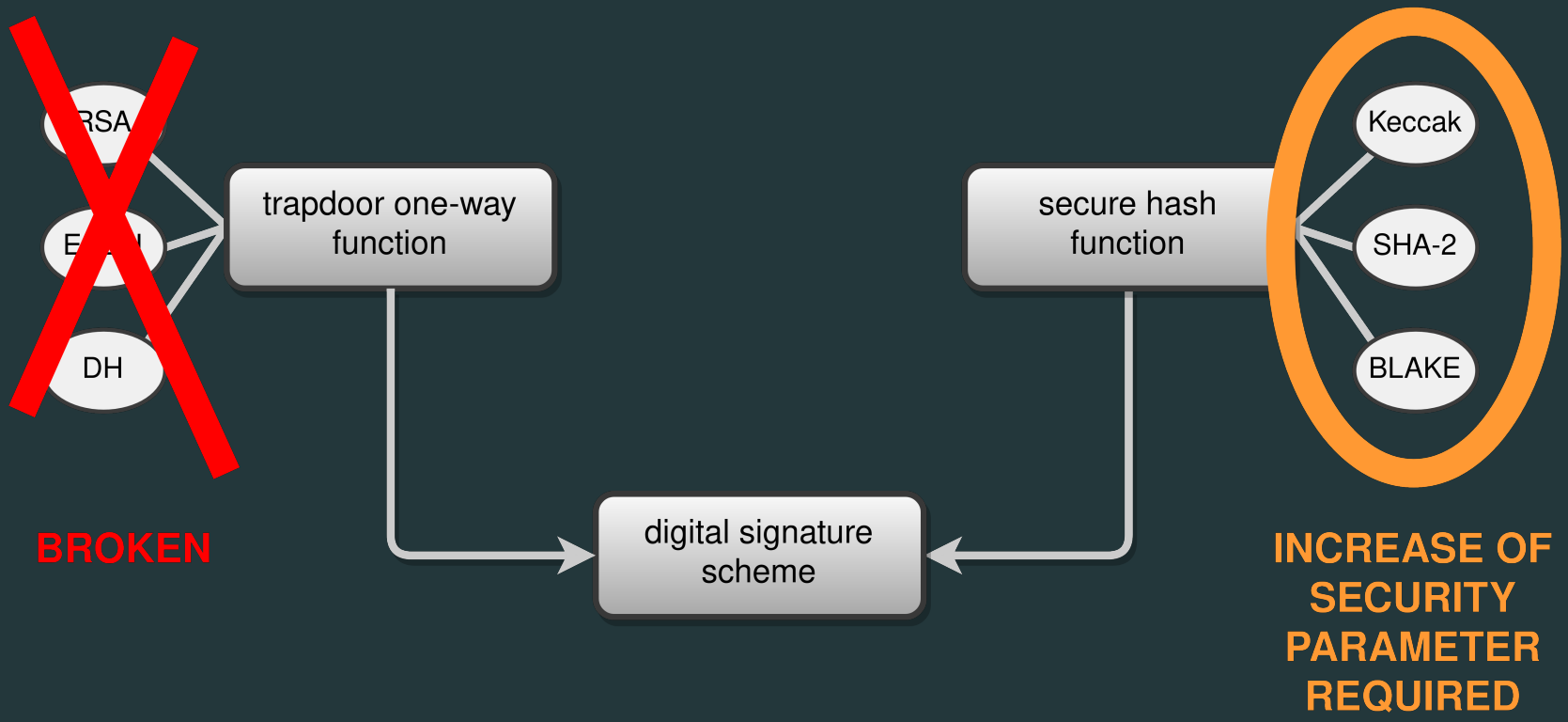

Fig.2: Due to Shor's and Grover's algorithm (and variants).

