

Post-Quantum

Cryptography Conference

Real-World Post-Quantum Migrations: Lessons Learned and Performance Results



Michiel Marcus

Cryptographer at TNO



KEYFACTOR

CRYPTO4A

SSL.com

ENTRUST

HID

October 28 - 30, 2025 - Kuala Lumpur, Malaysia

PKI Consortium Inc. is registered as a 501(c)(6) non-profit entity ("business league") under Utah law (10462204-0140) | pkic.org



Partnership for
Cyber Security
Innovation

Real-World Post-Quantum Migrations: Lessons Learned and Performance Results

PKI Consortium, October 30th 2025

Michiel Marcus



The Team



Until March 2025



Belastingdienst



About Me

Michiel Marcus

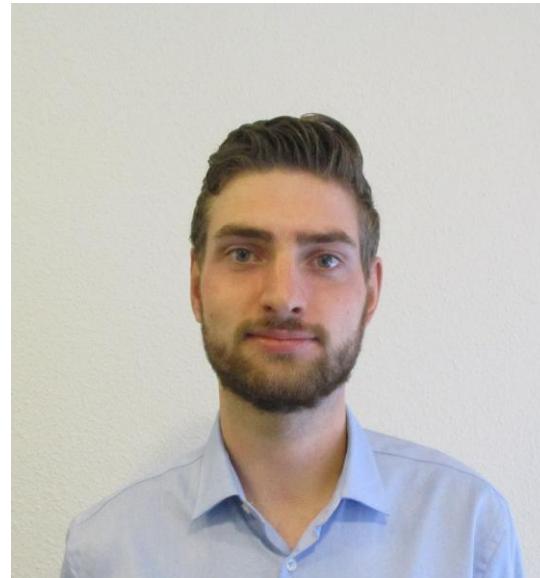
- Cryptographer at TNO

Research Interests

- Post-Quantum Cryptography
- Formal Methods
- Multi-Party Computation

PQC activities

- Delegate at the ETSI Quantum Safe Cryptography Working group
- PQC migration consultant for various Dutch organisations
- Researcher within various PQC research projects
 - HAPKIDO
 - PQC Benchmarking



The start

Trend - Maturing of quantum technology

- Summarized: Every Dutch organization must undergo the same complicated migration process to quantum safe cryptography.

Pain point: Lack of practical insights leads to hesitation to start

- There is not one replacement, but multiple,
- With specific (performance and functional) drawbacks and
- Multiple ways of replacing it (hybrid or not)
- The new algorithms differ from our current cryptography in terms of performance, size, bandwidth and capabilities

Available benchmarks are academic or very limited in scope

- Not capturing system effects: bottlenecks, unexpected behaviour, architectural findings
- Not sharing migration experiences (and pitfalls)



PCSI is a collaboration of



Proposed project:

Let's gain experience!

- Migrate one application each
- Benchmark performance of multiple algorithmic options
- Share experiences, pitfalls, difficulties and successes

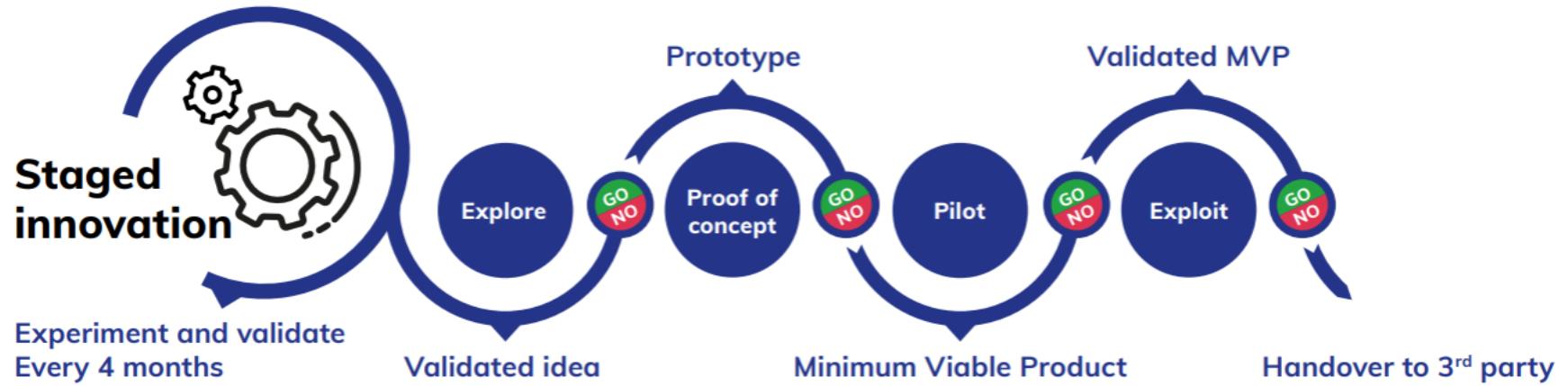
“Prepare now, relax later”

- Remove uncertainty of impact of migration to motivate to act now



PCSI is a collaboration of

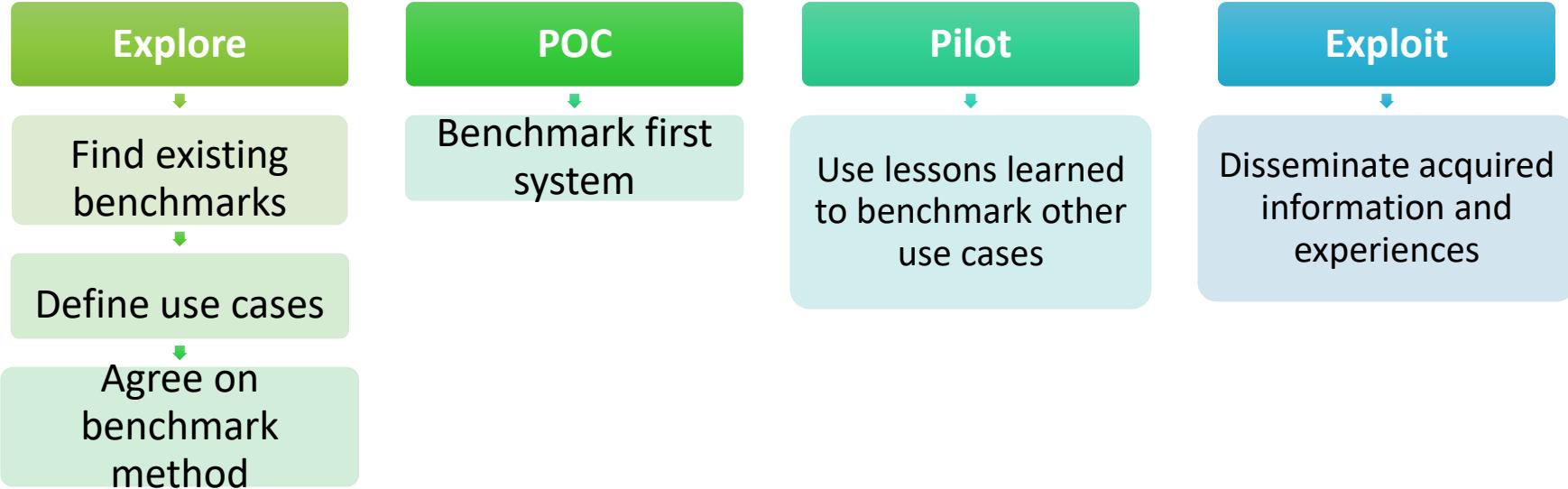
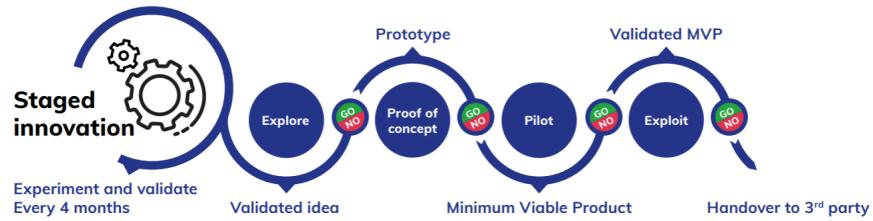




