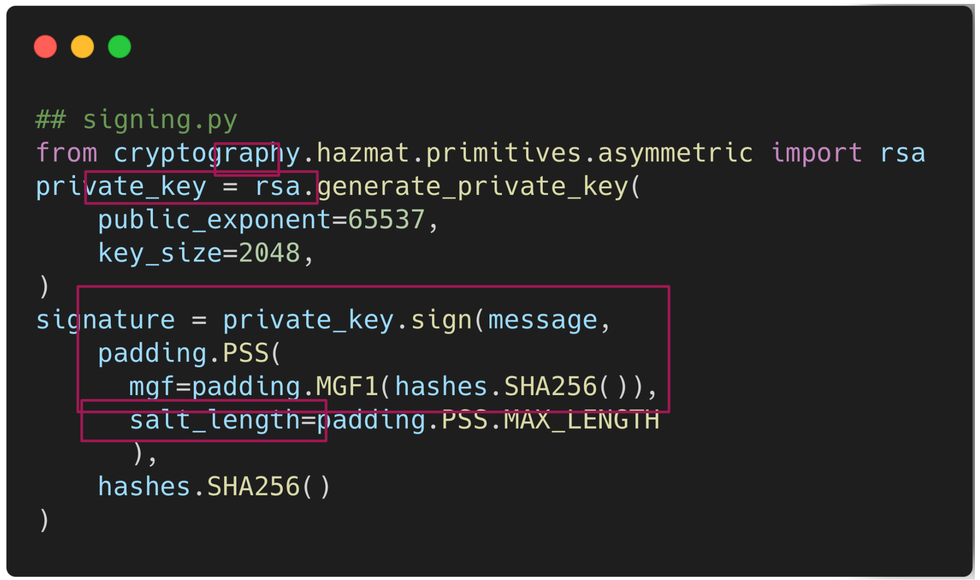
Cryptographic Agility for Applications: An Assessment Framework and Principled API Design

Messmer Grégoire
Research Engineer
gregoire.messmer@ibm.com

Migrating cryptography: an architectural problem

Algorithm identity and parameters embedded in application code



```
## signing.py
from cryptography.hazmat.primitives.asymmetric import rsa
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
signature = private_key.sign(message,
    padding.PSS(
      mgf=padding.MGF1(hashes.SHA256()),
      salt_length=padding.PSS.MAX_LENGTH
      ),
    hashes.SHA256()
)
```

A solution: **cryptographic agility**

Focus: **Agility for applications**

1. Replacing algorithm with no code change to the application → PQC migration scenario
2. Replacing an algorithm's implementation with no code change to the application → Weak implementation (eg. CVEs) scenario

Contributions

1. Assessment Framework

- 7 independent dimensions to characterize agility of cryptographic APIs

2. Evaluation of existing cryptographic APIs

- Evaluation of cryptographic libraries and systems against the assessment framework
- PKCS#11, OpenSSL 3.0, JCA, Tink, AWS KMS, Vault Transit

