`pqm4`: Benchmarking NIST Additional Post-Quantum Signature Schemes on Microcontrollers

Matthias J. Kannwischer, Markus Krausz, Richard Petri, and Shang-Yi Yang
matthias@chelpis.com

12 April 2024, Fifth PQC Standardization Conference, Rockville, Maryland, USA

Embedded PQC

- Cryptography needs to perform well on a large range of platforms
- NIST Additional Signatures: Small signatures, fast verification
  - Fast verification particularly important for smaller platforms
  - No performance concerns for Dilithium on large CPUs
- NIST: Arm Cortex-M4 as the primary microcontroller optimization target
  - Powerful instruction set $\Longrightarrow$ `UMAAL` and many more multiplication instructions
  - Cheap and widely available $\Longrightarrow$ \$20 dev board unless there is a pandemic
  - Huge $\Longrightarrow$ cores with 640 KB SRAM available; fits many of the PQC schemes
  - **Fun to optimize for**; great for teaching due to simple pipeline
- This talk: Initial benchmarking of NIST Additional Signature Schemes
  - Disclaimer: Many schemes have not been optimized for the Cortex-M4 yet

Embedded PQC

- Cryptography needs to perform well on a large range of platforms
- NIST Additional Signatures: Small signatures, fast verification
  - Fast verification particularly important for smaller platforms
  - No performance concerns for Dilithium on large CPUs
- NIST: Arm Cortex-M4 as the primary microcontroller optimization target
  - Powerful instruction set $\Longrightarrow$ `UMAAL` and many more multiplication instructions
  - Cheap and widely available $\Longrightarrow$ $20 dev board unless there is a pandemic
  - Huge $\Longrightarrow$ cores with 640 KB SRAM available; fits many of the PQC schemes
  - **Fun to optimize for**; great for teaching due to simple pipeline
- This talk: Initial benchmarking of NIST Additional Signature Schemes
  - Disclaimer: Many schemes have not been optimized for the Cortex-M4 yet

Embedded PQC

- Cryptography needs to perform well on a large range of platforms
- NIST Additional Signatures: Small signatures, fast verification
  - Fast verification particularly important for smaller platforms
  - No performance concerns for Dilithium on large CPUs
- NIST: Arm Cortex-M4 as the primary microcontroller optimization target
  - Powerful instruction set $\Longrightarrow$ `UMAAL` and many more multiplication instructions
  - Cheap and widely available $\Longrightarrow$ $20 dev board unless there is a pandemic
  - Huge $\Longrightarrow$ cores with 640 KB SRAM available; fits many of the PQC schemes
  - **Fun to optimize for**; great for teaching due to simple pipeline
- This talk: Initial benchmarking of NIST Additional Signature Schemes
  - Disclaimer: Many schemes have not been optimized for the Cortex-M4 yet

Embedded PQC

- Cryptography needs to perform well on a large range of platforms
- NIST Additional Signatures: Small signatures, fast verification
  - Fast verification particularly important for smaller platforms
  - No performance concerns for Dilithium on large CPUs
- NIST: Arm Cortex-M4 as the primary microcontroller optimization target
  - Powerful instruction set $\Longrightarrow$ `UMAAL` and many more multiplication instructions
  - Cheap and widely available $\Longrightarrow$ \$20 dev board unless there is a pandemic
  - Huge $\Longrightarrow$ cores with 640 KB SRAM available; fits many of the PQC schemes
  - **Fun to optimize for**; great for teaching due to simple pipeline
- This talk: Initial benchmarking of NIST Additional Signature Schemes
  - Disclaimer: Many schemes have not been optimized for the Cortex-M4 yet

Embedded PQC

- Cryptography needs to perform well on a large range of platforms
- NIST Additional Signatures: Small signatures, fast verification
    - Fast verification particularly important for smaller platforms
    - No performance concerns for Dilithium on large CPUs
- NIST: Arm Cortex-M4 as the primary microcontroller optimization target
    - Powerful instruction set $\Longrightarrow$ `UMAAL` and many more multiplication instructions
    - Cheap and widely available $\Longrightarrow$ \$20 dev board unless there is a pandemic
    - Huge $\Longrightarrow$ cores with 640 KB SRAM available; fits many of the PQC schemes
    - **Fun to optimize for**; great for teaching due to simple pipeline
- This talk: Initial benchmarking of NIST Additional Signature Schemes
    - Disclaimer: Many schemes have not been optimized for the Cortex-M4 yet

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation:**
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality:**
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- Reality:
  No improvement over NIST PQC I

My expectations and (sad) reality

Ideal World

- NISTPQC: Cryptographers learned that good reference code is important
- Project like **PQClean** raised quality bar
  $\Longrightarrow$ Increase awareness for useful SW-dev tools
  $\Longrightarrow$ Test automation many platforms
  $\Longrightarrow$ `-Wall -Wextra -Wpedantic -Werror`
- We wrote a paper with lessons learned
  $\Longrightarrow$ `https://eprint.iacr.org/2022/337`
  $\Longrightarrow$ Includes list of recommendations for NIST
- **Expectation**:
  My life is going to be much easier than 2018

