2025 KpqC 연구단 워크숍

암호민첩성 시스템 구축의 요구사항과 설계 방향

2025.07.16

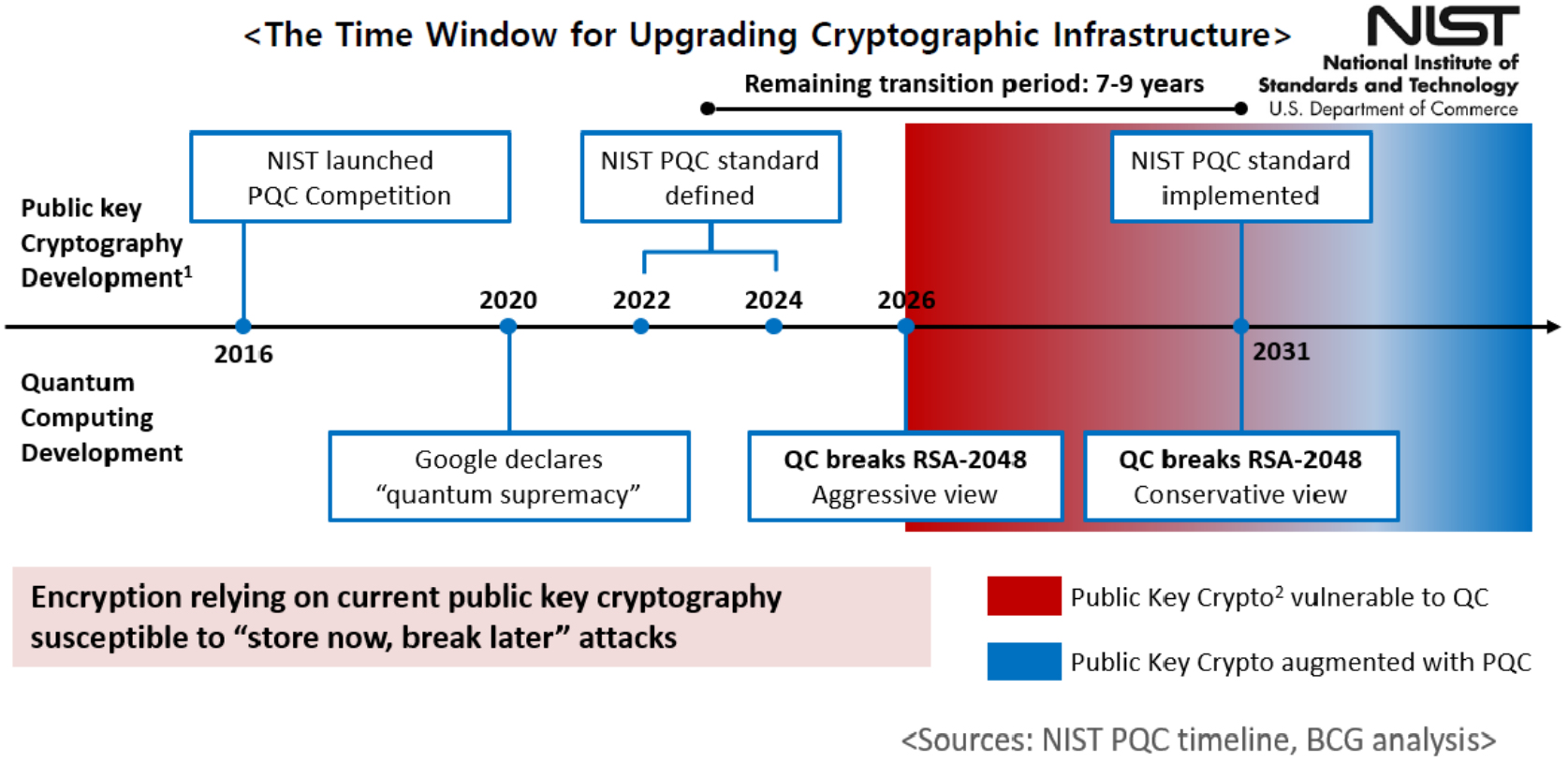
성신여자대학교 김성민

Contents

PQC 전환 동향

[NIST PQC 알고리즘 표준화 현황]

- 양자 컴퓨팅 환경에서 안전하게 암호기술을 이용할 수 있도록 하는 새로운 공개키 암호
- 양자 컴퓨터로도 해결 불가능한 다양한 수학적 난제에 안전성을 기반함



<Sources: NIST PQC timeline, BCG analysis>

<2022>

- 최종 표준 알고리즘 4종 발표

✓ Public-Key Encryption / Key Encapsulation Mechanism 1종 (CRYSTALS-KYBER)

✓ Digital Signature 3종 (CRYSTALS-Dilithium, FALCON, SPHINCS+)

<2025>

- 추가 KEM 알고리즘 1종 선정

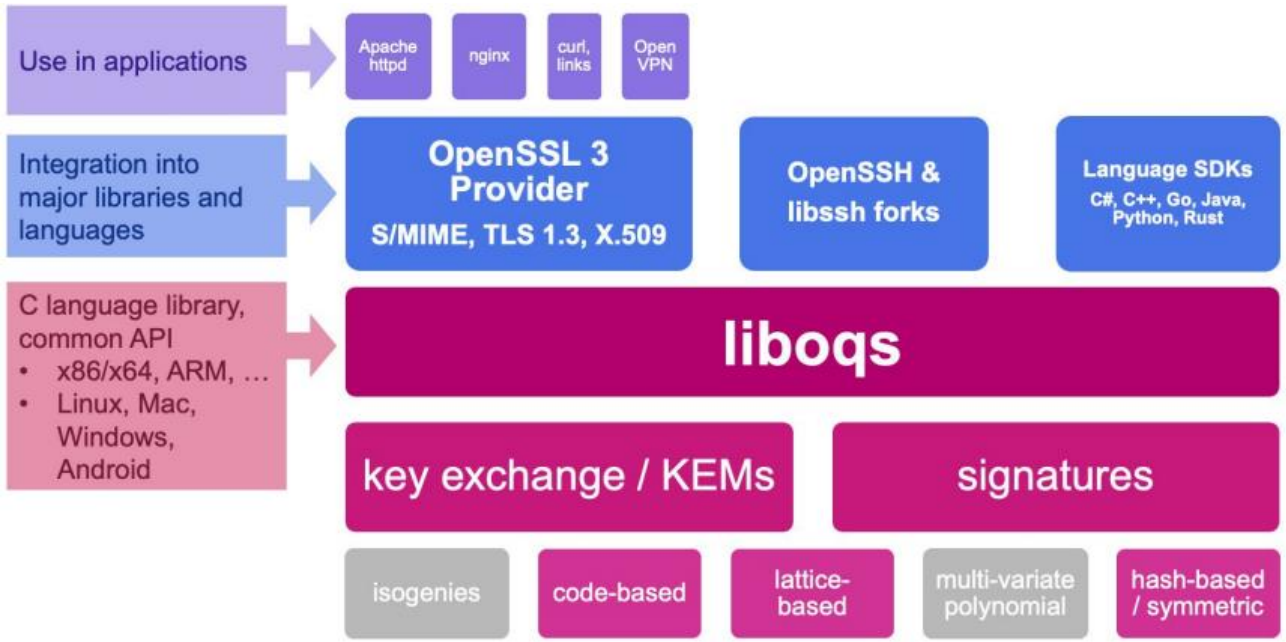
✓ Public-Key Encryption / Key Encapsulation Mechanism 1종 (HQC)

PQC 전환 동향

[PQC를 지원하는 암호 라이브러리]

- quantum-safe cryptographic algorithm을 support하는 liboqs project
- OpenSSL1.1.1 및 BoringSSL의 fork에 integration

  → TLS protocol 내에서의 prototype post-quantum key exchange, authentication, 및 ciphersuites 지원
- 어플리케이션 구현을 위한 다양한 programming language wrapper API들을 지원: C++, Python, Go, Rust



* OPEN QUANTUM SAFE, OQS, 2023. 11

PQC 전환 동향

[PQC 기반 암호화 프로토콜 현황]

- PQC 기반 TLS: 최신 TLS 1.3 수행 시 KEM 및 DSA를 PQC 기반으로 대체하기 위한 연구가 수행 중. benchmarking 및 최적화를 위한 연구가 주요 트렌드, 암호 민첩성을 위한 전환 관점에서의 연구는 초동 단계
  - E.g., KEM: ECDH/DH (legacy) → Kyber-crystal (PQC)
  - E.g., DSA: ECDSA/RSA (legacy) → Kyber-Dilithium/Falcon (PQC)

Table 2: Time, Power and Energy consumption of Traditional and PQ TLS 1.3 Handshake.

