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

Dimitrios Sikeridis*, David Ott†, Sean Huntley*, Shivali Sharma*, Vasantha Kumar Dhanasekar*,
Megha Bansal*, Akhilesh Kumar*, Anwitha U N*, Daniel Beveridge‡, Sairam Veeraswamy*
{sikeridisd,dott,shuntley}@vmware.com

* VMware xLabs, Palo Alto, California, USA
† VMware Research, Palo Alto, California, USA
‡ VMware Incubation, Palo Alto, California, USA

***Abstract*—Given the importance of cryptography to modern security and privacy solutions, it is surprising how little attention has been given to the problem of *cryptographic agility*, or frameworks enabling the transition from one cryptographic algorithm or implementation to another. In this paper, we argue that traditional notions of cryptographic agility fail to capture the challenges facing modern enterprises that will soon be forced to implement a disruptive migration from today's public key algorithms (e.g., RSA, ECDH) to quantum-safe alternatives (e.g., CRYSTALS-KYBER). After discussing the challenge of real-world cryptographic transition at scale, we describe our work on enterprise-level cryptographic agility for secure communications based on orchestrated *cryptographic providers*. Our policy-driven approach, prototyped in service mesh, provides a much-needed re-envisioning for cryptographic agility and highlights what's missing today to enable disruptive cryptographic change at scale.**

I. Introduction

Cryptography algorithms and standards (e.g., RSA, ECDH, DSA) have become so widely deployed and deeply woven into modern security solutions that it's hard to imagine the world without them. From data encryption and secure communications to identity management and decentralized ledgers, enterprise security solutions across the board rely on well-designed and -implemented cryptography standards. Yet despite our ubiquitous dependence on a handful of widely used cryptography standards, it is surprising how little attention has been given to the problem of *cryptographic agility* or securely transitioning a system or application from one cryptographic algorithm or implementation to another.

Historically, industry-wide transitions have been periodic and ongoing. For example, MD5 was deprecated after cryptographic attacks were discovered [63]. It was replaced by the SHA-1 standard initially announced by the National Institute of Standards and Technology (NIST) in 2001 [7]. However, subsequent collision attacks [66] led NIST to deprecate its use in 2013 in favor of SHA-2, with certificate authorities following suit in 2016 [12], [64]. Other historical transitions include the broader adoption of ECC (e.g., ECDH), the deprecation of DES and 3DES in favor of AES and its variants, and the IETF's evolving TLS standards–SSL 3.0 followed by TLS 1.0, then TLS 1.1 then TLS 1.2 and now TLS 1.3.

The latest disruptive change to public key cryptography is NIST's introduction of *post quantum cryptography (PQC)*. Intended to protect against attack by scaled quantum computers, NIST announced four candidate algorithms for standardization in July of 2022: CRYSTALS-KYBER for key-establishment, CRYSTALS-Dilithium for digital signatures, and alternate signature schemes Falcon and SPHINCS+. Complicating the picture, these algorithms will likely be deployed in *hybrid* form with existing RSA, EC, ECDH, DSA, ECDS standards before eventually becoming *de facto* standalones.

Companies, organizations, and governments around the world now face the problem of cryptographic transition in the complex infrastructures and software deployments they operate. For instance, the Biden National Security Memorandum (NSM-10) of May 2022 instructs a broad gamut of federal and civilian agencies to provide annual reporting on progress migrating to the new NIST PQC standards.

In this paper, we address the question of whether the frameworks for cryptographic transition even *exist* for enabling organizations to implement this transition at the size and scale of modern enterprise computing. In particular, we consider the domain of secure communications where enterprise transition to PQC is especially needed to avoid data exposure to future QC-enabled adversaries.

Traditional notions of cryptographic agility approach the problem of transition from software architecture and endpoint-based communication protocol points of view. The former addresses a developer challenge: how to encapsulate cryptography components in a software architecture and how to modify libraries without re-implementing the system. The latter addresses how two endpoints can interact with one another to negotiate the use of a new cryptographic standard as part of an authentication and key establishment sequence. TLS's cipher suite negotiation is a well-known example, although the pattern is reused elsewhere.

But while the developer-centric and protocol-centric notions of cryptographic agility are obviously important, are they adequate in addressing the holistic challenge faced by enterprises around the world – implementing the transition across complex enterprise computing infrastructures? For example, organizations deploy hundreds of applications, many of which do not expose cryptographic configuration or do not support PQC. Modern applications are increasingly deployed as the co-mingling of multiple services and clustered compute hosts. We believe that migration in these scaled domains represents

an important gap in applied cryptography.

In this paper, we address a complex, real-world challenge facing enterprises around the world – one that, to our knowledge, has not been addressed previously. We propose organizing this problem space through a new notion of *enterprise-level cryptographic agility*. Our contributions are as follows:

• We identify an expanded set of requirements for cryptographic agility, one that articulates the challenge from an operator point of view and comprehends scale,

• We present a key building block for decoupling cryptographic configuration from an application and supporting library extensibility,

• We use this building block to create a distributed management scheme that addresses the problem of orchestration in scaled enterprise infrastructure,

• We implement our scheme in service mesh using a modified version of Envoy proxy and evaluate overheads introduced by our cryptographic abstraction layer.

II. Related Work

In this section, we discuss prior bodies of work related to the challenge of enterprise-level cryptographic agility. In recent years, the notion of *cryptographic agility* has been associated with multiple contexts and has expanded to include various properties, fail-safes, and use-case-specific attributes. We note, however, that academic research better defining the challenge and examining the design space from a computer science and security point of view is noticeably weak.

A. Cryptographic Agility

Brian Sullivan used the term in 2009 to describe an application design practice that would allow developers to "replace broken algorithms on the fly without having to recompile" [61], [62][1]. While initially, he describes how such practice can abstract .NET code from using hard-coded schemes like hash algorithms (e.g., MD5), he also makes the case that the same logic can be applied to all cryptographic algorithm types, i.e., symmetric, asymmetric, hash-based message authentication codes, or keyed hashes.

Informational RFC 6421 [44] defines cryptographic agility as "the ability of a protocol to adapt to evolving cryptography and security requirements". The author also describes it as a modular way to avoid disruptions in implementations when algorithms need to be updated for security reasons. RFC 7696 [27] presents guidelines for ensuring that security protocols can transition from an old suite of algorithms to updated versions or completely new schemes when desired.

NAS authors from the 2017 workshop on "Cryptographic Agility and Interoperability" [43][§ 3.1] also approach the problem from the perspective of the protocol implementer. Design concerns include the ability to add new schemes following verification and designing protocols that sift through old, new, or mandatory algorithms providing interoperability.

[1]https://infocondb.org/con/black-hat/black-hat-usa-2010/cryptographic-agility-defending-against-the-sneakers-scenario

In [43][§ 2.1], Kerry McKay from NIST expands the notion of cryptographic agility to include: (a) real-time security scheme selection based on the system's security guidelines, (b) the ability to add new cryptographic schemes, and (c) the ability to retire obsolete or vulnerable cryptosystems efficiently. Authors in [47] and [38] add various abstract properties to the functional requirements of a cryptographic agility framework: enforceability, heterogeneous environment support, performance awareness, policy awareness, scalability, and full automation.

B. Risk Assessment Frameworks

Another body of related work examines cryptographic agility as a resource allocation problem from the prism of enterprise risk assessment evaluation. Authors in [37] present an enterprise-level framework that can proactively evaluate risks due to cryptographic transitions and draft a response strategy that will fit specific business needs. The proposed response consists of a five-phase solution that includes threat identification, available asset inventory, estimation of risk, risk mitigation, and results in the development of an organizational roadmap. Authors in [26] similarly introduce a maturity model for inferring an IT infrastructure's readiness for transition.

C. Implementation Frameworks

Prior treatments of cryptographic agility as a framework for transition are scattered and cross-cutting. Notions are seen in various spheres, from hardware modules to gateway applications to abstraction at a service software layer, to name but a few.

Several internet protocols adopt algorithm agility to enable replacement due to newfound insecurities or to enable more attractive alternatives, for example, new algorithms offering smaller signatures or keys. Cases statues can be seen in SSH [25], [9], [30] and TLS [54], [20] where cipher negotiation mechanisms have been integrated into the handshake protocol managing communication setup. The scheme can be used to enable the transition to PQC key exchange and authentication schemes in a straightforward way [58]. X.509v3 digital certificate extensions provide an additional agility mechanism for transition to PQC in PKI and network-based authentication schemes [47]. [42] looks at how DNSSEC has achieved partial algorithmic agility at the moment.

On the Industry side, Senetas offers a hardware encryptor that provides a Field Programmable Gate Arrays architecture [56] for network encryption and hardware agility. The solution, however, requires proprietary and dedicated hardware. Cryptomatic offers a gateway that utilizes a cryptographic control point to act as a policy manager and Hardware Security Module service[17]. The solution is, however, limited by the cryptographic algorithm and keys used in each specific application endpoint.