Real World

- NIST: No changes to SW requirements
- Many compiler warnings;
  not passing sanitizers
  $\Longrightarrow$ Revealing quite a few bugs
- Some use of static memory
  Some cheating: Large pre-computation
- 20 out of 40 submissions use dynamic memory allocations
  $\Longrightarrow$ Often without real need
- **Reality**:
  No improvement over NIST PQC I

pqm4: Platform and framework changes

- You may have seen a `pqm4` talk before; here are some recent changes
- Switched default platform to `STM32L4R5ZI`
  - 640 KB of RAM, 2 MB of flash
  - Instruction timing same as `STM32F407`; except for small differences for memory loads
  - Other supported platforms: `STM32F407`, `STM32L476RG`, `STM32F303RCT7` (ChipWhisperer F3), `MPS2-AN386` (qemu)
- 20% faster Keccak
  - Described in **An update on Keccak performance on ARMv7-M** by Adomnicai `https://eprint.iacr.org/2023/773`
  - `https://github.com/mupq/pqm4/pull/254`

pqm4: Platform and framework changes

- You may have seen a `pqm4` talk before; here are some recent changes
- Switched default platform to `STM32L4R5ZI`
  - 640 KB of RAM, 2 MB of flash
  - Instruction timing same as `STM32F407`; except for small differences for memory loads
  - Other supported platforms: `STM32F407`, `STM32L476RG`, `STM32F303RCT7` (ChipWhisperer F3), `MPS2-AN386` (qemu)
- 20% faster Keccak
  - Described in **An update on Keccak performance on ARMv7-M** by Adomnicai `https://eprint.iacr.org/2023/773`
  - `https://github.com/mupq/pqm4/pull/254`

pqm4: Platform and framework changes

- You may have seen a `pqm4` talk before; here are some recent changes
- Switched default platform to `STM32L4R5ZI`
  - 640 KB of RAM, 2 MB of flash
  - Instruction timing same as `STM32F407`; except for small differences for memory loads
  - Other supported platforms: `STM32F407`, `STM32L476RG`, `STM32F303RCT7` (ChipWhisperer F3), `MPS2-AN386` (qemu)
- 20% faster Keccak
  - Described in **An update on Keccak performance on ARMv7-M** by Adomnicai `https://eprint.iacr.org/2023/773`
  - `https://github.com/mupq/pqm4/pull/254`

Exclusion criteria

- **Vulnerable**: Brokenness of the scheme
- **PK too big**: Public key + private key + signature need to fit in 640 KB of RAM
- **Too much memory**: Keys + memory consumption needs to fit in 640 KB of RAM
- **External library**: Cannot have any external dependency (e.g., gmp, flint)
  $\Longrightarrow$ we do replace Keccak, SHA-2, AES with optimized code
- **Not portable**: Code that is not supported for 32-bit platforms (e.g., `__int128`)
- **Dynamic Memory allocations**: Dynamic memory allocations are undesirable
  $\Longrightarrow$ we try to fix it if it is straightforward

Exclusion criteria

- **Vulnerable**: Brokenness of the scheme
- **PK too big**: Public key + private key + signature need to fit in 640 KB of RAM
- **Too much memory**: Keys + memory consumption needs to fit in 640 KB of RAM
- **External library**: Cannot have any external dependency (e.g., gmp, flint)
  $\Longrightarrow$ we do replace Keccak, SHA-2, AES with optimized code
- **Not portable**: Code that is not supported for 32-bit platforms (e.g., `__int128`)
- **Dynamic Memory allocations**: Dynamic memory allocations are undesirable
  $\Longrightarrow$ we try to fix it if it is straightforward

Exclusion criteria

- **Vulnerable**: Brokenness of the scheme
- **PK too big**: Public key + private key + signature need to fit in 640 KB of RAM
- **Too much memory**: Keys + memory consumption needs to fit in 640 KB of RAM
- **External library**: Cannot have any external dependency (e.g., gmp, flint)
  $\Longrightarrow$ we do replace Keccak, SHA-2, AES with optimized code
- **Not portable**: Code that is not supported for 32-bit platforms (e.g., `__int128`)
- **Dynamic Memory allocations**: Dynamic memory allocations are undesirable
  $\Longrightarrow$ we try to fix it if it is straightforward

Exclusion criteria

- **Vulnerable**: Brokenness of the scheme
- **PK too big**: Public key + private key + signature need to fit in 640 KB of RAM
- **Too much memory**: Keys + memory consumption needs to fit in 640 KB of RAM
- **External library**: Cannot have any external dependency (e.g., gmp, flint)
  $\Longrightarrow$ we do replace Keccak, SHA-2, AES with optimized code
- **Not portable**: Code that is not supported for 32-bit platforms (e.g., `__int128`)
- **Dynamic Memory allocations**: Dynamic memory allocations are undesirable
  $\Longrightarrow$ we try to fix it if it is straightforward