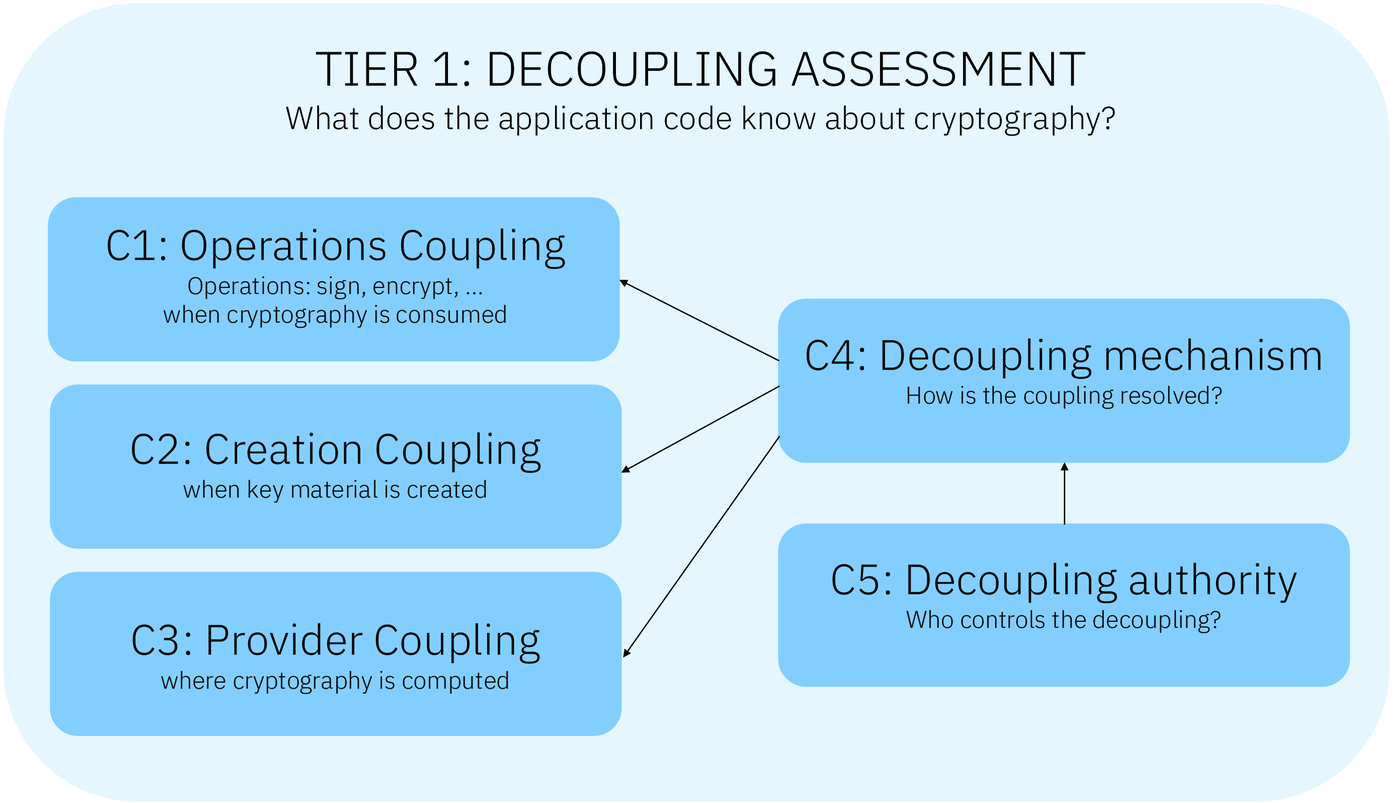
3. API Design Principles

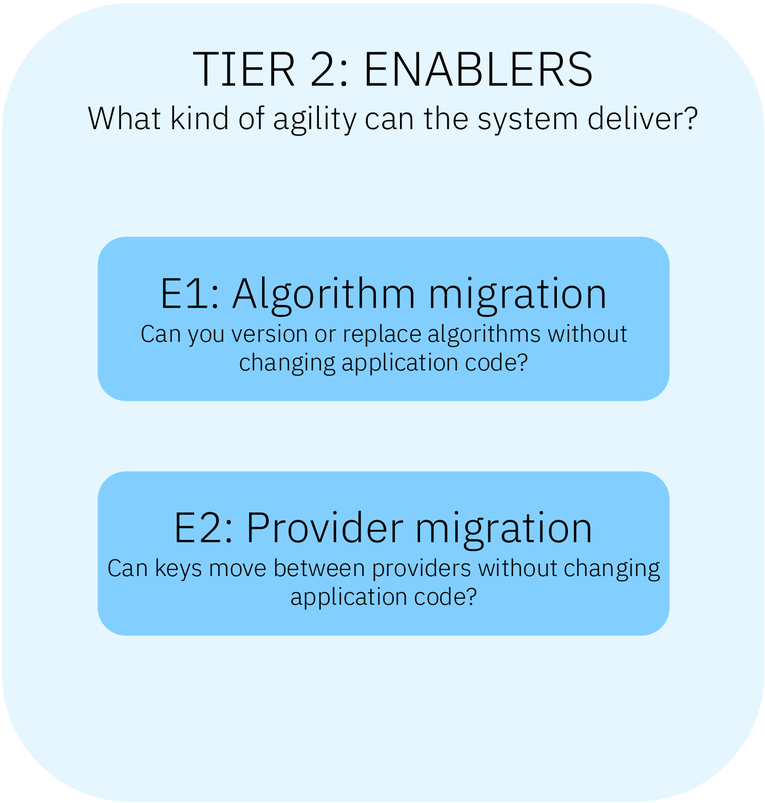
- Fills the gaps revealed by the evaluation
- Provides guidance for designing agile APIs

4. Concrete Patterns

- Protocol Buffers patterns that follow the design principles

Agility: two tiers, multiple dimensions





C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed

Algorithm-specific functions

rsa_sign(key, data)
ecdsa_sign(key, data)

1.0

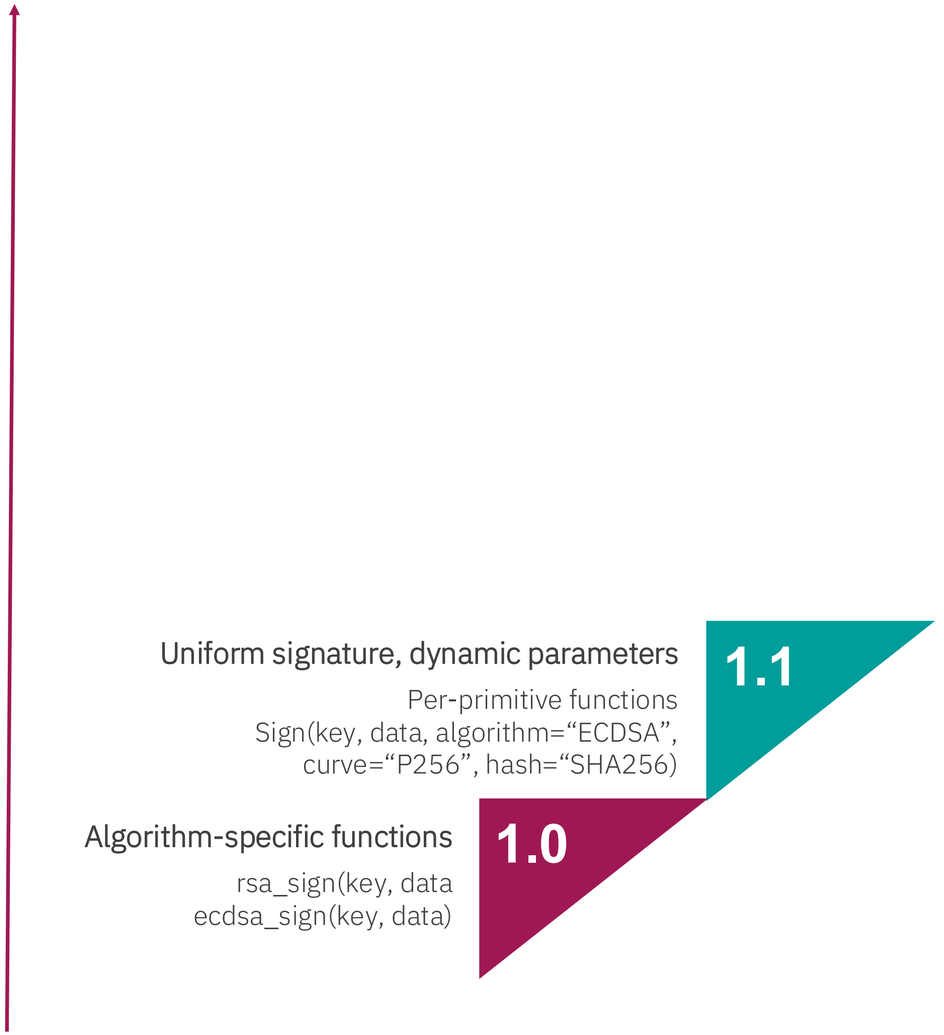
C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