PCSI is a collaboration of

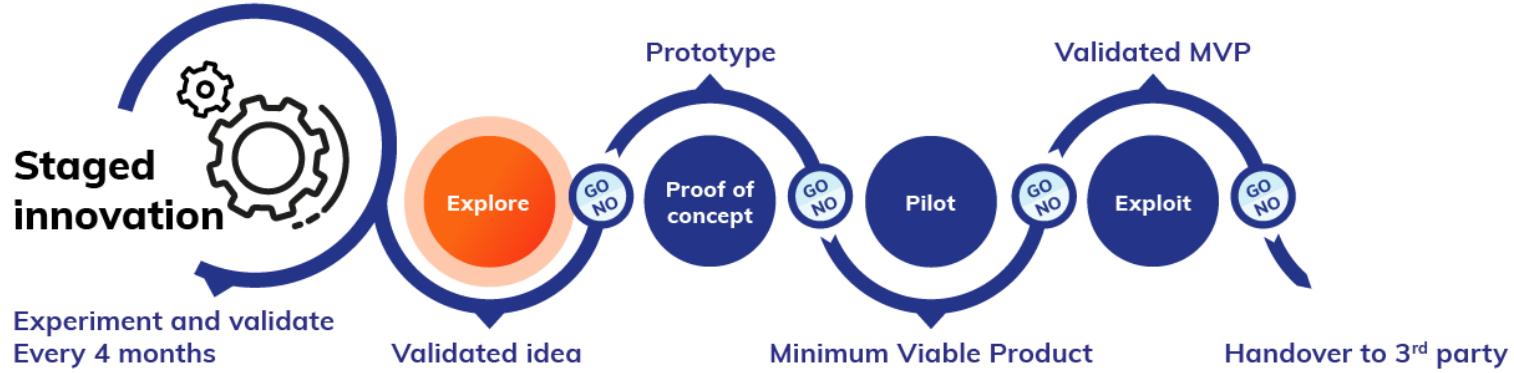


PQC Benchmark project vision



PCSI is a collaboration of

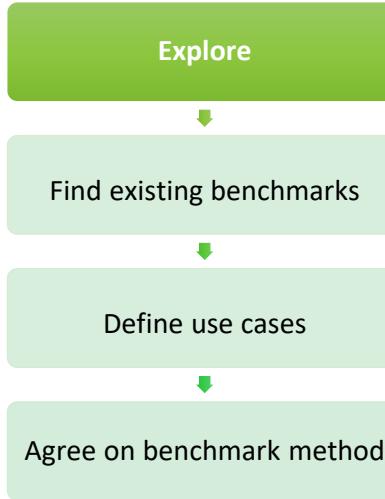
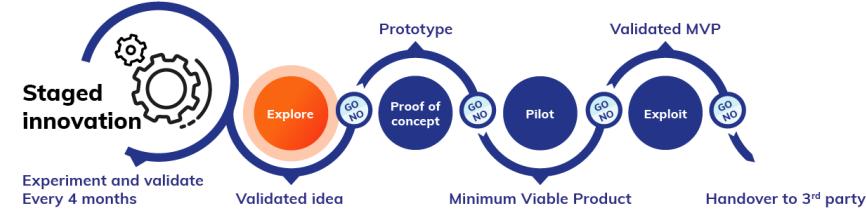




PCSI is a collaboration of



Explore phase results



Unexpectedly hard:
Finding suitable applications at partners
Arranging time and resources for POC



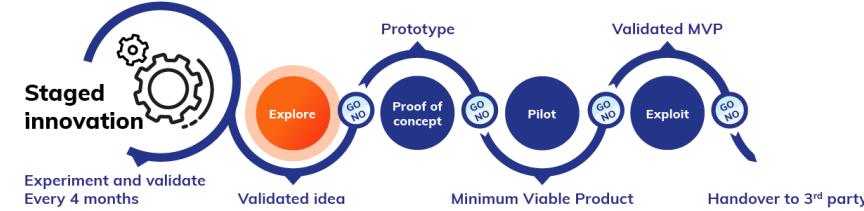
PCSI is a collaboration of



Explore phase results

Good POC Use Case Characteristics

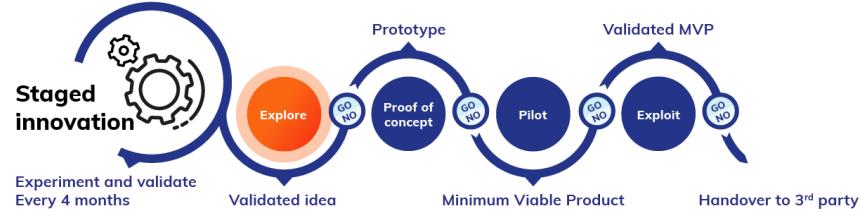
1. The application
 - Interesting for partner
 - Reusable result
 - Impacted by quantum threat
 - Has performance demands
2. The application team has time and is motivated
 - Dependence on suppliers can complicate
3. Library/code/software *can* support PQC algorithms
4. The application behaviour can be benchmarked
 - Load generation, performance measurements



PCSI is a collaboration of



Explore phase results



Benchmark method

PQC Algorithms (NIST level 1 or equivalent):

- Key exchange:
 - **A. Crystals-Kyber (ML-KEM)**,
 - B. Frodo-KEM,
 - C. McEliece
- Digital signature:
 - **D. Crystals-Dilithium (ML-DSA)**
 - E. Falcon
 - **F. Sphincs+ (SLH-DSA)**

Metrics:

- CPU usage
- Storage
- Memory
- Network information(bandwidth, package size, package drops, latency)
- Time (connections/signings/key exchanges per second)

Test cases:

	KEM	DSA	Algorithms to test
0.	Current (RSA)	Current (RSA)	First baseline
1.	ECC	ECC	Second baseline
2.	PQC	ECC	A,B,C
3.	ECC	PQC	D,E,F
4.	PQC	PQC	All combinations of A-C with D-F
5.	(ECC+PQC)	ECC	A,B,C, hybrid with ECC
6.	ECC	(ECC+PQC)	D,E,F, hybrid with ECC
7.	(ECC+PQC)	(ECC+PQC)	All combinations of A-C with D-F

