CRYSTALS - Dilithium

Shi Bai
Leo Ducas
Eike Kiltz
Tancrede Lepoint

Vadim Lyubashevsky
Peter Schwabe
Gregor Seiler
Damien Stehle

High-Level Scheme Overview

- “Schnorr-like” lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=Z_q[X]/(X^{256}+1)$ for $q=8,380,417$

High-Level Scheme Overview

```
KeyGen:
    $A \leftarrow R^{n \times m}$
    $s \leftarrow S^m$
    $t = \mathrm{Round}(As)$
    $pk=(A,t)$   $sk=s$
```

- “Schnorr-like” lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=Z_q[X]/(X^{256}+1)$ for $q=8{,}380{,}417$

High-Level Scheme Overview

```
KeyGen:
    $A \leftarrow R^{n \times m}$
    $s \leftarrow S^m$
    $t = \mathrm{Round}(As)$
    $pk=(A,t)$   $sk=s$
```

```
Sign(pk,sk,μ):
    $y \leftarrow Y^m$
    $w=\mathrm{Round}(Ay)$
    $c=\mathrm{Hash}(w,\mu)$
    $z=sc+y$
    $\mathrm{RejectionSample}(pk,sk,z)$
    $\omega = \mathrm{HintVector}(pk,sk,z)$
    $\sigma = (z, \omega, c)$
```

- “Schnorr-like” lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=Z_q[X]/(X^{256}+1)$ for $q=8,380,417$

High-Level Scheme Overview

```
KeyGen:
    $A \leftarrow R^{n \times m}$
    $s \leftarrow S^m$
    $t = \mathrm{Round}(As)$
    $pk=(A,t)$   $sk=s$
```

- “Schnorr-like” lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=Z_q[X]/(X^{256}+1)$ for $q=8{,}380{,}417$

```
Sign(pk,sk,μ):
    $y \leftarrow Y^m$
    $w=\mathrm{Round}(Ay)$
    $c=\mathrm{Hash}(w,\mu)$
    $z=sc+y$
    RejectionSample(pk,sk,z)
    $\omega$ = HintVector(pk,sk,z)
    $\sigma = (z, \omega, c)$
```

The main difference between lattice and Schnorr schemes

High-Level Scheme Overview

```
KeyGen:
    $A \leftarrow R^{n \times m}$
    $s \leftarrow S^m$
    $t = \mathrm{Round}(As)$
    $pk=(A,t)$   $sk=s$
```

- "Schnorr-like" lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=Z_q[X]/(X^{256}+1)$ for $q=8{,}380{,}417$

```
Sign(pk,sk,μ):
    $y \leftarrow Y^m$
    $w=\mathrm{Round}(Ay)$
    $c=\mathrm{Hash}(w,\mu)$
    $z=sc+y$
    RejectionSample(pk,sk,z)
    $\omega = \mathrm{HintVector}(pk,sk,z)$
    $\sigma = (z, \omega, c)$
```

The main difference between lattice and Schnorr schemes

```
Verify(μ,σ,pk):
    $w=\mathrm{UseHintVector}(pk,\sigma)$
    check that $c=\mathrm{Hash}(w, \mu)$  and $|z|$ is small
```

Round 2 → Round 3

Round 2 → Round 3

- No improved attacks

Round 2 → Round 3

- No improved attacks

- We did some parameter reshuffling
  - Chose parameters to fit with the NIST security levels
  - Added NIST Level 5 (was not mandatory, but became strongly recommended)

Round 2 → Round 3

- No improved attacks
- We did some parameter reshuffling
  - Chose parameters to fit with the NIST security levels
  - Added NIST Level 5 (was not mandatory, but became strongly recommended)
- Sampling in the signing procedure is now uniform within a range with 2^k elements – even simpler than before when the range wasn't a power-of-2
- Slightly simpler and shorter generation of the fixed-weight challenge polynomial

CRYSTALS-Dilithium

AVX2 + AES on Skylake
/ sec is assuming 3GHz freq.