Scheme inclusion

Code	Lattice	MPCitH	MQ	Other
CROSS	EagleSign	Biscuit	3WISE	AIMer
Enhanced pqsigRM	EHTv3 and EHTv4	MIRA	DME-Sign	ALTEQ
FuLeeca	HAETAE	MiRitH	HPPC	Ascon-Sign
LESS	HAWK	MQOM	MAYO	eMLE-Sig 2.0
MEDS	HuFu	PERK	PROV	FAEST
Wave	Raccoon	RYDE	QR-UOV	KAZ-SIGN
	SQUIRRELS	SDitH	SNOVA	Preon
			TUOV	SPHINCS-alpha
			UOV	SQIsign
			VOX	Xifrat1-Sign.I

Vulnerable (9) PK too big (4) Too much memory (7) External library / Not portable (3)

Scheme inclusion

Code	Lattice	MPCitH	MQ	Other
CROSS	~~EagleSign~~	Biscuit	~~3WISE~~	AIMer
Enhanced pqsigRM	~~EHTv3 and EHTv4~~	MIRA	~~DME-Sign~~	ALTEQ
~~FuLeeca~~	HAETAE	MiRitH	HPPC	Ascon-Sign
LESS	HAWK	MQOM	MAYO	~~eMLE-Sig 2.0~~
MEDS	HuFu	PERK	PROV	FAEST
Wave	Raccoon	RYDE	QR-UOV	~~KAZ-SIGN~~
	SQUIRRELS	SDitH	SNOVA	Preon
			TUOV	SPHINCS-alpha
			UOV	SQIsign
			~~VOX~~	~~Xifrat1-Sign.I~~

Vulnerable (9) PK too big (4) Too much memory (7) External library / Not portable (3)

Scheme inclusion

Code	Lattice	MPCitH	MQ	Other
CROSS	~~EagleSign~~	Biscuit	~~3WISE~~	AIMer
~~Enhanced pqsigRM~~	~~EHTv3 and EHTv4~~	MIRA	~~DME-Sign~~	ALTEQ
~~FuLeeca~~	HAETAE	MiRitH	HPPC	Ascon-Sign
LESS	HAWK	MQOM	MAYO	~~eMLE-Sig 2.0~~
MEDS	~~HuFu~~	PERK	PROV	FAEST
~~Wave~~	Raccoon	RYDE	QR-UOV	~~KAZ-SIGN~~
	~~SQUIRRELS~~	SDitH	SNOVA	Preon
			TUOV	SPHINCS-alpha
			UOV	SQIsign
			~~VOX~~	~~Xifrat1-Sign.I~~

Vulnerable (9) PK too big (4) Too much memory (7) External library / Not portable (3)

Scheme inclusion

Code	Lattice	MPCitH	MQ	Other
CROSS	~~EagleSign~~	Biscuit	~~3WISE~~	AIMer
~~Enhanced pqsigRM~~	~~EHTv3 and EHTv4~~	MIRA	~~DME-Sign~~	~~ALTEQ~~
~~FuLeeca~~	HAETAE	MiRitH	HPPC	Ascon-Sign
~~LESS~~	HAWK	MQOM	MAYO	~~eMLE-Sig 2.0~~
MEDS	~~HuFu~~	PERK	~~PROV~~	FAEST
~~Wave~~	Raccoon	RYDE	~~QR-UOV~~	~~KAZ-SIGN~~
	~~SQUIRRELS~~	~~SDitH~~	SNOVA	~~Preon~~
			~~TUOV~~	SPHINCS-alpha
			UOV	SQIsign
			~~VOX~~	~~Xifrat1-Sign.I~~

Vulnerable (9) PK too big (4) Too much memory (7) External library / Not portable (3)

Scheme inclusion

Code	Lattice	MPCitH	MQ	Other
CROSS	~~EagleSign~~	Biscuit	~~3WISE~~	AIMer
~~Enhanced pqsigRM~~	~~EHTv3 and EHTv4~~	MIRA	~~DME-Sign~~	~~ALTEQ~~
~~FuLeeca~~	HAETAE	MiRitH	~~HPPC~~	Ascon-Sign
~~LESS~~	HAWK	MQOM	MAYO	~~eMLE-Sig 2.0~~
MEDS	~~HuFu~~	PERK	~~PROV~~	FAEST
~~Wave~~	~~Raccoon~~	RYDE	~~QR-UOV~~	~~KAZ-SIGN~~
	~~SQUIRRELS~~	~~SDitH~~	SNOVA	~~Preon~~
			~~TUOV~~	SPHINCS-alpha
			UOV	~~SQIsign~~
			~~VOX~~	~~Xifrat1-Sign.I~~

Vulnerable (9) PK too big (4) Too much memory (7) External library / Not portable (3)

