Software-Defined Cryptography:
A Design Feature of Cryptographic Agility

Jihoon Cho, Changhoon Lee, Eunkyung Kim, Jieun Lee, and Beumjin Cho

Security Research Team, Samsung SDS, Republic of Korea
{jihoon1.cho, changhoon47.lee, ek41.kim, jieun78.lee,
beumjin.cho}@samsung.com

Abstract. Cryptographic agility, or crypto-agility, is a design feature that enables agile updates to new cryptographic algorithms and standards without the need to modify or replace the surrounding infrastructure. This paper examines the prerequisites for crypto-agility and proposes its desired design feature. More specifically, we investigate the design characteristics of widely deployed cybersecurity paradigms, i.e., zero trust, and apply its design feature to crypto-agility, achieving greater visibility and automation in cryptographic management.

1 Introduction

Quantum computers are expected to break conventional public key cryptography once they reach a certain level of performance. As a result, there have been efforts to transition to *post-quantum cryptography* (PQC), which offers resistance against attacks enabled by quantum computers. However, since the invention of public key cryptography due to the pioneering work of Diffie and Hellman [2] in their paper, *"new directions in cryptography"*, in 1976, we have never experienced a full-scale replacement of public key cryptography.

Rose *et al.* [4] explored the complexity and strategic requisites involved in this transition, claiming that many information systems cannot adopt new cryptographic algorithms or standards without extensive and time-consuming modification to their infrastructure. Ott *et al.* [6] pointed out the lack of related research in literature and questioned whether the applied cryptography and systems research communities have adequately understood and provided a framework for attaining so-called crypto-agility.

Cryptographic agility, or *crypto-agility*, is *a design feature* that enables agile updates to new cryptographic algorithms and standards without the need to modify or replace the surrounding infrastructure.[1] This paper examines the prerequisites for crypto-agility and proposes its desired design feature. More specifically, we investigate the design characteristics of widely deployed cybersecurity paradigms, i.e., zero trust, and apply its design feature to crypto-agility, achieving greater visibility and automation in cryptographic management.

---

[1] National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems

2   The Design Feature of Cybersecurity

This section examines the design feature of software-defined network (SDN) and how they have been applied to zero trust architecture (ZTA). It also explores the relationship between zero trust and crypto-agility.

2.1   From SDN to SDx

Software-Defined Networking (SDN) [3] refers to a novel approach for network programmability that allows a greater *visibility* into network behaviors and *automation* of network policy propagation. A key design feature of SDN is the introduction of abstractions between (traditional) forwarding and control planes, and between operational plane and management plane as in Figure 1[2].

```
               o------------------------------o
               |                              |
               |  +-------------+   +---------+ |
               |  | Application |   |  Service | |
               |  +-------------+   +---------+ |
               |          Application Plane    |
               o---------------Y--------------o
                               |
   *---------------------------Y------------------------------*
   |            Network Services Abstraction Layer (NSAL)      |
   *------Y---------------------------------------Y-------*
          |                                       |
          |              Service Interface        |
          |                                       |
   o------Y------------------o     o--------------------Y------o
   |       | Control Plane  |      | Management Plane   |      |
   | +----Y----+   +-----+  |      |   +-----+      +----Y----+ |
   | | Service |   | App |  |      |   | App |      | Service | |
   | +----Y----+   +--Y--+  |      |   +--Y--+      +----Y----+ |
   | |        |       |     |      |     |      |         |    |
   | *----Y-----------Y----* |      | *---Y--------------Y----* |
   | | Control Abstraction | |      | | Management Abstraction | |
   | |   Layer (CAL)       | |      | |     Layer (MAL)        | |
   | *----------Y---------* |      | *----------Y-----------* |
   | |          |          |        | |          |          |
   o------------|----------o        o-----------|------------o
                |                                |
                | CP                             | MP
                | Southbound                     | Southbound
                | Interface                      | Interface
                |                                |
   *------------Y---------------------------------Y----------------*
   |            Device and resource Abstraction Layer (DAL)        |
   *------------Y---------------------------------Y----------------*
   |            |                                 |                |
   |  o-------Y----------o   +-----+   o--------Y----------o       |
   |  | Forwarding Plane |   | App |   | Operational Plane |       |
   |  o------------------o   +-----+   o-------------------o       |
   |                      Network Device                          |
   +--------------------------------------------------------------+
```
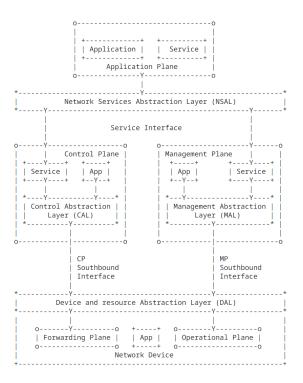
Fig. 1. SDN Layer Architecture [3]