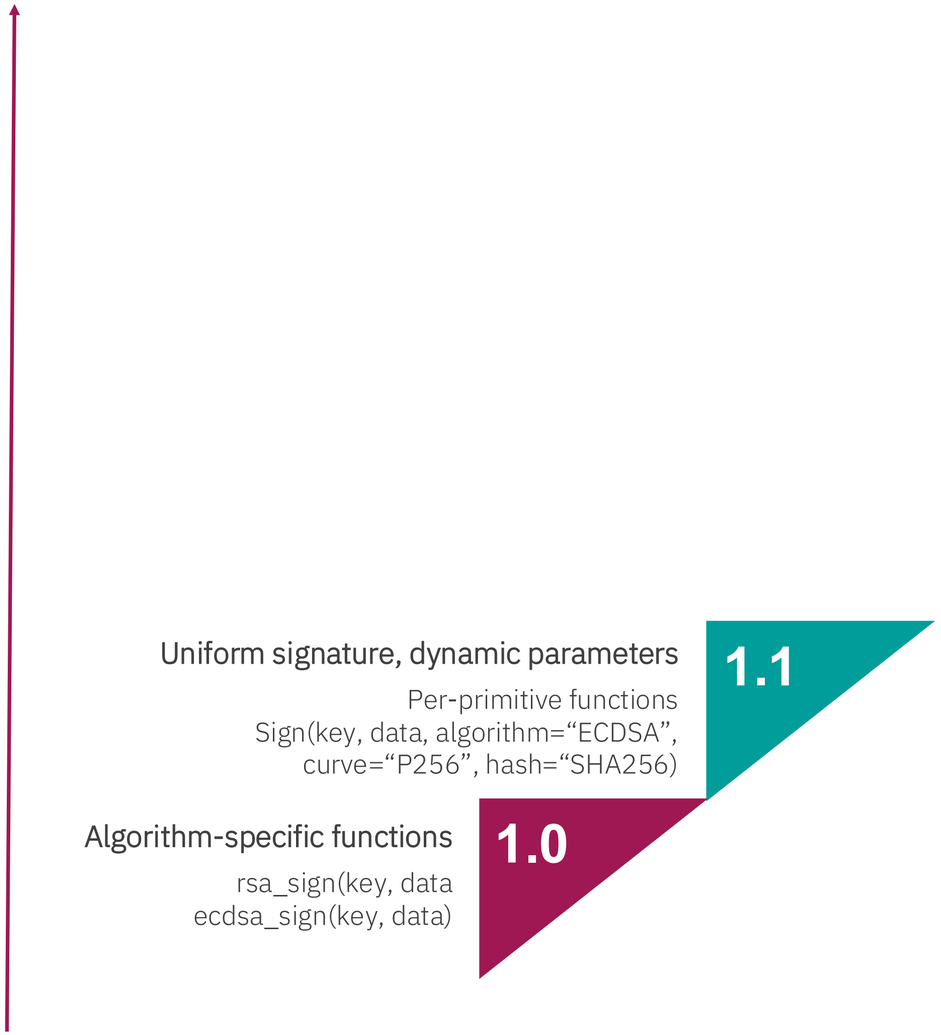
C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



Classification of parameters

Algorithm identity: indicates which configuration of an algorithm to use

System-generatable: IVs, nonces

Operational: additional inputs that only a user can supply (eg. Domain context for signatures, aad)

