Key Management in Cloud Services:

Understanding Encryption's Desired Outcomes and Limitations

The permanent and official location for Cloud Security Alliance Cloud Key Management research is: https://cloudsecurityalliance.org/research/working-groups/cloud-key-management/

Acknowledgments

Lead Authors:

Doug Egan
Ashish Kurmi
Paul Rich
Michael Roza
Mike Schrock

Contributors:

Marina Bregkou
Subhojit Goswami
Anup Marwaha
Christiane Peters

CSA Global Staff:

Marina Bregkou
Stephen Lumpe (Cover)
AnnMarie Ulskey (Layout)

Table of Contents

1. Introduction

The purpose of this document is to provide guidance for using Key Management Systems (KMS) with cloud services, whether the key management system is native to a cloud platform, external, self-operated, or another cloud service. Recommendations will be given to aid in determining which forms of key management systems are appropriate for different use cases.

Key Management Systems, including hardware security modules and other cryptographic tools, are commonly used to meet compliance and data control requirements in addition to providing security benefits. Examples of reference standards that are widely used to guide and constrain KMS designs include NIST (the United States National Institute of Standards and Technologies, which authors the Federal Information Processing Standards, Common Criteria, and PCI DSS (Payment Card Industry Data Security Standard). This document references some of these standards.

It is important to always remain cognizant that a KMS is a means to an end, not an end in itself. The capabilities enabled by KMS are tools that must serve business needs. Before KMS-based capabilities are built it is crucial to clarify business requirements. Some business requirements cannot be addressed through KMS and encryption; in addition, misconceptions about the capabilities of encryption persist. Furthermore, regulatory requirements for key management and encryption are commonly unclear, undefined, or poorly understood. This lack of clarity and understanding has many causes: regulators who have a poor understanding of the capabilities of encryption and cryptographic systems; assumptions on the part of security and compliance staff; and marketing and press coverage that misrepresent cryptographic capabilities or misinterpret regulatory or legislative intent or language. Cryptography is a relatively specialized field that generally requires topical education and hands-on experience, both of which are often lacking in the regulatory and journalism staff that writes on the topic. Frequently it is a misunderstanding of what encryption can accomplish rather than errors in understanding technical details that leads to incorrect conclusions and misplaced recommendations. This document will provide clarity regarding the uses of key management systems with SaaS, PaaS, and IaaS.

A cornerstone document for developing CSA EKM guidance is NISTIR 7956 (Cryptographic Key Management Issues & Challenges in Cloud Services. Published in September of 2013, NISTIR addresses the complexities of managing cryptographic keys in cloud environments compared to enterprise IT environments. However, NISTIR 7956 does not address the issue [highlighted in the previous paragraph] of the necessity to first understand the business requirements and determine if encryption and KMS are even appropriate technologies. Encryption, as a technology, is used for secrecy/privacy in the transmission and storage of data. However, encryption is not the only technology used for secrecy/privacy, and there are many cases where the use of encryption can be pointless, costly, and provide a false sense of security. An example of a pointless use of encryption with a false sense of secrecy is to leverage symmetric key encryption to protect an email sent to all employees that reveals the intent to acquire a rival business and expecting that the encryption will prevent a leak to the press or the rival. Encryption can neither enforce the privacy desired by the organization, nor can it help the organization determine who leaks the information.

NISTIR 7956 asserts that “difference in ownership” is an additional complexity but does not state why. In fact, NISTIR 7956 does not explain why ownership of an encryption key has a material impact on any outcome at all. Ownership of keys is not an end in itself; it is a means to an end. The desired goal of ownership of a key could be the assertion of a compliance control, or it could be an assertion of privacy, or both. However, ownership of a key does not directly result in any privacy outcome. Furthermore, the assertion of a compliance control may or may not achieve a privacy goal.[1] Therefore, it is imperative that the objective [of key ownership] is clear so that an assessment can be made regarding the effectiveness of the means to achieve that end.

Ownership of encryption keys is nearly always indicative of misplaced interest. To illustrate this, let us consider a physical storage unit under lease and used for some type of valuable goods such as legal documentation, jewelry, or collectible goods such as wine or fine art. In this case the lessee owns the keys for the leased space and [let us assume] signed a contract granting the lessee sole rights to access the leased container. Let us also assume that the space is locked using a traditional padlock. It is not uncommon for the lessee to choose to provide a padlock key to the lessor so that shipments can be received and placed in the leased container without the presence of the lessee. In this scenario all of the keys for the padlock can be said to be "owned" by the lessee. However, it is not ownership of the keys that determines access to the property stored in the container, it is possession of a key that will open the container. Thus, it is not ownership, but possession, that matters. Should a law enforcement agency approach the lessor with a search warrant that compels the release of the key, the lessor has the ability to open the container and allow the search; likewise, should the lessor suspect that the lessee is storing explosive chemicals, the lessor has a key that can be used to inspect the contents without the permission of the lessee, even if such a search is not in the contract. This is the nature of keys: it is possession that matters, not the ownership.