The *control plane* decides how packets should be forwarded by one or more network devices and pushes these decisions down to the network devices for execution. The *forwarding plane* handles packets on the data path based on the

---

instructions received from the control plane. On the other hand, the *management plane* is responsible for monitoring, configuring, and maintaining network devices, including making decisions about the state of network devices, while the *operational plane* is responsible for managing the operational state of the network devices, such as availability of network devices, the number of ports available, and the status of each port.

The core design features of SDN are then broken down into abstracting network resources for network control and management and providing interfaces for network services that can be used by network applications and other services. These abstractions (e.g., interfaces or APIs) reduce the complexity of network operations by using software applications to *dynamically program policies* for individual network devices and thereby control over the network behavior *as a whole.* This architecture has been applied to other areas, such as software-defined storage, software-defined perimeter, and software defined data center, giving rise to what is known as *software-defined everything* (SDx).

2.2   Zero Trust Architecture (ZTA)

Zero trust (ZT) is a term for an evolving set of cybersecurity paradigm that move defenses from a static, network-based perimeter to focusing on users, assets, and resources. Zero trust architecture (ZTA) employs zero trust principles to plan enterprise infrastructure and workflows. With ZTA, rather than implicitly trusting an asset or user based on physical or network location, authentication and authorization occur before a session is established to the resource.

Fig. 2. Core Zero Trust Logical Components [7]

In 2020, the National Institute of Standards and Technology (NIST) published a special publication, *Zero Trust Architecture* [7], outlining the design principles of zero trust. Figure 2 illustrates the overall architecture and the interactions among its components. Policy engine (PE) uses policies as well as input from other resources, such as the subject's security context and threat

intelligence, to decide whether to gran a particular subject access to enterprise resources. Policy administrator (PA) generates session-specific authentication and authorization tokens and manages the communication path between subjects and resources. Finally, policy enforcement point (PEP) manages connections between subjects and resources and communicates with the PA to forward requests or receive policy updates from the PA.

One important requirement of ZTA is to separate (logically or physically) the communication flows used to control and configure the network from the application/service communication flows used to perform the actual work of the organization [7]. That is, to support ZTA, the network should logically separate the *data plane* and the *control plane*, as seen in the design feature of SDN. The data plane handles the communication between subjects and enterprise resources, and subjects should not be able to connect to enterprise resources without accessing the PEP. Access polices can be programmed at the policy decision point in the control plane and enforced automatically by communicating with PEPs in the data plane. This *software-defined* approach allows a central security control tower to gain *visibility* into access activities and *automation* of security policy configuration to tens of thousands of security functions across the enterprise.