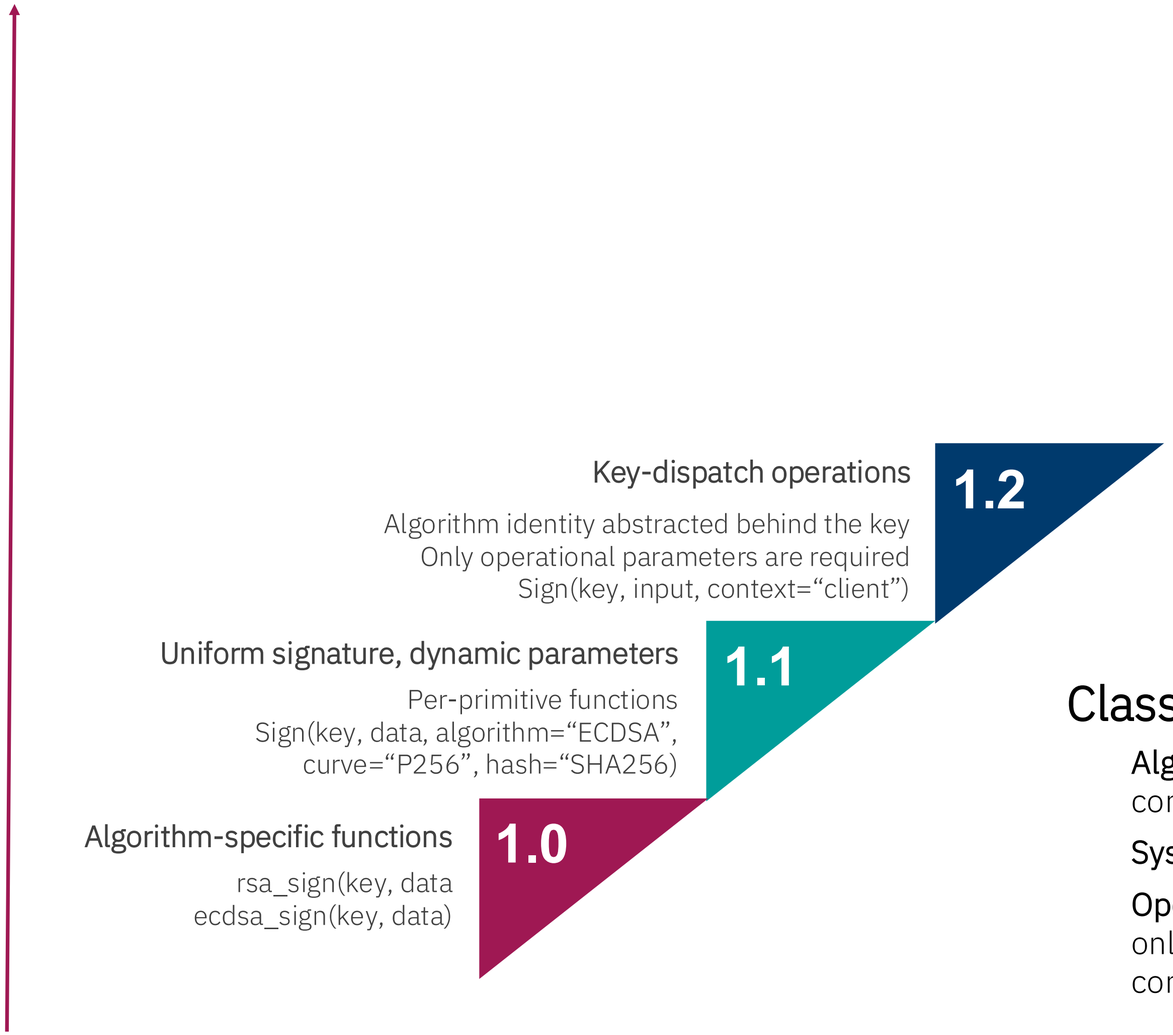
C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



Classification of parameters

Algorithm identity: indicates which configuration of an algorithm to use

System-generatable: IVs, nonces

Operational: additional inputs that only a user can supply (eg. Domain context for signatures, aad)

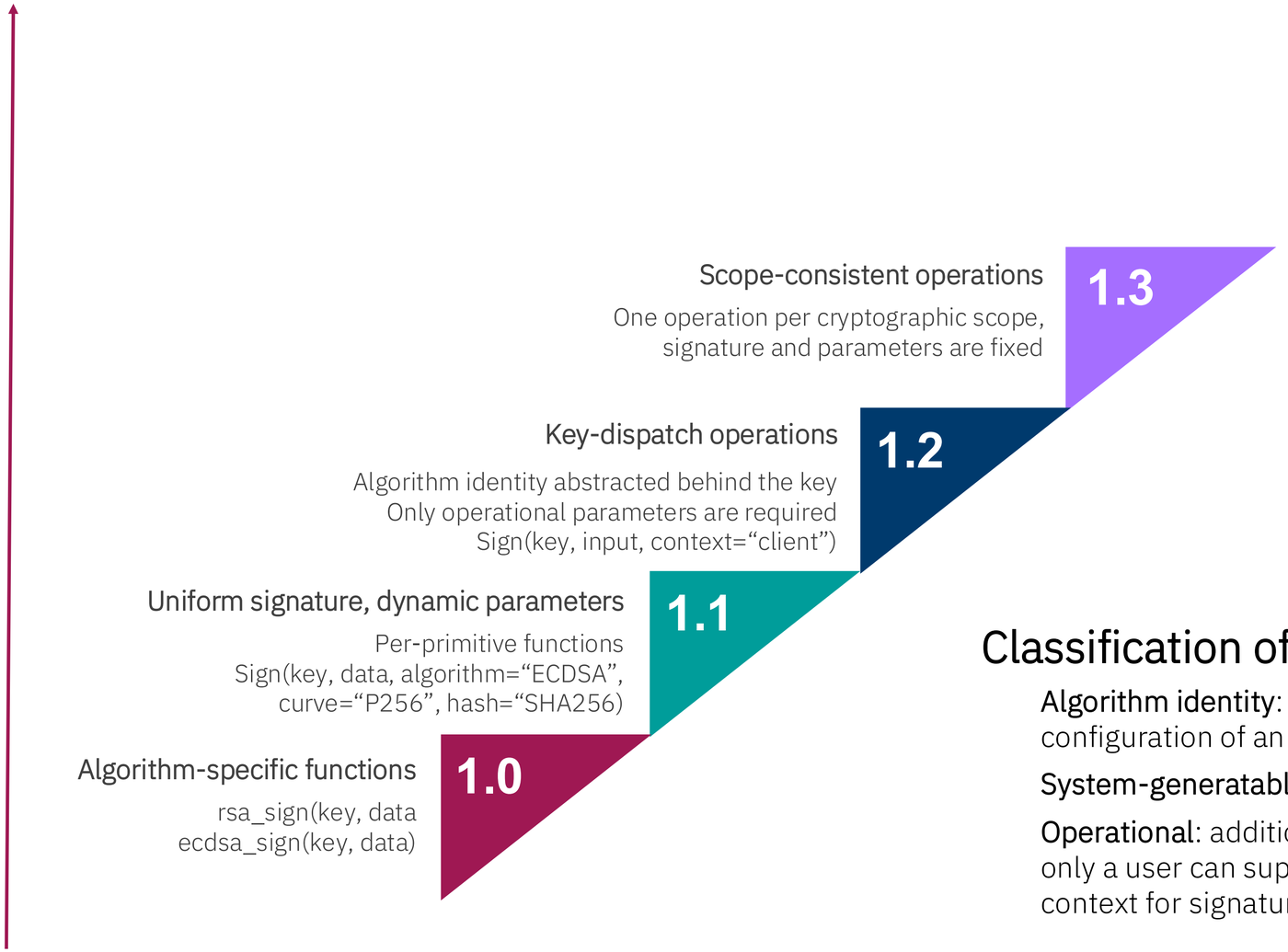
C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