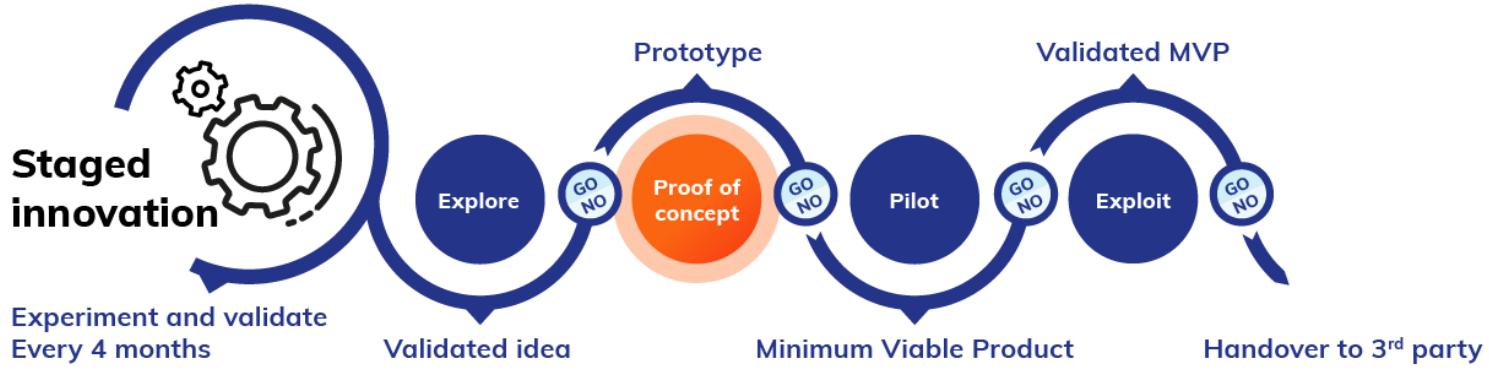
Information gathered:

- Performance indicators
- System details (hardware, platform, library version, etc)
- Algorithm details currently being tested (OID, key size, sphincs type)
- Encountered problems, bottlenecks, learned lessons about implementation, adjustments required outside of replacing the algorithm



PCSI is a collaboration of

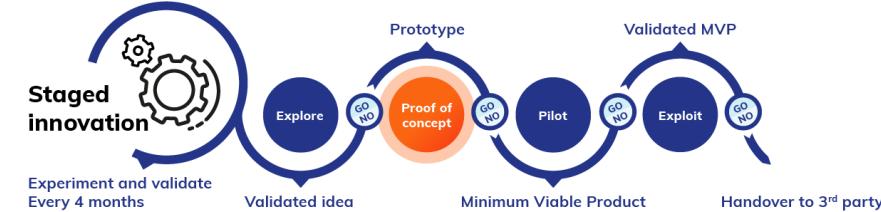




PCSI is a collaboration of



POC phase



POC



Benchmark first system

In-house application chosen at ABN Amro

- No vendor dependency, capable programmers
- High reliance on quick key exchange (done many times)

However, life is what happens when you're busy making other plans

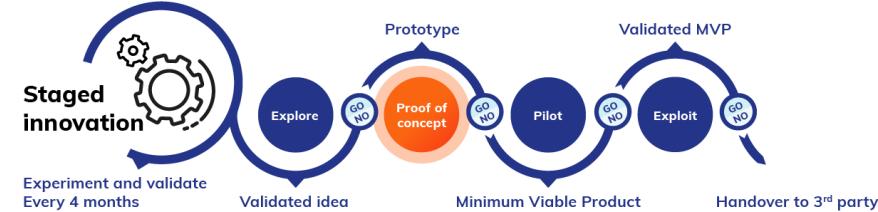
- Unforeseen unavailability of programming team
- Decision to transfer migration work to TNO
- Benefit: experience in outsourcing of migration!



PCSI is a collaboration of



POC phase results



POC

Benchmark first system

293 files changed, 3918 lines of code adjusted just to support PQC key exchange

- Most changes to make code crypto agile

An unexpected lesson in crypto agility:

- Cryptography was used in many locations in the code
- Abstracting the algorithm away was less time consuming
- Also makes the code more agile for future cryptographic changes
- Afterwards, new algorithms only requires adding the algorithm code itself and changing a setting

```
private void initKeys() {  
    keyPair1 = getRSA(0);  
    keyPair2 = getRSA(1);  
}
```



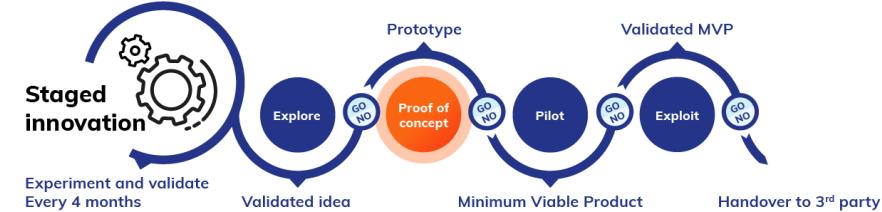
```
private void initKeys() {  
    keyPair1 = getKeyPair(RSA, 0);  
    keyPair2 = getKeyPair(RSA, 1);  
}
```



PCSI is a collaboration of



POC phase lessons learned



POC



Benchmark first system

TNO

- Comprehending someone else's multi-year project is quite hard
- Ensure good code practices now (crypto-agility, modularity, documentation) will decrease migration cost
- A more crypto-agile system is way easier to migrate
- Outsourcing PQC migration limits experience gain at the partner, and has quite high overhead

ABN Amro

- Learned a lot about processes, technical impact and what to do with approvals from management.
- Overall conclusion: Crypto Agility enables Functional Agility

Important points for pilot

- Reserve enough time
- Hybrid cryptography is hard - make sure you know which "hybrid flavor" you'll use
- Consider first making code crypto agile, then adding PQC support



PCSI is a collaboration of



Technical details ABN Amro



PCSI is a collaboration of



Key Code Concepts

1

Use Universal APIs

```
public AsymmetricKeysGenerator get(AsymmetricKeyType keyType) {  
  
    return switch (keyType) {  
        case EC -> new AsymmetricGenericKeysGenerator("EC", "BC");  
        case RSA -> new AsymmetricGenericKeysGenerator("RSA", "BC");  
        case MLKEM -> new AsymmetricGenericKeysGenerator("KYBER512", "BCPQC");  
        case FRODO -> new AsymmetricGenericKeysGenerator("Frodo", "BCPQC");  
        case MCELIECE -> new AsymmetricGenericKeysGenerator("CMCE", "BCPQC");  
    };  
}
```

- Logic for cryptographic algorithms remains in separate files to maintain a structured overview
- Other modules can retrieve a key pair generator with a unified API

2

Use Dynamic Memory Structures

```
private Map<AsymmetricKeyType, EncryptionKeys> encryptionKeysMap;
```

- A dynamic structure can map algorithm identifiers to specific instances for that algorithm
- This replaces hardcoded variables for each algorithm