Security Level	Public Key (Bytes)	Signature (Bytes)	pkgen	sign	verify
60	864	1196			
100	992	1843			
128 (NIST II)	1312	2420	50K cyc 60K / sec	150K cyc 20K / sec	65K cyc 45K / sec
2^{159} gates					
2^{98} memory					
192 (NIST III)	1952	3293	80K cyc 35K / sec	200K cyc 15K / sec	95K cyc 30K / sec
2^{217} gates					
2^{139} memory					
256 (NIST V)	2592	4595	125K cyc 24K / cyc	230K cyc 13K / sec	135K cyc 22K / sec
2^{285} gates					
2^{187} memory					
320	2912	5246			
384	3232	5892			

Many Efficiency Trade-Offs Possible

Implementation of Dilithium Signing on Cortex M3 and M4:

[Greconici, Kannwischer, Sprenkels 2020] (Speed numbers extrapolated because the number of repetitions changed)

NIST Level 3	Speed	RAM
Cortex M3	12M cycles	70KB
Cortex M3	37M cycles	11KB
Cortex M3	9M cycles	21KB
		+ 48KB Flash
Cortex M4	8M cycles	70KB
Cortex M4	26M cycles	11KB
Cortex M4	6M cycles	21KB
		+ 48KB Flash

Many Efficiency Trade-Offs Possible

Implementation of Dilithium Signing on Cortex M3 and M4:

[Greconici, Kannwischer, Sprenkels 2020] (Speed numbers extrapolated because the number of repetitions changed)

NIST Level 3	Speed	RAM
Cortex M3	12M cycles	70KB
Cortex M3	37M cycles	11KB
Cortex M3	9M cycles	21KB
		+ 48KB Flash
Cortex M4	8M cycles	70KB
Cortex M4	26M cycles	11KB
Cortex M4	6M cycles	21KB
		+ 48KB Flash

[Gonzalez, Hulsing, Kannwischer, Kramer, Lange, Stottinger, Waitz, Wiggers, Yang 2021]

Verification can fit in under 8kB of RAM

Design Criteria for CRYSTALS-Dilithium

1. Significant speed / size advantage over hash-based schemes, even when comparing 256-bit Dilithium vs. 128-bit SHA-based schemes

Design Criteria for CRYSTALS-Dilithium

1. Significant speed / size advantage over hash-based schemes, even when comparing 256-bit Dilithium vs. 128-bit SHA-based schemes

	SPHINCS+ Small (NIST Level 1)	**SPHINCS+ Fast (NIST Level 1)**
Dilithium (NIST Level 5)	3000X faster signing / same size	150X faster signing / 2.5X smaller

Design Criteria for CRYSTALS-Dilithium

1. Significant speed / size advantage over hash-based schemes, even when comparing 256-bit Dilithium vs. 128-bit SHA-based schemes

	SPHINCS+ Small (NIST Level 1)	SPHINCS+ Fast (NIST Level 1)
Dilithium (NIST Level 5)	3000X faster signing / same size	150X faster signing / 2.5X smaller

2. Easy to implement
   - no Gaussian sampling (not even the “easy kind” always centered at 0)
   - should be easy to ~~avoid~~ detect bugs

Design Criteria for CRYSTALS-Dilithium

1. Significant speed / size advantage over hash-based schemes, even when comparing 256-bit Dilithium vs. 128-bit SHA-based schemes

	SPHINCS+ Small (NIST Level 1)	SPHINCS+ Fast (NIST Level 1)
Dilithium (NIST Level 5)	3000X faster signing / same size	150X faster signing / 2.5X smaller

2. Easy to implement
    - no Gaussian sampling (not even the "easy kind" always centered at 0)
    - should be easy to ~~avoid~~ detect bugs

Dilithium > ~~Dilithium-G~~ > ~~BLISS~~

Gaussians

Gaussians + NTRU assumption

Design Criteria for CRYSTALS-Dilithium

1. Significant speed / size advantage over hash-based schemes, even when comparing 256-bit Dilithium vs. 128-bit SHA-based schemes

	SPHINCS+ Small (NIST Level 1)	SPHINCS+ Fast (NIST Level 1)
Dilithium (NIST Level 5)	3000X faster signing / same size	150X faster signing / 2.5X smaller

2. Easy to implement
   - no Gaussian sampling (not even the "easy kind" always centered at 0)
   - should be easy to ~~avoid~~ detect bugs

Dilithium > ~~Dilithium-G~~ > ~~BLISS~~ > Falcon

↑ Gaussians

↑ Gaussians + NTRU assumption

Comparison with Falcon

CRYSTALS-Dilithium

Security Level	Public Key (Bytes)	Signature (Bytes)
128	1312	2420
192	1952	3293
256	2592	4595

Falcon

Security Level	Public Key (Bytes)	Signature (Bytes)
128	897	666
-	-	-
256	1793	1280