Classification of parameters

Algorithm identity: indicates which configuration of an algorithm to use

System-generatable: IVs, nonces

Operational: additional inputs that only a user can supply (eg. Domain context for signatures, aad)

C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed

Algorithm substitutability

Algorithms with same scope can be exchanged with one another without any change to the application code

1.3 Scope-consistent operations

One operation per cryptographic scope, signature and parameters are fixed

1.2 Key-dispatch operations

Algorithm identity abstracted behind the key
Only operational parameters are required
Sign(key, input, context="client")

1.1 Uniform signature, dynamic parameters

Per-primitive functions
Sign(key, data, algorithm="ECDSA", curve="P256", hash="SHA256)

1.0 Algorithm-specific functions

rsa_sign(key, data
ecdsa_sign(key, data)

Classification of parameters

Algorithm identity: indicates which configuration of an algorithm to use

System-generatable: IVs, nonces

Operational: additional inputs that only a user can supply (eg. Domain context for signatures, aad)

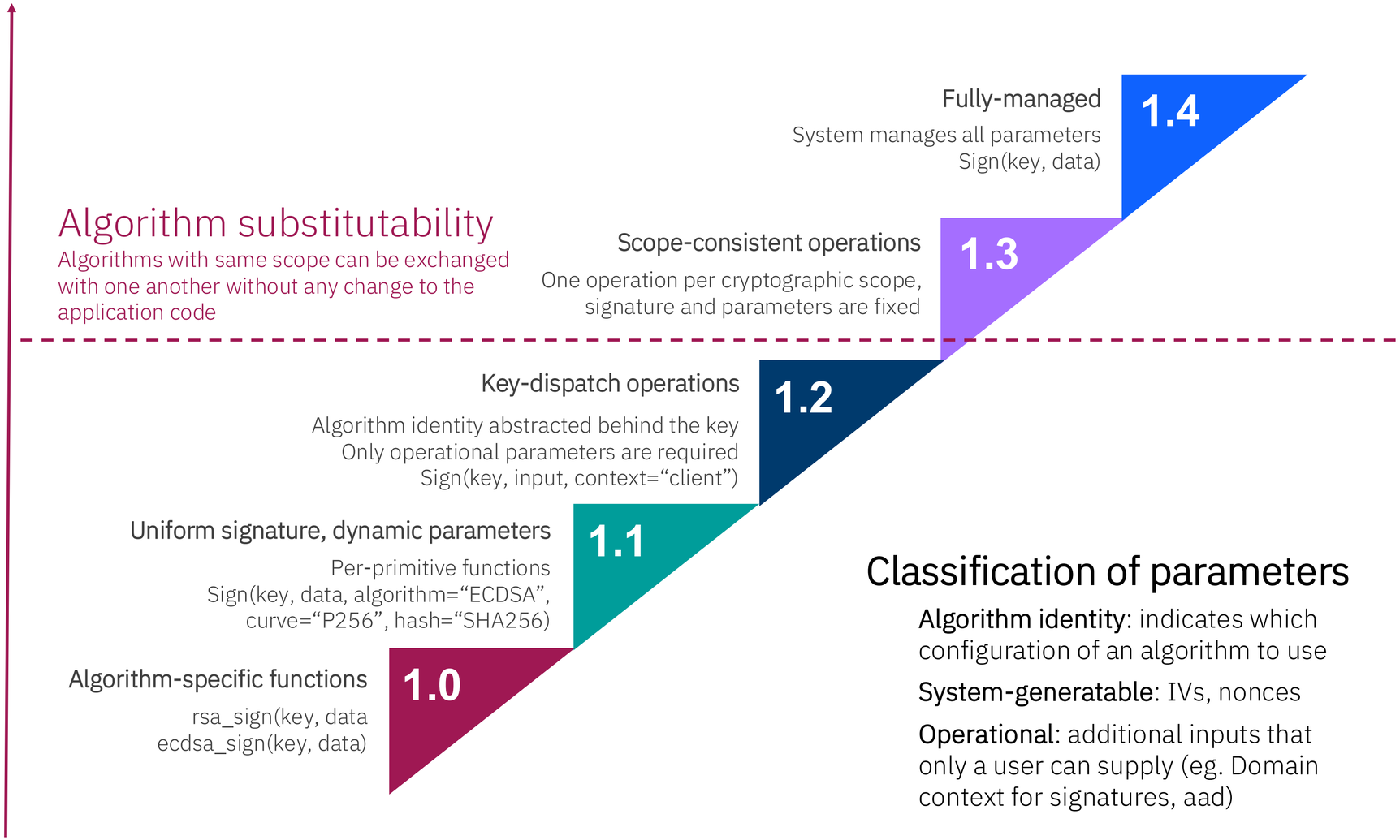
C1: Operations coupling

How much do applications know about cryptography when operations are computed?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