Regarding NISTIR 7956, with respect to cloud services, the same "ownership versus possession" logic applies. An organization that shares keys with a cloud service provider accepts the same possibility that the cloud service provider can access the organization's data. In the case of SaaS, the organization normally expects that the cloud service provider will access the customer's data. This is because SaaS is largely useless or of very little value without access to the customer's data. For instance, in the case of artificial intelligence (AI) or machine learning (ML) algorithms offered as PaaS or SaaS, without access to the customer's data there is nothing for the AI or ML algorithms to process. The earliest SaaS capability was search - the ability for Yahoo or Lycos to index content so that people could search for and find that content on the World Wide Web. If a website had encrypted its content and the search provider did not have a key to decrypt the content, no indexing could occur.

Furthermore, even ownership and possession do not encompass all possible scenarios of data access. Consider the case in which law enforcement presents an order to a possessor of a key. Even if the possessor is not the owner of the key, the law may explicitly instruct the possessor to use the

[1] If only data storage and retrieval [from a cloud service] is desired, then sole ownership, possession and use of encryption keys may be used to achieve a privacy goal. That is, the owner of the key is also the sole entity that possesses and can use the key. The cloud service provider is never in possession of the key and therefore can provide only storage and retrieval of the data in its encrypted form. This renders the data unavailable to the cloud service provider and thus blocks third party access via legal demand to the service provider. If, however, data processing is desired - as is the case with SaaS - then data must be rendered unencrypted. When a SaaS provider has access to unencrypted data, ownership of keys is irrelevant with respect to the privacy of the data.

key in the service of a legal demand, while prohibiting the possessor from informing the owner of the key. In this circumstance, law enforcement is never directly in possession of a key but it is still able to access the contents through the compelled use of the key by another entity.

Rather than ownership of encryption keys, web security and privacy standards should focus on possession and use of keys. Ownership is technically irrelevant in terms of access to data, though it certainly may implicitly or explicitly provide legal rights or obligations. With regard to encryption used to provide data privacy and security in cloud services, we must examine ownership, possession, and use of encryption keys.

The above explanation of the use of cryptography and the dynamics of the ownership, use, and possession of keys illustrates the need for the reader of this guide to begin any approach to leveraging Key Management Systems with cloud environments by focusing first on the desired outcomes when using cryptography before drawing conclusions or making assumptions about KMS design and implementation choices. Any Key Management System is a tool - a means to an end - and so it is critical to understand the desired outcomes as well as the limitations of encryption to protect data.

1.1 Key Management History

Key management has a very long history, with some evidence that management of encryption keys originated in ancient times as far back as the 4th or 5th century BCE. For the purposes of this document only the modern era of cloud computing will be discussed. The earliest written mentions of "cloud computing" appear in the mid-1990s, and the emergence of popularized cloud services could be reasonably dated to 2006 when the Amazon "Elastic Compute" service launched. In 2010 the Key Management Interoperability Protocol was published, establishing a beachhead for what would become an increasingly multi-instance, heterogenous, and interconnected technology world for the enterprise. Many technology standards had Public Key Infrastructure (PKI) and key management foundations, though these technologies have not yet adapted to the evolution of cloud computing. It was not until eight years after Amazon's EC2 service launch that Amazon would announce the launch of Amazon Key Management Service.[2] Prior to that launch, customers of major cloud service providers effectively had no managed service option for key management and were using legacy (built for on-premise deployment) key management products or choosing encryption options available on a cloud-service-by-service basis. For example, Amazon initially shipped bespoke encryption solutions for S3 and Redshift[3], Relational Database Service (RDS) for Oracle, and RDS for Microsoft SQL Server. These per-service encryption options offered no real key management services and used only cloud keys. The legacy key management applications typically required additional software development to integrate with cloud applications and services because they had been architected with only on-premise usage in mind.

Beginning with the launch of Amazon KMS in 2013, customers could choose a cloud-native solution and have control over the generation and use of keys for both Amazon services and the customer's own applications, both on-premises and cloud. Later, the option of integrating keys generated and/or hosted on-premises would become available. Around this same time Microsoft released its own basic

[2] https://aws.amazon.com/blogs/aws/new-key-management-service/

[3] https://aws.amazon.com/redshift/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc

key management [cloud] service called Azure Key Vault[4], offering many of the same capabilities as Amazon's option/system. Soon these cloud services would expand their capabilities beyond encryption keys and add passwords, connection strings, and other secrets. They also shipped software development kits (SDKs) for integration into any application, cloud or not. This evolution redefined cloud key management services as secrets management services, with key management features such as attestation, version control, and richer delegation. In 2017 Google shipped its own key management service[5] with capabilities similar to those of Amazon and Microsoft.

In the past few years, the major cloud providers have enhanced their key management services, adding new key sizes, administrative functions, dedicated tenant capabilities, and co-location options. These changes reflect a desire to meet the requirements of more sophisticated customers such as financial services, defense, pharmaceuticals, and government entities. Cloud providers have also commoditized the provisioning of encryption keys for enabling transport Layer Security (TLS) for sites hosted on the cloud provider's platforms, taking market share from long-established incumbent providers such as DigiCert, GlobalSign, and Entrust. This may be a precursor to the major cloud providers taking a larger role in key management, particularly as more devices become integrated with cloud services and use keys for identity and authentication.

1.2 Goals

This document offers recommendations for using KMS in conjunction with cloud services to aid (users/developers, designers, architect, administrators, operators, developers and auditors) in meeting security and compliance requirements. Additional recommendations are provided for cloud service providers offering key management functionality to customers.