	Time (ms)			*Power (mW)*			*Energy (mJ)*		
Notation	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***	***Client (mut)***	***Server (mut)***	***Client (uni)***
Dil1+Kyb1	96.318	91.062	69.639	155.800	161.300	176.300	15.006	14.688	12.277
Falc1+Kyb1	288.305	285.951	44.548	139.500	136.800	165.500	40.219	39.118	7.373
Dil3+Kyb3	157.126	153.492	98.481	161.600	164.200	178.400	25.392	25.203	17.569
Falc5+Kyb3	594.495	589.058	66.254	137.100	133.900	168.900	81.505	78.875	11.190
Dil3+Kyb5	165.590	152.537	105.865	161.200	162.900	180.100	26.693	24.848	19.066
Falc5+Kyb5	601.827	592.302	73.967	138.000	133.700	172.500	83.052	79.191	12.759
Sph1s+Kyb1	66 977.000	66 776.000	911.000	175.700	175.500	163.400	11 767.859	11 719.188	148.857
Dil2+Bike1	878.818	143.28	768.000	154.200	160.900	175.200	135.514	23.054	134.554
Dil2+Hqc1	198.603	145.989	183.622	156.000	158.800	174.800	30.982	23.183	32.097
Dil2+Frodo1AES	–	–	936.294	–	–	180.400	–	–	168.907
Dil2+Frodo1SHAKE	–	–	1252.250	–	–	199.700	–	–	250.074
RSA+ECDHE	540.220	538.158	77.237	144.500	150.400	168.200	78.062	80.939	12.991
ECDSA+ECDHE	109.171	106.927	102.109	167.100	175.500	181.500	18.242	18.766	18.533

*Tasopoulos, George, et al. "Energy consumption evaluation of post-quantum TLS 1.3 for resource-constrained embedded devices." Proceedings of the 20th ACM International Conference on Computing Frontiers. 2023.

PQC 전환 동향

[PQC 기반 암호화 프로토콜 현황]

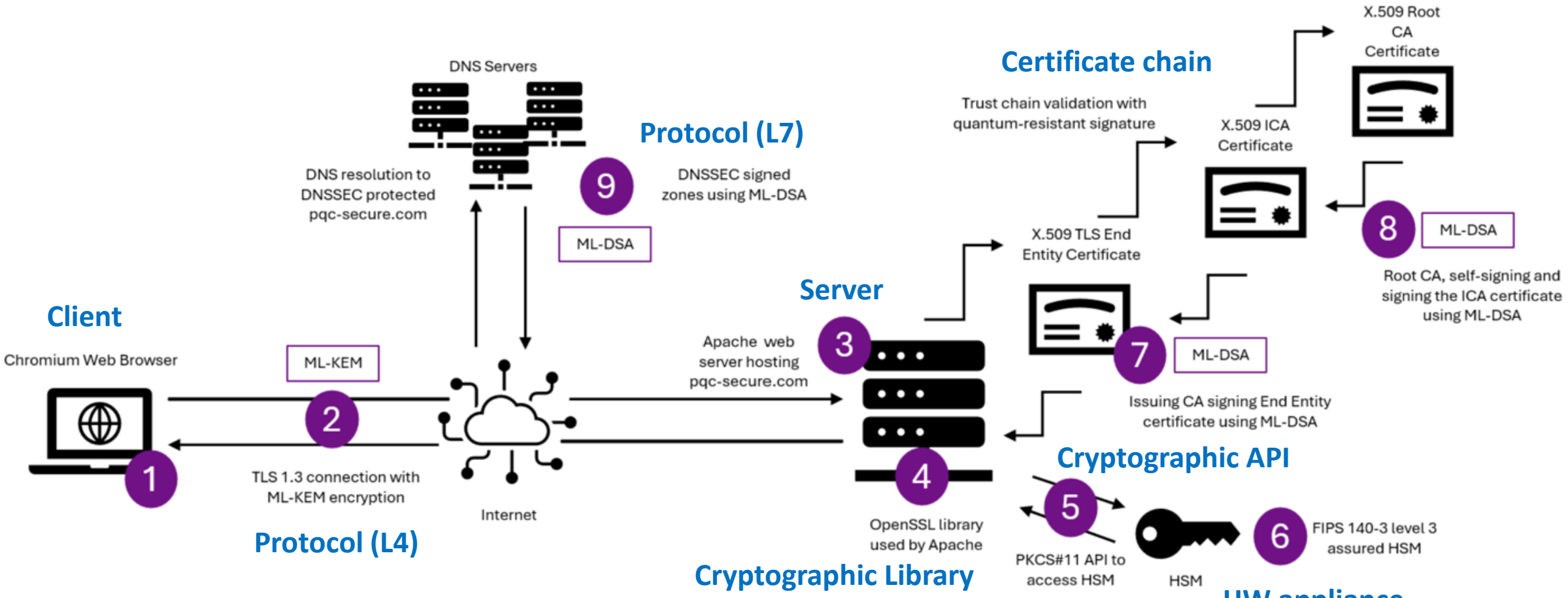
- TLS 1.3 외에도 WireGuard, IPSec, DNSSEC에 PQC를 적용하거나 PQC+레거시 전자서명을 결합하여 사용하는 이중서명 방식 등이 제안되고 있음

프로토콜	사용 계층	암호 스킴 구성	PQC 사용	대표 알고리즘
IPSec	OSI 3 layer	키 교환 프로토콜(IKE) + 대칭키 암호화(ESP)	Kyber, NTRU, Saber [1]	DH/ECDH, RSA AES, ChaCha20
DNSSEC	OSI 7 layer	전자서명 (DSA)	FALCON, DILITHIUM2-AES, SPHINCS+ [2]	ECDSA, RSA
WireGuard	Tunneling, OSI 3~4 layer	키 교환 프로토콜(IKE) + 대칭키 암호화(ESP)	Classic McEliece, Saber [3]	Curve25519(X25519) ChaCha20-Poly1305

[1] BAE, Seungyeon, et al. A performance evaluation of ipsec with post-quantum cryptography. In: International Conference on Information Security and Cryptology. Cham: Springer Nature Switzerland, 2022. p. 249-266.
[2] PAN, Syed W. Shah, et al. Double-Signed Fragmented DNSSEC for Countering Quantum Threat. *arXiv preprint arXiv:2411.07535*, 2024.
[3] HÜLSING, Andreas, et al. Post-quantum wireguard. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021. p. 304-321.

PQC 전환 동향

[PQC 기반 TLS connection: end-to-end workflow]



*Appviewx, https://www.appviewx.com/blogs/practical-advice-for-pqc-migration-for-tls-1-3/

암호 민첩성 정의

[BSI Definition]

- A cryptosystem is considered crypto-agile if its components, for example cryptographic algorithms, key lengths, key generation schemes or technical implementation, can be replaced by other components without having to make significant changes to the rest of the overall system

✓ Real-time/Deployment 관점에서의 고려 사항이 포함되지 않은 정의

✓ Algorithm/Protocol-centric agility에 가까운 개념

[NIST Definition]

- The ability 1) for machines to select their security algorithms in real time and based on their combined security functions; 2) to add new cryptographic features or algorithms to existing hardware or software, resulting in new, stronger security features; 3) to easily retire cryptographic systems that have become either vulnerable or obsolete.