On the other hand, there is relatively little research on practical crypto-agility approaches, especially in enterprise contexts. In [36], the authors discuss security considerations of a cryptographic API, while in [50], the authors support the idea of crypto-agility APIs that are developer-friendly in terms of source code adjustments and maintenance. The authors of [68]

introduce eUCRITE[2] a general-purpose API for Java aimed at enabling crypto-agility for developers. CogniCrypt [34] is a framework that simplifies the use of crypto APIs through an Eclipse plugin. The framework generates secure implementations for simple cryptography-related programming tasks. The authors in [53] present EverCrypt, a provider of cryptographic functionalities that offers an API for both choosing among various implementations of an algorithm and choosing between different algorithms for the same functionality. The authors validate their design via case studies that consider the performance of QUIC's transport security and cryptographic operations in Merkle trees.

Another body of related work has looked at cryptographic libraries that simplify the use of cryptography via simpler interface design, including Charm[8] for Python and Keyczar [19] for Python, Java, and C++, or Tink [10] for multiple languages. This is achieved through handling routine tasks, default configuration support, exclusion of unsafe suites, schemes, or algorithms, and reducing the volume of the inputs for the developers. However, most of these schemes have shortcomings, namely simplified use for limited operations only. Tink and Keyczar support only signing, message authentication codes, hybrid encryption, and shared key authenticated encryption, while Charm only supports the latter. Finally, the authors in [31] introduce high-level abstractions for cryptographic operations to simplify their use with full declarative configurations. The abstractions can be implemented on top of any crypto library and language, leading to simplified writing of security protocols with a small computational overhead.

Lastly, the recent release of OpenSSL 3.0 is a testament to the need for additional implementation agility with the introduction of `providers` that implement an abstraction for accessing different algorithm implementations[2]. This is achieved through OpenSSL's high-level API, with providers able to be loaded at any time, while parameter passing is supported in an implementation-agnostic logic. Some OpenSSL's 3.0 providers are the legacy provider for algorithms that have commonly fallen out of use, the default provider, the FIPS provider for schemes that conform to the Federal Information Processing Standard FIPS 140-2, and the base provider that supplies the encoding for OpenSSL's asymmetric cryptography [65].

D. Post-Quantum Cryptography Transition

In anticipation of NIST's quantum-resistant public key standards [45], more and more authors discuss how enterprises should be planning for transition to a whole new era of cryptography. Unsurprisingly, nearly all mention cryptographic agility as a *de facto* path for transition.

In [51], for instance, authors focus on a financial institution's networking infrastructure and investigate its quantum resistance as a function of its current cryptographic agility. The importance of agility in cryptosystems is also highlighted in [50] from the domain of industrial automation where production system components are becoming Web accessible.

[47] aims to inform enterprises of the post-quantum era in cryptography and guide them through an organization roadmap that includes a migration plan, actions on long-lived digital assets, and embracing crypto agility. Similarly, Joseph et al. in [28] provide recommendations on the transition to post-quantum schemes, with crypto-agility being the first mentioned. The authors suggest adopting abstraction layers on toolkits that are managed centrally —e.g., crypto libraries that abstract algorithms from infra teams— and underline the need for embedding crypto-agility in any new standard like 6G. Crypto-agility is also identified as a PQC migration solution by Brian Lamacchia in [35] under the condition that possible adversarial tampering and the introduction of new attack surfaces are thoroughly investigated.

In [46], Ott et al. extensively discuss the challenges associated with PQC migration, also considering complex infrastructures. In addition, the authors reintroduce the notion of crypto agility in the context of the upcoming public key cryptographic algorithm replacement and expand its definitions to eight possible scopes, among them being implementation and compliance agility [46]. Finally, the authors in [68] offer a survey of existing works on PQC and crypto agility.

While the discussion on PQC migration is intensifying, none of the above literature offers detailed blueprints of how an enterprise-level cryptographic agility scheme is implemented or integrated into existing software solutions, which is precisely the gap that this work targets to fill.

III. Enterprise Requirements

For clarity, we define an *enterprise* as an organization (e.g., a small- or medium-sized business, a multi-national company, a government entity, a university) overseeing a compute *infrastructure* in service of its day-to-day business functions. For example, a large- or mid-sized company will employ an IT operations team to procure and manage on-premise data centers, cloud hosting, and service arrangements, client devices for employees, edge office or retail deployments, and more. Today's enterprises, furthermore, make use of hundreds (sometimes thousands) of software applications and services to manage customer accounts, personnel, internal communications, web content, product support, and much more. Some companies additionally develop in-house software, but most rely on third-party suppliers.

As seen in section 2, cryptographic agility is often loosely thought of as a system attribute: the ability of a system, application, or protocol to securely transition from one cryptographic algorithm or implementation to another. In this paper, we propose defining crypto-agility as *an architectural framework for implementing cryptographic transition and change*. That discussion of crypto-agility can and should be a discussion of architectural schemes for enabling change within a system or application.

We furthermore make the distinction between *monolithic* cryptographic agility frameworks and *enterprise-level* frameworks. Prior work on cryptographic agility looks largely at how a software library can be organized to facilitate changes by an application developer or how a communication protocol can be designed to add or deprecate a cipher suite within a secure connection. This first essential building block for cryptographic agility can be summarized as follows:

[2]https://cspub.h-da.io/eucrite/

• *Cryptographic Transition*. Enables transition from one cryptographic algorithm to another.

To add scale, we propose the notion of *enterprise-level cryptographic agility (ELCA)* to address the problem of cryptographic migration across larger units of enterprise infrastructure. We see ELCA as adding four key requirements to address the problems of scale, control, policy, and monitoring:

• **Enterprise Control**. Enterprise operators are given control over the cryptographic configuration of their infrastructure.

• **Orchestrated Migration**. Operators can migrate larger units of infrastructure (e.g., data center, mesh) in coordinated ways.

• **Policy Governance**. Configuration can be expressed as policies that map infrastructure to cryptographic configurations.

• **Monitoring/Auditing**. Infrastructure configuration is transparent and tools are available to monitor and audit configuration state.

The above list of ELCA requirements reflect our belief that the problem of cryptographic migration across complex infrastructures is an enterprise-level challenge that cannot be solved by the aforementioned building blocks. As such, frameworks should be designed to enable enterprise operators to have full control over the cryptographic configuration. This may seem obvious, but recall that agility in software library domains largely serves developers, and many software applications and services put cryptographic configuration in the software provider's hands. In fact, many enterprise operators find cryptographic change daunting exactly because it means corralling hundreds of software providers to support the change and with very little control over the process.

We argue, furthermore, that only enterprise operators can know the domain-specific considerations underlying configuration decisions. For example, early PQC protections against harvest now, decrypt later attacks imply an understanding of where long-lived information assets flow over the wire and where FIPS protection (without PQC overhead) is more desirable.

The scope of migration for ELCA is expanded from single-application or single-connection contexts to larger units of infrastructure. For example, orchestrated migration may be needed for a cluster of web or content caching servers, a large pool of client devices, or a set of data centers distributed across North America. Alternatively, migration may be applied to bundled software services hosted within private cloud infrastructure or across services hosted by a public cloud provider. Enterprises may divide their configuration domains by geography, applications, data classification, logical subnet, administrative domain, or myriad other schemes. A common consideration will be compliance domains defined by government (e.g., NIST) or industry (e.g., HIPAA, PCI DSS) regulatory bodies.

To address large domains of configuration and to better aid in defining principled configuration, ELCA frameworks should support the creation, deployment, and maintenance of enterprise policies. A *policy* (discussed in Section V), encapsulates a cryptographic configuration that will be deployed across a unit of enterprise infrastructure. It might, for example, include a designated public key cryptography algorithm, secure hash function, acceptable TLS version, and more.

Finally, ELCA frameworks must provide monitoring and auditing capabilities for an enterprise operations or SOC team. First, the framework should allow an operator to know where cryptography is used within an infrastructure and the current configuration state. Second, it should provide verification artifacts – preferably those with cryptographic proofs ensuring authenticity and integrity – that document configuration state and reconfiguration events. These can be used for verification, third-party auditing, and general debugging.

A summary of comparison points between monolithic and enterprise-level cryptographic frameworks is shown in Tab. I.

	Application-level	*Enterprise-level*
Admin Control:	Developer	Enterprise IT/SOC
Configuration:	Direct	Policy-based
Deployment:	Manual	Orchestrated
Monitoring:	App-specific	Infrastructure-wide

Table I: Comparing application-level and enterprise-level cryptographic agility

IV. Cryptographic Provider

To address the above requirements, we have developed an enterprise-level cryptographic agility framework based on the notion of a *cryptographic provider*. The provider decouples cryptographic services from the application and creates a configurable control point for enterprise operators. It also provides such features as routing calls across cryptographic libraries, the use of policy-based configuration, and monitoring services. In this section, we describe the basic architectural scheme, including its key components and interfaces. In Section 5, we will describe how the architectural primitive can be scaled to support orchestrated agility across larger units of enterprise infrastructure.

A. Architecture

A functional diagram and overview of the Cryptographic Provider is shown in Fig. 2. A key point to note is the need for application independence. As mentioned above, enterprise control over cryptographic configuration implies the need to decouple the cryptographic module from application control, thus migrating configuration decisions from the developer to the enterprise deployer and user of the software. While some argue that developer decisions on behalf of a naive user may be a good thing, we believe that developers frequently lack the information they need to anticipate the configuration needs of an enterprise. For example, an enterprise IT team may have differentiated compliance requirements to observe across deployment geographies, specific roadmaps for PQC testing and migration, fast-paced remediation procedures for an unexpected library vulnerability discovery, or even custom cryptography libraries to integrate.