1.3 Target Audience

The target audience includes cloud providers and cloud customers, CISOs, regulators, developers, architects, security and compliance staff who are concerned with key security and the secure development, operation and use of key management services in cloud services.

2. KMS Conceptual Architecture

2.1 Definitions

A Cryptographic Module is "a set of hardware, software, firmware, or some combination thereof that implements cryptographic logic or processes, including cryptographic algorithms, and is contained within the module's 'cryptographic boundary,' which is an explicitly defined contiguous perimeter that establishes the physical bounds of the module." [RFC4949], [NIST FIPS 140-2]. Examples of hardware-based cryptographic modules are Trusted Platform Modules (TPMs), smartcards, hardware security modules (HSM); software-based cryptographic modules include any type of cryptographic libraries, crypto software etc.

[4] https://azure.microsoft.com/en-us/services/key-vault/

[5] https://cloud.google.com/security-key-management

The FIPS (Federal Information Processing Standard) 140-2 and ISO/IEC 15408 Common Criteria for Information Technology Security Evaluation (referred to as Common Criteria or CC) are international standards for certifying protection levels of cryptographic modules.

Regulators in various industries (particularly the financial sector) demand that only FIPS 140-2 certified cryptographic modules are used to protect critical data and processes. The protection level depends on the usage of the cryptographic module.

Particularly, regulators require usage of FIPS 140-2 Level 2 or higher certified cryptographic modules for protecting business-critical cryptographic keys. These modules often come in the form of so-called Hardware Security Modules (HSMs) used to manage, generate and securely store cryptographic keys.

While cryptographic modules automate cryptographic algorithms, they do not necessarily provide key lifecycle management including key distribution, key registration and de-registration, key backup and restore, key renewal, key revocation, preservation of key management, etc.
It is important to note that auditors will not audit only the used cryptographic algorithms and key storage, but also how cryptographic keys are used.

A Key Management System is a system that is built around a cryptographic module and leverages the cryptographic functionality and the key lifecycle management to connect applications and services. The functionality of a key management system can also be extended to manage secrets and certificates. NIST [NIST SP [800-57] Part1] defines the KMS as a system for the management of cryptographic keys and their metadata (e.g. generation, distribution, storage, backup, archive, recovery, use, revocation, and destruction). An automated key management system may be used to oversee, automate, and secure the key management process.

2.1.1 The Meaning of BYOK

The acronym "BYOK" (Bring Your Own Key) as well as related constructs "HYOK" (Hold Your Own Key) and "CYOK" (Control Your Own Key) came into use around 2013 as marketing terms used by technology companies. The meaning of these acronyms differed not only among companies, but even within a single company offering many cloud services (one service might use the term BYOK to mean bringing a key from a customer-owned KMS, while another service from the same company might mean allowing customer ownership and management of a key from the cloud provider's own KMS). This lack of standardization has, of course, led to a great deal of confusion, and that confusion persists today. Many more technology companies have brought products and solutions to market using these acronyms, further eroding any common meaning for these terms. This is not likely to change since there is no technical reason why a standards body such as the IETF would publish a reference standard. There is no mention of "BYOK" or "Bring Your Own Key" in NIST documentation such as NIST SP 800-57 and in that document the definition of "Owner" for a private key is any entity "authorized to use the private key" and, for a symmetric key is "any entity that is authorized to use the key." This very definition of owner negates the "O" in most, if not all, BYOK schemes on the market for SaaS, since the private key or symmetric key ends up being used by the cloud service provider. An organization need only ask "What entities can use this key?" to determine ownership. Ownership is not limited to the originator of the key when that key is then configured to be used by another party.

What is important to realize is that BYOK is not one particular implementation — the term/acronym does not clearly indicate any technical design or outcome, nor the resulting legal risk. It is just a marketing term. Therefore, technical professionals should not use the term and should actively discourage its use among the professional community as well as the press.

This document conveys technical information and recommendations, and terms such as BYOK and HYOK are not technical terms; they are marketing terms. Since what those terms are intended to do is express some desired outcome — normally, actual control over access to data or the appearance of control over access to data — what technical professionals must do is describe in factual, technical terms what outcomes can or cannot be achieved through the use of a particular KMS model or pattern.

2.2 Common Cloud KMS Patterns

As with most modern computing technologies that intersect with cloud computing, there are many possible architectural combinations of solutions to cloud computing and key management. The primary questions that affect how organizations approach the architectural choice include:

> 1. What KMS systems are already in use by the organization?
> 2. What are the desired functional, operational, and business outcomes for use of the cloud service?

The answer to the first question is less pressing than the answer to the second question. Indeed, it is possible that the answer to the first question may be irrelevant, as some cloud services do not allow or support integration with an existing KMS. The first question is better left to address once the organization has clearly documented the answer to the second question. The second question may elicit a long list of desired outcomes, with considerations of cost, complexity, stability, service level agreement, privacy, transparency, regulatory compliance, and speed of adoption. These outcomes may further be driven by considerations that include interoperability, configurability, conformance to standards, and the KMS-related skills available to the organization. The various architectural patterns exhibit strengths and weaknesses relative to these considerations.

It is important to note that in the architectural patterns that follow, any cloud shown could be either public or private. What is illustrated are the patterns most commonly seen, and the architectures can be generalized for implementations that mix multiple patterns or leverage inter-pattern integration (e.g. a multi-cloud KMS).