The NIST PQC (not a) competition[1]

- 2016: NIST calls for proposals for key encapsulation and digital signatures
- 2017: 69 schemes accepted for the first round of evaluation
- 01.2019: 26 schemes (9 digital signature) advance to round 2
- 08.2019: Second NIST PQC Standardization Conference
- 2022-2024: NIST PQC standards

[1] `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`

Recommendation[2] for stateful hash-based signature schemes

- NIST: *" ... NIST is proposing to supplement FIPS 186 by approving the use of two stateful hash-based signature schemes: the eXtended Merkle Signature Scheme (XMSS) and the Leighton-Micali Signature system (LMS) ... Stateful hash-based signature schemes are not suitable for general use since they require careful state management in order to ensure their security. ... An application that may fit this profile is firmware updates for constrained devices."*
- We: *"So, let's try it!*

[2] `https://csrc.nist.gov/News/2019/draft-sp-800-208-stateful-hash-based-sig-schemes`

Embedded PQC

- pqm4[3]: Post-quantum crypto library for the ARM Cortex-M4
- STM32F4DISCOVERY-Board
  - ARM Cortex-M4 (recommended by NIST for PQC evaluation)
  - 32-bit, 192 KiB RAM, 168 MHz
  - ARMv7E-M
  - cheap ($< \$30$)
- Challenge: Do LMS/XMSS even fit in limited RAM + Flash?

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

Background

Many-time Signature Schemes

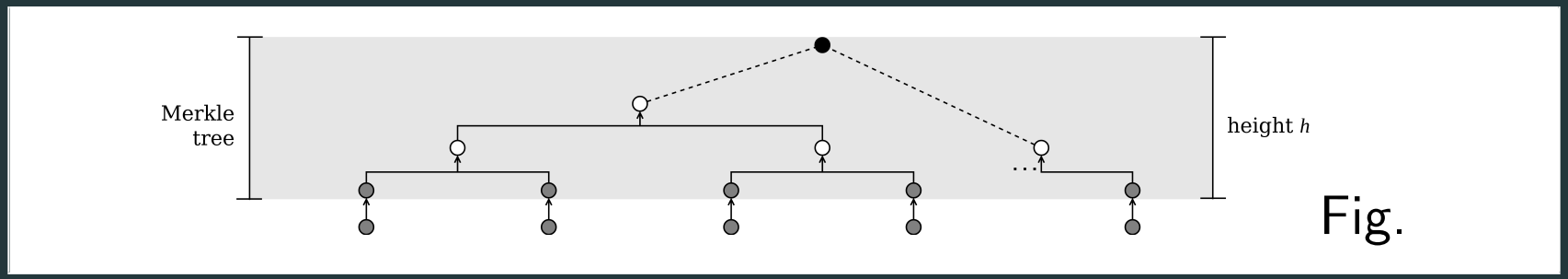

Fig.

Fig.3: Balanced binary tree (Merkle Tree) enables the use of a single public key (root of the tree) for verifying several messages. Grey nodes represents the one-time signatures. LMS and XMSS use variants of the Winternitz One-time Signature Scheme (WOTS).

Construction

Tweakable hash function

Definition 1: *Let* $n, \alpha \in \mathbb{N}, \mathcal{P}$ *be the public parameters space, and* $\mathcal{T}$ *be the tweak space. A tweakable hash function is an efficient function*

$$\mathsf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{\alpha} \to \{0,1\}^{n}, \qquad \mathsf{MD} \leftarrow \mathsf{Th}(P, T, M)$$

mapping an α*-bit message* M *to an* n*-bit hash value* MD *using a public parameter* $P \in \mathcal{P}$*, also called function key, and a tweak* $T \in \mathcal{T}$*.*

LMS / prefix construction

Construction 1*: Given a hash function* $H : \{0,1\}^{2n+\alpha} \to \{0,1\}^n$, *we construct* Th *with* $\mathcal{P} = \mathcal{T} = \{0,1\}^n$, *as*

$$\mathsf{Th}(P, T, M) = H(P||T||M).$$

XMSS / prefix and bitmask construction

Construction 2*: Given two hash functions*
$H_1 : \{0,1\}^{2n} \times \{0,1\}^{\alpha} \to \{0,1\}^{n}$ *with* $2n$*-bit keys, and*
$H_2 : \{0,1\}^{2n} \to \{0,1\}^{\alpha}$, *we construct* Th *with* $\mathcal{P} = \mathcal{T} = \{0,1\}^{n}$, *as*

$$\mathsf{Th}(P, T, M) = H_1(P||T, M^{\oplus}), \text{ with } M^{\oplus} = M \oplus H_2(P||T).$$