✓ Runtime adoption/Deployment 중심의 정의

✓ Platform Operator/Developer-centric agility에 가까움

암호 민첩성 정의

	BSI	NIST
관점	기술/설계 중심	운영/배포 중심
대상	프로토콜, 시스템, 제품 전체	제품(Product)에 집중
강조점	구현 단계에서의 알고리즘 교체 용이성	배포 후 알고리즘 업데이트 가능성
목표	유연하고 모듈화된 아키텍처	제품 교체 없이 유지관리 가능

[Protocol(Algorithm)-centric agility]

- Cryptographic primitive 및 이를 기반으로 한 알고리즘이 암호민첩성-friendly하게 설계
- 교체 용이성에 방점 (보안 문제 시 대체)
- Functional consistency vs. Interoperability

[Developer(Operator)-centric agility]

- Candidate cryptographic primitive들 중 시스템이 유연하게 선택 가능하도록 설계
- 상호운용성(interoperability)에 방점
- Functional consistency vs. Interoperability

현재 암호 민첩성 개념의 정의는 암호 민첩성 시스템을 구현함에 있어 notion 정의의 모호성 및 design space가 존재

암호 민첩성 시스템: Motivation

[PQC를 지원하는 암호민첩성 시스템]

- 암호 민첩성은 실행 중인 시스템의 흐름을 방해하지 않으면서도 프로토콜, 애플리케이션, 하드웨어 및 인프라 등에서 암호 알고리즘을 교체하고 적응할 수 있는 능력을 통해 복원력을 확보하기 위한 기술

- HSM(Hardware Security Module) 및 KMS(Key Management Service) 산업을 이끄는 기업들 중심으로 PQC를 지원하는 암호 민첩성 시스템 개발에 착수 중

- 그러나 대부분 실질적으로 PQC와 레거시 암호 스킴을 동시에 지원한다는 수준이며, 진정한 암호민첩성을 보장한다고 규정하기 어려운 것이 현 실태

- 양자내성암호 알고리즘의 암호 민첩성의 설계 요구사항을 도출하여, 다양한 양자내성암호 알고리즘이 유연하게 교체되도록 호환될 수 있는 구조와 프로토콜 설계가 선행되어야 함

암호 민첩성 시스템: Design goal

[암호민첩성 주요 속성]

<Primary Goal>

- Security, Policy-awareness: 보안성을 보장해야 함 (e.g., 특정 암호 primitive가 compromised되도 문제 없음)
- Heterogeneity, interoperability: 이종의 암호 primitive들을 지원하고, 상호운용 가능해야 함

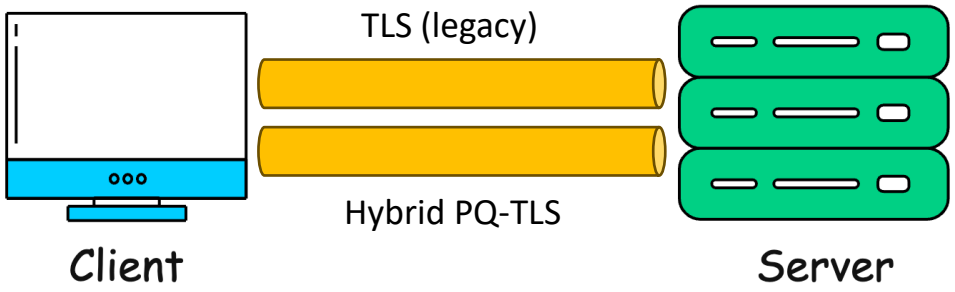
But also need to provide

- Backward-compatibility: 레거시 프로토콜/알고리즘 기반의 라이브러리/어플리케이션에 수정 없이 적용 가능해야 함
- Effectiveness, performance: 성능 효율성을 보장해야 함
- Extendibility, flexibility, upgradeability: 암호 프로토콜/알고리즘을 추가하거나 제거, 확장 가능해야 함

암호 민첩성 시스템: Design goal

[암호민첩성을 위한 주요 속성: Security, Policy-awareness]

- Option1: Security problem 발생 시 crypto-primitive를 전면 교체하도록 설계 (BSI definition)
  - 사후 scheme 교체 및 키 생성, 생성된 키로 ciphertext 변환
  - Key bootstrapping 및 handshake 과정을 미리하지 않음으로 인한 consistency 측면에서의 지연 시간
- Option2: Security problem 발생 시 alternative들 중 automatic하게 선택하도록 정책 기반으로 설계 (NIST definition)
  - Handshake 시 양종단에서 사용 가능한 primitive에 대해 미리 setup을 operational consistency 달성
  - 지속적인 storage space 사용 및 metadata 관리 등으로 인한 runtime overhead



Revoke TLS channel → Hybrid PQ-TLS handshake & Session establishment

or

TLS/Hybrid PQ-TLS handshake & maintain sessions → Revoke TLS channel

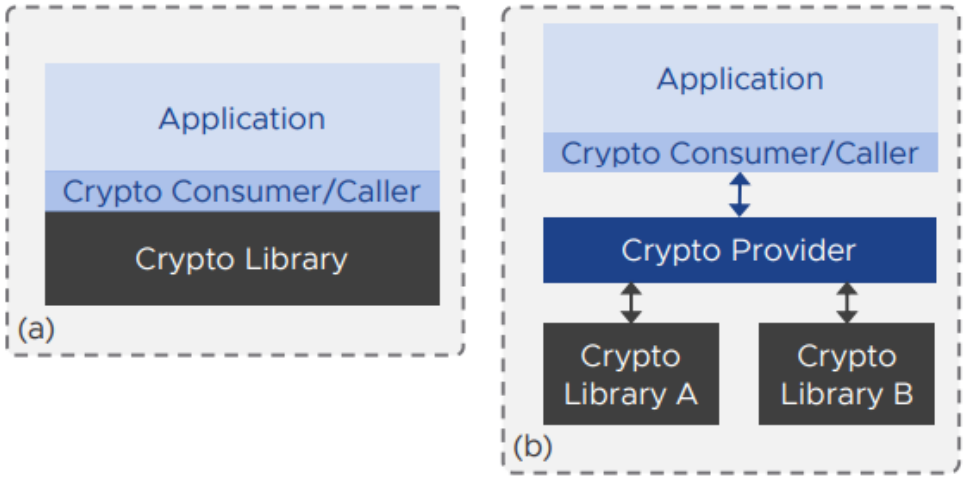
```
premaster_secret = ECDHE_KEY

seed = "master secret"
          || ClientHello.random
          || ServerHello.random


master_secret = HMAC(premaster_secret, seed)


premaster_secret = ECDHE_KEY || PQ_KEY

seed = "hybrid master secret"
          || ClientHello.random
          || ServerHello.random
          || ClientKeyExchange

master_secret = HMAC(premaster_secret, seed)
```

암호 민첩성 시스템: Design goal

[암호민첩성을 위한 주요 속성]

Heterogeneity, interoperability: 이종의 crypto primitive들을 지원하고, 상호운용 가능해야함

Backward-compatibility: 레거시 프로토콜/알고리즘 기반의 라이브러리/어플리케이션에 수정 없이 적용 가능해야 함



- System Requirement 1: 어플리케이션과 특정 암호 라이브러리 간의 종속성 제거
- System Requirement 2: Crypto-agnostic한 어플리케이션과 암호 로직 간 generic한 인터페이스 (API)
- System Requirement 3: 알고리즘/프로토콜 수준에서의 교체 또는 암호 라이브러리 수준의 교체를 위한 모듈화

암호민첩성 시스템 설계 PoC 구현

[암호민첩성을 위한 주요 속성: Effectiveness, performance]

Challenge 1: PQC 키 및 서명, 인증서 사이즈

- PQC 키 사이즈의 경우, conventional cryptographic algorithm 대비 큼 **(25x ~ 49x)**
- Ciphertext 사이즈의 크기 또한 비슷하게 증가함
- Signature의 경우도 **10x ~71x** 크기가 증가함

Challenge 2: 프로토콜 메세지 사이즈의 증가

- PQC 기반 암호화 프로토콜에서의 메세지 사이즈 또한 dramatic하게 증가 **(17x ~ 67x)**

Algorithm	Size	Algorithm	Size
ECDH+ECDSA(P-256)	224	DH+RSA (2048)	1,280
KYBER512+DILITHIUM2	7,720	KYBER512+FALCON512	3,797
KYBER768+DILITHIUM3	10,810	KYBER1024+FALCON1024	7,489
KYBER1024+DILITHIUM5	14,918		

KEM	NIST Level	Size (byte)		
		public key	*private key*	*ciphertext*
ECDH-P256	0	32	32	32
DH-2048	0	256	256	256
KYBER512	1	800	1,632	768
KYBER768	3	1,184	2,400	1,088
KYBER1024	5	1,568	3,168	1,568

DSA	NIST Level	Size (byte)		
		public key	*private key*	*signature*
ECDSA-P256	0	32	32	64
RSA-2048	0	259	256	256
DILITHIUM2	2	1,312	2,528	2,420
DILITHIUM3	3	1,952	4,000	3,293
DILITHIUM5	5	2,592	4,864	4,595
FALCON512	1	897	1,281	666
FALCON1024	5	1,793	2,305	1,280

*KIM, Jane, et al. ExpressPQDelivery: Toward Efficient and Immediately Deployable Post-Quantum Key Delivery for Web-of-Things. In: *Proceedings of the ACM on Web Conference 2025*. 2025. p. 2969-2980.