The past decade has seen a good deal of evolution in the architectures of cloud services and the Key Management Systems that underpin, or can be integrated with, many cloud services. KMS features traditionally conceived of as enterprise data center hardware owned by a single organization have been unleashed with cloud scale and performance. This evolution will attract many organizations to cloud KMS options due to the historical challenges with legacy enterprise KMS implementations: cost, complexity, brittleness, and a tendency to reach "senior citizen" status with little appetite for revitalization.

Because the industry is in a period of rapid change, it is recommended that organizations consider the ability to pivot, migrate, or adopt new patterns of cloud KMS. In the evaluation of a vendor's specific implementation of any cloud KMS pattern, the ability to pivot to a different pattern, or to merge patterns, should be determined. This flexibility will enable business agility and reduce friction in mergers, acquisitions, and divestitures.

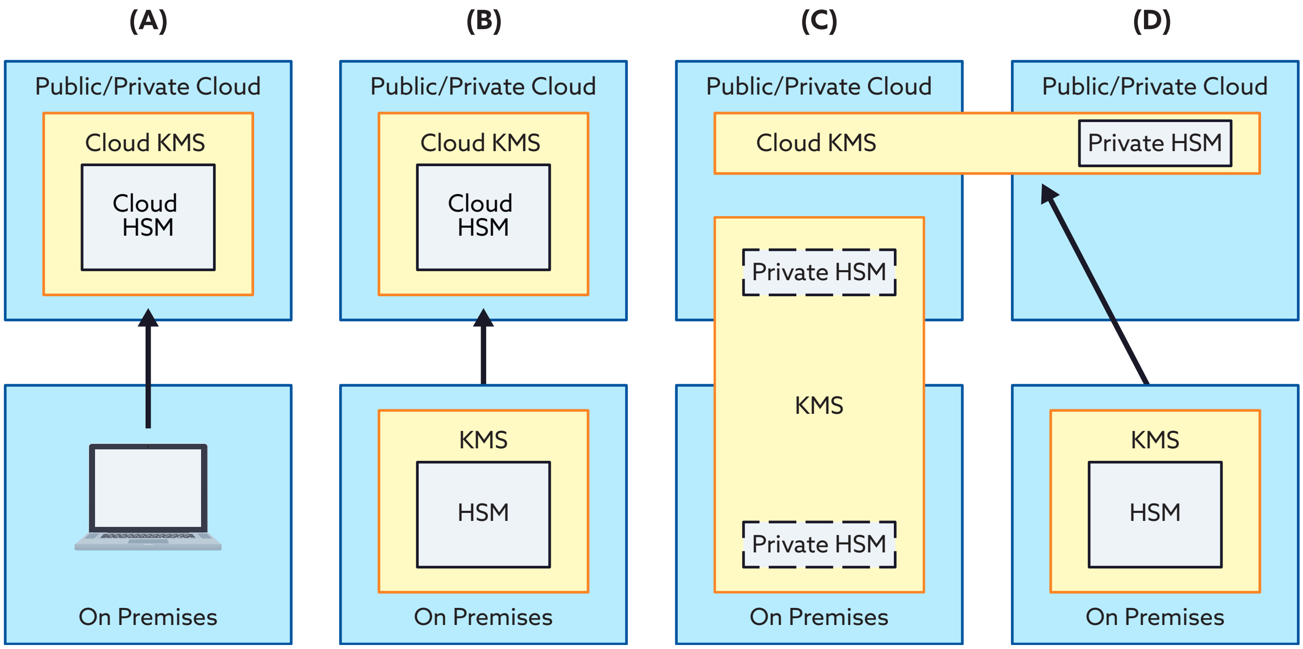

Figure 1: Cloud Services and Key Management System Patterns

Figure 1 illustrates the following cloud KMS patterns:

> 1. (A) is a cloud service that leverages a KMS (including HSM) within that same cloud;
> 2. (B) expands pattern (A) to allow for the import of key material from an external KMS;
> 3. (C) is a cloud KMS with a dedicated (private) HSM that is under control of the owning organization, but is physically hosted within the cloud provider's data center(s);
> 4. (D) illustrates an on-premises KMS that is used for multi-cloud KMS integration/management that can be hosted either on premise or in the cloud and is linked to an on-premise cryptographic module such as an HSM or crypto card.

The above figure is not solely intended to represent four architectural patterns. It illustrates that it is possible to compose solutions where the fundamental cloud key management components can be confined to a single cloud or extended across multiple cloud services or providers, and even inclusive of systems that are resident within an organization's traditional data center environment. In the following sections we examine the four primary cloud key management patterns that have emerged over the past decade.

A list of existing implementations of cloud KMS systems is found in Appendix B.