XMSS / WOTS public key compression with L-trees

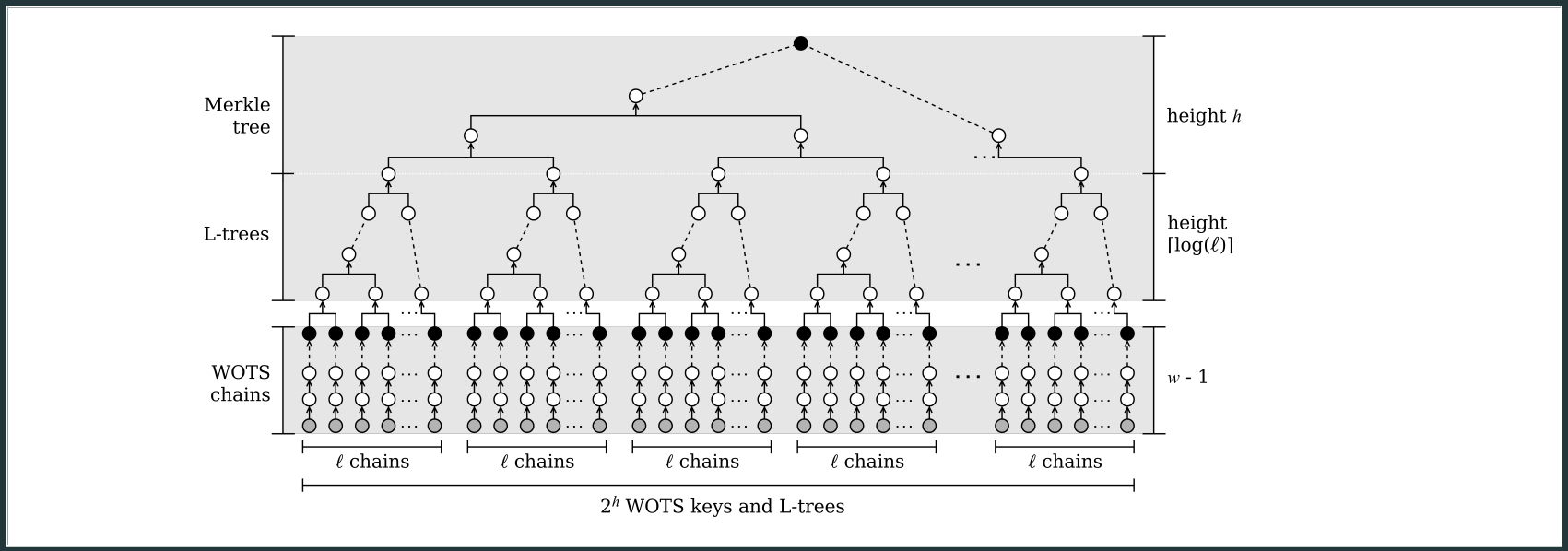

Fig.4: Overview with L-trees and WOTS chains. Grey nodes are the private keys and the black nodes the public keys of the WOTS chains. The black node at the top is the public key.