PCSI is a collaboration of



Key Code Concepts

3

Use Parametrized Tests

```
public static Stream<Arguments> testGenerateKeyPair() {  
    return Arrays.stream(AsymmetricKeyType.values()).map(  
        (x) -> arguments(x)  
    );  
}  
  
@ParameterizedTest  
@MethodSource  
public void testGenerateKeyPair(AsymmetricKeyType keyType) {  
    // test code body  
}
```

4

Use an Appropriate Key Store

```
@ElementCollection  
@CollectionTable(name = "example_table_name", joinColumns = @JoinColumn(name = "example_id"))  
@MapKeyColumn(name = "key_type")  
private Map<AsymmetricKeyType, @NotNull @Valid EncryptionKeys> encryptionKeysMap;
```

- When a new key type is added, the test code does not need to be updated
- Parametrized tests usually test specific behaviour independent of the algorithm used
- Keys should be stored in a type-agnostic way, such that the infrastructure does not break when different key types are stored
- If a database layout is generated through code annotations, appropriate annotations should be used on dynamic memory structures



PCSI is a collaboration of



Key Code Concepts

5

Use a Modular class structure

```
graph TD; A["FrodoKemAesEncapsulationDecapsulationTest  
McElieceAesEncapsulationDecapsulationTest  
MIKemAesEncapsulationDecapsulationTest  
RsaOaepSha1AesEncryptionDecryptionTest  
RsaOaepSha256AesEncryptionDecryptionTest"]; B["EndToEndFrodoKeyAcceptorTest  
EndToEndKeyAcceptorTest  
EndToEndMCELIECEKeyAcceptorTest  
EndToEndMLKEMKeyAcceptorTest"]; C["FrodoKemAesEncapsulationDecapsulationTest  
McElieceAesEncapsulationDecapsulationTest  
MIKemAesEncapsulationDecapsulationTest  
RsaOaepSha1AesEncryptionDecryptionTest  
RsaOaepSha256AesEncryptionDecryptionTest"]
```

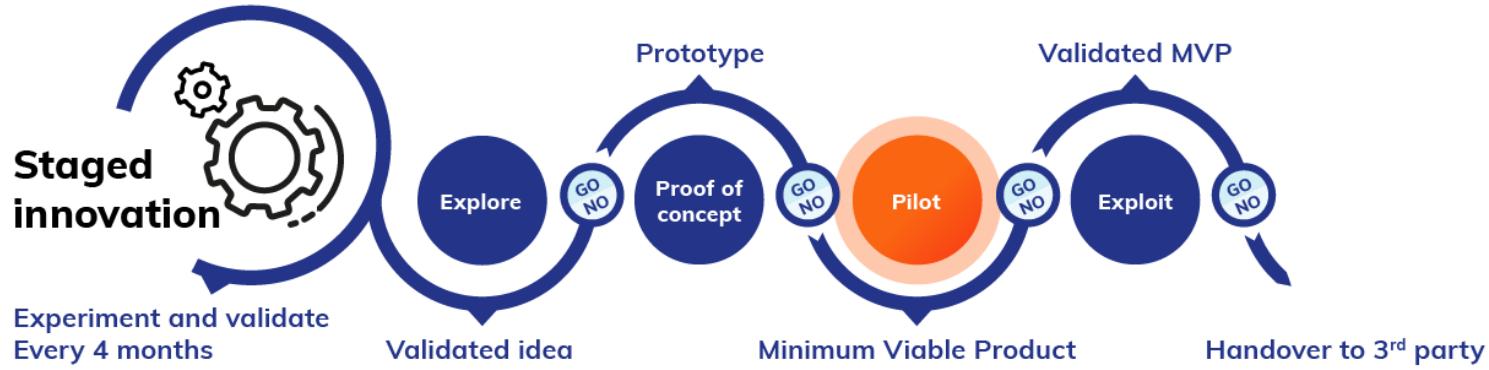
Do not do this!

- When every new key type requires multiple new classes to be created, the code is not crypto agile
- Ideally, abstract constructions are created in a single class that can reference concrete algorithms
- This way, modular tests can also be written more easily



PCSI is a collaboration of

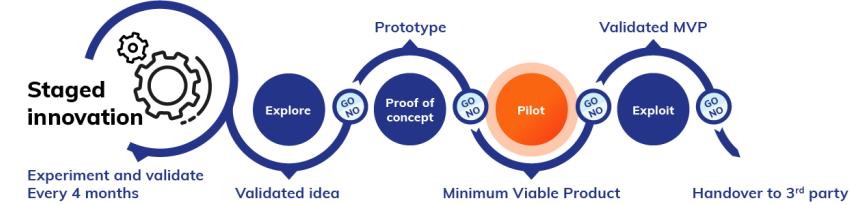




PCSI is a collaboration of



Pilot phase



Pilot

Use lessons learned to benchmark other use cases

Belastingdienst:

- High performance application
- Vendor dependency

ING:

- Critical component PKI
- Including hardware support



PCSI is a collaboration of

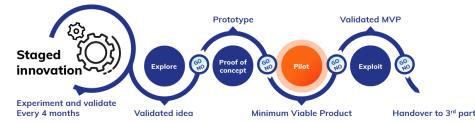


Belastingdienst



PCSI is a collaboration of





Pilot phase results Belastingdienst

Pilot

Use lessons learned to benchmark other use cases

1. Vendor PQC support not available yet, pivot to:
2. Connection setup towards application (PQC certificates), but certificates not accepted yet by application, pivot to:
3. Insert reverse proxy (OpenSSL + OQS) on application server, test connection setup with clients using PQC certificates

- ➔ Actually an interesting architectural solution
- ➔ Enables tests of most algorithmic combinations, including hybrid

Results (sneak peek)*:

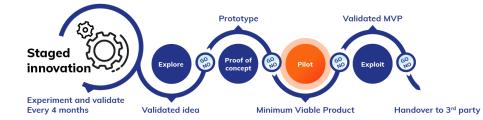
- Not as bad as expected! 20% performance hit for full hybrid replacement
- FrodoKEM did better than expected
- SLH-DSA much slower, but parameters matter much

* full result in blog



PCSI is a collaboration of





Pilot phase lessons learned Belastingdienst

Pilot

Use lessons learned to benchmark other use cases

- Don't be delayed by vendor products not being quantum ready.
- Start your integration journey by decoupling
 - ✓ Enables first learning experiences with PQC technology
 - ✓ Enables availability of working test setup
 - ✓ Enables benchmark data for later comparison
 - ✓ Can be used to protect internet facing side



PCSI is a collaboration of



Technical details Belastingdienst



PCSI is a collaboration of



Belastingdienst use case on Certificates

Benchmark a message queueing application in combination with PQC certificates

Considerations:

- Important high performance component for many IT companies
- Closely tied to daily work for many teams
- Most likely resources already available, like (parts of) tooling, test environment, knowledge