2.2.1 Cloud Native Key Management System

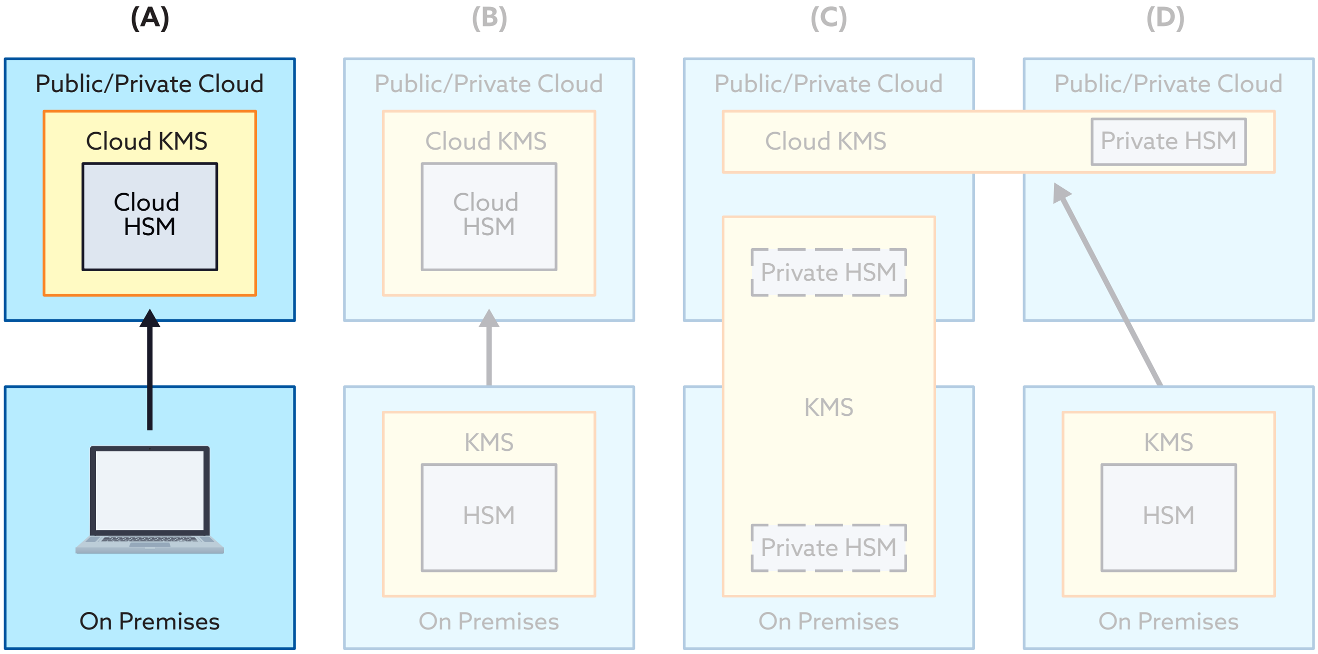

Figure 2: Cloud Native Pattern

Definition

The defining characteristics of this pattern are that the KMS is built and owned by the same provider that delivers the cloud service the customer consumes, and all components of the KMS are in the cloud. Figure 1 illustrates this as pattern (A). The KMS may be an integral part of the service being consumed, or it may be a KMS service that is provided separately from within the same cloud service provider. This model uses cryptographic algorithms and modules chosen and operated by the cloud provider.

Attributes

- All hardware and software components are under the control of the provider, and the KMS is typically a "black box" to the customer, other than public documentation
- Typically, has limited or no separation of duties between the cloud service and the KMS features
- Least disruptive model for SLA enforcement
- Cloud environment has, or can have, access to customer plaintext data
- This pattern may have low extensibility of the KMS functions outside of the provider, though often provides easy extension to other services within the same provider
- Pricing is commonly built into the cloud service the customer purchases, and is therefore hidden, difficult to discern, or assumed to be zero
- Cost is driven by technical expertise necessary to administer and operate the KMS functions
- Fastest implementation time
- Typically, high performance
- Typically, high scalability

- Latency insensitive
- May support evolution to other cloud KMS patterns
- FIPS support is limited by the cloud KMS
- Typically, cannot support customer key ceremony requirements

Challenges

This model implies design differences between cloud providers and will therefore require unique technical knowledge and skills for each cloud provider used. Additionally, often the administrative model does not distinguish between the application/service administrator and the KMS administrator and so has poor separation of duties. However, the simplicity of implementation and feature set available to the consumer may mitigate the need to develop much technical domain knowledge and administrative burden.

Typically, details about specific algorithms and protocols are not visible to the consumer, though broad technical parameters may be stated in cloud provider documentation or auditor materials. Because the consumer typically has little or no control over KMS configuration or actions, this model may not meet all compliance requirements such as key roll (or rotation) period or revocation and recovery. Lastly, this pattern does not ensure the total privacy of customer data from the cloud provider, since the CSP (Cloud Service Provider) has the keys.

Recommendations for Use

- When the consumer has no clear reasons to use other models
- When the consumer's cost is driven by factors other than staffing for KMS expertise, such as storage and transactions
- When organizations lack KMS expertise and/or operational support capability
- When speed of implementation is a critical factor