LMS / WOTS public key compression w/o L-trees

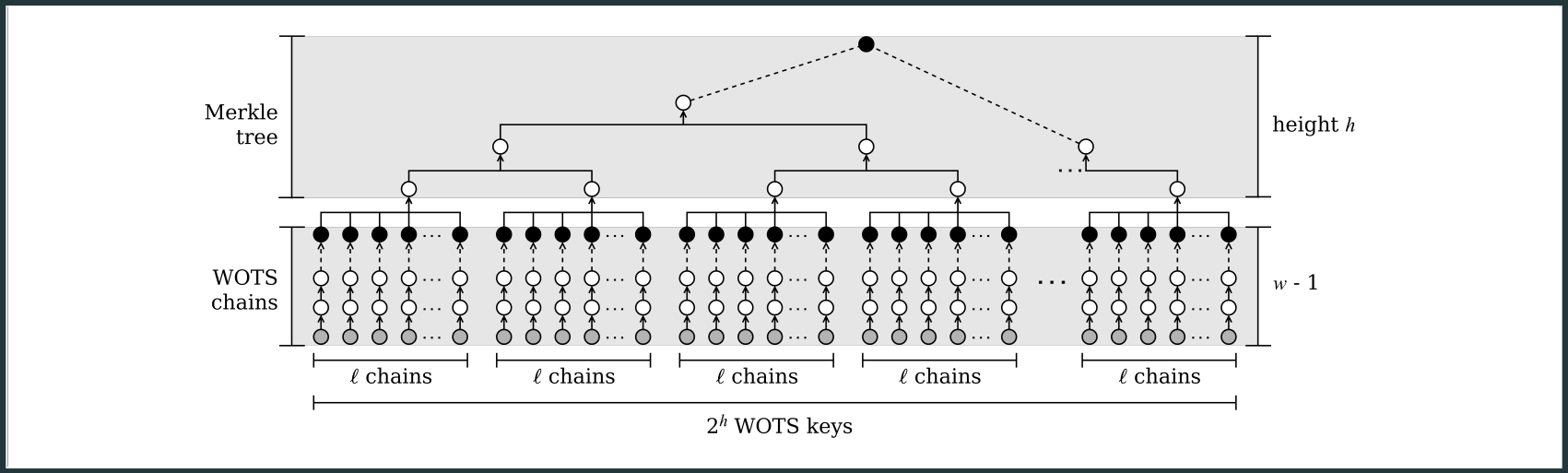

Fig.5: Overview without L-trees. Grey nodes are the private keys and the black nodes the public keys of the WOTS chains. The black node at the top is the public key.

Speeding up XMSS

Hash pre-computation

For a given key pair and a security parameter n, the first $2n$-bit block of the input to the pseudo-random function is the same for all calls.

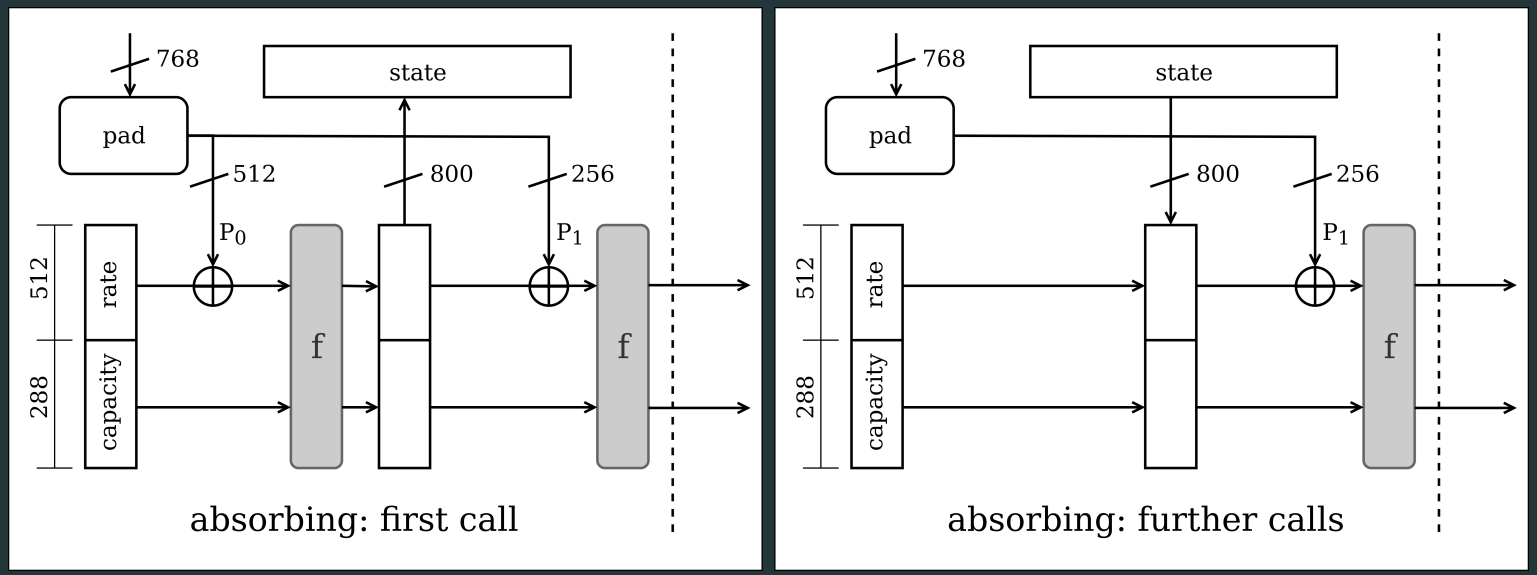

Fig.6: Hash pre-computation within KECCAK-f[800] with a rate of 512 bits.