Comparison with Falcon

CRYSTALS-Dilithium

Security Level	Public Key (Bytes)	Signature (Bytes)
128	1312	2420
192	1952	3293
256	2592	4595

- ≈ 2.3 X larger (pk + sig)

Falcon

Security Level	Public Key (Bytes)	Signature (Bytes)
128	897	666
-	-	-
256	1793	1280

+ ≈ 2.3 X smaller (pk + sig)

Comparison with Falcon

CRYSTALS-Dilithium

Security Level	Public Key (Bytes)	Signature (Bytes)
128	1312	2420
192	1952	3293
256	2592	4595

- ≈ 2.3 X larger (pk + sig)

+ Signing uses only uniform sampling in a (power-of-2) range. Easier to detect bugs.

Falcon

Security Level	Public Key (Bytes)	Signature (Bytes)
128	897	666
-	-	-
256	1793	1280

+ ≈ 2.3 X smaller (pk + sig)

- Signing uses high-precision Gaussian sampling with high-precision changing centers. Hard to detect subtle implementation mistakes which can leak the secret key
- Very difficult to mask

Comparison with Falcon

CRYSTALS-Dilithium

Security Level	Public Key (Bytes)	Signature (Bytes)
128	1312	2420
192	1952	3293
256	2592	4595

- ≈ 2.3 X larger (pk + sig)

+ Signing uses only uniform sampling in a (power-of-2) range. Easier to detect bugs.

Falcon

Security Level	Public Key (Bytes)	Signature (Bytes)
128	897	666
-	-	-
256	1793	1280

+ ≈ 2.3 X smaller (pk + sig)

- Signing uses high-precision Gaussian sampling with high-precision changing centers. Hard to detect subtle implementation mistakes which can leak the secret key
- Very difficult to mask

Signing a few messages (≈ 100?) shouldn’t leak enough even if the sampling is leaky

Runtime Efficiency Comparison

Dilithium [Greconici, Kannwischer, Sprenkels 2020]

Falcon [Pornin, 2019]

Runtime Efficiency Comparison

Dilithium [Greconici, Kannwischer, Sprenkels 2020]

NIST Level 3	Key Gen. Speed	Key Gen. RAM
Cortex M4	6M cycles	10KB

Falcon [Pornin, 2019]

NIST Level 1	Key Gen. Speed	Key Gen. RAM
Cortex M4	171M cycles	16KB

Runtime Efficiency Comparison

Dilithium [Greconici, Kannwischer, Sprenkels 2020]

NIST Level 3	Key Gen. Speed	Key Gen. RAM
Cortex M4	6M cycles	10KB

NIST Level 3	Sign Speed	Sign RAM
Cortex M4	8M cycles	70KB
Cortex M4	26M cycles	11KB
Cortex M4	6M cycles	21KB
		+ 48KB Flash

Falcon [Pornin, 2019]

NIST Level 1	Key Gen. Speed	Key Gen. RAM
Cortex M4	171M cycles	16KB

NIST Level 1	Sign Speed	Sign RAM
Cortex M4	40M cycles	40KB
Cortex M4	21M cycles	25KB
		+ 57KB Flash

Runtime Efficiency Comparison

Dilithium [Greconici, Kannwischer, Sprenkels 2020]

NIST Level 3	Key Gen. Speed	Key Gen. RAM
Cortex M4	6M cycles	10KB

NIST Level 3	Sign Speed	Sign RAM
Cortex M4	8M cycles	70KB
Cortex M4	26M cycles	11KB
Cortex M4	6M cycles	21KB
		+ 48KB Flash

NIST Level 3	Ver. Speed	Ver. RAM
Cortex M4	2.7M cycles	11KB

Falcon [Pornin, 2019]