2.3   The Correlation between Crypto-Agility and Zero Trust

Since the path to zero trust is an incremental process that may take years to implement, CISA released the Zero Trust Maturity Model (ZTMM) [8] to provide a guideline for the transition to zero trust. More specifically, the ZTMM suggests that gradual advancement, from the starting point *'Traditional'* to *'Initial'*, *'Advanced'* and *'Optimal'*, can be achieved over time across five distinct pillars: identity, devices, networks, application & workloads, and data, as in Figure 3.
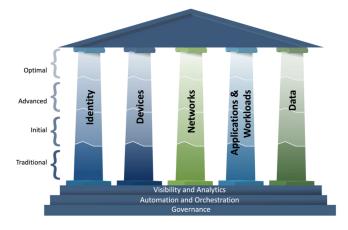

Fig. 3. Zero Trust Maturity Evolution [8]

Each pillar contains details about the cross-cutting capabilities, including visibility and analytics, automation and orchestration, and governance, and the maturity level of each pillar is determined by the level of optimizations of these capabilities. It is interesting to note that the optimal level of *networks* pillar demands integration of best practices for crypto-agility in the ZTMM. Specifically, achieving the highest level of cross-cutting capabilities, such as *visibility* and *automation*, is necessary to support crypto-agility.

3   Towards Software-defined Cryptography

This section proposes the design feature of crypto-agility considering both development and operational environment, and describes its implementation.

3.1   Prerequisite

We have observed that a software-defined approach allows greater levels of visibility and automation in the operational environment. Since transitioning to new cryptographic standards like PQC requires efficient and effective software updates, it is important to decouple static cryptographic configuration from applications. More specifically, applications should not invoke cryptographic APIs directly, otherwise applications should be modified whenever cryptographic algorithms and standards are updated.

This design feature of decoupling configuration from application can be found in the Java Cryptographic Architecture (JCA). JCA provides extensions to accommodate multiple cryptographic libraries, and their usage can be configured by updating the built-in 'java.security' configuration file. It is also possible to enforce to use specific algorithm to use by writing a policy in the file above. However, enterprise IT environments still need a framework for managing cryptography deployed at scale across both development and operational environments as a whole, and greater *visibility* into cryptographic usage and *automation* of cryptographic configuration for hundreds of thousands of applications and services.

3.2   Proposed Design Feature of Crypto-Agility

Enterprise applications are increasingly adopting a standardized architecture consisting of multiple *loosely* coupled components called microservices, often deployed as containers. These applications are supported by an infrastructure that provides application services, such as a service mesh. Applications and application services are typically hosted on container orchestration and resource management platforms. This architecture enables application environment to be defined and managed with source codes as discussed in NIST SP800-204C [1], including application code for business logic, *application services code* for services such as session establishment, network connection, etc., *infrastructure as code* to provision and configure compute, networking, and storage resources,
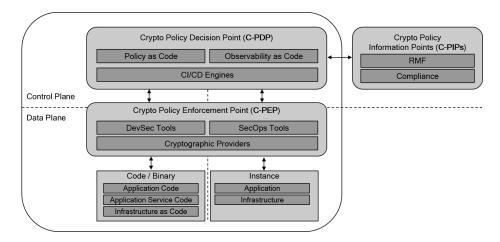
Fig. 4. The Design Feature of Crypto-Agility

policy as code to define runtime policies such as zero trust, and *observability as code* to monitor application runtime state.

This application environment, or application architecture, facilitates effective DevSecOps implementation, where development, deployment, and operation of the application could be agile and automated with primitives such as continuous integration, continuous delivery, and continuous deployment (CI/CD) pipelines. That is, a service mesh allows cryptographic policies, including detecting vulnerable cryptography or mandating particular cryptographic modules or algorithms, to be specified as code and automatically implemented. Furthermore, the architecture of service mesh also separates the control plane and data plane, adhering to the soft-defined approach.