Dynamic memory allocations

- Dynamic memory allocations should be avoided in embedded implementations
  $\Longrightarrow$ expensive, often not well supported
- Schemes that actually need them (usually > 4 MB memory), not suitable anyway
- `pqm4`: No dynamic memory allocations allowed
- Schemes with dynamic memory allocations (20): ~~Enhanced pqsigRM~~, ~~LESS~~, ~~Wave~~, ~~SQIsign~~, ~~EHTv3 and EHTv4~~, ~~HuFu~~, MIRA, MQOM, RYDE, ~~SDitH~~, ~~PROV~~, ~~QR-UOV~~, ~~TUOV~~, ~~VOX~~, AIMer, FAEST, ~~ALTEQ~~, ~~eMLE-Sig 2.0~~, ~~KAZ-SIGN~~, ~~Preon~~
- Easily fixed: MQOM, AIMer
- Excluded due to dynamic memory allocations (3): MIRA, RYDE, FAEST

Dynamic memory allocations

- Dynamic memory allocations should be avoided in embedded implementations $\implies$ expensive, often not well supported
- Schemes that actually need them (usually > 4 MB memory), not suitable anyway
- `pqm4`: No dynamic memory allocations allowed
- Schemes with dynamic memory allocations (20): ~~Enhanced pqsigRM~~, ~~LESS~~, ~~Wave~~, ~~SQIsign~~, ~~EHTv3 and EHTv4~~, ~~HuFu~~, MIRA, MQOM, RYDE, ~~SDitH~~, ~~PROV~~, ~~QR-UOV~~, ~~TUOV~~, ~~VOX~~, AIMer, FAEST, ~~ALTEQ~~, ~~eMLE-Sig 2.0~~, ~~KAZ-SIGN~~, ~~Preon~~
- Easily fixed: MQOM, AIMer
- Excluded due to dynamic memory allocations (3): MIRA, RYDE, FAEST

Dynamic memory allocations

- Dynamic memory allocations should be avoided in embedded implementations $\implies$ expensive, often not well supported
- Schemes that actually need them (usually > 4 MB memory), not suitable anyway
- `pqm4`: No dynamic memory allocations allowed
- Schemes with dynamic memory allocations (20): ~~Enhanced pqsigRM~~, ~~LESS~~, ~~Wave~~, ~~SQIsign~~, ~~EHTv3 and EHTv4~~, ~~HuFu~~, MIRA, MQOM, RYDE, ~~SDitH~~, ~~PROV~~, ~~QR-UOV~~, ~~TUOV~~, ~~VOX~~, AIMer, FAEST, ~~ALTEQ~~, ~~eMLE-Sig 2.0~~, ~~KAZ-SIGN~~, ~~Preon~~
- Easily fixed: MQOM, AIMer
- Excluded due to dynamic memory allocations (3): MIRA, RYDE, FAEST

Dynamic memory allocations

- Dynamic memory allocations should be avoided in embedded implementations $\Longrightarrow$ expensive, often not well supported
- Schemes that actually need them (usually > 4 MB memory), not suitable anyway
- `pqm4`: No dynamic memory allocations allowed
- Schemes with dynamic memory allocations (20): ~~Enhanced pqsigRM~~, ~~LESS~~, ~~Wave~~, ~~SQIsign~~, ~~EHTv3 and EHTv4~~, ~~HuFu~~, MIRA, MQOM, RYDE, ~~SDitH~~, ~~PROV~~, ~~QR-UOV~~, ~~TUOV~~, ~~VOX~~, AIMer, FAEST, ~~ALTEQ~~, ~~eMLE-Sig 2.0~~, ~~KAZ-SIGN~~, ~~Preon~~
- Easily fixed: MQOM, AIMer
- Excluded due to dynamic memory allocations (3): MIRA, RYDE, FAEST

Included implementations

	PR	ref	m4f	params	eprint
CROSS	#309	✓		12/24	
MEDS	#324	✓		2/6	
HAETAE	#313	✓	✓	3/3	ia.cr/2023/624
HAWK	#305	✓		3/3	
Biscuit	#314	✓		3/6	
MiRitH	#315	✓	✓	16/32	ia.cr/2023/1666
MQOM	#322	✓		2/12	
PERK	#318	✓	✓	12/12	ia.cr/2024/088
MAYO	#302	✓	✓	3/4	ia.cr/2023/1683
SNOVA	#311	✓		7/18	
UOV	#300	✓	✓	3/12	ia.cr/2023/059
AIMer	#323	✓		3/12	
Ascon-Sign	#308	✓		8/8	
SPHINCS-alpha	#312	✓		6/24	
		14	5		

HAETAE

- Both reference and M4-optimized code has been integrated
- Contributed by the HAETAE team $\Longrightarrow$ Thank you!
- Described in **HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures**, by Cheon, Choe, Devevey, Güneysu, Hong, Krausz, Land, Möller, Stehlé, and Yi
  `https://eprint.iacr.org/2023/624`
- Supported parameter sets: `HAETAE-{2,3,5}`
- Incompatible with the original specification