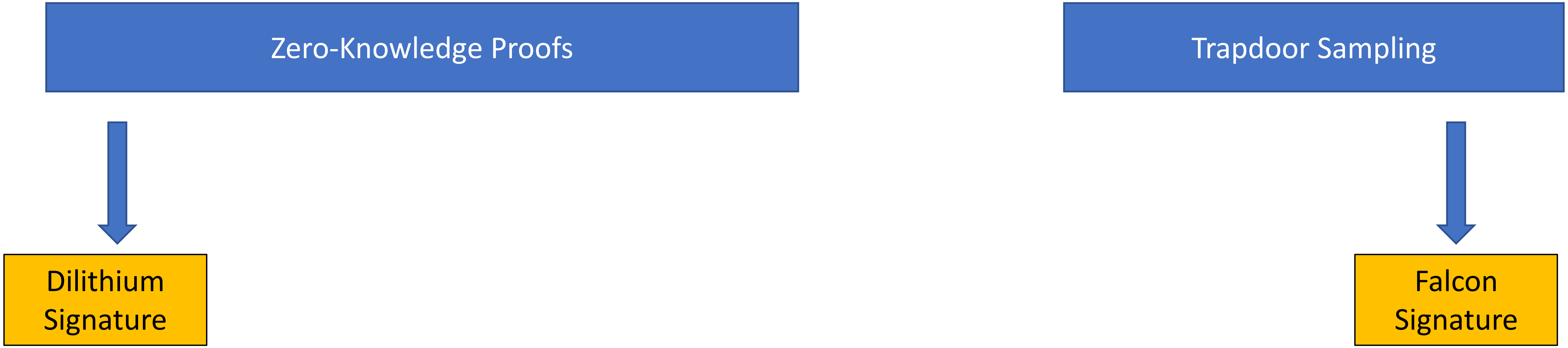
NIST Level 1	Key Gen. Speed	Key Gen. RAM
Cortex M4	171M cycles	16KB

NIST Level 1	Sign Speed	Sign RAM
Cortex M4	40M cycles	40KB
Cortex M4	21M cycles	25KB
		+ 57KB Flash