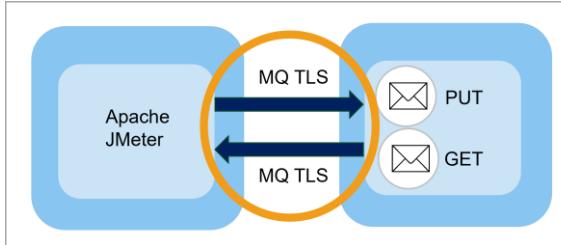


PCSI is a collaboration of



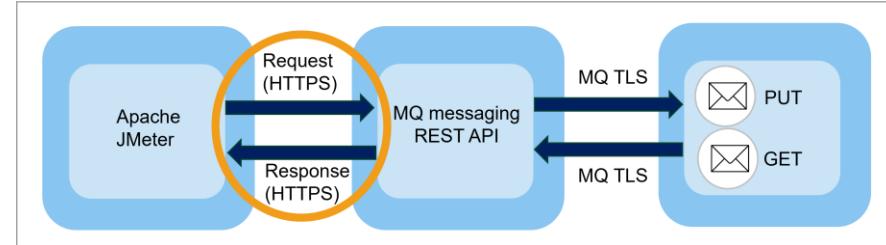
Belastingdienst use case on Certificates

Original idea:



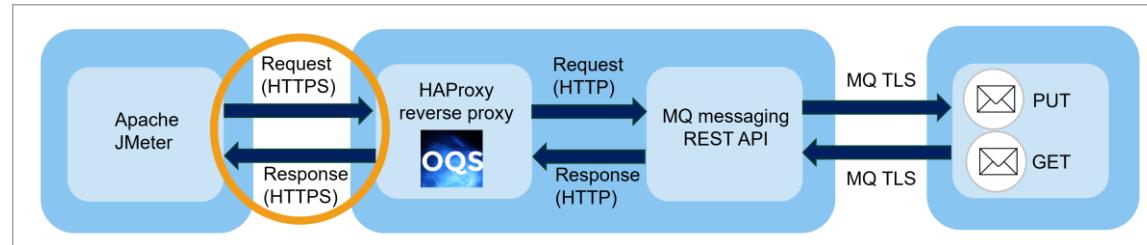
We ran into unavailability of certificate tooling team, as a result no vendor certificate could be used. Also MQ product not supporting PQC yet.

First alternative:



Tried with REST API and OpenSSL certificate. REST API also not supporting PQC yet. OpenSSL with support for PQC available.

Final setup:



Decoupled architecture using reverse proxy from framework OpenQuantumSafe^{*)}. SSL offloading for the time being until vendors support PQC. Focus on baseline benchmarking and automation. Later proxy can be removed and test re-run.



= PQC configured connection

^{*)} <https://openquantumsafe.org>



PCSI is a collaboration of



Test setup

PQC Algorithms (NIST level 1 or equivalent):

- Key exchange:
 - Crystals-Kyber (ML-KEM),
 - Frodo-KEM,
 - McEliece
- Digital signature:
 - Crystals-Dilithium (ML-DSA)
 - Falcon
 - Sphincs+ (SLH-DSA)

Metrics:

- CPU usage
- Storage
- Memory
- Network information(bandwidth, package size, package drops, latency)
- Time (connections/signings/key exchanges per second)

Test cases:

- 1. Current algorithms
- 2. Everything ECC
- 3. Only key exchange using PQC
- 4. Key exchange and digital signatures to PQC
- 5. Key exchange **hybrid**
- 6. Digital signature **hybrid**
- 7. Digital signatures replaced by PQC
- 8. Key exchange and digital signatures **hybrid**

Information to gather:

- Performance indicators
- Systemdetails (hardware, platform, version of bouncycastle, etc)
- Algorithm details currently being tested (key size, SPHINCS+ type, library version)
- Encountered problems, bottlenecks, learned lessons about implementation, adjustments required outside of replacing the algorithm

Tested/measured Not explicitly measured Not tested



PCSI is a collaboration of



Additional details

- In all tests TLS_AES_256_GCM_SHA384 was used for symmetric encryption
- Metric measured: number of messages per second
- CPU usage, memory and network information were observed, not measured
- Message size varied for test configurations
- 1 run = 2.500 messages
- First run used 1 thread, later runs used respectively
4 and 8 threads, number of messages 10.000 and 20.000

Software	Version
Linux	RHEL 9
TLS	1.3
OpenSSL	3.2.3
Liboqs	0.12.1
OQS Provider	0.8.1
cURL	8.12.1
HAProxy	3.1.0
JMeter	5.6.3

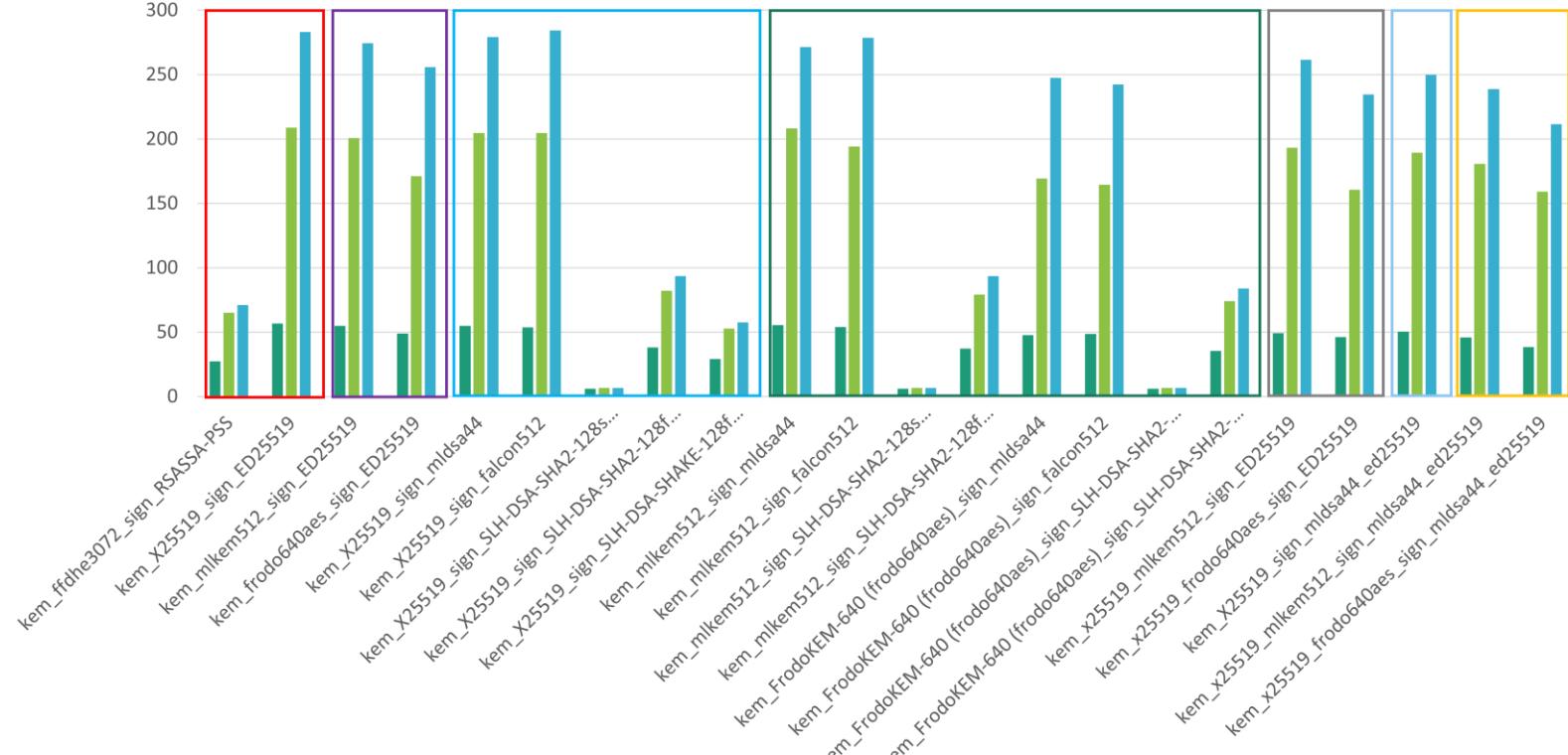


PCSI is a collaboration of



Benchmark data

22 test configurations (baselines in red frame)