HAETAE

- Both reference and M4-optimized code has been integrated
- Contributed by the HAETAE team $\Longrightarrow$ Thank you!
- Described in **HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures**, by Cheon, Choe, Devevey, Güneysu, Hong, Krausz, Land, Möller, Stehlé, and Yi `https://eprint.iacr.org/2023/624`
- Supported parameter sets: `HAETAE-{2,3,5}`
- Incompatible with the original specification

HAETAE

- Both reference and M4-optimized code has been integrated
- Contributed by the HAETAE team $\Longrightarrow$ Thank you!
- Described in **HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures**, by Cheon, Choe, Devevey, Güneysu, Hong, Krausz, Land, Möller, Stehlé, and Yi `https://eprint.iacr.org/2023/624`
- Supported parameter sets: `HAETAE-{2,3,5}`
- Incompatible with the original specification

MiRitH

- Both reference and M4-optimized code has been integrated
- Described in **MiRitH: Efficient Post-Quantum Signatures from MinRank in the Head** by Adj, Barbero, Bellini, Esser, Rivera-Zamarripa, Sanna, Verbel, and Zweydinger `https://eprint.iacr.org/2023/1666`
- Reference: 16 out of 32 parameter sets are working
- M4-optimized: `mirith_hypercube_Ia_{fast,short}`

MiRitH

- Both reference and M4-optimized code has been integrated
- Described in **MiRitH: Efficient Post-Quantum Signatures from MinRank in the Head** by Adj, Barbero, Bellini, Esser, Rivera-Zamarripa, Sanna, Verbel, and Zweydinger `https://eprint.iacr.org/2023/1666`
- Reference: 16 out of 32 parameter sets are working
- M4-optimized: `mirith_hypercube_Ia_{fast,short}`

MiRitH

- Both reference and M4-optimized code has been integrated
- Described in **MiRitH: Efficient Post-Quantum Signatures from MinRank in the Head** by Adj, Barbero, Bellini, Esser, Rivera-Zamarripa, Sanna, Verbel, and Zweydinger `https://eprint.iacr.org/2023/1666`
- Reference: 16 out of 32 parameter sets are working
- M4-optimized: `mirith_hypercube_Ia_{fast,short}`

PERK

- Both reference and M4-optimized code has been integrated
- Contributed by the PERK team $\Longrightarrow$ Thank you!
- Described in **Enabling PERK on Resource-Constrained Devices** by Bettaieb, Bidoux, Budroni, Palumbi, and Lucas Pandolfo Perin
  `https://eprint.iacr.org/2024/088`
- All 12 parameter sets are supported by the M4 implementation

PERK

- Both reference and M4-optimized code has been integrated
- Contributed by the PERK team $\Longrightarrow$ Thank you!
- Described in **Enabling PERK on Resource-Constrained Devices** by Bettaieb, Bidoux, Budroni, Palumbi, and Lucas Pandolfo Perin
  `https://eprint.iacr.org/2024/088`
- All 12 parameter sets are supported by the M4 implementation

MAYO

- Both reference and M4-optimized code has been integrated
- Described in **Nibbling MAYO: Optimized Implementations for AVX2 and Cortex-M4** by Beullens, Campos, Celi, Hess, and Kannwischer `https://eprint.iacr.org/2023/1683`
- Supported parameter sets: `MAYO{1,2,3}` (`MAYO5` requires too much RAM as of now)
- Current `pqm4` implementation is compatible with round-1 specification
- Can be slightly faster when using different representation (see Nibbling MAYO talk)

MAYO

- Both reference and M4-optimized code has been integrated
- Described in **Nibbling MAYO: Optimized Implementations for AVX2 and Cortex-M4** by Beullens, Campos, Celi, Hess, and Kannwischer `https://eprint.iacr.org/2023/1683`
- Supported parameter sets: `MAYO{1,2,3}` (`MAYO5` requires too much RAM as of now)
- Current `pqm4` implementation is compatible with round-1 specification
- Can be slightly faster when using different representation (see Nibbling MAYO talk)

MAYO

- Both reference and M4-optimized code has been integrated
- Described in **Nibbling MAYO: Optimized Implementations for AVX2 and Cortex-M4** by Beullens, Campos, Celi, Hess, and Kannwischer `https://eprint.iacr.org/2023/1683`
- Supported parameter sets: `MAYO{1,2,3}` (`MAYO5` requires too much RAM as of now)
- Current `pqm4` implementation is compatible with round-1 specification
- Can be slightly faster when using different representation (see Nibbling MAYO talk)

UOV

- Both reference and M4-optimized code has been integrated
- Described in **Oil and Vinegar: Modern Parameters and Implementations** by Beullens, Chen, Hung, Kannwischer, Peng, Shih, and Yang `https://eprint.iacr.org/2023/059`
- Supported parameter sets: `ov-Ip-{,pkc,pkc-skc}`
- `ov-Is` requires offloading keys to flash to fit within 640 KB of RAM (see paper)
- `ov-III` and `ov-V` is out of reach due to public key sizes