암호민첩성 시스템 설계 PoC 구현

[암호민첩성을 위한 주요 속성: Effectiveness, performance]

Challenge 3: 서명 및 인증서 사이즈 증가로 인해 프로토콜 Window 사이즈보다 커져 fragmentation이 일어남

- Handshake time에 영향을 줄 수 있음 (e.g., PQC-TLS handshake)
- 이중서명과 같은 hybrid approach 사용 과정에서 maximum message 사이즈를 초과 (e.g., DNSSEC)

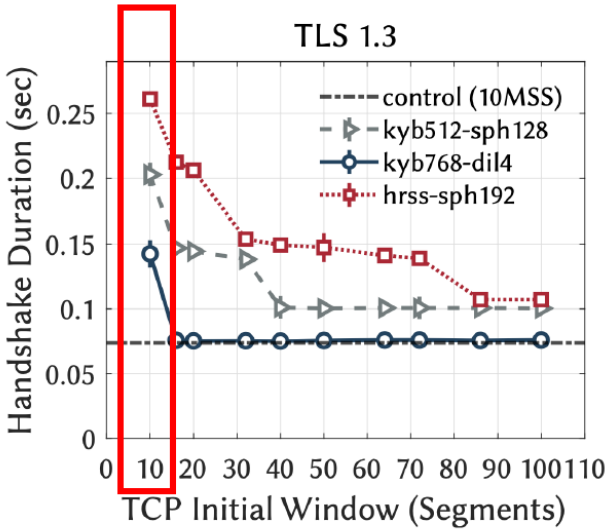


Table 1: Total Number of Fragments with Various Combinations of Double-Signatures

Digital Signature	Fragments in Respons to QTYPE A	Fragments in Respons to QTYPE DNSKEY
FALCON	2	3
FALCON+ECDSA	3	4
FALCON+RSA	3	4
DILITHIUM	7	7
DILITHIUM+ECDSA	8	8
DILITHIUM+RSA	8	8
SPHINCS	23	15
DILITHIUM+ECDSA	23	15
DILITHIUM+RSA	23	15

Systematic Indicators for crypto agility

[암호민첩성을 위한 주요 속성: Effectiveness, performance]

- 암호 민첩성 시스템의 효율적인 성능 보장 → 레거시 환경에서의 새로운 crypto constraint들을 통제하는 것이 필요
- Synchronization between constraints is important
    - Legacy system constraints (e.g., fixed key sizes, formats)
    - New crypto constraints (e.g., larger PQC keys)

      → leads to overly narrow key slots blocking PQC adoption

256-bit Key Slot
[■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■]
32 Bytes (256 bits)

Hold RSA/ECC keys, but
Cannot hold ML-KEM(512 bits for seed)
or SLH-DSA (768 bits)

- security vs. performance trade-off가 필연적으로 발생: 운영자가 최소한의 기준을 지키는 선에서 security-level을 탄력적으로 설정할 수 있도록 정책 기반의 관리가 필요

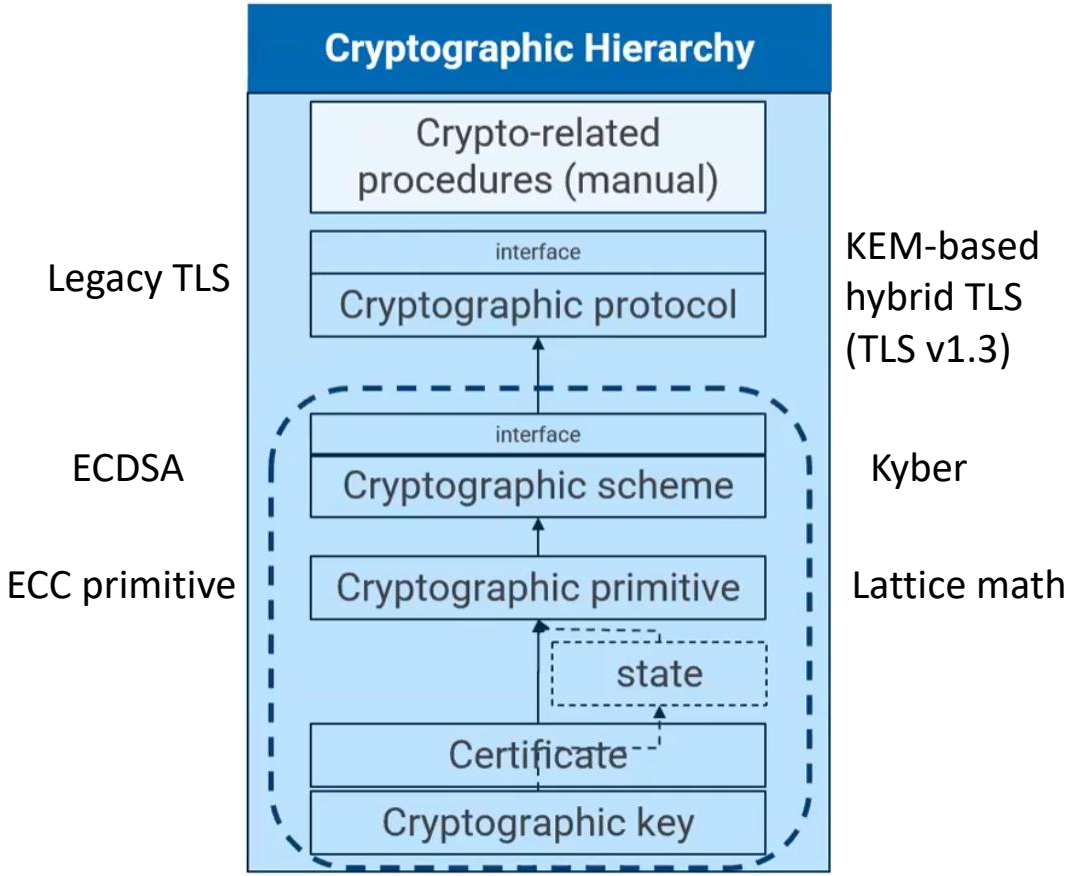
  (e.g., TLS ciphersuite, x509 certificate에서 레거시 KEM/DSA과 PQC 기반을 hybrid하게 사용)
- Recommendation and demand of enterprises: NIST should offer clear **guidance on safe configurations**

구현 전략 및 요구사항: Overview

[Cryptographic Hierarchy]

- **From HW-based root key to applications**
  - **Cryptographic primitive**: most basic mathematical operations, foundation layer (e.g., hash function, symmetric/asymmetric cipher)
  - **Cryptographic schemes**: how to use primitives to perform a cryptographic task (e.g., signature, encryption, KEM)
  - **Cryptographic protocols**: multi-step processes between parties to achieve secure communication, authentication, or digital identity

→ These layers would be candidates for implementing and supporting crypto-agility in a modular manner



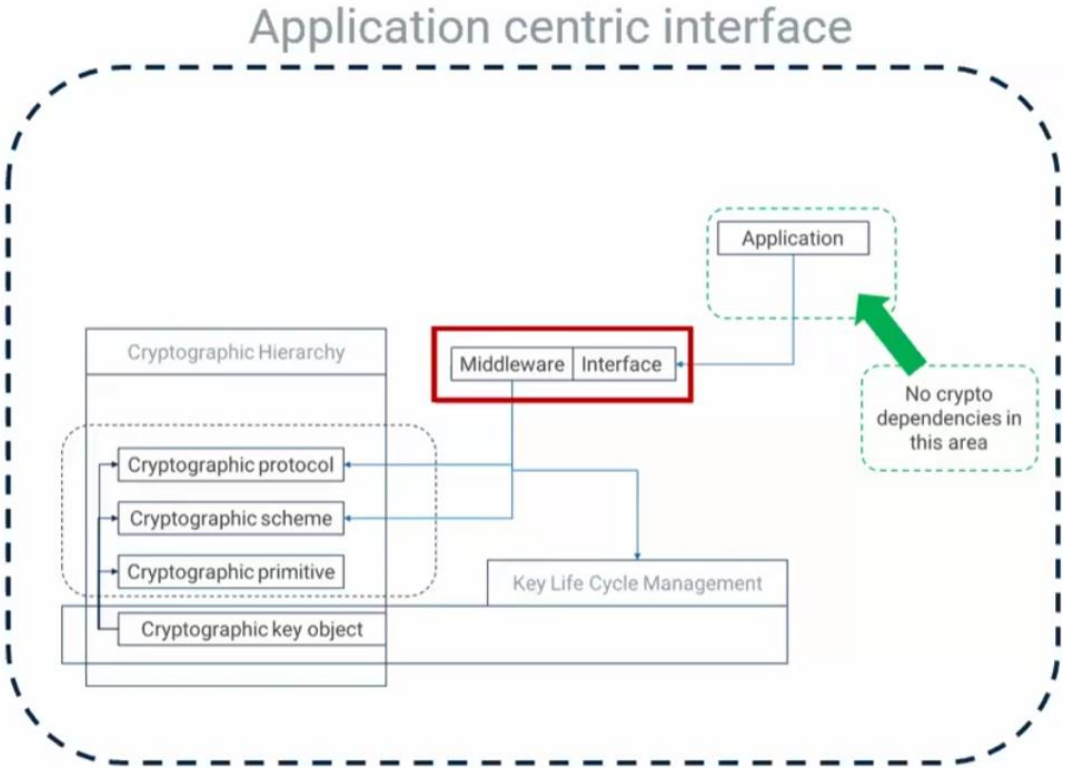
*https://csrc.nist.gov/events/2025/crypto-agility-workshop