The decoupled Crypto Provider can be integrated with the application in multiple ways. Our work has experimented with a wide variety of configurations exploring both in-process and out-of-process alternatives, statically linked and dynamically linked modules, and proxy-based schemes that provide cryptographic agility to key segments of the end-to-end communication path. While configuration control is given to

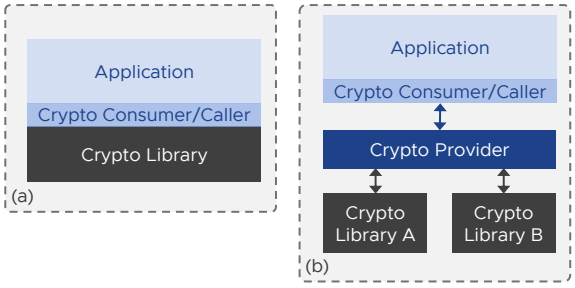

Figure 1: (a) Conventional crypto library usage. (b) Application-independent Crypto Provider framework.

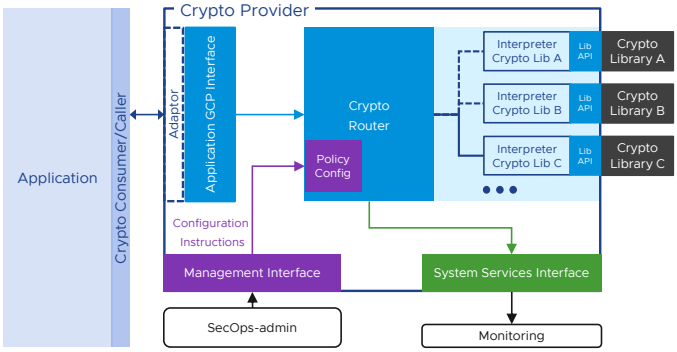

Figure 2: Generic Cryptographic Provider Functional Diagram

the enterprise, application developers may or may not directly integrate the Crypto Provider into their code. As described in Section 6, our Envoy-based implementations combine several of these alternatives in a proxy-based scheme that can be used with unchanged legacy applications.

In what follows, we describe the internal components and interfaces of the Cryptographic Provider by tracing an application crypto call toward execution. An application can directly interact with CP's Application Interface, which exposes a set of high-level cryptography interfaces that abstract the complexities of the underlying crypto implementations from the developer. The generic crypto service call will then be passed on to the *Crypto Router* component, which is responsible for a runtime decision on which crypto library will service the call and the details of the call to be executed. This decision will be made using a configuration lookup scheme. There the CP maps information about the caller and the cryptographic service requested to an available crypto library and a specific algorithmic standard to be used (e.g., ciphers, key sizes, protocol versions). For example, a TLS connection should use TLS v. 1.3 and the ECDHE-RSA-CHACHA20-POLY1305 cipher suite in OpenSSL 1.0.2 (e.g., see Figure 4 for a detailed illustration). In that sense, the CP should be considered a cryptographic Policy Enforcement Point (PEP) where policies defined at the management layer (see section V for our definition of policy) are translated to detailed crypto configurations ready to be applied to the different data-streams.

The crypto call and associated configuration information are then passed to the *Interpreter* for the selected backend library. The Interpreter translates the information to a corresponding low-level function call (or set of function calls) in terms that are customized to the individual library. An Interpreter mapping generic calls and configuration information to a specific crypto library is needed but can be written once and widely reused across CPs. The appropriate set of functions are executed by the backend crypto library to perform the requested crypto service, while the results of the call are passed, through the component chain, back to the application via the original API call.

To enable enterprise operators to create, delete, and update cryptography policies and therefore adjust crypto configuration on the CP, a *Management Interface* is introduced. The interface is also used to manage crypto libraries, for example, adding an additional OpenSSL version library or updating a library with a patch to address a recently announced vulnerability. A *Systems Services Interface* is also provided within the Cryptographic Provider to manage reporting and auditing functions. With this API, an enterprise operator application can inventory the policy configuration of the current CP instance and obtain crypto call statistics. The mechanism provides information in a digitally signed fashion and leverages modern attestation methods.

Finally, to avoid the need for application code changes (e.g., the case of legacy applications), there can be an optional *Adapter* component. This component will translates a library-specific crypto call to a generic *Crypto Provider (CP)* call and thus acts as a shim between an application and the CP.

B. Proxy-based Crypto Agility

Since implementing cryptographic transition for legacy applications is a crucial challenge for enterprise operators, our approach to Cryptographic Provider design options explores the use of proxies in data communications. In this scheme, an application TLS connection is segmented or tunneled so as to provide configurable cryptographic protections for data traveling over a public network, through an untrusted provider network, or across broadcast wireless networks. The CP-enabled proxy can be placed in front of one or both endpoints of the end-to-end communication.

A key benefit of this scheme is that it can be used to address the widespread problem facing enterprise operators of what to do with hundreds of legacy applications that do not support quantum-safe cryptographic configurations (i.e., PQC). But the scheme also addresses the problem of scalability in migration since automation and scaling, as described in Section V, can be used to relieve operators of the burden of re-configuring hundreds of applications on a per-application basis, each with slightly different configuration schemes. We believe that proxy schemes can be a crucial component in PQC migration broadly as the industry slowly improves its support for PQC standards across thousands of application domains and industry products.

V. Scaling to Enterprise-level Agility

In this section, we discuss how the Crypto Provider of Section IV can be scaled to provide enterprise operators with cryptographic agility for larger units of infrastructure. As discussed in Section III, we're looking to support orchestrated migration and policy support in a way that addresses the problem of cryptographic migration in realistic enterprise operating environments.

The challenge of scale may be thought of as a distributed control problem where a centralized agent (i.e., the infosec operator) must coordinate a replicated set of control points (i.e.,

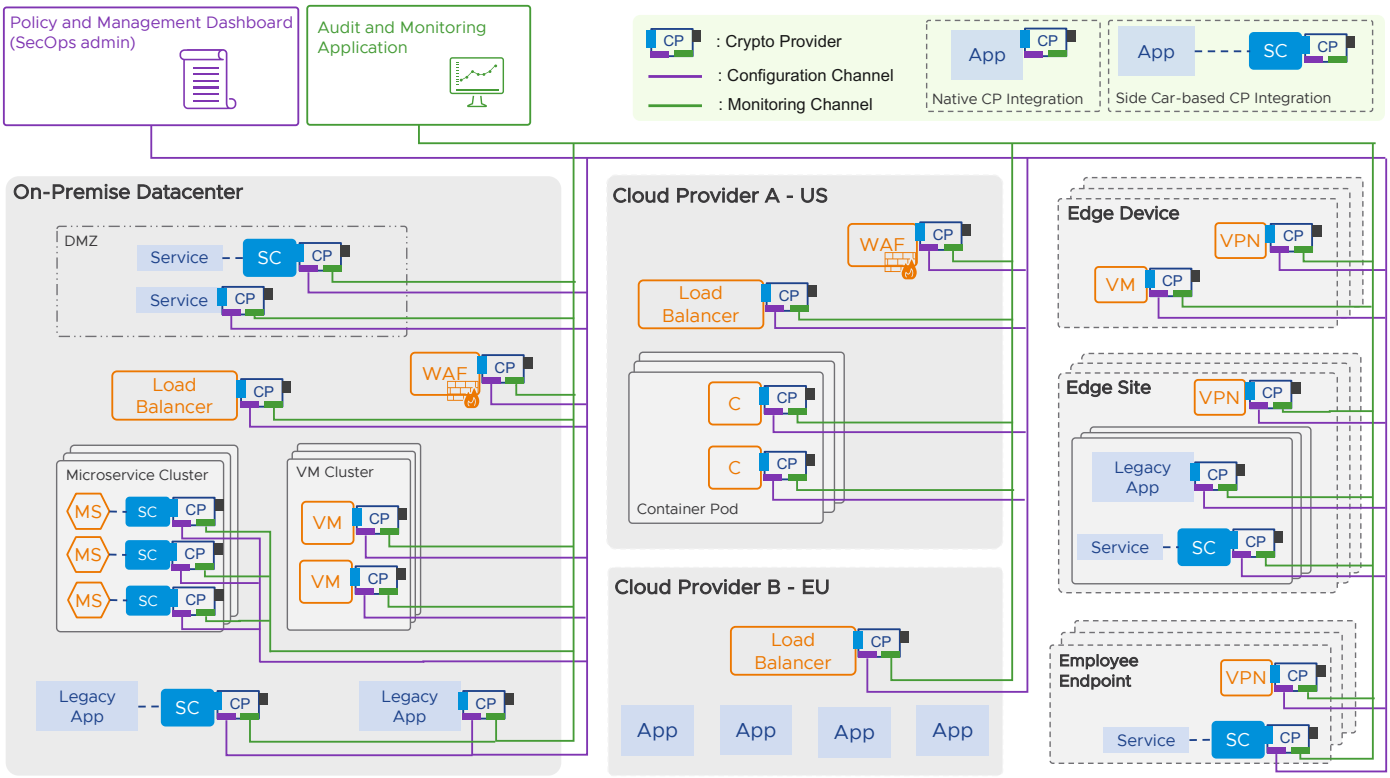

Figure 3: Enterprise-scale CP deployment: On-Premise Datacenter (left), Cloud Providers A and B (center), Edge Deployments (right)

Cryptographic Providers) distributed across the infrastructure. Our work draws upon extensive literature in this space to define a *distributed control plane* supporting a centralized dashboard for infrastructure operators and automated mechanisms to manage configuration, policy definition, and monitoring across CP nodes.