UOV

- Both reference and M4-optimized code has been integrated
- Described in **Oil and Vinegar: Modern Parameters and Implementations** by Beullens, Chen, Hung, Kannwischer, Peng, Shih, and Yang
  `https://eprint.iacr.org/2023/059`
- Supported parameter sets: `ov-Ip-{,pkc,pkc-skc}`
- `ov-Is` requires offloading keys to flash to fit within 640 KB of RAM (see paper)
- `ov-III` and `ov-V` is out of reach due to public key sizes

Performance

- Let's look at some performance numbers
- As always: Downclock device to avoid wait states for flash access (20 MHz)
- Only very limited selection here
    - Full results in the paper and `https://github.com/mupq/pqm4/blob/master/benchmarks.md`
    - Warning: Early in the competition; reference implementation performance is meaningless
- Selection criteria
    - For each submission: Fastest parameter set
    - May not be the same for signing and verification
    - Mostly security level 1 (exception: sphincs-a-sha2-192f)
- Addition: SQISign verification
    - Work in progress by Décio Luiz Gazzoni Filho and Krijn Reijnders

Performance

- Let's look at some performance numbers
- As always: Downclock device to avoid wait states for flash access (20 MHz)
- Only very limited selection here
  - Full results in the paper and `https://github.com/mupq/pqm4/blob/master/benchmarks.md`
  - Warning: Early in the competition; reference implementation performance is meaningless
- Selection criteria
  - For each submission: Fastest parameter set
  - May not be the same for signing and verification
  - Mostly security level 1 (exception: sphincs-a-sha2-192f)
- Addition: SQISign verification
  - Work in progress by Décio Luiz Gazzoni Filho and Krijn Reijnders

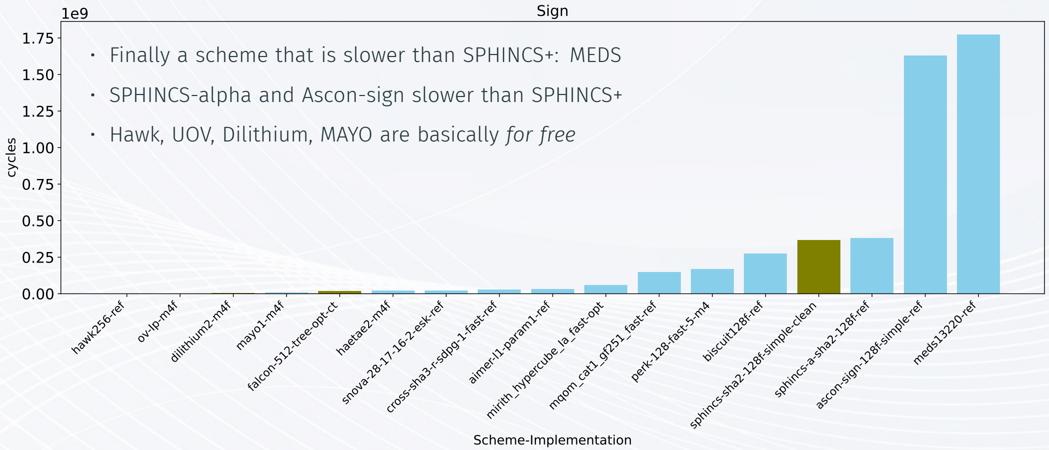
Performance

- Let's look at some performance numbers
- As always: Downclock device to avoid wait states for flash access (20 MHz)
- Only very limited selection here
  - Full results in the paper and `https://github.com/mupq/pqm4/blob/master/benchmarks.md`
  - Warning: Early in the competition; reference implementation performance is meaningless
- Selection criteria
  - For each submission: Fastest parameter set
  - May not be the same for signing and verification
  - Mostly security level 1 (exception: sphincs-a-sha2-192f)
- Addition: SQISign verification
  - Work in progress by Décio Luiz Gazzoni Filho and Krijn Reijnders