2.2.2 External Key Origination

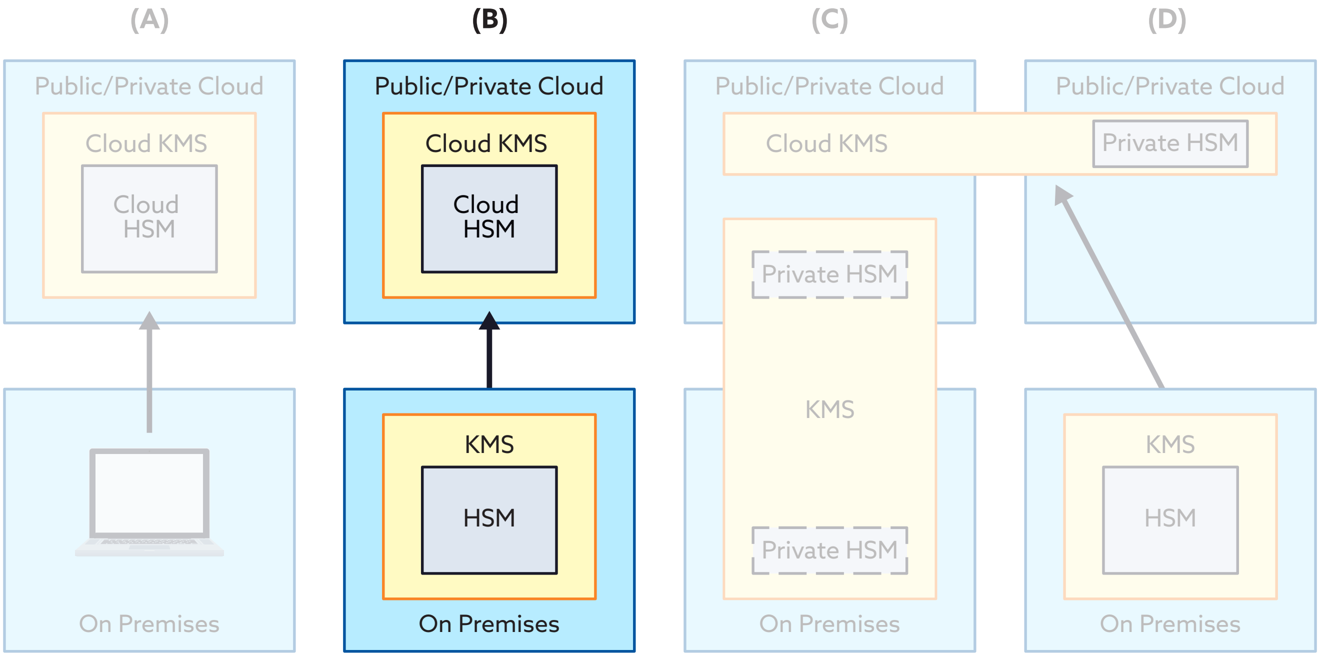

Figure 3: External Key Origination

Definition

The *External Key Origination* pattern builds upon the Cloud Native model, allowing for key generation ceremonies[6] that originate with an external KMS. This architecture is typically chosen in response to a compliance requirement that targets key "ownership" in some way, often using language that is vague and targets implementation details rather than business and/or technical outcomes. In this pattern there is no integration of the external KMS with the cloud KMS, so the external KMS acts only as a key generation facility. In some emerging use cases the cloud Service Provider can be forced to make calls outbound to the external key manager, though this has been generally frowned upon by CSPs due to the potential implications for incident handling as well as the latency impact to overall performance.

The unfortunate acronyms BYOK and HYOK can both reflect the expectation that this model is being used, though as we see in section 2.1.1 neither of these terms have any standard technical definition and so there is no definitive way to assess any pattern against those terms. The main feature of this pattern is that the customer is in control of the software and hardware used in one or more key generation processes. Typically, the pattern supports external generation of only root keys, and all other keys are generated within the cloud KMS.

[6] See https://en.wikipedia.org/wiki/Key_ceremony

Attributes

- Cloud hardware and software components are under the control of the provider; the customer controls a KMS external to the cloud and typically uses it for root key generation and import to the cloud KMS
- Provides clear separation of duties for root key generation
- Minimal potential to impact cloud SLA, though the external key import process may impact recovery operations
- Cloud environment has, or can have, access to customer plaintext data
- This pattern may have low extensibility of the KMS functions outside of the provider, though often provides easy extension to other services within the same provider
- Pricing is typically driven by the licensing of an external-key import feature
- Cost is typically driven by pricing of licensing for key import functionality, the number of keys imported, and managing the external KMS from which keys are imported
- Implementation time is typically driven by acquisition and configuration of the external KMS that will be used for key generation and import
- Performance is typically the same as the Native KMS pattern
- Scalability is typically the same as the Native KMS pattern
- Latency sensitivity is typically the same as the Native KMS pattern
- Typically, this pattern provides greater support for evolution to other patterns than the Native KMS pattern
- Can provide increased FIPS compliance capabilities for keys generated by the external KMS, though these are typically limited to key encryption keys
- Typically, can satisfy customer key generation ceremony requirements for root keys

Challenges

All of the challenges of the Cloud Native pattern apply. Additionally, the cloud provider may support only a limited set of hardware manufacturers for external key generation. Accordingly, it may be necessary to acquire compatible KMS hardware/software, and this may necessitate skills acquisition, potentially lengthening the time to implementation.