Classification of parameters

Algorithm identity: indicates which configuration of an algorithm to use

System-generatable: IVs, nonces

Operational: additional inputs that only a user can supply (eg. Domain context for signatures, aad)

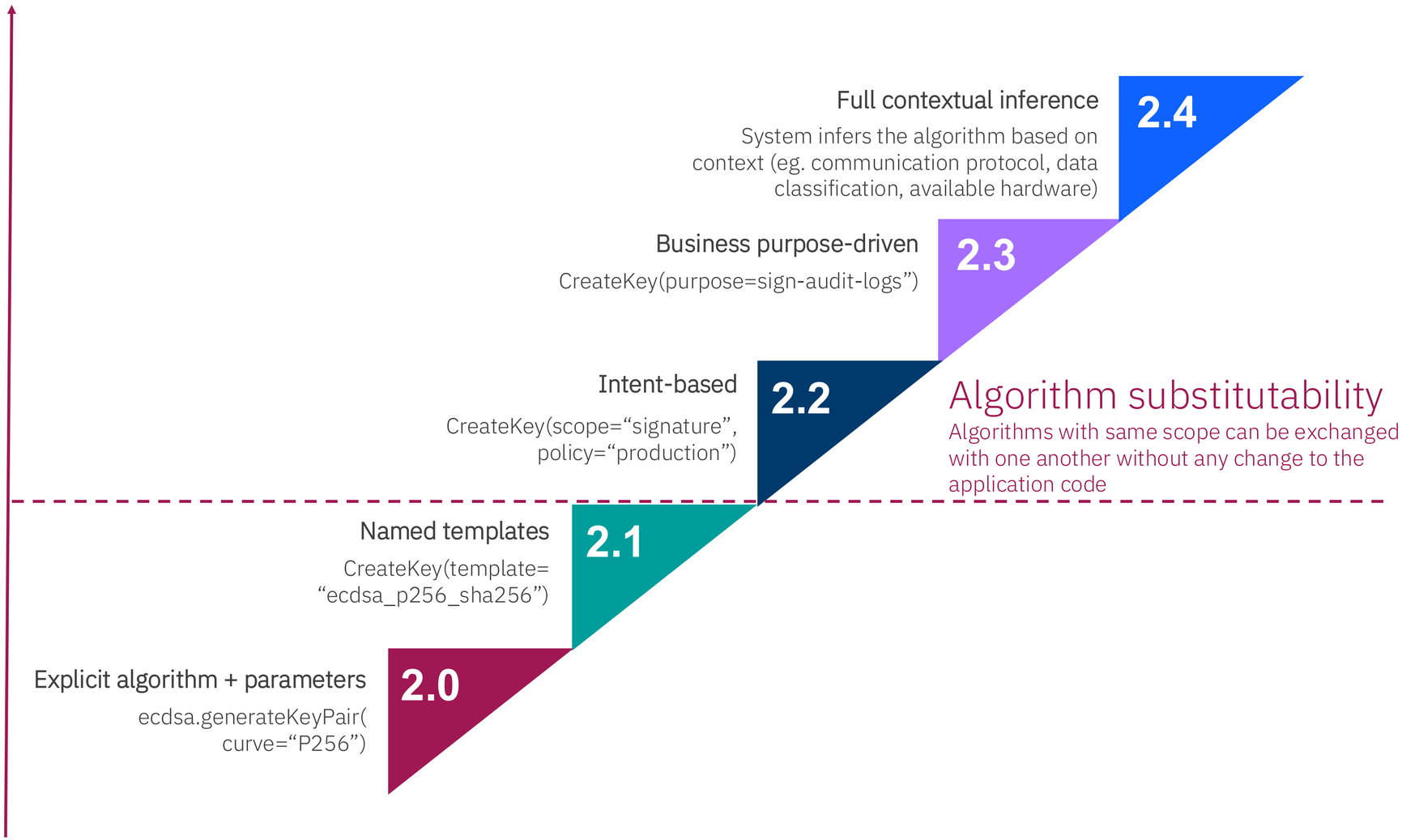
C2: Creation coupling

How much do applications know about algorithms when the key material is created?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