Performance

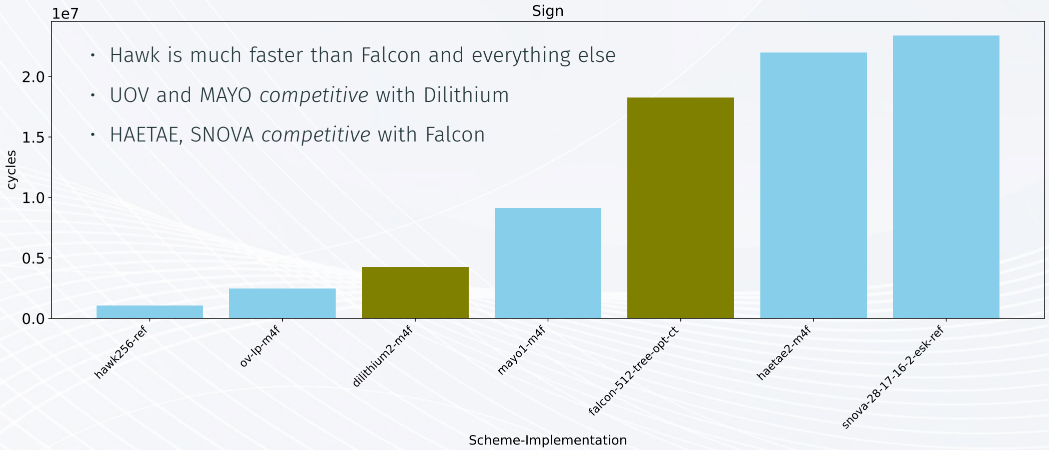
- Let's look at some performance numbers
- As always: Downclock device to avoid wait states for flash access (20 MHz)
- Only very limited selection here
  - Full results in the paper and `https://github.com/mupq/pqm4/blob/master/benchmarks.md`
  - Warning: Early in the competition; reference implementation performance is meaningless
- Selection criteria
  - For each submission: Fastest parameter set
  - May not be the same for signing and verification
  - Mostly security level 1 (exception: sphincs-a-sha2-192f)
- Addition: SQISign verification
  - Work in progress by Décio Luiz Gazzoni Filho and Krijn Reijnders

Performance: Signing



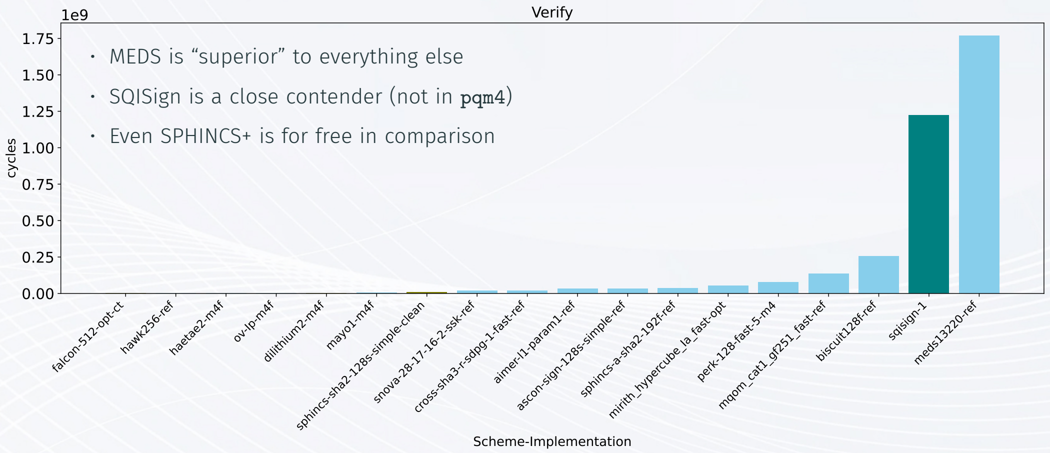
- Finally a scheme that is slower than SPHINCS+: MEDS
- SPHINCS-alpha and Ascon-sign slower than SPHINCS+
- Hawk, UOV, Dilithium, MAYO are basically *for free*

Performance: Signing



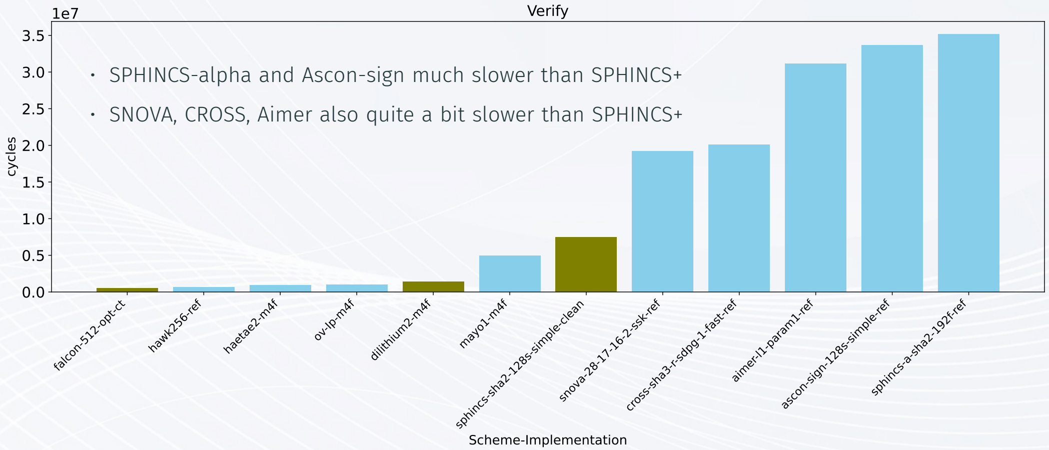
- Hawk is much faster than Falcon and everything else
- UOV and MAYO *competitive* with Dilithium
- HAETAE, SNOVA *competitive* with Falcon