PCSI is a collaboration of



■ msg/sec 1 thread (2.500)

■ msg/sec 4 threads (10.000)

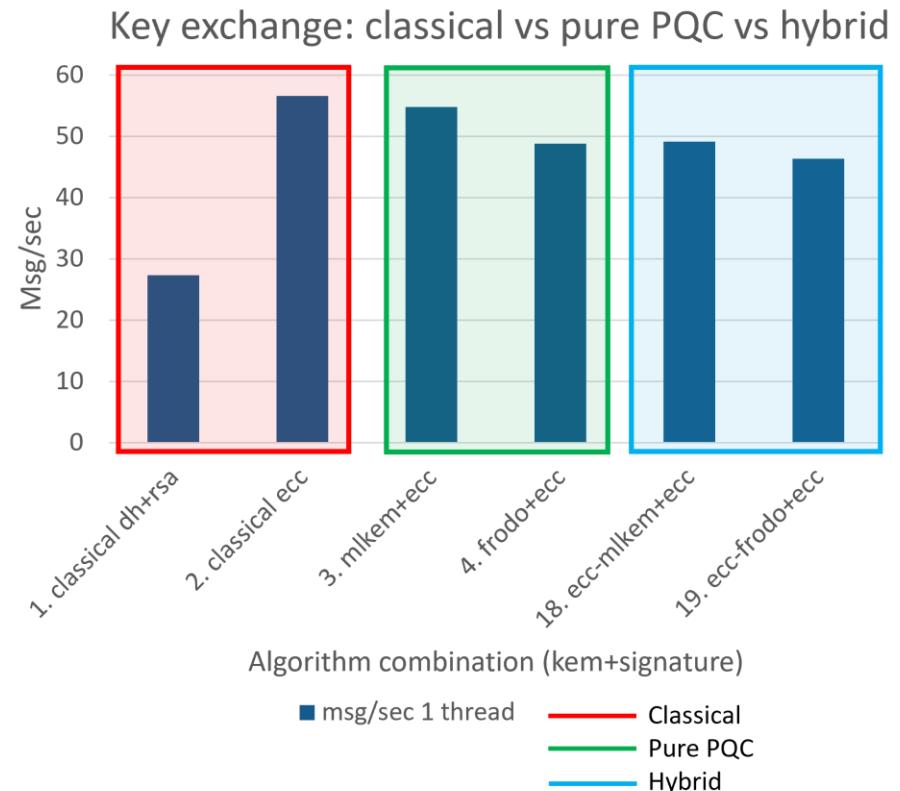
■ msg/sec 8 threads (20.000)

Benchmark Results – KEMs

Comparing PQC KEMs, in pure or hybrid configuration, to the RSA and ECC baselines.

The results for frodo640aes and mlkem512 are quite close to that of traditional ECC, and faster than RSA.

The performance loss of the hybrid KEM versus the single KEMs is a lot less than expected, at most 11%.



PCSI is a collaboration of



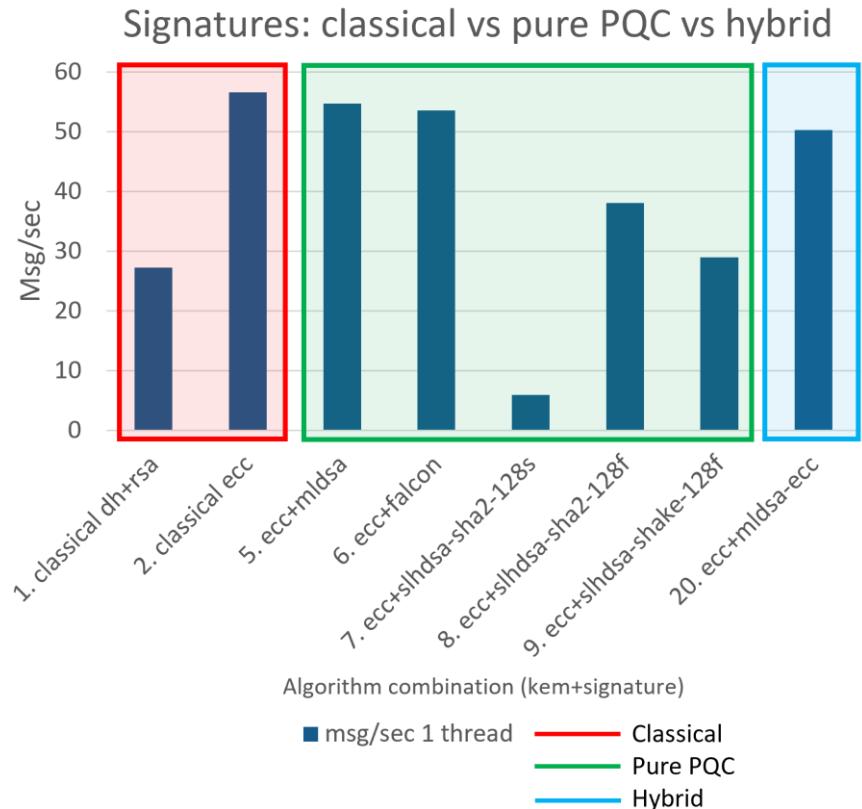
Benchmark Results – Signatures

Comparing PQC Signatures, in pure or hybrid configuration, to the RSA and ECC baselines.

Digital signature results vary much more. Falcon and ML-DSA are slightly slower than ECC. SLH-DSA is significantly slower: 40% slower for the speed optimized variant, 90% slower for the size optimized variant.

The hybrid test with ML-DSA was 12% slower than the classical baseline with ECC, but still faster than RSA.

Size-optimized SLH-DSA showed max CPU usage, that seems to be a bottleneck.