구현 전략 및 요구사항: Common

[Application-centric interface]

- **Recommendation1: Separate app logic from the crypto layer**
  - HSM은 application-centric interface를 외부로 노출해야 함
  - Low-level implementation detail이나 complexity of crypto operations에 대한 부분들은 abstract away
  - Cryptographic hierarchy 내 특정 layer가 modular하게 구현, Middleware가 event 발생 시 모듈을 교체

→ Enables agility by allowing cryptographic primitives to evolve (e.g. RSA → PQC) without rewriting application code



*https://csrc.nist.gov/events/2025/crypto-agility-workshop

구현 전략 및 요구사항: Common

[Centralized Policy Control]

- **Recommandation2: Introducing a centralized a policy-driven crypto layer (crypto provider)**
  - 어플리케이션과 crypto logic을 분리하기 위해서는 각 모듈에 대한 선택을 관리하는 정책을 결정하는 layer가 필요
  - Crypto provider/administrator라는 middleware를 통해, configuration(키 길이, 프로토콜 버전 등) 및 정책을 관리
  - Central한 암호화 정책 설정을 통해 구성요소를 명확히 정의 → 코드 수정 없이 새로운 암호 라이브러리 교체 가능
  - Aligned with **the application-centric interface**

<VMWare 사의 Crypto agility framework design>

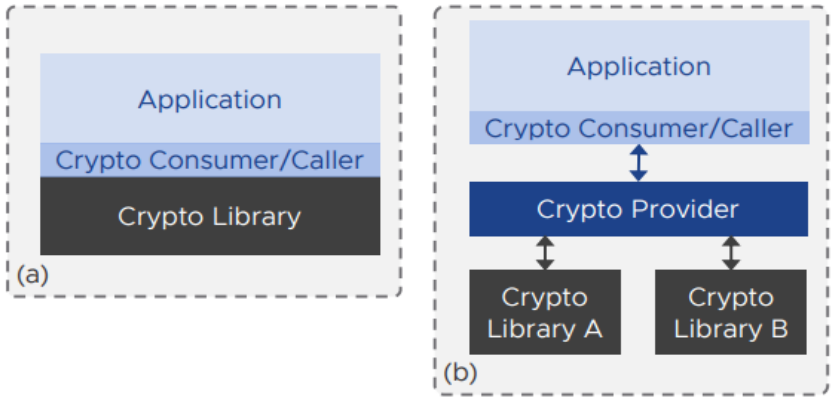


Figure 1: (a) Conventional crypto library usage. (b) Application-independent Crypto Provider framework.

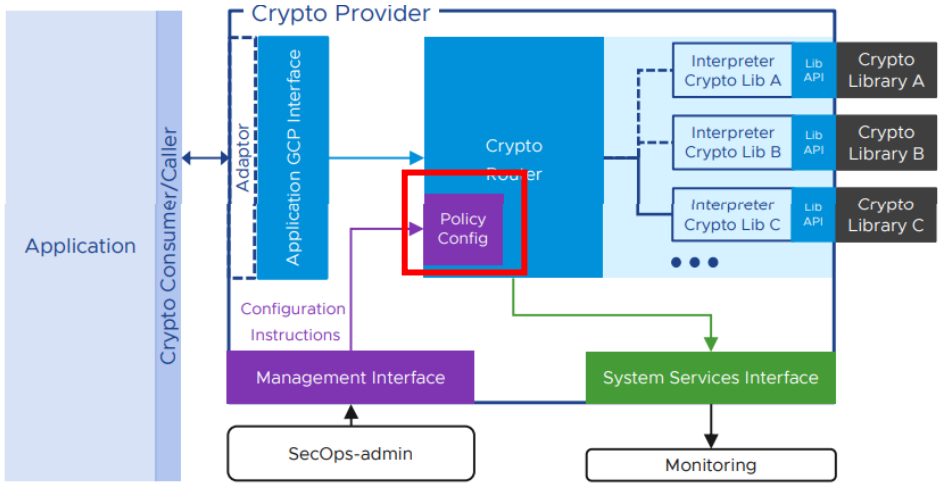


Figure 2: Generic Cryptographic Provider Functional Diagram

* ELCA: Introducing Enterprise-level Cryptographic Agility for a Post-Quantum Era, VMWare

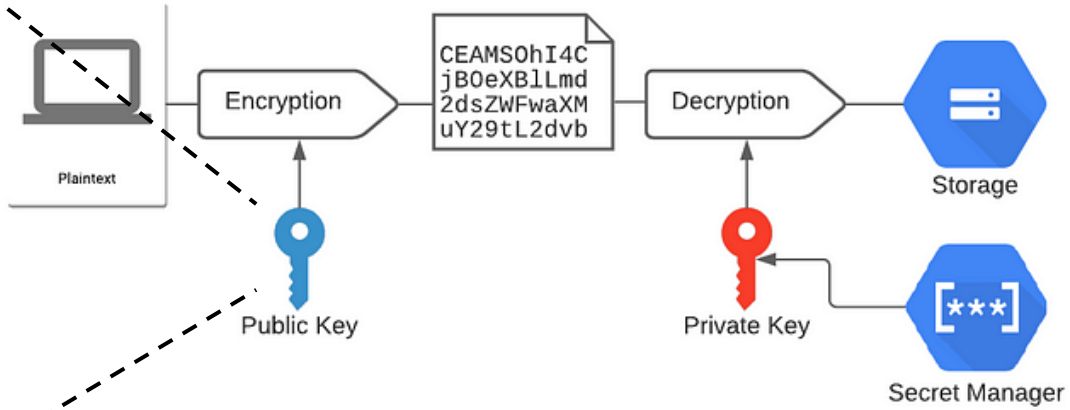
구현 전략 및 요구사항: Common

[Key-centric design]

- **Recommendation3: Use a key-centric abstraction model**
  - Key object가 단순히 raw key material만 담는 것이 아닌, algorithm, parameter, constraint과 같은 metadata를 포함
    → Treat it as a first-class object: "Each key should fully define how it behaves"
  - Key object를 "self-descriptive"하게 정의함으로써, crypto provider가 자동으로 암호 모듈을 configure & switch, 어플리케이션 로직 내 특정 암호 알고리즘에 대한 종속성이 사라짐 → making PQC upgrades seamless
  - HW (HSM), cryptographic API (PKCS #11), cryptographic library (liboqs) 모두 key-centric abstraction에 aware

Key metadata as JSON

```
$ cat pubkey.json
{
  "primaryKeyId": 415969939,
  "key": [
    {
      "keyData": {
        "typeUrl":
"type.googleapis.com/google.crypto.tink.EciesAeadHkdfPublicKey",
        "value": "...snip...",
        "keyMaterialType": "ASYMMETRIC_PUBLIC"
      },
      "status": "ENABLED",
      "keyId": 415969939,
      "outputPrefixType": "TINK"
    }
  ]
}
```