Fig. 3 illustrates the framework across three illustrative modern Enterprise environments: an on-premise data center, a public cloud provider, and an edge computing environment. At the upper left are the administrative applications serving the centralized operations team: the *Policy and Management Dashboard* application which provides an enterprise operations team with tools for defining custom units of infrastructure and policies that will govern their cryptography configuration and the *Audit and Monitoring Application* which will provide inventory, monitoring, and audit services across the infrastructure. With these tools, SecOps admins can create and transition cryptography configurations based on the application context, differentiated data protection needs, geographic location, regulatory considerations (e.g., FIPS), and more.

Within the on-premise datacenter, we observe that Crypto Providers can be deployed in various ways to provide configurable protection for data in motion. A CP may be deployed within a network demilitarized zone (DMZ) on behalf of external-facing services and leveraging secure tunnel or proxy-based communications. It may be placed alongside web application firewalls (WAFs) or load-balancers, which are also common staples in enterprise networks. For modern applications, CPs can be instrumented within microservices and or within individual VMs as part of the operating system communications stack. Finally, a legacy application can use a CP either through code native application support or the use of a proxy mechanism (see Section VI for our prototype).

The same range of options extends to cloud provider and edge computing environments, as shown in the middle and to the right of Fig. 3. Within the former, Crypto Providers may be placed at load balancers, WAFs, within containers, or directly within an application architecture. For the latter, CPs may be used within enterprise tunnel or proxy software, integrated into VMs or containers, used as part of a service mesh proxy scheme, or used directly with a specific application architecture. In addition, the proposed CP component can be easily modified to be integrated into databases, or even handle low-level crypto calls outside of the narrow encrypted communication spectrum.

Our discussion here is necessarily high-level, and the details of any given context of CP deployment and integration warrant further description. But we argue that the use of Crypto Providers can be integrated into many well-studied and widely deployed components within enterprise networks.

A. Policy Handling

In this section, we describe our notion of a cryptographic policy, the process of distributing that policy to CPs, and the lifecycle management behind enterprise configuration management at scale.

Broadly, a policy is created by the enterprise security operator and is deployed across a cluster or grouping of Cryptographic Providers. We refer to this distributed configuration scheme as the *management plane*. Note that a given

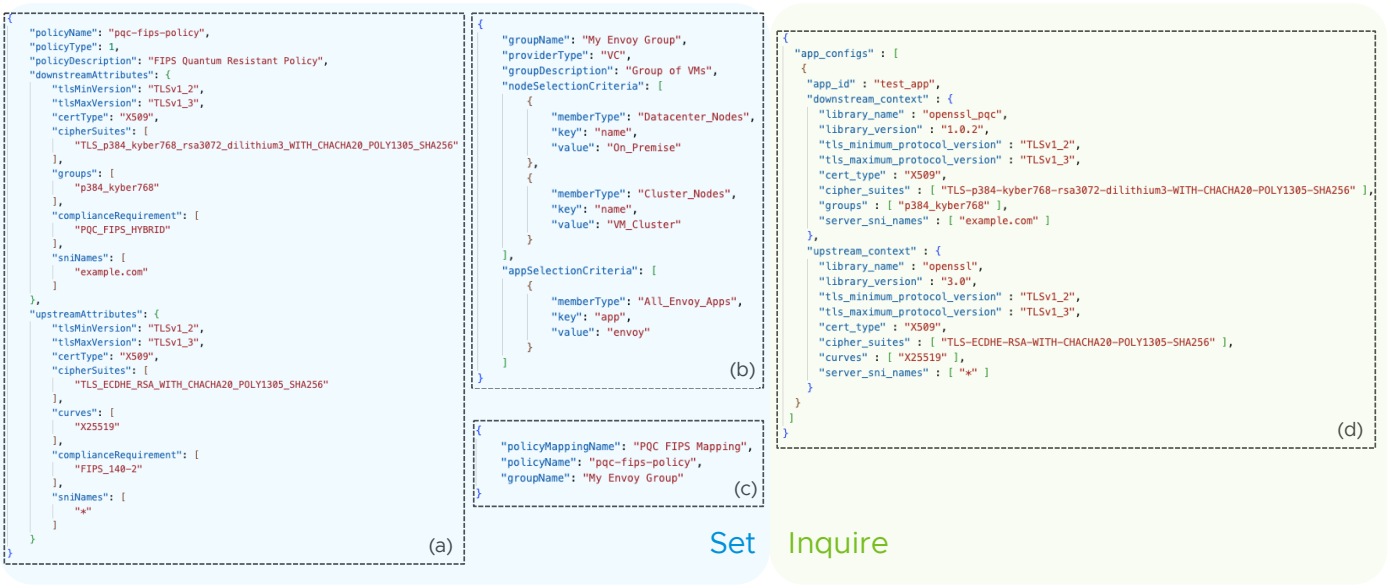

Figure 4: Illustration of policy-related management structure samples: (a) Single policy definition, (b) Cryptography group, (c) Group-Policy Mapping, (d) Get policy updates response

CP receives only necessary configuration information and is not necessarily aware of the overall global policy set by the operator for the infrastructure as a whole.

The proxy-based approach we explore positions the Crypto Provider between the production Internet and an application. Hence, a cryptographic policy must define the configuration to be used by two different communications interfaces: incoming connections from a remote endpoint and outgoing connections from the application server. (In Envoy parlance, these are referred to as ”downstream” and ”upstream”, respectively.) Fig.4 shows an illustrative policy.

In section (a) of the policy, the TLS configuration is defined for outbound traffic (downstream), including the cryptographic library and version to be used, the TLS protocol version(s) that are acceptable, the cipher suite and elliptic curves to be used, any compliance requirements, the cert type to be used, and so on. In addition, as part of the policy, the operators can specify Server Name Indications (sni) or specific IPs for a more fine-grained application of the policy on incoming connections. Similarly, the TLS configuration is also shown for the backend traffic (upstream) with a similar set of configuration items.

Note that the policy as a whole is titled ”pqc-fips-policy” because it requires the use of PQC on the inbound configuration but fips-only configuration on the inbound configuration. The idea is to protect information assets as they are transmitted across the production internet while not requiring PQC on the internal communication with the application server.

Section (b) of the policy defines the infrastructure domain over which the policy will be applied. In other words, it names the set of CP nodes that will receive the configuration, including both the CP nodes themselves and the application endpoints that they will be communicating with. In our example, the policy refers to selection criteria with member components defined using a key-value format. Here, the example refers to a node as any machine equipped with the proposed CP component that runs cryptography-enabled applications. The application is simply a test application referred to as ”gcltest”.

Finally, section (c) of the policy creates the mapping between the cryptographic configuration and the unit of infrastructure to be configured. It maps the defined policy name (”pqc-fips-policy”) to the CP node group (”Dev Group”). Note that the scheme supports the creation of many different configuration policies and infrastructure groupings and that an operator can select mappings in any way required to meet the crypto configuration requirements of their infrastructure. For example, policy mappings may be used to cover each of the infrastructure types seen in Fig.3. We envision a user-friendly graphical application (”Policy and Management Dashboard” in 3) to aid the operator in creating and managing policies.

Section (d) of Fig.4 shows the report generated by querying a configured CP using systems Services Interface described in Section 4.1. Once again, we envision a user-friendly graphical application (”Audit and Monitoring Application” in 3) to aid the operator in querying the current configuration state for a given infrastructure grouping or specific CP node and application pairing.

VI. Implementation

In this section we present a prototype implementation of our proposed crypto agility framework in the context of Envoy Proxy [32]. Note that our work implements a subset of the proposed interfaces–specifically those that enable an operator to characterize performance for new cryptographic configurations.

A. Cryptographic Provider API

The Cryptographic Provider must expose a generic set of interfaces that service a wide range of application cryptography calls. Our prototype has focused on those interfaces required to implement cryptographic operations in secure communications, and specifically the cryptographic libraries needed for SSL/TLS. Table II summarizes the minimal set of API calls

Functionality	CP API	OpenSSL Mapping	GnuTLS Mapping
Server context creation	gcl_create_server_context	SSL_CTX_new SSL_CTX_use_certificate_file SSL_CTX_use_PrivateKey_file SSL_CTX_set_verify	gnutls_certificate_allocate_credentials gnutls_certificate_set_x509_trust_file gnutls_certificate_set_x509_key_file
Session creation per socket (Server)	gcl_create_session gcl_read gcl_write	SSL_new SSL_set_fd SSL_accept SSL_read SSL_write	gnutls_init gnutls_credentials_set gnutls_transport_set_int gnutls_handshake gnutls_record_recv gnutls_record_send
Client context creation	gcl_create_client_context	SSL_CTX_new	
Session creation per socket (Client)	gcl_create_session gcl_read gcl_write	SSL_new SSL_set_fd SSL_connect SSL_read SSL_write	gnutls_init gnutls_server_name_set gnutls_session_set_verify_cert gnutls_transport_set_int gnutls_handshake gnutls_record_recv gnutls_record_send
Modify Cipher Key exchange TLS version	Not exposed (Managed through config)	APIs available	APIs available
Cert Validation CRL/OCSP stapling	Not exposed (Managed through config)	APIs available	APIs available
Non-blocking Read/Write	gcl_create_session_async gcl_read_async gcl_write_async	 SSL_read SSL_write	gnutls_handshake gnutls_record_recv gnutls_record_send
TLS session management	Not exposed	SSL_CTX_set_session_cache_mode SSL_set_session	

Table II: Abstracted CP API and Functionality Mapping

that were created to support the functionality of TLS-based client-server communications. Note that our CP component reduces the number of cryptographic calls made by the actual application as the individual functionality is implemented under the hood (i.e., by the CP as it translates a generic call to a library-specific call or sequence of calls).