Implemented variants of XMSS

Based on the different constructions presented, we implemented and evaluated the following XMSS variants:

design	multi-tree	tree-less WOTS	bitmask-less hashing[4]	pre-computation
XMSS_ROBUST				
XMSS_SIMPLE		×	×	
XMSS_SIMPLE+PRE		×	×	×
XMSS^{MT} ROBUST	×			
XMSS^{MT} SIMPLE	×	×	×	
XMSS^{MT} SIMPLE+PRE	×	×	×	×

[4] ≈ Construction 1: LMS / prefix construction

Evaluation

Setup

- STM32F4DISCOVERY board
- reference implementation of LMS[5] and XMSS[6]
- based on pqm4 framework
- optimised assembly implementations of:
  - Gimli-Hash
  - Keccak (Keccak-$p[800, 22]$ and Keccak-$p[800, 12]$)
  - SHAKE256, and
  - SHA-256

[5] `https://github.com/cisco/hash-sigs`, commit 5efb1d0

[6] `https://github.com/joostrijneveld/xmss-reference`, commit fb7e3f8

Selected parameter sets 1/2

symbol	meaning	XMSS	LMS
n	security parameter $\simeq$ length of the hash digest (in bits)	n	n
h	height of the tree or hypertree in a multi-tree variant	h	h
d	number of Merkle Trees in the multi-tree variant	d	L
w	Winternitz parameter	w	2^w
ℓ	number of Winternitz chains used in a single OTS operation	len	p

Selected parameter sets 1/2

scheme	n	w	h	layer	signature size (bits)
LMS	256	16	5	1	2352
LMS	256	256	5	1	1296
LMS	256	16	10	1	2512
LMS	256	256	10	1	1456
XMSS	256	16	5	1	2340
XMSS	256	16	10	1	2500
HSS	256	16	10	2	4756
HSS	256	256	10	2	2644
XMSS^{MT}	256	16	10	2	4642

HSS = multi-tree LMS (Hierarchical Signature System)

XMSS^{MT} = multi-tree XMSS

Speedup in XMSS and XMSS^{MT} exemplary with SHA-256

design	w	h	layer	key gen	sign	verify
XMSS_ROBUST	16	5	1	738.46	747.85	13.84
XMSS_SIMPLE	16	5	1	243.25	247.72	3.20
			speedup factor	3.03	3.01	4.32
XMSS_SIMPLE+PRE	16	5	1	237.27	241.02	3.73
			speedup factor	3.11	3.10	3.71
XMSS_ROBUST	16	10	1	23631.70	23642.03	13.07
XMSS_SIMPLE	16	10	1	7784.50	7788.56	3.67
			speedup factor	3.03	3.03	3.56
XMSS_SIMPLE+PRE	16	10	1	7586.15	7589.49	4.20
			speedup factor	3.11	3.11	3.11
XMSS^{MT}ROBUST	16	10	2	738.43	1498.06	27.67
XMSS^{MT}SIMPLE	16	10	2	243.49	494.55	7.77
			speedup factor	3.03	3.03	3.56
XMSS^{MT}SIMPLE+PRE	16	10	2	237.26	481.73	7.77
			speedup factor	3.11	3.11	3.56

All results (apart from speedup) are given in 10^6 clock cycles.

Performance comparison LMS vs XMSS

	LMS	XMSS_ROBUST	ratio[7]	XMSS_SIMPLE	ratio[8]	XMSS_SIMPLE+PRE	ratio[9]
key gen	3774.88	23631.70	6.26	7792.23	2.06	7586.15	2.01
sign	3791.15	23642.03	6.23	7796.39	2.05	7596.24	2.00
verify	2.65	13.07	4.93	3.57	1.34	4.20	1.58

All results for SHA-256, $n = 256$, $w = 16$, and $h = 10$ are given in 10^6 clock cycles.

[7] XMSS_ROBUST/LMS
[8] XMSS_SIMPLE/LMS
[9] XMSS_SIMPLE+PRE/LMS

LMS $\stackrel{?}{=}$ XMSS_SIMPLE

	LMS	XMSS_SIMPLE	ratio[10]	HSS	XMSS^{MT}SIMPLE	ratio[11]
key gen	1105990	1100800	0.99	34566	34400	0.99
sign	2216417	2202194	0.99	112542	104371	0.93
verify	2217208	2202686	0.99	113493	105359	0.93

Number of hash operations for SHA-256, $n = 256$, and $w = 16$.