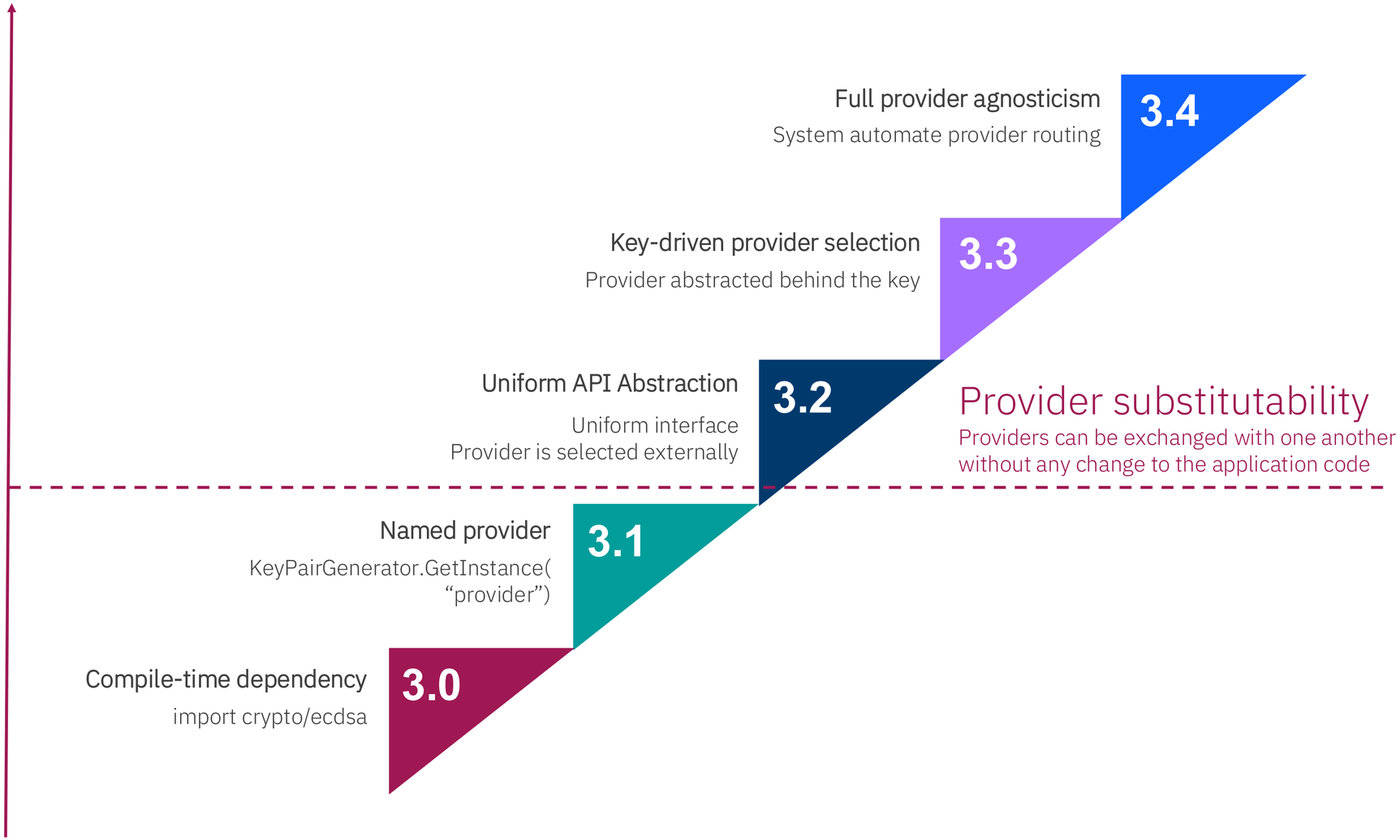
C3: Provider coupling

How much do applications know about the providers when operations are computed and/or key material is created?

C1: Operations Coupling
Operations: sign, encrypt, ...
when cryptography is consumed

C2: Creation Coupling
when key material is created

C3: Provider Coupling
where cryptography is computed



Decoupling mechanism and authority

C4: Decoupling mechanism

How is the coupling value computed?

Applies to

C5: Decoupling authority

Who controls the decoupling?

Contextual inference

Decision delegated to system, based on operational context

Policy engine

Rule-based mapping that enable expressive constraints eg. “require FIPS-approved algorithms with 256 bits security”

Configuration

Decision defined by an external configuration

Hardcoded

Decision is in source code

Cryptographic governance

Federated governance

Hierarchical cryptographic governance

Governance domains

Different organizational contexts (eg. Production vs development) have independent policies, managed by different authorities

Role-based access control

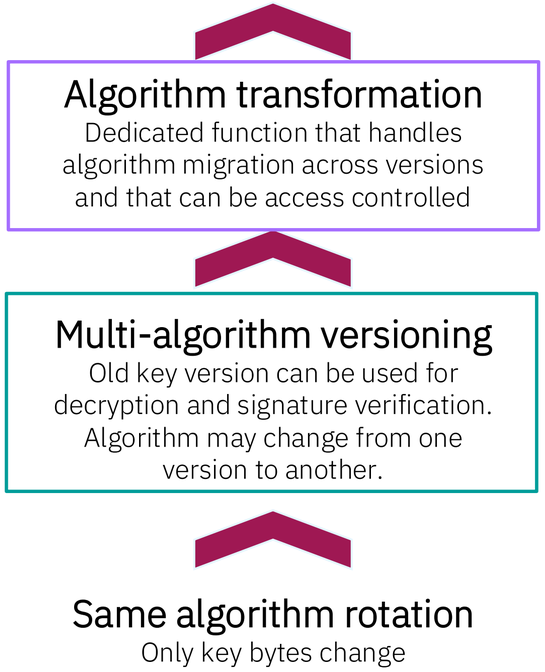
Permission for operations
Who may create keys, who may use which keys