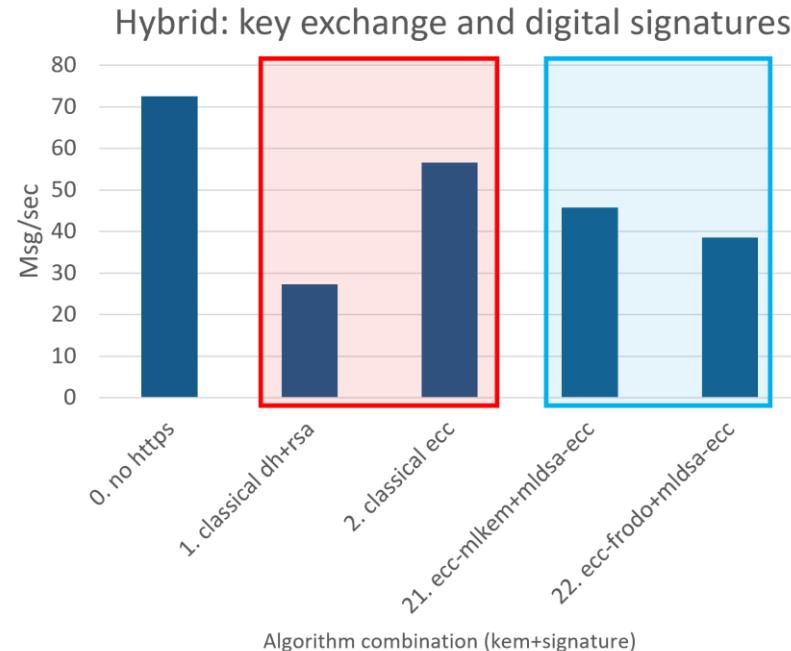
PCSI is a collaboration of



Benchmark Results – All hybrid

Comparing hybrid configuration of PQC KEMs and signatures to the RSA and ECC baselines.

Combining the fastest classical algorithms with the fastest post-quantum algorithms for KEMs and DSAs, we get about a 20% performance decrease, compared to ECC. It is still faster than the RSA baseline.



PCSI is a collaboration of
ING ABN AMRO TNO



■ msg/sec 1 thread — Classical
 — Hybrid

Benchmark Results – Message size

Even though TLS 1.3 message size in this test did not seem to matter much, it can be important for other use cases.

FrodoKEM and SLH-DSA with speed optimization produce significantly bigger messages.

Hybrid solutions with ML-KEM and ML-DSA have 2-6x bigger messages, with the certificate message increasing the most.

KEM algorithms influence Client hello and Server hello messages

	Baseline		KEM PQC		KEM hybrid	
	1. RSA	2. ECC	3. MLKEM	4. FrodoKEM	18. ECC-MLKEM	19. ECC-FrodoKEM
Client hello	778	512	1.194	10.010	1.226	10.042
Server hello	474	122	858	9.810	890	9.842

DSA algorithms influence Certificate and Cert Verify messages

	Baseline		Signature PQC					Sign. hybrid
	1. RSA	2. ECC	5. MLDSA	6. Falcon	7. SLHDSA-	8. SLHDSA-	20.	
Certificate	1.310	803	4.891	4.472	3.604	3.604	3.604	4.937
CERT verify	520	72	2.428	664	7.864	17.096	17.096	2.504



PCSI is a collaboration of



ING



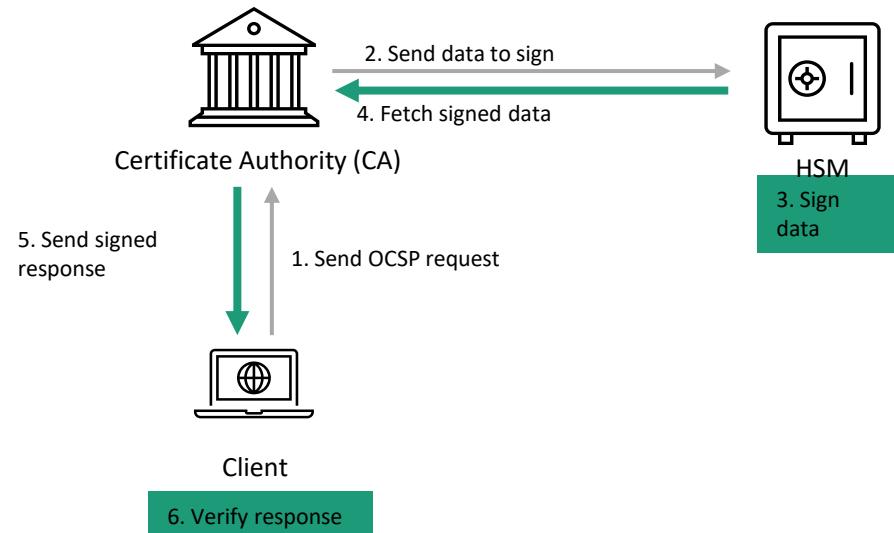
PCSI is a collaboration of



ING Use case on PKI

Benchmarking OCSP (Online Certificate Status Protocol) Performance

- The setup also involves issuing/revoking certificates, storing keys in HSMs.
- OCSP performance is critical in PQC transition due to the increased computation and communication complexity.
- Despite decreasing demand on the usage of OCSP in the TLS ecosystem, in banking usage of OCSP is inevitable. We foresee that OCSP will be needed longer due to business and risk requirements.

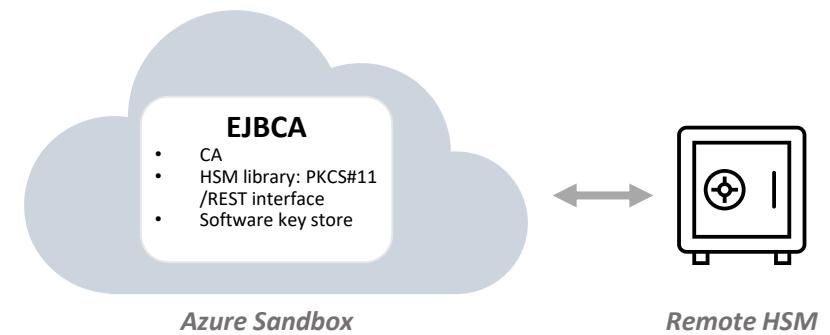


PCSI is a collaboration of



Deployment

- HSM
 - External HSMs are used to avoid networking complications.
- PKI Software: EJBCA
 - Cloud sandbox deployment for the PKI appliances instead of on-prem deployment to avoid complications in networking with the usage of external HSM.
 - 9.3.0 Alpha-1 – Test container
- Tested PQC Algorithms
 - ML-DSA
- Required interface
 - PKCS#11
 - REST API
- OCSP Functionality
 - Functionality implemented for this use case in the test container
- Client-side configuration:
 - Bouncy Castle 1.80
 - OpenSSL 3.5 (Alpha release)



KEYFACTOR



PCSI is a collaboration of



Benchmark Results

- Vendor readiness
 - 7 HSM vendors were contacted
 - 3 vendors only provided **older** versions of ML-DSA (ML-DSA-IPD or Crystals-Dilithium), not the standardized version ML-DSA
 - 1 requires additional configuration that couldn't be achieved within PoC timelines
 - PKCS#11 support often not (yet) ready
 - Hardware acceleration often not yet available
- Protocol readiness
 - No hybrid support in OCSP protocol
- Library readiness
 - Creating CSRs for hybrid not supported by OpenSSL

Assessment period Q1 2025.