NIST Level 1	Ver. Speed	Ver. RAM
Cortex M4	0.5 M cycles	4KB

- 80% of Dilithium Verification Time is Keccak

Future Uses of Dilithium / Falcon Techniques



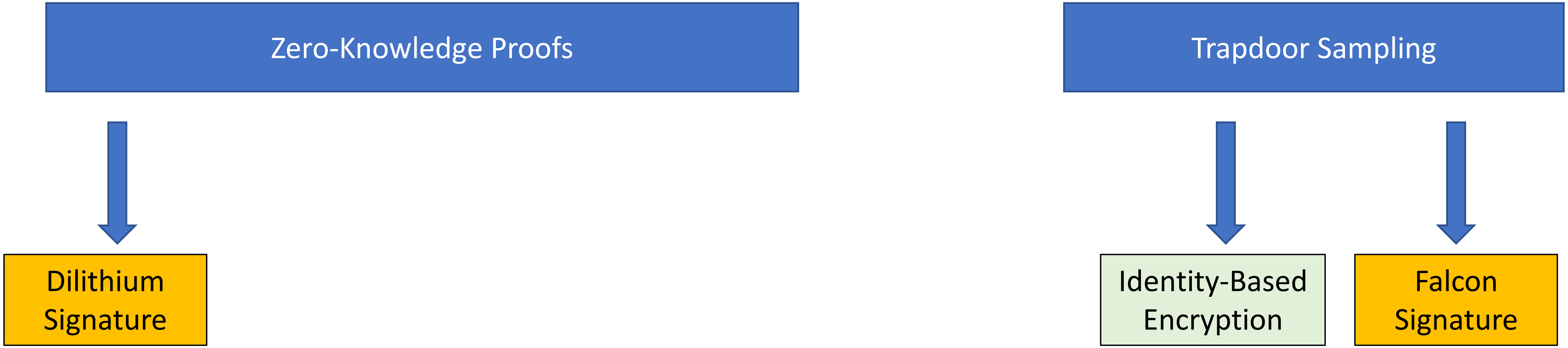
Zero-Knowledge Proofs

Dilithium Signature

Trapdoor Sampling

Falcon Signature

Future Uses of Dilithium / Falcon Techniques



Zero-Knowledge Proofs

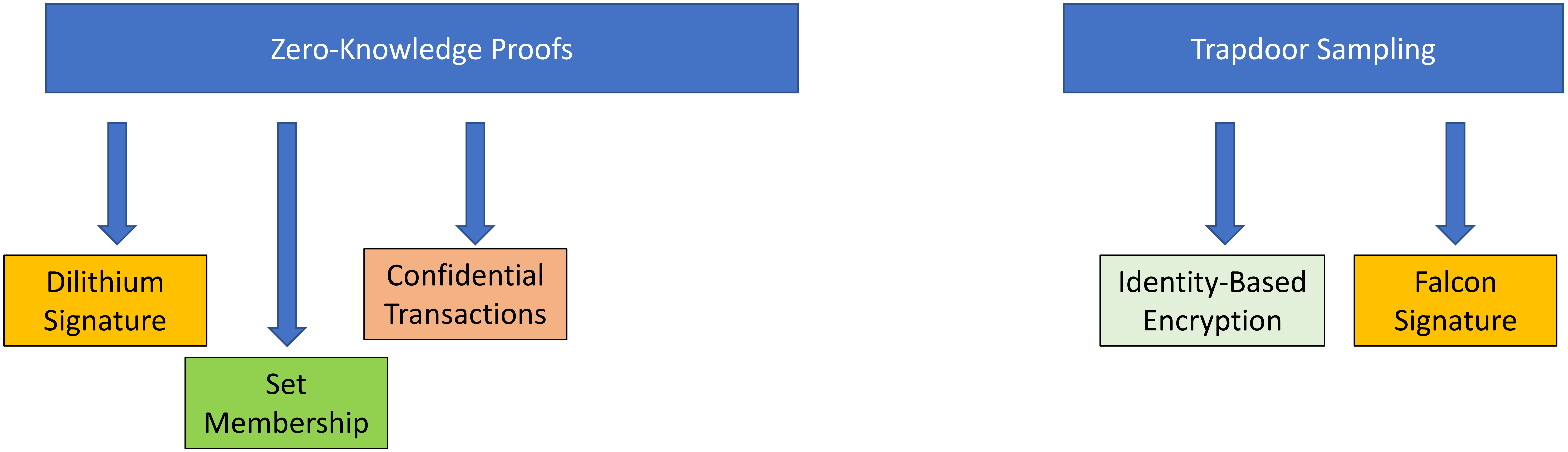
Dilithium Signature

Trapdoor Sampling

Identity-Based Encryption

Falcon Signature

Future Uses of Dilithium / Falcon Techniques



Zero-Knowledge Proofs

Dilithium Signature

Set Membership

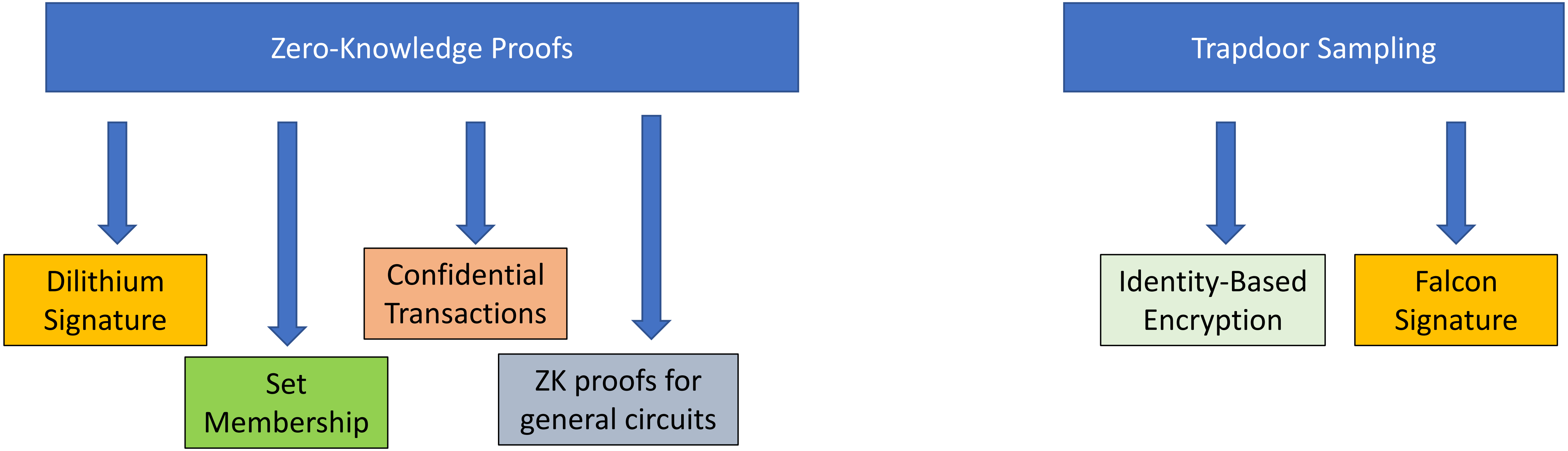
Confidential Transactions

Trapdoor Sampling

Identity-Based Encryption

Falcon Signature

Future Uses of Dilithium / Falcon Techniques



Zero-Knowledge Proofs

Dilithium Signature

Set Membership

Confidential Transactions

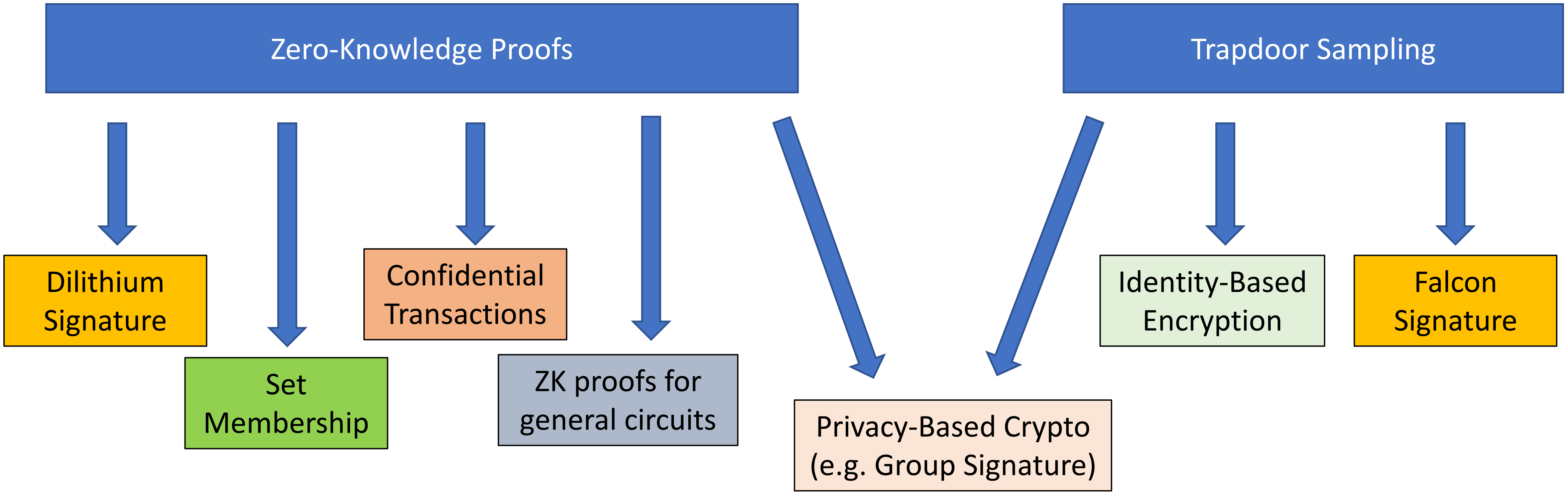
ZK proofs for general circuits

Trapdoor Sampling

Identity-Based Encryption

Falcon Signature

Future Uses of Dilithium / Falcon Techniques



Zero-Knowledge Proofs

Trapdoor Sampling

Dilithium Signature

Set Membership

Confidential Transactions

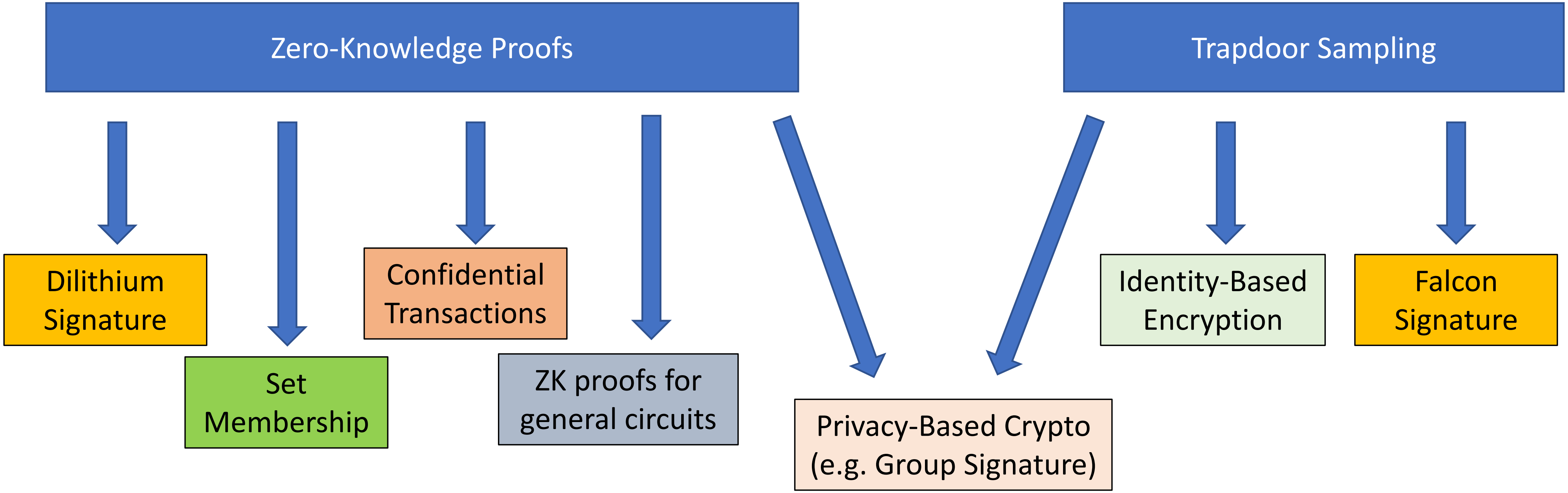
ZK proofs for general circuits

Privacy-Based Crypto (e.g. Group Signature)

Identity-Based Encryption

Falcon Signature

Future Uses of Dilithium / Falcon Techniques



Zero-Knowledge Proofs

Trapdoor Sampling

Dilithium Signature

Set Membership

Confidential Transactions

ZK proofs for general circuits

Privacy-Based Crypto (e.g. Group Signature)