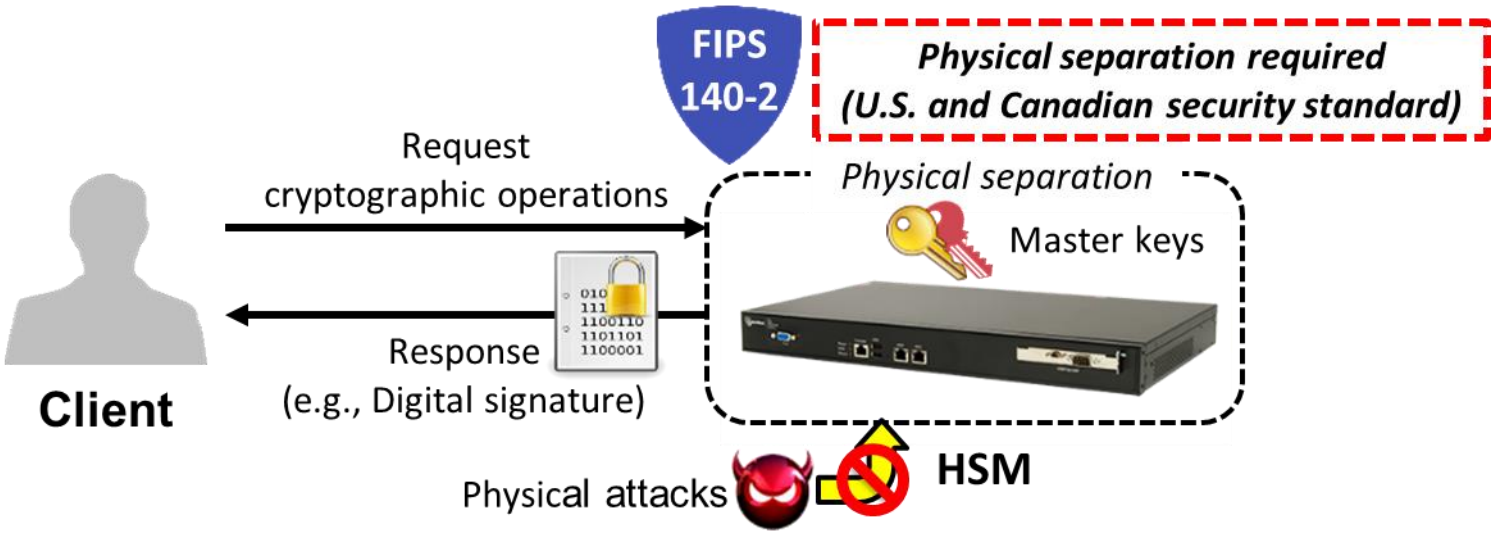


e.g., Google's Tink library
→ Bundling algorithm information inside each key

구현 전략 및 요구사항: HW-vendor 관점

[Current root-of-trust: HSM (Hardware Security Module]

- 다양한 Key Management Services (KMS)를 위한 **de facto Root of trust**
- FIPS 140-2 Level3 이상을 security regulation을 충족, physical security 지원 (e.g., tamper-evident & tamper-resistant)
- "HSM is a cryptographic black box around crypto algorithm implementations"

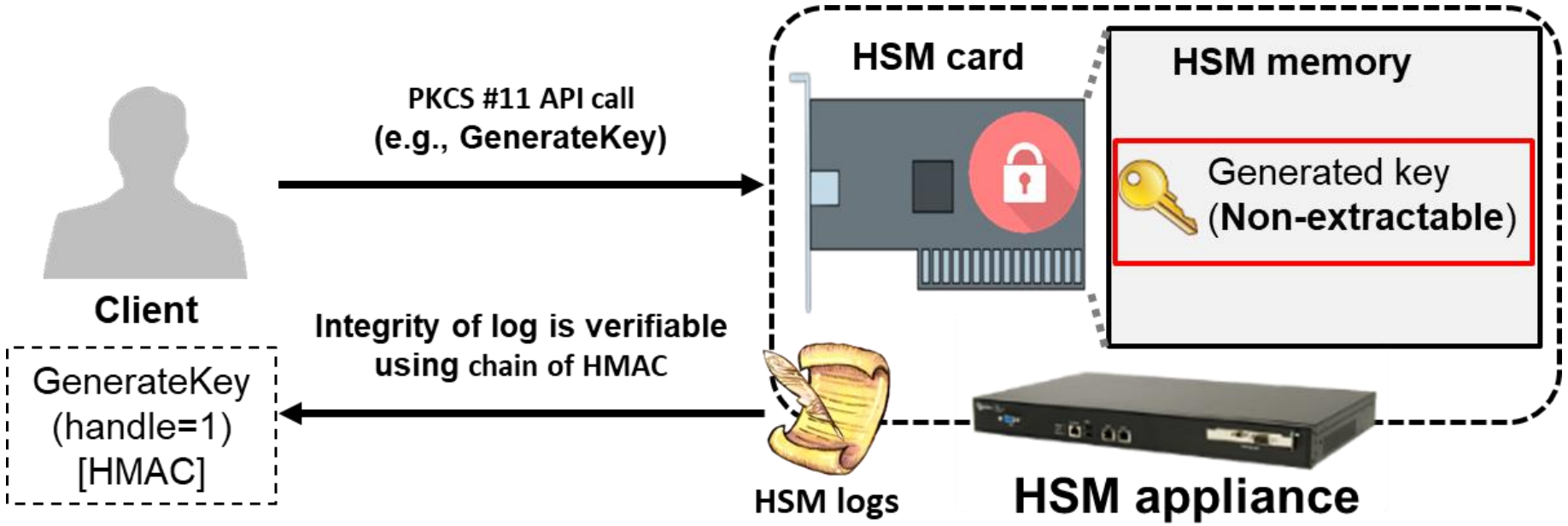


*2021 Global Encryption Trends Study, ENTRUST

구현 전략 및 요구사항: HW-vendor 관점

[Current root-of-trust: HSM (Hardware Security Module]

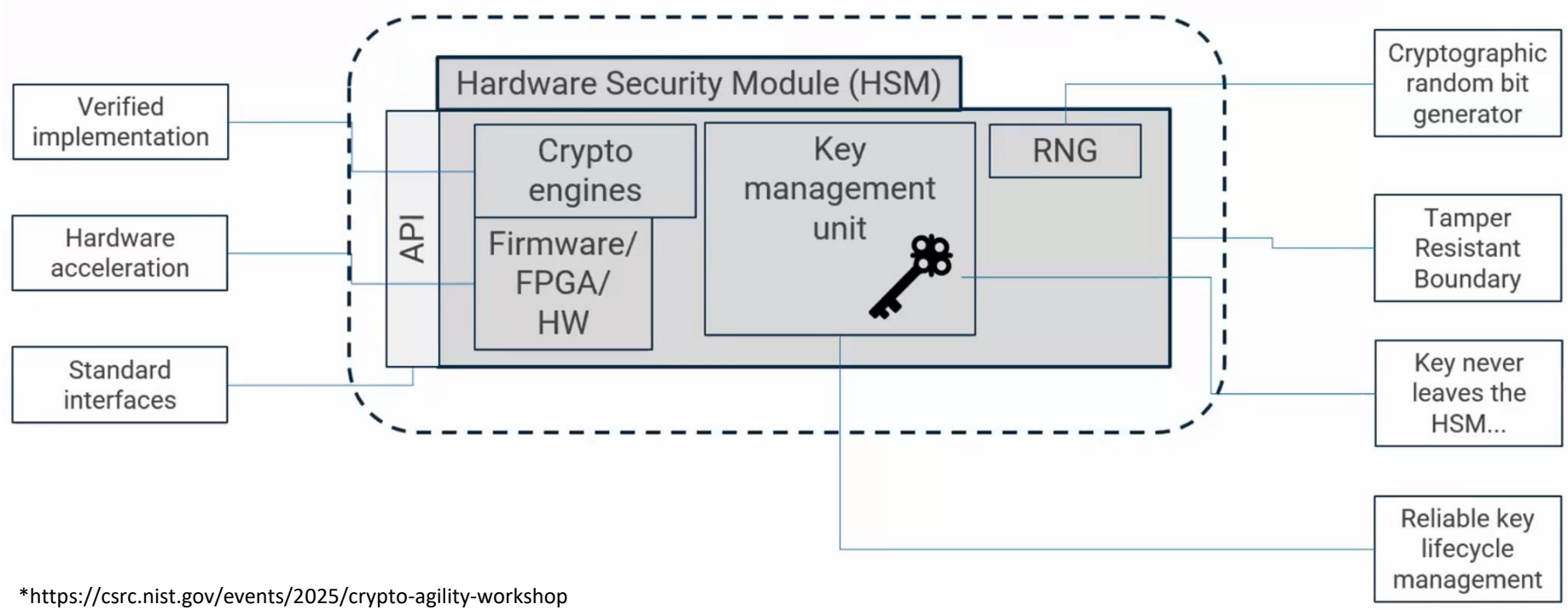
- Standard interface(e.g., **PKCS #11 API** – cryptographic token interface)를 open, API를 통해 어플리케이션이 request
- 다양한 cryptographic functions (e.g., key generation, signing) 및 안전한 키 관리를 지원
- "HSM is a **cryptographic black box**". 다만, HSM event에 대한 감사를 위한 logging system을 지원
- **Non-security relevant functions outside** (e.g., middleware)