Vendor	Supported Algorithms	Interface	Suitable for PoC	Additional Info
Vendor 1	Dilithium Falcon SPHINCS+	REST API	No	
Vendor 2	ML-DSA-IPD Falcon, SPHINCS+ XMSS, LMS/HSS	PKCS#11	No	
Vendor 3	Dilithium	PKCS#11	No	
Vendor 4	ML-DSA-IPD ML-DSA LMS	REST API	Yes, due to specific support.	Access to special cluster with ML-DSA support is provided. Not on dedicated hardware appliances.
Vendor 5	ML-DSA SLH-DSA XMSS, LMS/HSS	REST API PKCS#11 -unclear	Unclear	
Vendor 6	ML-DSA SLH-DSA	PKCS#11 REST API	Yes	Not used due to config issues
Vendor 7	ML-DSA	PKCS#11	No	Uses SoftHSM emulator

*Vendor names are anonymized, NDA in place.



PCSI is a collaboration of



Benchmark Results

Algorithm Names & Versions: Initial Public Draft (IPD) vs Final Version (FIPS)

- *The algorithms went through several changes from their announcement as final candidates (IPD) and their final version (FIPS).*
- *The changes may not necessarily affect the algorithm output, but they are represented by different Object Identifiers (OIDs) and algorithm names.*
- *It is important for vendors to provide the final version of the algorithms for compatibility with correct OID and algorithm name.*

	Object Identifier (OID)	Algorithm name
Pre-standard (IPD)	1.3.6.1.4.1.2.267.7.4.4	Dilithium2 / ML-DSA-IPD-44
	1.3.6.1.4.1.2.267.7.6.5	Dilithium3 / ML-DSA-IPD-65
	1.3.6.1.4.1.2.267.7.8.7	Dilithium5 / ML-DSA-IPD-87
Standard (FIPS 204)	2.16.840.1.101.3.4.3.17	ML-DSA-44
	2.16.840.1.101.3.4.3.18	ML-DSA-65
	2.16.840.1.101.3.4.3.19	ML-DSA-87

*Only OIDs and algorithm names used in this benchmarking project is listed for brevity.



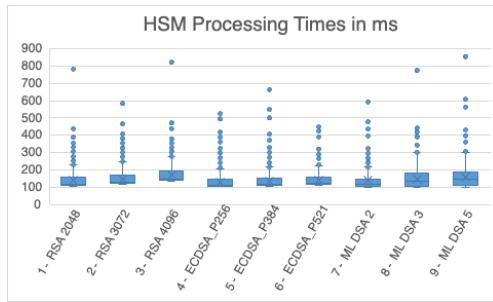
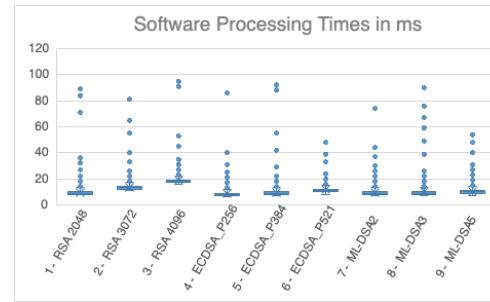
PCSI is a collaboration of



Benchmark Results

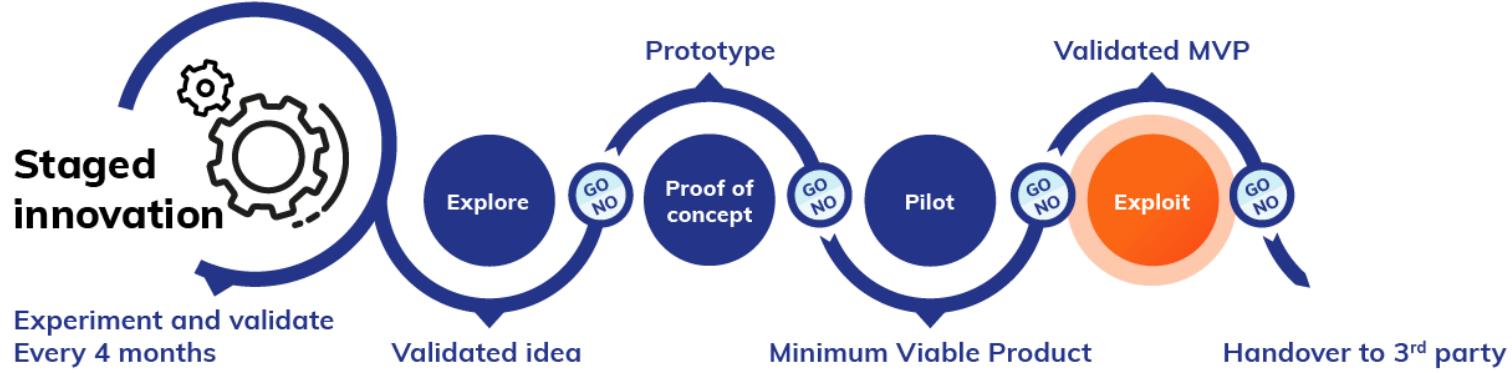
- Performance
 - *PQC algorithm is faster than classical counterpart RSA.*
 - *Key sizes are much larger, but they didn't impact OCSP performance.*
 - *The performance difference between hardware (HSM) and software is mostly caused by additional network delays.*

CA Name	Request length	Response length	Time (Software)	Time (HSM)
RSA 2048	142	1333	0.0132	0.139
RSA 3072	142	1717	0.0168	0.149
RSA 4096	142	2101	0.0221	0.171
ECDSA p256	142	751	0.0115	0.130
ECDSA p384	142	844	0.0114	0.133
ECDSA p521	142	953	0.0123	0.140
ML-DSA-2	142	6695	0.0129	0.138
ML-DSA-3	142	9113	0.0127	0.147
ML-DSA-5	142	12389	0.0134	0.159



PCSI is a collaboration of

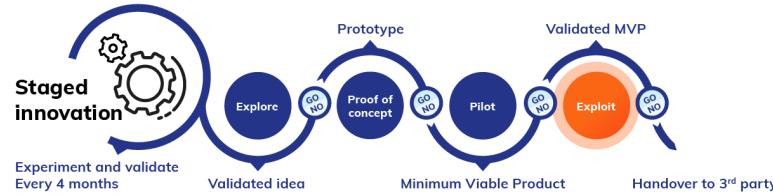




PCSI is a collaboration of



Exploit phase



Exploit

↓
Disseminate acquired
information and experiences

Blog series

- On: The essential role of the manager
- On: Code agility and the programming side of the migration
- On: Vendor management and HSM readiness
- On: Architecture considerations, alternative strategies and performance impact



PCSI is a collaboration of



Why work together on experiments?

Experiments

- Highlight organisational hurdles
- Creates internal knowledge and awareness: kickstarts the migration
- When doing early: motivates vendors

Cooperation

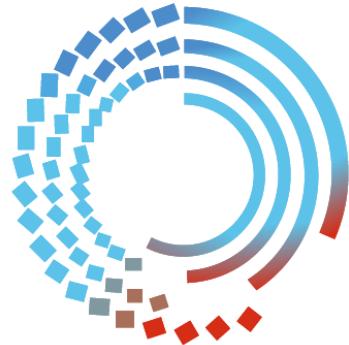
- Lowers time, risk and costs
- Helps identify different strategies, blind spots
- Accelerates knowledge gain
- Creates enthusiasm and
- Helps to keep momentum when things are tough



PCSI is a collaboration of



Partnership for Cyber Security Innovation is a collaboration of



Belastingdienst



ABN·AMRO

TNO

achmea

www.pcsi.nl



follow us