In our implementation, the application consumes the CP as a shared library, while the exact configuration instructions are provided by a configuration file that is loaded during initialization. (This emulates future management interface operations.) This is a JSON file that contains the specific library to be loaded, along with information related to cryptographic operations such as key-value pairs that vary depending on the utilized library. Following that, the specified crypto library is dynamically loaded at runtime from the predefined path. Fig. 5 shows the initialization steps, and the relations between the application, the generic cryptography provider, and the underlying libraries.

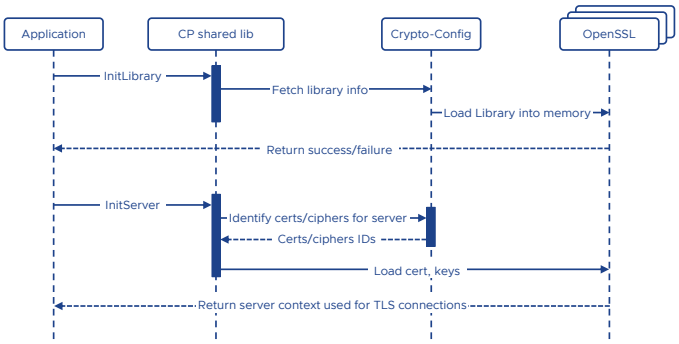

Figure 5: Sequence diagram of the CP component prototype consumption in a Server establishment use-case.

Initiating a server with a cryptographic library typically requires a certificate and an associated private key. In our CP prototype, these two parameters are provided by the CP layer, while the certificate's common name (CN) is provided by the application layer. Using this name, the certificate can be extracted from the certificate repository available at the node. The certificates for a given node are fetched from the policy service, a mechanism we have simplified by using a simple configuration file. The overall intention is that no information is needed from the application as the CP configuration is decoupled entirely from the application code. The CP certificate, for example, can be updated independently. This allows a SecOps admin to switch from one crypto implementation to another using the Management Dashboard.

B. Envoy Proxy Integration

1) Understanding Envoy Proxy: Envoy is an open-source proxy designed for high-performance large-scale service-mesh deployments[33], [22]. In this context, it is a self-contained process that runs as a sidecar alongside a specific service or application server. The communication mesh created by all the sidecars enables applications to exchange messages without the need for network topology information. Envoy is widely used for application layer load-balancers, ingress/egress proxying, HTTP application layer routing, and TLS termination. It is

also popular for its management, service discovery, and load observability functions within edge computing environments.

Envoy is typically deployed in front of application servers that serve clients' requests. The main Envoy building blocks that guide client requests to their respective endpoints are Listeners, Routes, and Clusters. *Listeners* are essentially named network locations (e.g., Unix domain sockets, ports, etc.) where clients will make requests or connect. *Routes* are attached to listeners and use rules to map virtual hosts to *Clusters*, which are groups of logically similar upstream hosts which manage requests in a load-balanced manner. Finally, traffic is diverted to an *Endpoint* (i.e., physical IP address) within a Cluster.

Finally, critical to Envoy's request handling architecture is the notion of *Filters*[3], which are a set of stages that requests go through at different abstraction layers. There are three types of filters that form a hierarchical chain in request handling. First, Listener filters access raw data and manipulate metadata in Layer 4 connections. For example, the TLS inspector filter is able to identify a connection's encryption scheme and extract TLS attributes. Network filters that work on raw data are in the transport layer. For example, the TCP proxy filter can route client connection data to upstream hosts and generate connection statistics. HTTP filters operate at the application layer and are used for all HTTP-related processing, from manipulating HTTP requests/responses to finalizing request route selection to clusters.

2) Prototype Implementation : Request handling in our prototype follows the Envoy functionality. A TCP connection is accepted by an Envoy Listener, which applies filter chain actions. There may be multiple filter chains per Listener to support different functionalities, different IP ranges, and specific server name indication (SNI) groups or ports. Associated with each chain is a transport socket that is in charge of reading and writing into the network buffer and keeping track of all lifecycle events for the TCP connection.

Support for encrypted tunnels is handled by Envoy's `TLS transport socket`, which first decrypts data read from an incoming TCP connection to create a decrypted stream for further processing and then encrypts outgoing data streams before writing them to the appropriate TCP socket. As such, it is the `TLS transport socket` that handles TLS handshakes for Envoy in both directions.

Our initial Crypto Provider prototype within Envoy was statically linked to BoringSSL, but later work introduced a dynamically linked mechanism and multiple versions of OpenSSL intended to offer enterprise operators configuration and compliance options. In both cases, the integration is achieved through the `TLS transport socket` that exposes abstract interfaces to handle incoming and outgoing traffic.

Fig. 6 shows our integration of the CP into Envoy. The TLS transport socket natively consumes a subset of BoringSSL APIs to satisfy a set of configuration requirements for handling the traffic. In our prototype implementation, these API calls were replaced with the equivalent CP calls (see Table II) to accommodate the equivalent functionality. Our initial prototype offers support for OpenSSL, BoringSSL, and the post-quantum crypto-enabled variant of OpenSSL known as OQS OpenSSL [52].

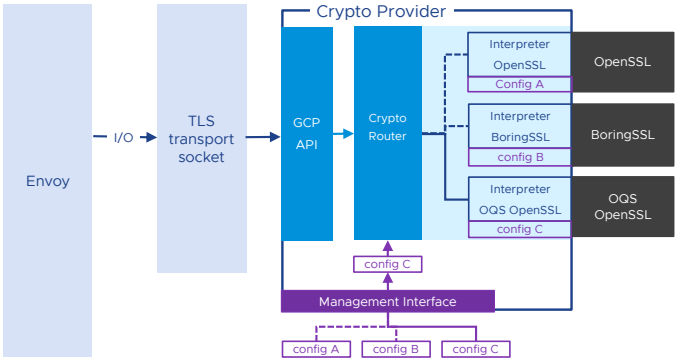

Figure 6: Cryptographic Provider Integration into Envoy

VII. Experimental Evaluation

In this section, we evaluate our proposed architecture for enterprise-level cryptographic agility. First, we measure the overhead introduced by the Crypto Provider using an implementation for Envoy Proxy. Next, we examine the full distributed architecture using a modern application driver within the context of a Kubernetes cluster and a service mesh architecture.

A. Cryptographic Provider Overhead

In this section, we evaluate the performance impact of introducing a cryptographic services component into the end-to-end TLS communication path. Our experimental testbed consists of a remote AWS instance running Ubuntu 18.04 on an x86_64 hardware platform equipped with an Intel Skylake Xeon (4 cores at 2.0 GHz) and 16 GB RAM. We utilize two Envoy variants: (a) the original open-source Envoy 1.20.4 that utilizes BoringSSL, and (b) our CP implementation that made use of Envoy 1.20.4 and two additional cryptographic libraries, namely the OpenSSL 1.1.1m, and the PQC-equipped OQS OpenSSL 1.1.1k[4]. Additionally, we made use of two load testing/benchmarking utilities to measure the overhead of our proposed solution in client-server communication scenarios, namely Siege[5][24], and Nighthawk[6][1].

We used Siege to simulate multiple clients and set up concurrent TLS connections with web server endpoints running on four remote cloud instances. We picked clients that were relatively close to the server to emulate location-based content hosting and services. The clients were uniformly distributed across the four locations. Simultaneous TLS connections were attempted with the server for one minute. The requested web pages had a size of 0.6KB, while for all experiments, we utilized ECDH with the NIST P-256 curve for key exchange, and RSA 2048 for authentication. The aim was to capture the request rate that the server was able to handle, i.e., the number of successful transactions per second. In addition, we captured

[3]https://www.envoyproxy.io/docs/envoy/latest/api-v3/config/filter/filter

[4]https://github.com/open-quantum-safe/openssl

[5]https://www.joedog.org/siege-home/

[6]https://github.com/envoyproxy/nighthawk

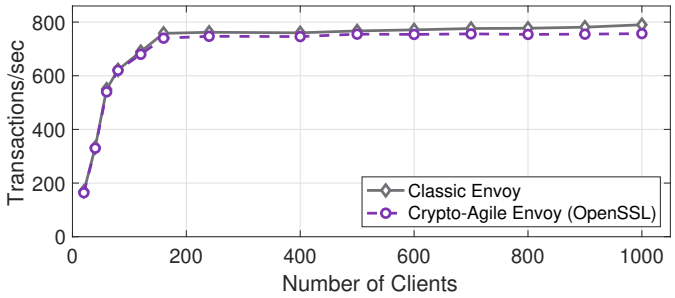

Figure 7: Transaction rate vs Number of consecutive clients

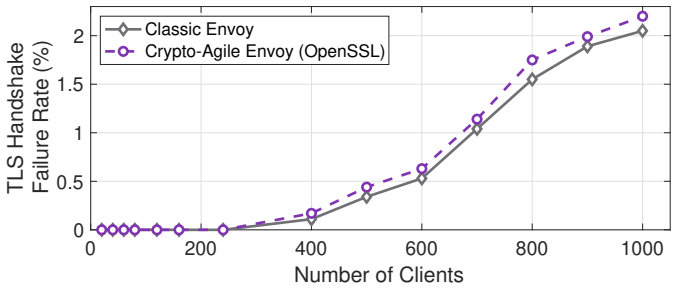

Figure 8: TLS handshake failure rate vs Number of consecutive clients

the server's overall availability by measuring the number of TLS handshake failures during the load-testing period.