Single entity

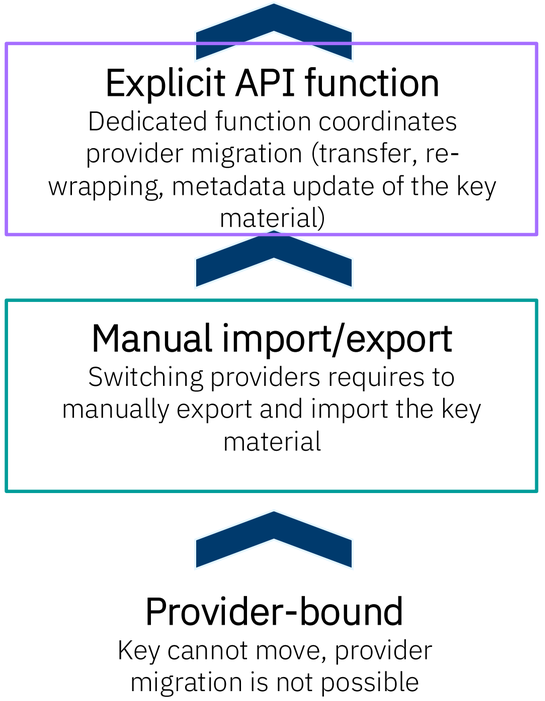
The entity producing the application selects the algorithms

Enablers: Exploiting the decoupling potential

E1: Algorithm migration
without changes to the application code



E2: Provider migration
without changes to the application code



Migration is actionable

Migration is possible but manual

Where do cryptographic libraries, frameworks and APIs stand?

System	C1 Ops	C2 Create	C3 Provider	C5 Gov	E1 Algo Migr	E2 Prov Migr
PKCS#11	1 Uniform signature with algorithm-specific params	0 Algorithm-specific params	2 Uniform API	0	0	1 Import / Export
OpenSSL 3.0	1	0	1 Named provider	0	0	0
JCA	1	0	1	0	0	0
Tink	4 Uniform per-primitive	1 Named templates	0 Hardcoded	0	2 Multi-algorithm versioning	0
AWS KMS	1	1	2	2 Access control	1 Rotation	1
Vault Transit	2 Key-dispatch	1	3 Key-driven	2	1	1

Where do cryptographic libraries, frameworks and APIs stand?

System	C1 Ops	C2 Create	C3 Provider	C5 Gov	E1 Algo Migr	E2 Prov Migr
PKCS#11	1 Uniform signature with algorithm-specific params	0 Algorithm-specific params	2 Uniform API	0	0	1 Import / Export
OpenSSL 3.0	1	0	1 Named provider	0	0	0
JCA	1	0	1	0	0	0
Tink	4 Uniform per-primitive	1 Named templates	0 Hardcoded	0	2 Multi-algorithm versioning	0
AWS KMS	1	1	2	2 Access control	1 Rotation	1
Vault Transit	2 Key-dispatch	1	3 Key-driven	2	1	1

Finding #1: Partial operation decoupling exists but key creation decoupling is absent

Where do cryptographic libraries, frameworks and APIs stand?

System	C1 Ops	C2 Create	C3 Provider	C5 Gov	E1 Algo Migr	E2 Prov Migr
PKCS#11	1 Uniform signature with algorithm-specific params	0 Algorithm-specific params	2 Uniform API	0	0	1 Import / Export
OpenSSL 3.0	1	0	1 Named provider	0	0	0
JCA	1	0	1	0	0	0
Tink	4 Uniform per-primitive	1 Named templates	0 Hardcoded	0	2 Multi-algorithm versioning	0
AWS KMS	1	1	2	2 Access control	1 Rotation	1
Vault Transit	2 Key-dispatch	1	3 Key-driven	2	1	1

Finding #2: Access control exists but cryptographic governance does not

Where do cryptographic libraries, frameworks and APIs stand?

System	C1 Ops	C2 Create	C3 Provider	C5 Gov	E1 Algo Migr	E2 Prov Migr
PKCS#11	1 Uniform signature with algorithm-specific params	0 Algorithm-specific params	2 Uniform API	0	0	1 Import / Export
OpenSSL 3.0	1	0	1 Named provider	0	0	0
JCA	1	0	1	0	0	0
Tink	4 Uniform per-primitive	1 Named templates	0 Hardcoded	0	2 Multi-algorithm versioning	0
AWS KMS	1	1	2	2 Access control	1 Rotation	1
Vault Transit	2 Key-dispatch	1	3 Key-driven	2	1	1

Finding #3: Enablers are missing

Designing an agile cryptographic API

1

Abstraction and Stability

Key abstracts algorithm identity and implementation
Scoped operation with consistent interface: Sign ≠ SignWithContext

2

Cryptographic governance

Policy engine governs which algorithms satisfy which intent
Separation of concerns: governing cryptography ≠ using cryptography

3

Evolution and extensibility

Dedicated operations to evolve keys: **rotation**, **transformation**, provider **migration**

↪ Principles and concrete patterns are described in the paper

Takeaways

Takeaway #1

Cryptographic migration requires code migration

Takeaway #2

Assessment framework characterizes agility of cryptographic APIs along independent dimensions

Takeaway #3

Design principles and concrete Protocol Buffers patterns pave the way towards agile APIs

- Agility requires decoupling, externalized governance, and enablers
- Patterns define a common ground for agile cryptographic systems

Takeaways

Takeaway #1

Cryptographic migration requires code migration

Takeaway #2

Assessment framework characterizes agility of cryptographic APIs along independent dimensions

Takeaway #3

Design principles and concrete Protocol Buffers patterns pave the way towards agile APIs

- Agility requires decoupling, externalized governance, and enablers
- Patterns define a common ground for agile cryptographic systems

Cryptographic agility for applications: an assessment framework and principles API design
Navaneeth Rameshan, Gregoire Messmer

Thank you, questions?