It should be noted that keys generated using the external KMS may not be useful as backup copies. Providers may add critical metadata to keys that the consumer imported, rendering the original keys useless for restore operations.[7]

Recommendations for Use

- When the consumer has a requirement for a particular key ceremony that cannot be met with the Cloud Native pattern
- When a consumer wants additional External Key Management key custody via supported emerging models

[7] Both Salesforce and Microsoft have implementations or features with this characteristic. See https://developer.salesforce.com/docs/atlas.en-us.securityImplGuide.meta/securityImplGuide/security_pe_byok_setup.htm and https://docs.microsoft.com/en-us/microsoft-365/compliance/customer-key-overview?view=o365-worldwide

2.2.3 Cloud Service Using External Key Management System

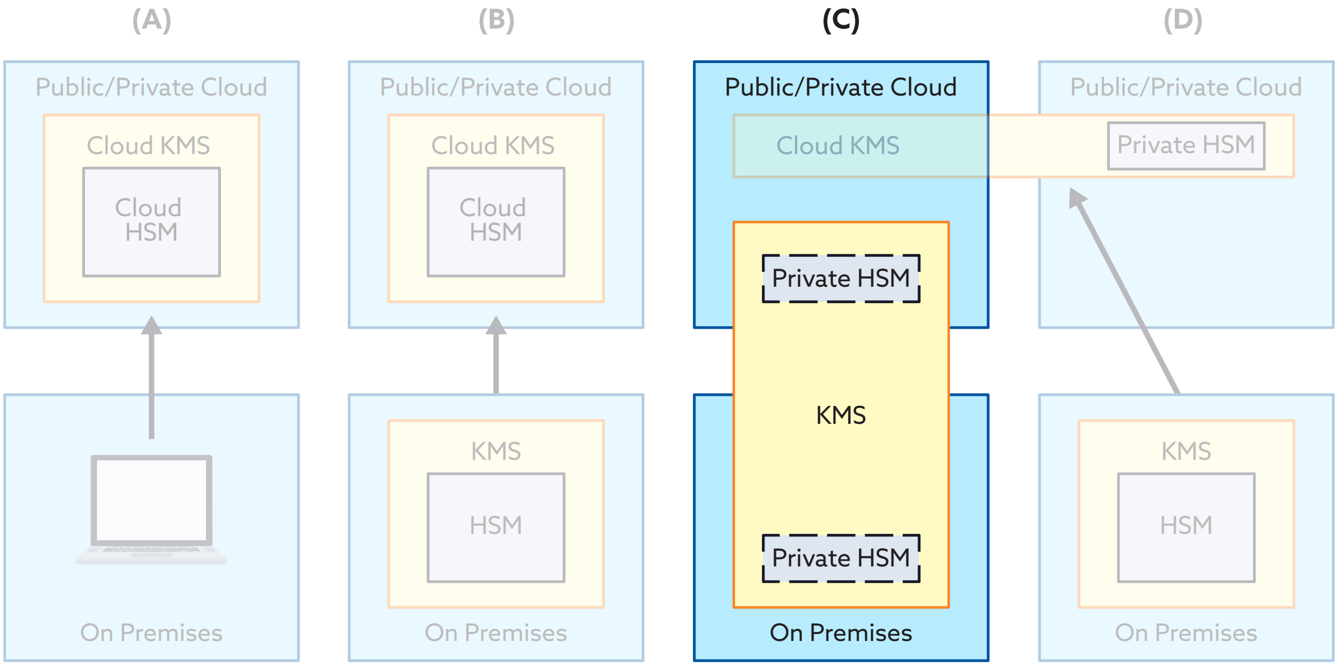

Figure 4: Cloud Service Using External Key Management System

Definition

The External Key Management System is the use of a cloud service where the KMS is hosted entirely external to the cloud service, either wholly on the customer's premises, wholly hosted by a third party chosen by the customer, or a combination of the two. Hardware may be the property of the consumer or the cloud provider but is provisioned solely for use by the consumer. A cloud provider may support a dedicated cloud HSM service offering or co-location model where the consumer hardware is hosted in the same facility where the cloud provider has a presence. The consumer manages all aspects of the KMS and typically agrees to service-level agreement considerations in the event the KMS is the source of a service incident. To achieve total data privacy from the cloud provider, no key wrapping or unwrapping can be performed by the CSP.

Attributes

- High degree of customer control and configuration. Algorithms, protocols, and key lengths are known and configurable by the consumer
- Can support complete separation of duties for KMS and cloud service activities, as well as within the KMS
- May support unification of KMS into single point of management
- Can ensure that key material is never shared with a cloud service provider, ensuring no consumer plain-text data is exposed to the provider
- This pattern provides the greatest portability since most or all KMS functions are implemented outside the cloud service

- Higher cost due to dedicated hardware and choice of single-tenant KMS model; staffing is a factor; CSP encryption features are typically a minor cost driver
- Cost is most often driven by the choice of external KMS and administration of all components
- This pattern can incur a long implementation time; organizations that have already established this pattern may be able to significantly reduce implementation delays for new cloud services
- Performance can be a limiting factor; use of a public or private cloud KMS may mitigate performance limitations
- Scalability can be a limiting factor; use of a public or private cloud KMS as the external KMS can mitigate scalability limitations
- Cloud services may incur latency impact when utilizing this pattern
- Typically, this pattern provides complete freedom to add and remove patterns
- Can provide customer-driven FIPS compliance for all keys
- Can satisfy customer key generation ceremony requirements

Challenges

This model is not as common as other patterns and so may not be available for the service chosen.[8] Use of this model has service-level agreement ramifications and the consumer may be unable to get compensated in the event of service failures. Because the general basis for wanting to use this model is to maintain complete secrecy of customer plaintext, this model is generally incompatible with SaaS or any service where the cloud provider's systems need to process customer data in plaintext. The consumer must have adequate skills to fully manage the KMS.