Fig. 7 shows the achievable transaction rate for the two different configurations of Envoy as the total number of clients increases from 20 to 1000. In addition, Fig. 8 shows the associated failure rate of TLS handshakes for an unmodified Envoy server versus a crypto-agile server using our CP component. During periods of low load, the performance of both versions is identical. However, as the saturation point of the server is reached, we observe a relatively minor impact to the CP-enabled version as its transaction rate is reduced by $\sim 4\%$ and the TLS handshake failure rate is increased by $\sim 7\%$.

Next, we used Envoy's Nighthawk tool to examine the CP's performance impact on client request rate. We used a client request rate of 1000 requests per second. Fig. 9 shows the cumulative density function of response times for the different configurations of Envoy tested. Again, we observe that responses for the crypto-agile version of Envoy are only marginally slower compared to unmodified Envoy with a difference of ~ 5 msec in the 50^{th} percentile.

Overall, our measurements indicate that the introduction of CP-based request handling has only a moderate impact on performance at the highest load levels. Cryptographic agility using a proxy-based approach appears feasible and without undue tradeoffs for the flexibility and features introduced.

B. Modern Application Integration

In this section, we demonstrate and evaluate our full cryptographic agility framework in the context of a real-world application architecture.

Modern cloud applications rely on the microservice architecture where internal functional components are organized into loosely coupled services that communicate via remote

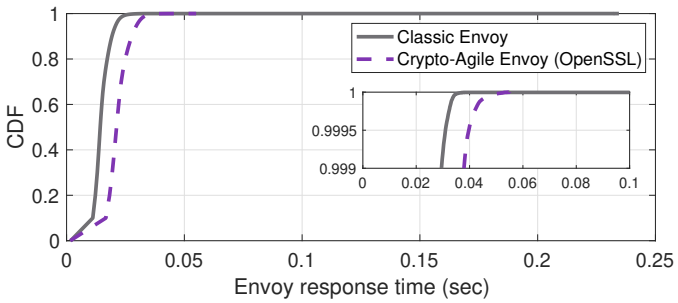

Figure 9: Empirical CDF of Envoy response times

API calls [18], [39]. Microservices are commonly implemented using Linux containers [13] while automatic deployment, management, and scaling is done through Kubernetes [4], [16]. Communication between microservices is handled by proxy-based mediator services known as sidecars. Recently, sidecar connectors have become platforms for enabling service-to-service communications (i.e., a service mesh [15], [40], [41]), offering security, monitoring, failure handling, and other functions.

Our prototype implements these concepts using the ACME Fitness Store application [3], a showcase open-source, distributed application in which every function is a microservice. Fig. 10 shows the architecture, including the communication links between microservices. Service-to-service communications are mTLS [67] encrypted using Tanzu Service Mesh (TSM) [6], [5], which is based on Istio. All services were deployed as part of the same Kubernetes cluster residing on Google Cloud Platform and managed by the Tanzu Mission Control service. Cluster on-boarding into the Tanzu Service Mesh automatically injects Envoy containers into each microservice pod. Note that the attached sidecars included our Cryptographic Provider prototype (see Section VI) with the same Envoy and crypto library versions as mentioned in Subsection VII-A. Envoy sidecar services are given an upper resource limit of 2 CPU cores and 1 GB of RAM.

During the default service mesh onboarding process, the developer normally selects an mTLS configuration for the end-to-end encryption of application communications. However, our prototype modifies Envoy to enable policy-based TLS configuration at any time. To enable such policy orchestration across sidecars, we further modified Tanzu's Management Plane to create a pipeline that transfers crypto policy information from the admin through the TSM controller to each Crypto Provider's management interface. Specifically, a security administrator will create a cryptography policy capturing the low-level configuration details for incoming and outgoing sidecar connections. This policy is applied to a targeted set of microservices through a Crypto Policy Mapping as seen in Section V.

To communicate the Crypto Policy Mapping to CP-enabled Envoy sidecars, we utilize the Kubernetes Custom Resource Definitions[7], a mechanism that allows for retrieval and storage of structured data. The Crypto Policy Mapping is passed to the backend service running in the TSM Management Plane and gets stored as a Custom Resource Definition in Management

[7]https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

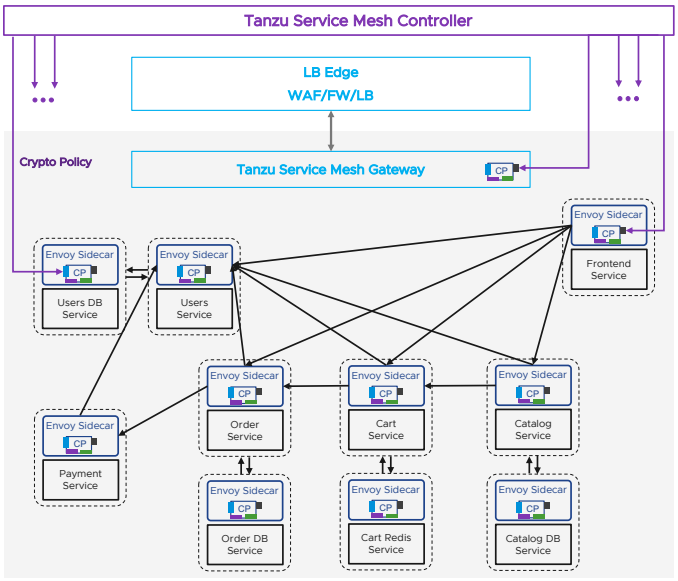

Figure 10: ACME application architecture

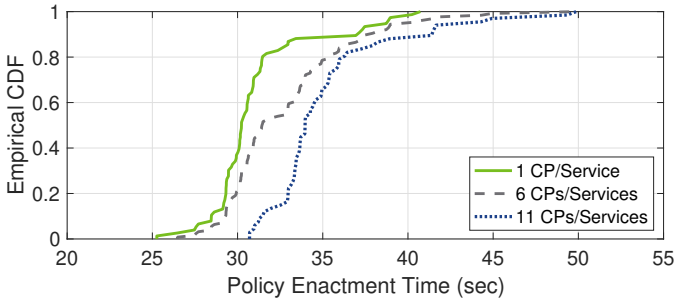

Figure 11: Empirical CDF of new Crypto Policy enactment times

Plane's datastore. A watcher service on the TSM Management Plane replicates the Custom Resource Definition to the Control Plane as soon as a new instance of Policy Mapping is found. In turn, the TSM Controller service reads a new Policy Mapping Custom Resource Definition and translates it to an EnvoyFilter, a mechanism that customizes sidecar proxy configurations[8]. Finally, the new Mapping is passed to the Istio Pilot Agent[9] which translates the EnvoyFilter to an Envoy cryptographic configuration and applies it to the target microservice sidecars.

1) Addressing ELCA Requirements: In this section, we briefly point out how our prototype addresses the ELCA requirements described in Section 3. Enterprise administrators are given configuration control through the Tanzu Service Mesh Controller interface. As described in the previous section, differentiated configuration policies may be created and pushed to any arbitrary microservice within the ACME Fitness Store application through a Policy Mapping. Orchestrated migration may be implemented through the creation of new policies and mappings that are automatically loaded by the TSM Management Plane. Finally, monitoring is handled through the TSM management and control planes which, for now, simply produce detailed logs. In addition, Kubernetes container pods are able to produce monitoring logs, while Envoy offers the capability of Telemetry Filters. Future work will focus on the CP System Services API on each sidecar to support a complete auditing and monitoring structure as imagined in Fig. 3.

[8]https://istio.io/latest/docs/reference/config/networking/envoy-filter/
[9]https://istio.io/latest/docs/reference/commands/pilot-agent/

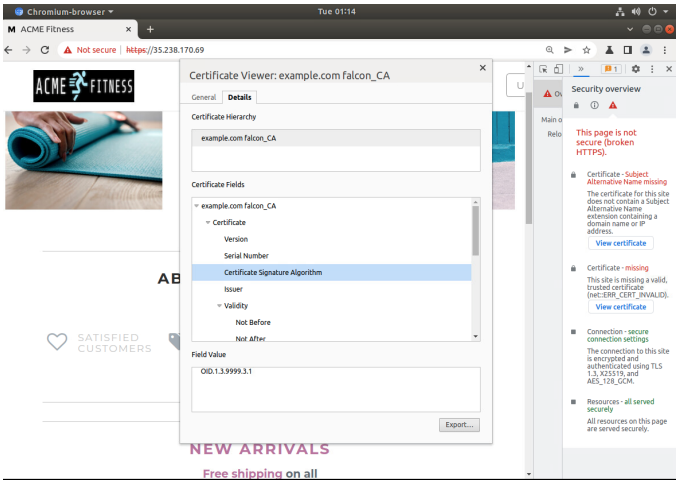

Figure 12: ACME app page authenticated with the PQC Falcon-512 digital signature algorithm

2) Demonstrating Scaled Orchestration: A major feature of our approach, as mentioned above, is the automated deployment of enterprise operator-defined configuration policies. To showcase the practicality of the approach, in this section, we present measurement data on the time taken to push a configuration policy across multiple CPs. More specifically, we measure the delay between the command to update a cryptographic policy and the actual enactment of the new crypto configurations in TLS. Our experiments compare the number of CPs configured within the ACME application, where each CP is associated with a different microservice. In each policy change the OQS OpenSSL 1.1.1k library and the regular OpenSSL 1.1.1m are enforced alternately, and for each case, we performed 100 policy transitions.