구현 전략 및 요구사항: HW-vendor 관점

[Current root-of-trust: HSM (Hardware Security Module]

- **Agility-related features of HSM**
  - Cryptographic API를 open, SDK를 통한 programmability
  - FPGA를 통한 extensible firmware
  - Multi-tenancy를 지원 (for isolating security risks)

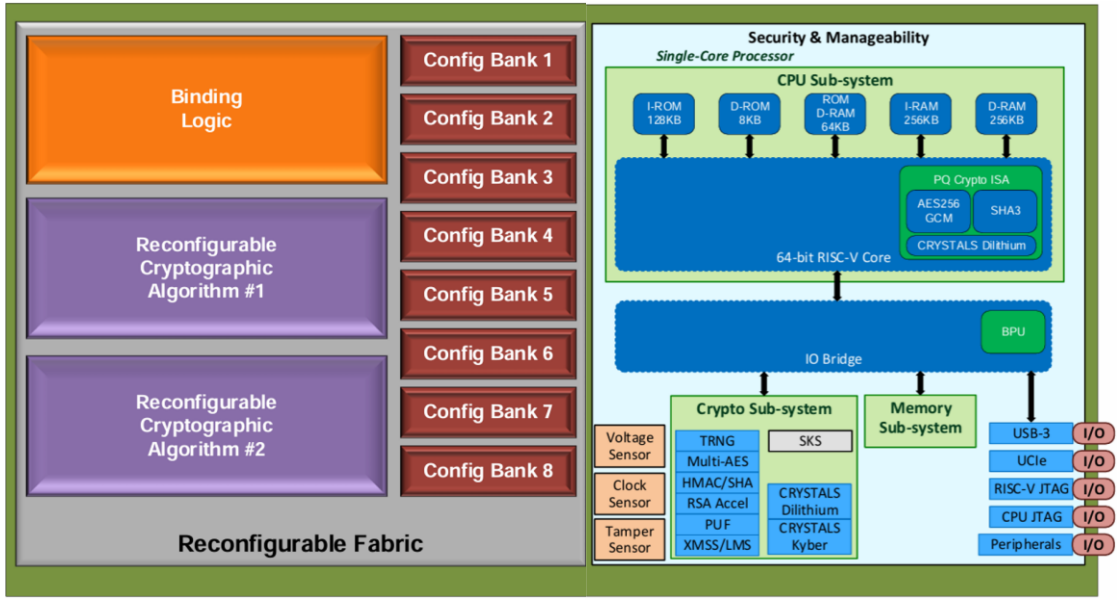


*https://csrc.nist.gov/events/2025/crypto-agility-workshop

구현 전략 및 요구사항: HW-vendor 관점

[HW requirements for crypto agility]

- HSM 내 **eFPGA Tile**(Reconfigurable Fabric)과 같은 재구성 가능한 알고리즘들을 구현할 수 있는 유연한 하드웨어를 포함
- **Secure controller (CPU 서브 시스템, Promising option: RISC-V와 같은 Open ISA를 이용)**
  - 암호: PQC와 legacy primitive를 모두 지원하는 ISA를 RISC-V로 구현
  - 코드 무결성: 재구성 로직에서 메모리 접근 시, 명령어 RAM에 코드 적재 전, 무결성 검사
  - Required operational HW primitives: 장치 식별 및 인증 (attestation), 난수 생성, Secure boot



*https://csrc.nist.gov/events/2025/crypto-agility-workshop

구현 전략 및 요구사항: SW-vendor 관점

[API Design]

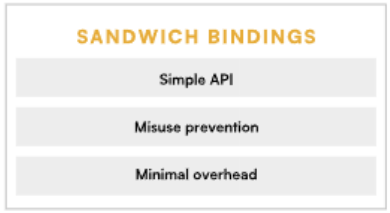
- **암호 라이브러리의 동적 교체를 위한 인터페이스 설계 및 구현이 필요함**
  - 특정 암호화 라이브러리에 대한 어플리케이션의 종속성의 최소화가 필요
- **API should be high-level and explicit**
  - "How you do"이 아닌 "What you do"를 abstract하기 위한 API 설계가 필요
  - "encrypt," "sign," "verify"와 같이 generic한 method 형태로 API를 expose
  - hiding the implementation details (e.g., RSA vs. module-lattice-based KEM)

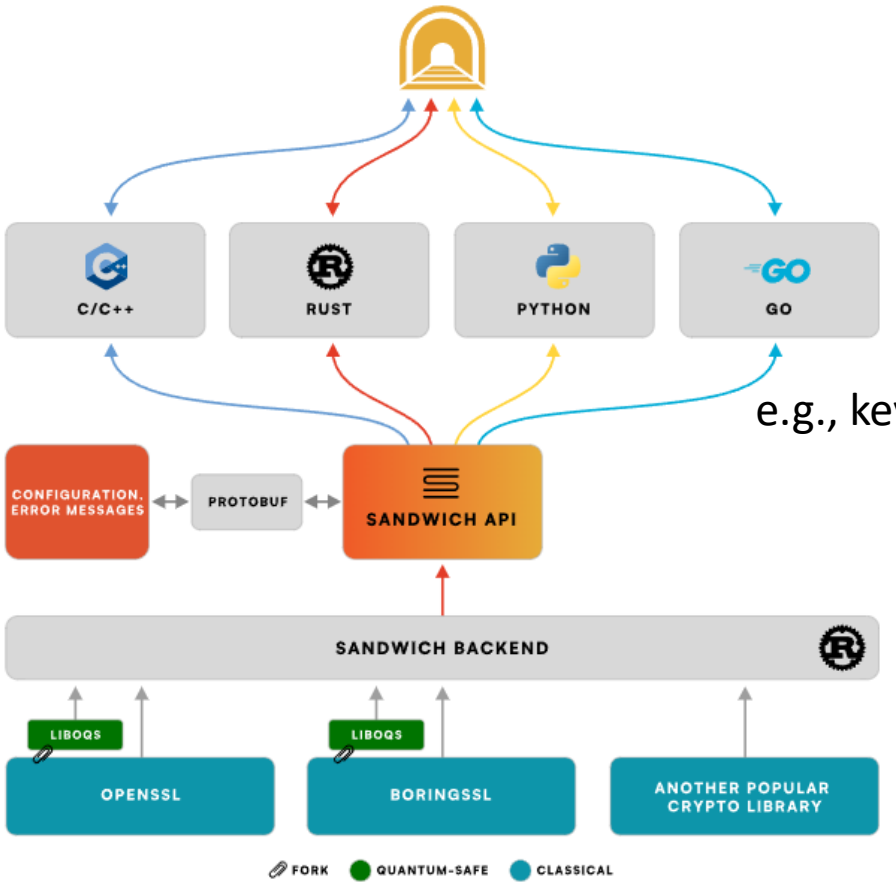
  → Allows the underlying cryptographic algorithms to change without affecting the interface exposed to developers

구현 전략 및 요구사항: SW-vendor 관점

[API Design]

- **Potential baseline for agility-aware API design: Sandwich API**
  - 동적인 cryptographic agility를 구현하기 위한 오픈 소스 프로젝트
  - libOQS, OpenSSL, BoringSSL 지원