Identity-Based Encryption

Falcon Signature

Lattice-based ZK proofs improved by 3 orders of magnitude in the last 2 years

Lattices are currently the most efficient quantum-safe solution for many of these applications

We should probably get good at the techniques behind them

Only Standardize Dilithium xor Falcon?
(as NIST suggested)

Only Standardize Dilithium xor Falcon? (as NIST suggested)

- Maybe … but I think more discussion is needed.
  - They may both be useful as signatures even if they are both lattice schemes
  - Both, or at least techniques from both, will be useful in the future

Only Standardize Dilithium xor Falcon? (as NIST suggested)

- Maybe … but I think more discussion is needed.
  - They may both be useful as signatures even if they are both lattice schemes
  - Both, or at least techniques from both, will be useful in the future

- Main question for now: are *serious* footguns a serious problem?

  “Dilithium uses only uniform sampling, and is in general much easier to implement than Falcon; on the other hand, Falcon produces much shorter signatures“ [Pornin, 2019]

Only Standardize Dilithium xor Falcon? (as NIST suggested)

- Maybe … but I think more discussion is needed.
  - They may both be useful as signatures even if they are both lattice schemes
  - Both, or at least techniques from both, will be useful in the future

- Main question for now: are *serious* footguns a serious problem?

  "Dilithium uses only uniform sampling, and is in general much easier to implement than Falcon; on the other hand, Falcon produces much shorter signatures" [Pornin, 2019]

- A possible compromise is:
  - Make Dilithium the default option
  - Allow Falcon to be used in certain situations that only require k-time signatures (k ≈ 100 – 1000 is reasonable)
  - NIST standardized 1-time signatures, why not k-time ones?

Only Standardize Dilithium xor Falcon? (as NIST suggested)

- Maybe … but I think more discussion is needed.
  - They may both be useful as signatures even if they are both lattice schemes
  - Both, or at least techniques from both, will be useful in the future

- Main question for now: are *serious* footguns a serious problem?

  "Dilithium uses only uniform sampling, and is in general much easier to implement than Falcon; on the other hand, Falcon produces much shorter signatures" [Pornin, 2019]

- A possible compromise is:
  - Make Dilithium the default option
  - Allow Falcon to be used in certain situations that only require k-time signatures (k ≈ 100 – 1000 is reasonable)
  - NIST standardized 1-time signatures, why not k-time ones?
  - Could also consider the Falcon ideas with less compact, but easier to use and mask (still Gaussian, though) samplers that don't require floating point ops in the "4th round" :
    - MITAKA [Espitau, Takahashi, Tibouchi, Wallet 2020]
    - Zalcon [Fouque, Gerard, Rossi, Yu 2021]

CRYSTALS – Dilithium

https://pq-crystals.org/dilithium/index.shtml

https://github.com/pq-crystals/dilithium

https://github.com/pq-crystals/security-estimates