Performance: Verification



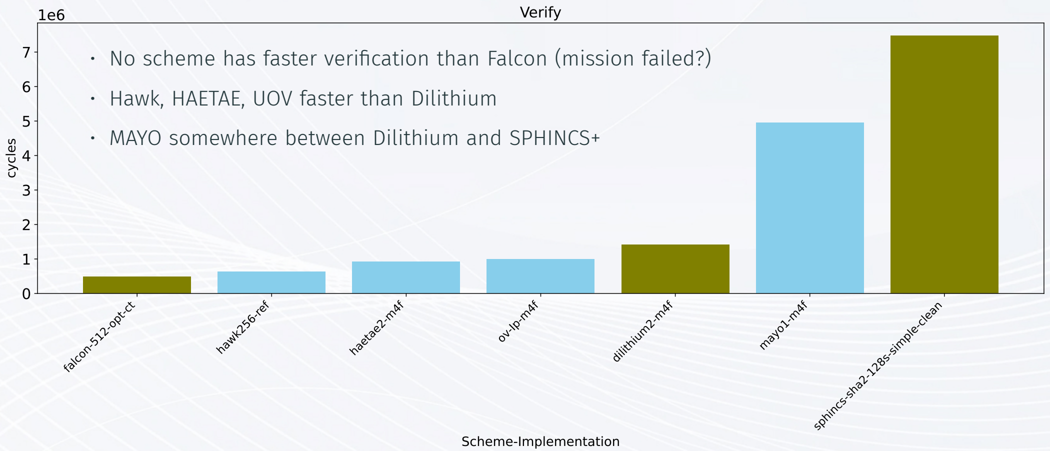
- MEDS is “superior” to everything else
- SQISign is a close contender (not in `pqm4`)
- Even SPHINCS+ is for free in comparison

Performance: Verification



- SPHINCS-alpha and Ascon-sign much slower than SPHINCS+
- SNOVA, CROSS, Aimer also quite a bit slower than SPHINCS+

Performance: Verification



- No scheme has faster verification than Falcon (mission failed?)
- Hawk, HAETAE, UOV faster than Dilithium
- MAYO somewhere between Dilithium and SPHINCS+

Conclusion and next steps

- Some cryptographers have weird understanding of **fast verification**
- We are very early in the competition
  - Still many low-hanging cycles to be eliminated
- Interesting question: Can we make schemes fit that are not yet included
  - Eliminate dynamic memory allocations
  - Replace external libraries
  - Reduce memory consumption
- If your favorite scheme is not included or slow
  - Do not send angry e-mails to the mailing list
  - Instead: Write a fast implementation and submit a pull request
- Work in progress: Partial scheme benchmarking (e.g., SQISign verification)
- NIST: Please significantly lower the number of candidates

Conclusion and next steps

- Some cryptographers have weird understanding of **fast verification**
- We are very early in the competition
  - Still many low-hanging cycles to be eliminated
- Interesting question: Can we make schemes fit that are not yet included
  - Eliminate dynamic memory allocations
  - Replace external libraries
  - Reduce memory consumption
- If your favorite scheme is not included or slow
  - Do not send angry e-mails to the mailing list
  - Instead: Write a fast implementation and submit a pull request
- Work in progress: Partial scheme benchmarking (e.g., SQISign verification)
- NIST: Please significantly lower the number of candidates

Conclusion and next steps

- Some cryptographers have weird understanding of **fast verification**
- We are very early in the competition
  - Still many low-hanging cycles to be eliminated
- Interesting question: Can we make schemes fit that are not yet included
  - Eliminate dynamic memory allocations
  - Replace external libraries
  - Reduce memory consumption
- If your favorite scheme is not included or slow
  - Do not send angry e-mails to the mailing list
  - Instead: Write a fast implementation and submit a pull request
- Work in progress: Partial scheme benchmarking (e.g., SQISign verification)
- NIST: Please significantly lower the number of candidates

Conclusion and next steps

- Some cryptographers have weird understanding of **fast verification**
- We are very early in the competition
    - Still many low-hanging cycles to be eliminated
- Interesting question: Can we make schemes fit that are not yet included
    - Eliminate dynamic memory allocations
    - Replace external libraries
    - Reduce memory consumption
- If your favorite scheme is not included or slow
    - Do not send angry e-mails to the mailing list
    - Instead: Write a fast implementation and submit a pull request
- Work in progress: Partial scheme benchmarking (e.g., SQISign verification)
- NIST: Please significantly lower the number of candidates

Conclusion and next steps

- Some cryptographers have weird understanding of **fast verification**
- We are very early in the competition
    - Still many low-hanging cycles to be eliminated
- Interesting question: Can we make schemes fit that are not yet included
    - Eliminate dynamic memory allocations
    - Replace external libraries
    - Reduce memory consumption
- If your favorite scheme is not included or slow
    - Do not send angry e-mails to the mailing list
    - Instead: Write a fast implementation and submit a pull request
- Work in progress: Partial scheme benchmarking (e.g., SQISign verification)
- NIST: Please significantly lower the number of candidates

Thank you very much for your attention!

ia.cr/2024/112