Figure 4 depicts the conceptual framework, or design feature, of crypto-agility, and its basic components and their interaction. Cryptographic policies are defined in *observability-as-code* and *policy-as-code*, based on policies from compliance or risk management framework from the Cryptographic Policy Information Point (C-PIP). The CI/CD engines in the Cryptographic Policy Decision Point (C-PDP) within the control plane invokes policies written as codes and deploy them to DevSec tools and SecOps tools, as well as cryptogrpahic providers, in the Cryptogrpahic Policy Enforcement Point (C-PEP) in the data plane. The cryptographic providers may provide interfaces for observability and/or configurations, e.g., enforcing to use specific cryptographic modules or algorithms. The proposed design feature enables cryptographic policies to be *software-defined* and enforced throughout CI/CD pipelines.

3.3   Implementation

To implement the proposed architectural design, we can construct a system comprising a control plane (C-PDP) and a data plane (C-PEPs), both of which are tightly integrated with the DevSecOps platform.

The initial development phase begins with the DevSecOps platform initiating dependency scanning to identify specific libraries and versions used by applications, resulting in the generation of a manifest, such as a Software Bill Of Materials (SBOM). Alternatively, the development chain may employ a static analysis tool to extract detailed semantics, such as target URLs and communication methods of RPCs [5].

However, since static analysis can lead to false positives, runtime behavioral analysis becomes crucial for acquiring accurate runtime states. To understand the runtime behaviors of applications, the system can execute them within a sandboxed environment to capture actual cryptographic usage and service interactions during the staging phase. This aggregation of data from both static and dynamic analyses provides a comprehensive system view, effectively serving as a cryptography inventory that not only details cryptographic usage but also maps the interactions between services.

In this framework, cryptography inventory is instrumental in assessing security risks through compliance audits, which in turn informs the generation of security policies that specify default libraries for the application based on insights gained during the staging phase.

In a production environment, the operational components of the system can leverage a service mesh. A key element of the service mesh is the sidecar proxy attached to each micerservice that acts as a C-PEP, managing both inbound and outbound traffic. This setup facilitates the collection of various metrics, traffic control, policy evaluation, and data encryption. C-PEP can leverage these features to select appropriate cryptographic libraries based on established security policies. At the same time, it provides the C-PDP with detailed information about cryptographic usage, such as the specific cipher suites employed by the application. By synthesizing metrics from C-PEPs, the C-PDP can ascertain whether workloads are in the desired state to establish a zero-trust environment.

4   Conclusion

The *software-defined* approach is evident in the cloud, where functions such as networking and storage in a data plane are software-defined. This facilitates visibility and automation via a control plane which interacts with data plane through APIs. The paradigm of cybersecurity exemplified by the zero trust architecture is increasingly embracing the software-defined approach, and it is now time for cryptography to follow suit. This shift is particularly crucial as we confront the challenge of migrating to post-quantum cryptography – a path cryptography has not traversed before.

References

1. Ramaswamy Chandramouli. Implementation of devsecops for a microservices-based application with service mesh. *NIST Special Publication*, 800-204C, March 2022.
2. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.
3. Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, Jamal Hadi Salim, David Meyer, and Odysseas Koufopavlou. Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, January 2015.
4. David Joseph, Rafael Misoczki, Marc Manzano, Joe Tricot, Fernando Pinuaga, Olivier Lacombe, Stefan Leichenauer, Jack Hidary, Phil Venables, and Royal Hansen. Transitioning organizations to post-quantum cryptography. *Nature*, 605:237–243, 05 2022.
5. Xing Li, Yan Chen, Zhiqiang Lin, Xiao Wang, and Jim Hao Chen. Automatic policy generation for Inter-Service access control of microservices. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3971–3988. USENIX Association, August 2021.
6. David Ott, Kenny Paterson, and Dennis Moreau. Where is the research on cryptographic transition and agility? *Communications of the ACM*, 66:29–32, March 2023.
7. Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. *NIST Special Publication*, 800-207, August, 2020.
8. temp. Zero trust maturity model 2.0. https://www.cisa.gov/zero-trust-maturity-model, April 2023. Cybersecurity and Infrastructure Security Agency (CISA).