Fig. 11 shows the empirical cumulative density function of the crypto policy enactment times for simultaneous policy change of 1, 6 and 11 sidecar CPs. In the latter case, the entire ACME application is configured, including the CP integrated into the TSM Gateway service. As seen, the actual time to make the crypto transition is a function of the number of constituent CPs (microservices). Specifically, when moving from setting policy to one provider (single microservice) to six of them, we observe a 4% increase in the median case of enactment duration and a 5% in the 95^{th} percentile. In addition, when configuring the whole ACME application (i.e., all 11 microservices/CPs), the median is increased by 11%, while in the 95^{th} percentile the policy enactment time is increased by 14.4% in comparison to the single CP case.

The takeaway here is, in contrast to the manual configuration of cryptographic libraries within a microservice infrastructure, our orchestrated agility framework handles configuration scaling with relative ease. Future work can consider even larger scaling factors and potential optimizations which were out of scope for our initial prototype.

3) Case Study: Enterprise PQC Migration Testing: In this section, we illustrate how our proposed cryptographic

agility framework might be used by an enterprise to facilitate migration testing for quantum-safe cryptography. In practice, an infosec team might begin migration by provisioning a realistic staging environment to deploy PQC libraries and investigate impacts across specific applications. They might want to compare the impacts of different PQC algorithm settings and roll back configuration changes at will. Our ELCA framework easily facilitates this through features described in Sections VI and VII.

In the case of our deployed ACME application, a simple policy update was generated, changing the cryptographic library from BoringSSL (i.e., the Envoy default) to the OQS OpenSSL alternative. Specifically, this translates to conventional ECDH with the X25519 curve and the newly selected quantum-resistant Falcon 512. Fig. 12 shows the ACME app frontpage as accessed by the OQS Chromium web browser[10] that is able to support the majority of the new NIST PQC schemes[45]. (Note that the page warnings are due to the fact that the utilized certificate is self-signed.) The certificate's Object Identifier (OID) confirms that it was signed with Falcon 512[11]. To the best of our knowledge, this is the first microservices-based app able to swap cryptographic schemes and support mTLS with pure PQC algorithms.

Finally, we further demonstrate the flexibility in setting specific crypto configurations, especially in the case of the various PQC algorithm combinations available today. To do so, we setup the PQC-equipped OQS CURL application[12] to request data from the ACME frontpage from a client-side by specifying specific crypto algorithm configurations. The client was a local host running Ubuntu 18.04 on the x86_64 architecture, equipped with an Intel i7-8665u that utilized two cores at 1.9 GHz each, and 4 GBs of RAM. The average round trip time between the client and the ACME cluster was measured at 64.2 ms. In all cases, we configured the ACME app through a policy update to only accept incoming connections with specific key exchange/signature algorithm combinations. Apart from conventional crypto schemes, we considered the PQC algorithms selected by NIST [45], namely Dilithium (dil2) [21], Falcon (fal512) [23], and SPHINCS$^+$ (sph128) [11] for authentication, and Kyber [14] for key-exchange. In addition, since the majority of existing PQC migration strategies consider the use of hybrid key exchange [60], [48], where conventional key agreement (i.e., ECDH with p256 curve here) is combined with a PQC KEM algorithm (Kyber) to principally protect against "record now, decrypt later" attacks [47].

Fig. 13 shows the 50th and 95th percentiles of the time to the first byte for the examined cryptographic policies. We observe that our measurements are in agreement with more extensive PQC performance studies found in literature [48], [59], [58], [49], [29]. The main differentiating factor concerning performance lies to the different certificate sizes among the authentication algorithms. For instance, SP generates larger certificates and about $\approx$ 30 times more authentication data during the TLS handshake [57] leading to more roundtrips, and thus a larger TTFB for the client.

[10]https://github.com/open-quantum-safe/oqs-demos/tree/main/chromium

[11]See Codepoints and OIDs section in [69].

[12]https://github.com/open-quantum-safe/oqs-demos/tree/main/curl

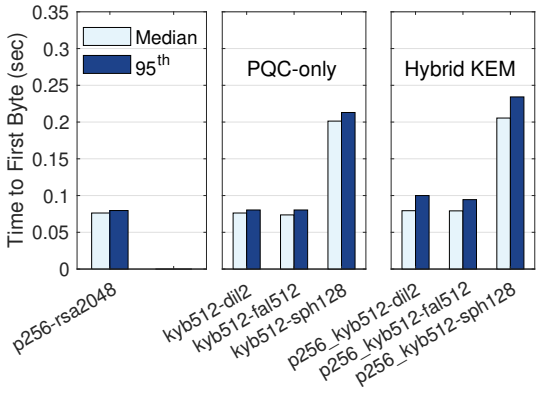

Figure 13: Time to First Byte for different cryptographic policies

The point here is to demonstrate that such experiments are readily enabled by the cryptographic framework described. Both infrastructure-wide impacts and in-depth studies of algorithms and parameters, for example, are now easily managed given an agile framework and policy-driven automation. Enterprises now have a productive framework making the challenge of PQC migration approachable.

VIII. Future Work: Security Protections

The cryptographic agility framework presented in this paper will require additional work to address security protections for the components and mechanisms we describe. These protections represent a significant body of forthcoming future work, and we describe a few preliminary directions below.

The Cryptographic Provider scheme described in Fig. 2 will need robust access control protections that enable exclusive access for security operators interacting with the Management Interface in order to set policy configuration and manage cryptographic libraries. Operators will also need a certificate management framework to address, among other things, authentication between CPs and backend sidecars. We believe that existing technologies can be leveraged to address these issues since we are not the first to face the challenge of access control in distributed proxies (e.g., load balancers, web proxies).

Mechanisms are needed to verify the authenticity and integrity of CP modules implementing configuration management. We believe that confidential computing attestation [55] and associated building blocks could be leveraged as robust verification mechanisms. In such approaches, cryptographic proofs signed by trusted agents or platform components are used to verify the integrity and authenticity of software components, configuration data, and the parties involved.

Another area to be addressed is denial-of-service (DDoS). DoS attacks could be used to block CP availability. Notice that this can happen when the application is not able to reach the provider to perform crypto operations, or when configuration instructions from the management plane cannot reach the CP. Once again, we believe existing approaches can be leveraged since other distributed schemes share the same threat.

IX. Conclusion

We have argued that traditional notions of cryptographic agility, while providing essential building blocks, are hardly

adequate for addressing the challenges facing modern enterprises in transitioning to quantum-safe public key cryptography. Our work highlights what's missing and formulates an expanded vision that we refer to as *enterprise-level cryptographic agility*. We propose an orchestrated, proxy-based architecture that highlights key features and requirements in enabling and managing cryptographic migration at scale. Some of these include policy definition and governance, orchestrated configuration management, monitoring and auditing capabilities, and enterprise administrative control. Using service mesh, we demonstrate how future cryptographic agility frameworks can be constructed to better address the real-world needs of scale.

REFERENCES

[1] "Nighthawk: A l7 (http/https/http2) performance charac- terization tool," https://github.com/envoyproxy/nighthawk, 2022, Web page. Accessed 2022-02-06.

[2] "Openssl 3.0.0 design," https://www.openssl.org/docs/OpenSSL300Design.html, 2022.

[3] "Acme fitness store," https://github.com/vmwarecloudadvocacy/acme_fitness_demo, 2023, Web page. Accessed 2023-02-06.

[4] "Kubernetes: Production-grade container orchestration," https://kubernetes.io/, 2023, Web page. Accessed 2023-02-06.

[5] "Tanzu service mesh materials," https://github.com/Tanzu-Solutions-Engineering/tanzu-service-mesh, 2023, Web page. Accessed 2023-02-06.

[6] "Vmware tanzu service mesh," https://tanzu.vmware.com/service-mesh, 2023, Web page. Accessed 2023-02-06.

[7] D. E. E. 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3174

[8] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[9] M. R. Albrecht, J. P. Degabriele, T. B. Hansen, and K. G. Paterson, "A surfeit of ssh cipher suites," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1480–1491.

[10] H. Andrianakis, D. Bleichenbacher, T. Duong, and e. a. Holenstein, Thomas, "Tink cryptographic library," 2018, https://developers.google.com/tink.

[11] J.-P. Aumasson, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange *et al.*, "SPHINCS+ - Submission to the 2nd round of the NIST post-quantum project," https://sphincs.org/data/sphincs+-round2-specification.pdf, 2019, Specification document (part of the submission package).