[10] XMSS_SIMPLE/LMS

[11] XMSS^{MT}SIMPLE/HSS

Speed in clock cycles for XMSS and LMS for $h = 5$

design	hash type	w	h	d	key gen	sign	verify
XMSS_ROBUST	Gimli-Hash	16	5	1	1048850892	1063994437	17850167
XMSS_SIMPLE	Gimli-Hash	16	5	1	345097734	351135622	4843341
XMSS_SIMPLE+PRE	Gimli-Hash	16	5	1	35652023	341236863	4991976
LMS	Gimli-Hash	16	5	1	210439959	226186258	4601931
XMSS_ROBUST	Keccak-p[800, 22]	16	5	1	1162653236	1179847660	19384572
XMSS_SIMPLE	Keccak-p[800, 22]	16	5	1	380333946	387149205	5183652
XMSS_SIMPLE+PRE	Keccak-p[800, 22]	16	5	1	369894358	375718141	5838576
LMS	Keccak-p[800, 22]	16	5	1	180384764	193651049	4108963
XMSS_ROBUST	Keccak-p[800, 12]	16	5	1	699127232	709176591	11945544
XMSS_SIMPLE	Keccak-p[800, 12]	16	5	1	230594112	234234392	3625308
XMSS_SIMPLE+PRE	Keccak-p[800, 12]	16	5	1	**225063121**	**228715963**	**3444956**
LMS	Keccak-p[800, 12]	16	5	1	**106406966**	**114348011**	**2325050**
XMSS_ROBUST	SHAKE256	16	5	1	1569880839	1593969977	25282729
XMSS_SIMPLE	SHAKE256	16	5	1	515089881	523679528	7643266
LMS	SHAKE256	16	5	1	482690432	519083330	10541350
XMSS_ROBUST	SHA-256	16	5	1	738461396	747855715	13842083
XMSS_SIMPLE	SHA-256	16	5	1	243254582	247726301	3207473
XMSS_SIMPLE+PRE	SHA-256	16	5	1	237275019	241026688	3735483
LMS	SHA-256	16	5	1	117988963	126516806	2576515

Stack memory usage (bytes) for XMSS and LMS for $h = 5$

design	hash type[12]	w	h	layer	key gen	sign	verify
XMSS_ROBUST	Gimli-Hash	16	5	1	3784	3832	3604
XMSS_SIMPLE	Gimli-Hash	16	5	1	**3712**	**3760**	**3556**
XMSS_SIMPLE+PRE	Gimli-Hash	16	5	1	3728	3776	3572
LMS	Gimli-Hash	16	5	1	**3528**	**2240**	**876**
XMSS_ROBUST	Keccak-p[800, x]	16	5	1	3896	3944	3720
XMSS_SIMPLE	Keccak-p[800, x]	16	5	1	3824	3872	3672
XMSS_SIMPLE+PRE	Keccak-p[800, x]	16	5	1	3840	3888	3688
LMS	Keccak-p[800, x]	16	5	1	3644	2356	988
XMSS_ROBUST	SHAKE256	16	5	1	4224	4272	4088
XMSS_SIMPLE	SHAKE256	16	5	1	4176	4200	4024
LMS	SHAKE256	16	5	1	3844	2532	1164
XMSS_ROBUST	SHA-256	16	5	1	4032	4080	3912
XMSS_SIMPLE	SHA-256	16	5	1	3984	4032	3832
XMSS_SIMPLE+PRE	SHA-256	16	5	1	3976	4016	3840
LMS	SHA-256	16	5	1	3764	2460	1044

[12] Results for Keccak valid for Keccak-p[800, 22] and Keccak-p[800, 12].

Conclusion

Conclusion

- the reference implementation of LMS with some required modifications achieves good performance on Cortex-M4
- the presented variants of XMSS achieved speedups of up to 4.32×
- XMSS_SIMPLE, the variant without L-trees using Construction 1, differs structurally marginally from LMS
- reducing the number of rounds in KECCAK-f[800] to 12 instead of 22 yields a speedup of up to 1.76×
- the round-reduced version of KECCAK (KECCAK-p[800, 12]) achieved the best performance
- Gimli-Hash achieved the lowest stack consumption