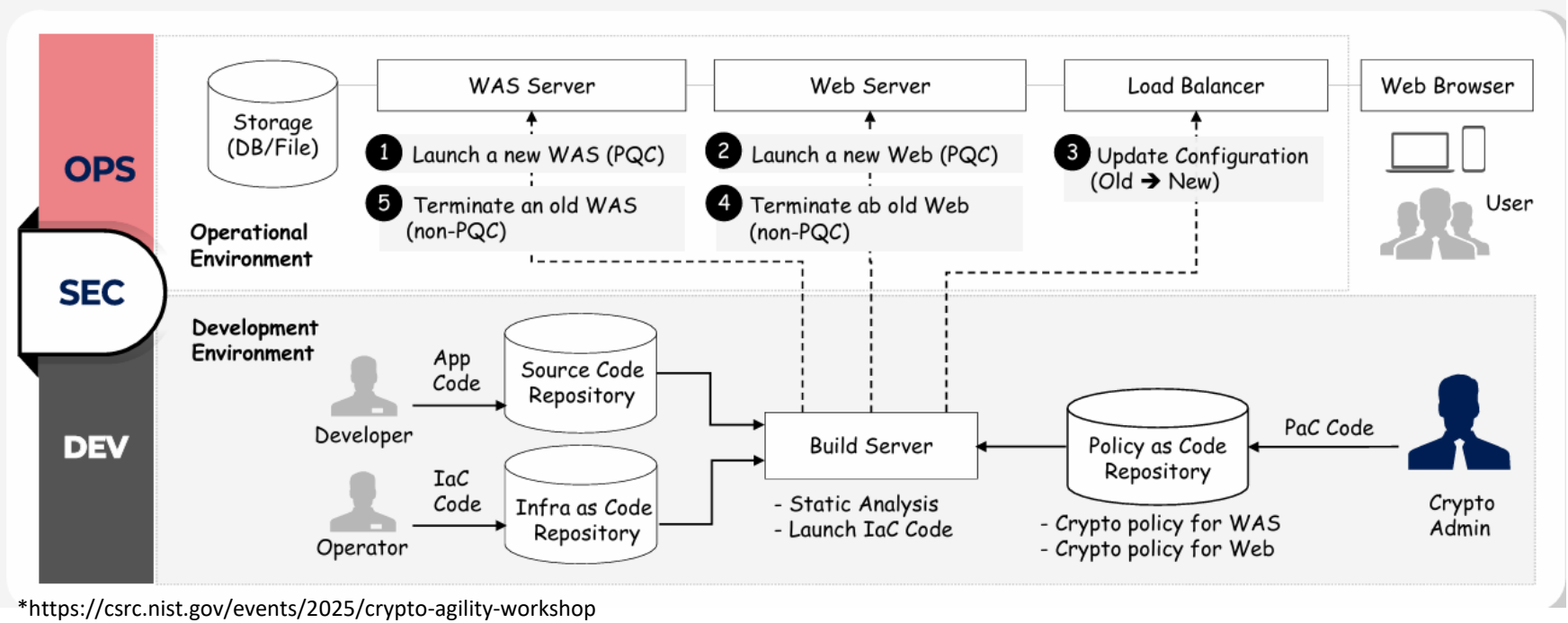
e.g., keygen()

Default: Use TLS 1.2 ECDHE-RSA-AES256, OpenSSL
If key_leaked:
Change suite to TLS 1.3 RLWE-BCNS15- ECDHE-RSA-AES256

구현 전략 및 요구사항: SW-vendor 관점

[Policy-centric Software-Defined Cryptography]

- **DevSecOps with crypto-agility [Samsung SDS, 2025 NIST crypto-agility workshop]**
  - crypto admins: centrally define cryptographic policies and write them as code (Policy-as-Code, PaC)
  - Embedded PaC into infrastructure as code (IaC) during the build process
  - Integrate it with CI/CD pipeline

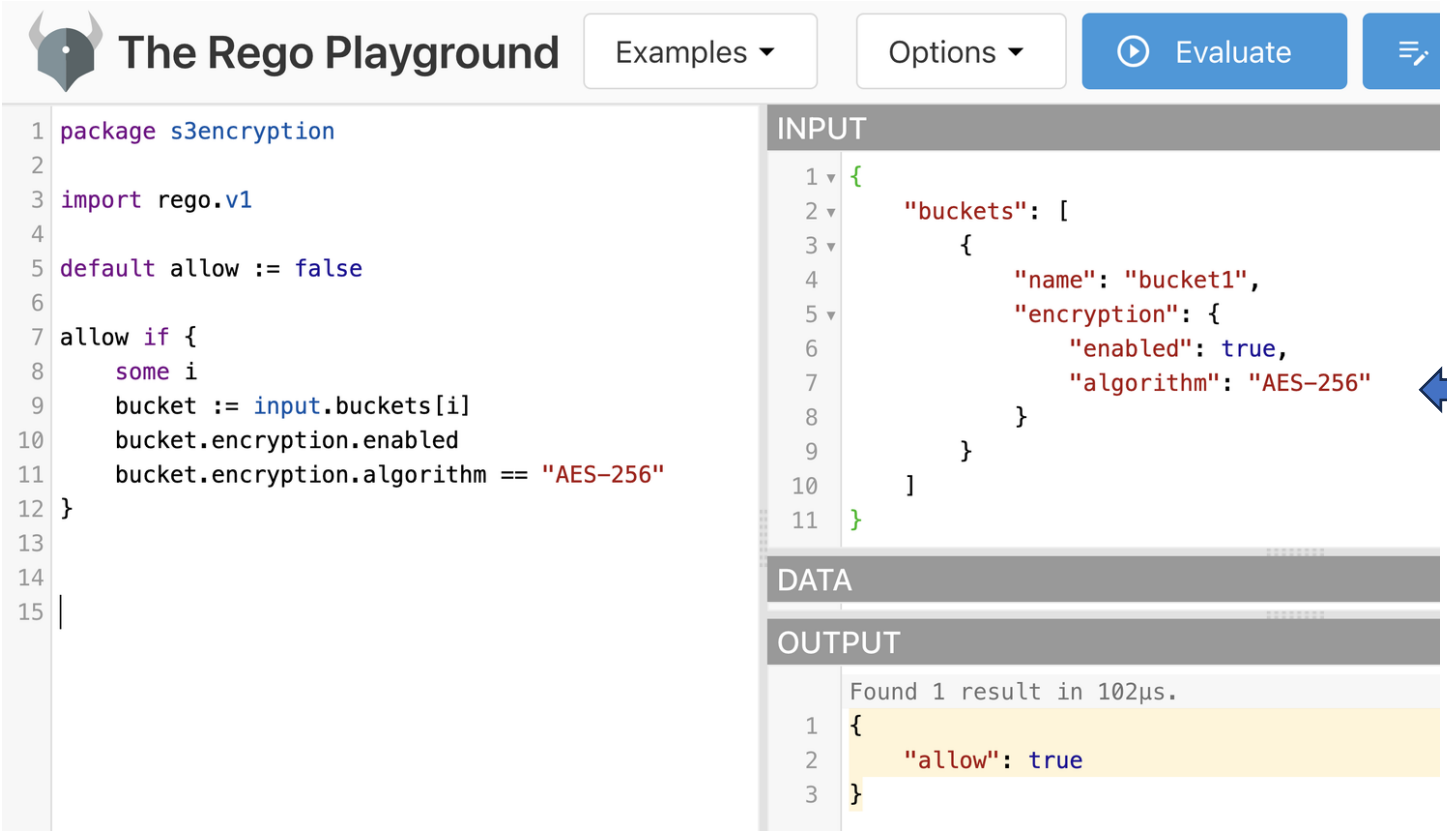


*https://csrc.nist.gov/events/2025/crypto-agility-workshop

구현 전략 및 요구사항: SW-vendor 관점

[Policy-centric Software-Defined Cryptography]

- **DevSecOps with crypto-agility**
  - Aligned with the concept of **policy-driven configurations and centralized policy control**



Making algorithms Switchable to a PQC-based primitive by “crypto admins”

*https://play.openpolicyagent.org/, OPA

Conclusion

- Crypto agility를 지원하는 시스템의 핵심적인 요구사항
  - 암호 primitive 간 switch를 위한 centralized & policy-driven configuration
  - Foundation(FPGA/RISC-V: 이종의 crypto primitive를 지원) – Intermediate(Middleware: Engine 간 교체) – Top(Library: Exposing API)로 구성, 모듈화 된 형태로 application logic과 분리
  - Key-centric design: Key로부터 알고리즘, 파라미터 등을 포함한 meta-data를 파악하고 이를 교체할 수 있도록 하는 data structure의 설계 및 이에 대한 관리 메커니즘이 필요

- 현재 crypto-provider에 해당되는 Intermediate layer를 어떻게 PoC 형태로 구현할 것인가에 대한 design space가 crypto-agility system 설계에 있어 가한 unclear한 부분: Research opportunities!
  - How to abstract the policy and configuration for crypto agility?
  - How to dynamically adopt cryptographic policies at runtime while guaranteeing backward compatibility?

Q & A

Thank you