[12] E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," National Institute of Standards and Technology, Tech. Rep., 2018.

[13] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE cloud computing*, vol. 1, no. 3, pp. 81–84, 2014.

[14] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367.

[15] L. Calcote, *The enterprise path to service mesh architectures*. O'Reilly Media, Incorporated, 2020.

[16] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[17] Cryptomatic, "Csg case study," 2021, https://www.cryptomathic.com/hubfs/Documents/Case_Studies/Cryptomathic_CSG_Case_Study_-_Barclays.pdf.

[18] A. Detti, L. Funari, and L. Petrucci, "bench: An open-source factory of benchmark microservice applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 968–980, 2023.

[19] A. Dey and S. Weis, "Keyczar: A cryptographic toolkit," *as early as*, 2008, https://www.webencrypt.org/resource/keyczar05b.pdf.

[20] M. Driscoll, "The Illustrated TLS 1.3 Connection: Every byte explained," https://tls13.ulfheim.net, 2018, Web page. Accessed 2019-21-08.

[21] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation," https://pq-crystals.org/dilithium/resources.shtml, 2018, Submission to round 2 of the NIST post-quantum project.

[22] A. Ellis, "Emplacing new tracing: Adding opentelemetry to envoy," Ph.D. dissertation, Tufts University, 2022.

[23] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2018, Specification v1.1.

[24] J. Fulmer, "Siege HTTP regression testing and benchmarking utility," https://www.joedog.org/siege-home/, 2019, Web page. Accessed 2019-02-09.

[25] O. Gasser, R. Holz, and G. Carle, "A deeper understanding of ssh: Results from internet-wide scans," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[26] J. Hohm, A. Heinemann, and A. Wiesmaier, "Towards a maturity model for crypto-agility assessment," *arXiv preprint arXiv:2202.07645*, 2022.

[27] R. Housley, "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms," RFC 7696, Nov. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7696

[28] D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. D. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, "Transitioning organizations to post-quantum cryptography," *Nature*, vol. 605, no. 7909, pp. 237–243, 2022.

[29] P. Kampanakis and D. Sikeridis, "Two post-quantum signature use-cases: Non-issues, challenges and potential solutions," Cryptology ePrint Archive, Report 2019/1276, 2019. https://eprint. iacr. org ..., Tech. Rep., 2019.

[30] P. Kampanakis, D. Stebila, M. Friedl, T. Hansen, and D. Sikeridis, "Post-quantum public key algorithms for the secure shell (ssh) protocol," *Internet Engineering Task Force, Internet-Draftdraft-kampanakis-curdle-pq-ssh-00*, 2020.

[31] C. Kane, B. Lin, S. Chand, S. D. Stoller, and Y. A. Liu, "High-level cryptographic abstractions," in *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*, 2019, pp. 31–43.

[32] M. Klein, "Lyft's envoy: Experiences operating a large service mesh." San Francisco, CA: USENIX Association, Mar. 2017.

[33] ——, "Lyft's envoy: Experiences operating a large service mesh," 2017.

[34] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler *et al.*, "Cognicrypt: Supporting developers in using cryptography," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 931–936.

[35] B. LaMacchia, "The long road ahead to transition to post-quantum cryptography," *Communications of the ACM*, vol. 65, no. 1, pp. 28–30, 2021.

[36] K. Lee, Y. Lee, J. Park, K. Yim, and I. You, "Security issues on the cng cryptography library (cryptography api: Next generation)," in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2013, pp. 709–713.

[37] C. Ma, L. Colon, J. Dera, B. Rashidi, and V. Garg, "Caraf: Crypto agility risk assessment framework," *Journal of Cybersecurity*, vol. 7, no. 1, p. tyab013, 2021.

[38] T. Macaulay and R. Henderson, "Cryptographic agility in practice: emerging use-cases." *Infosec Global*, 2021, https://assets.website-files.com/612fec6a451c71c9308f4b69/614b712e53ce8f7fad0c3c4a_ISG_AgilityUseCases_Whitepaper-FINAL.pdf.

[39] N. Mendonca and C. Aderaldo, "Towards first-class architectural connectors: The case for self-adaptive service meshes," in *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, ser. SBES '21. New York, NY, USA: Association for Computing Machinery,

2021, p. 404–409. [Online]. Available: https://doi.org/10.1145/3474624.3477072

[40] W. Morgan, "The service mesh: What every software engineer needs to know about the world's most over-hyped technology," *URL: https://buoyant. io/servicemesh-manifesto (visited on 2021-05-07)*, 2019.

[41] F. Moyer, "Comprehensive container-based service monitoring with kubernetes and istio," *USENIX Association: Berkeley, CA, USA*, 2018.

[42] M. Müller, W. Toorop, T. Chung, J. Jansen, and R. van Rijswijk-Deij, "The reality of algorithm agility: Studying the dnssec algorithm lifecycle," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 295–308.

[43] E. National Academies of Sciences and Medicine, *Cryptographic Agility and Interoperability: Proceedings of a Workshop*, A. F. Johnson and L. I. Millett, Eds. Washington, DC: The National Academies Press, 2017. [Online]. Available: https://nap.nationalacademies.org/catalog/24636/cryptographic-agility-and-interoperability-proceedings-of-a-workshop

[44] D. B. Nelson, "Crypto-Agility Requirements for Remote Authentication Dial-In User Service (RADIUS)," RFC 6421, Nov. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6421

[45] NIST, "Post-Quantum Cryptography selected algorithms 2022," 2022, https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[46] D. Ott, D. Moreau, and M. Gaur, "Planning for cryptographic readiness in an era of quantum computing advancement." in *ICISSP*, 2022, pp. 491–498.

[47] D. Ott, C. Peikert *et al.*, "Identifying research challenges in post quantum cryptography migration and cryptographic agility," *arXiv preprint arXiv:1909.07353*, 2019.

[48] C. Paquin, D. Stebila, and G. Tamvada, "Benchmarking post-quantum cryptography in tls," in *International Conference on Post-Quantum Cryptography*. Springer, 2020, pp. 72–91.

[49] S. Paul, Y. Kuzovkova, N. Lahr, and R. Niederhagen, "Mixed certificate chains for the transition to post-quantum authentication in tls 1.3," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 727–740.

[50] S. Paul and M. Niethammer, "On the importance of cryptographic agility for industrial automation," *at-Automatisierungstechnik*, vol. 67, no. 5, pp. 402–416, 2019.

[51] K. Petrenko, A. Mashatan, and F. Shirazi, "Assessing the quantum-resistant cryptographic agility of routing and switching it network infrastructure in a large-size financial organization," *Journal of Information Security and Applications*, vol. 46, pp. 151–163, 2019.

[52] O. Project, "OQS OpenSSL," https://github.com/open-quantum-safe/openssl, 2020, Web page. Accessed 2020-02-06.

[53] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet *et al.*, "Evercrypt: A fast, verified, cross-platform cryptographic provider," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 983–1002.

[54] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-dtls13-00, Apr. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-00

[55] M. Schneider, R. J. Masti, S. Shinde, S. Capkun, and R. Perez, "Sok: Hardware-supported trusted execution environments," 2022. [Online]. Available: https://arxiv.org/abs/2205.12742

[56] Senetas, "Certified high-assurance network encryption," 2021, https://www.senetas.com/wp-content/uploads/Certified_High_Assurance_Encryption.pdf.

[57] D. Sikeridis, S. Huntley, D. Ott, and M. Devetsikiotis, "Intermediate certificate suppression in post-quantum tls: an approximate membership querying approach," in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, 2022, pp. 35–42.

[58] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh," in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 149–156.

[59] ——, "Post-Quantum Authentication in TLS 1.3: A Performance Study," in *Network and Distributed Systems Security (NDSS) Symposium 2020 23-26 February 2020, San Diego, CA, USA*. The Internet Society, 2020.

[60] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-hybrid-design-05, Aug. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/05/

[61] B. Sullivan, "Cryptographic agility," *MSDN Magazine*, vol. 24, 2009, https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/august/cryptographic-agility.

[62] ——, "Cryptographic agility: Defending against the sneakers scenario," in *BLACK HAT USA 2010*, 2010.

[63] S. Turner, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms," RFC 6151, Mar. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6151

[64] L. Velvindron, K. Moriarty, and A. Ghedini, "Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2," RFC 9155, Dec. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9155

[65] J. Walden, "Openssl 3.0.0: An exploratory case study," in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 735–737. [Online]. Available: https://doi.org/10.1145/3524842.3528035

[66] X. Wang and Y. Yin, "H. yu. finding collisions in the full sha-1. in proc," in *25th Annual International Cryptology Conference (Crypto'05)*, 2005, https://www.iacr.org/archive/crypto2005/36210017/36210017.pdf.

[67] D. Warburton, "What is mtls?" https://www.f5.com/labs/learning-center/what-is-mtls, 2023, Web page. Accessed 2023-02-06.

[68] A. Wiesmaier, N. Alnahawi, T. Grasmeyer, J. Geißler, A. Zeier, P. Bauspieß, and A. Heinemann, "On pqc migration and crypto-agility," *arXiv preprint arXiv:2106.09599*, 2021.

[69] wolfSSL Manual, "Experimenting with post-quantum cryptography," https://www.wolfssl.com/documentation/manuals/wolfssl/appendix07.html, 2023, Web page. Accessed 2023-02-06.