Recommendations for Use

- When a cloud service provider has no native KMS
- When the consumer desires a single KMS that is used both on-premises and with cloud services
- Can support a requirement to keep plaintext entirely private from the cloud provider
- Can provide the ability to achieve a higher FIPS certification level

[8] In June 2020 the WG made a comparison of Google, Salesforce, Oracle and Dropbox and found that they do not support this model. It appears that only Microsoft and Amazon have a "bring-your-own-HSM" or "dedicated HSM" and it may not be available for all services.

2.2.4 Multi-Cloud Key Management Systems (MCKMS)

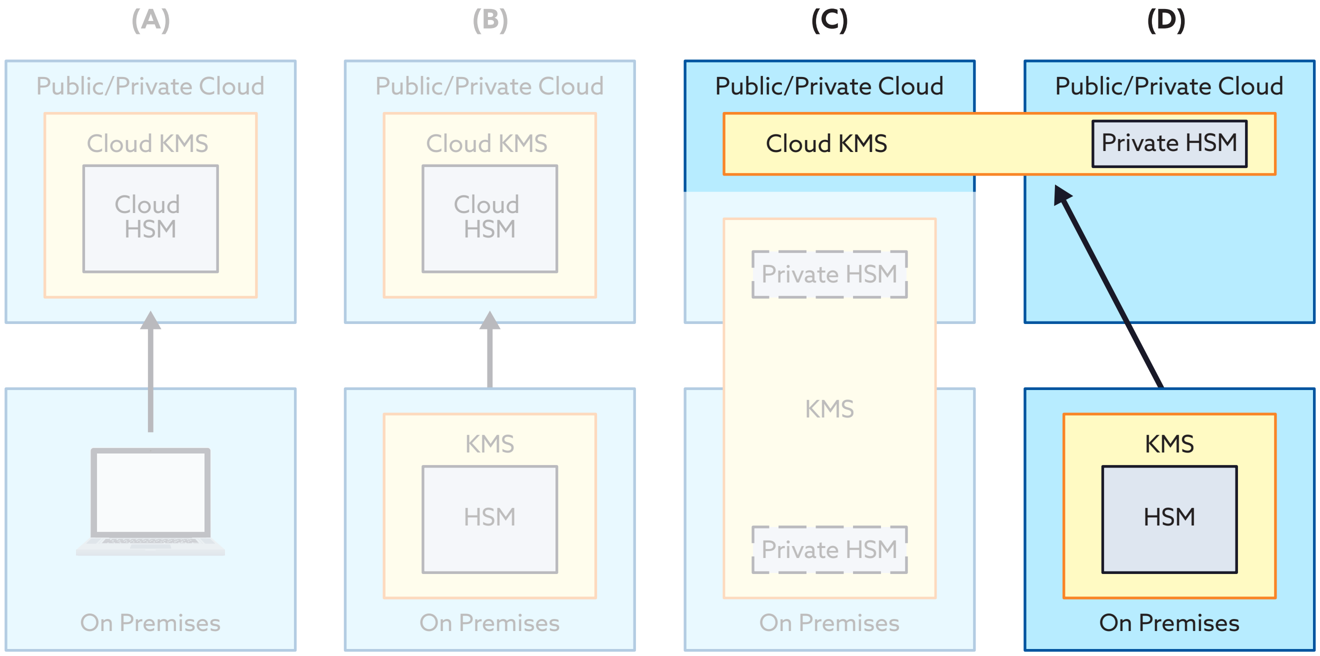

Figure 4: Cloud Service Using External Key Management System

Definition

The Multi-Cloud KMS pattern illustrates the ability to blend approaches for KMS implementations and cloud services. There are existing cloud services that support an external KMS and thus it is possible for the KMS to span many clouds and for the cloud to span many KMS choices.

Attributes

- This pattern's attributes (cost, complexity, control, implementation time, scale, performance and interoperability) are a function of the other cloud KMS patterns implemented
- Provides cloud-scale fault-tolerance due to the ability to leverage one cloud KMS as backup to another cloud KMS
- Can have side effect of speeding up adoption of additional cloud services due to past investments in expertise and resources

Challenges

This pattern is generally reserved for a strategic approach to Key Management. It requires the broadest range of expertise, the most time to engineer and implement, and the highest cost in either capital (licensing) or operational costs, or both. It does, however, provide the greatest degree of portability, fault tolerance and scalability due to leveraging multiple public cloud providers.

Recommendations for Use

- When resilience and scalability are highly valued
- Organizations with a dynamic composition - frequent acquisitions and divestitures
- Complex compliance requirements
- For all organizations with mature technology architectures

3. Encryption Key Management and Control

Encryption key management refers to control of the operations that are necessary to encrypt and decrypt data. This section covers the control environment — the setting which facilitates the operation of the controls and the controls themselves.

3.1 Key Management Control Environment

The key management control environment refers to the tone that management sets regarding the importance of key management control in the company. The control environment consists of all the policies and procedures supported by and approved by management as well as their action regarding deviations from policy. The rest of this section reviews some of the major policies necessary to ensure a proper control environment.

Governance and Policy Management

Defining and enforcing encryption key management policies affects every stage of the key management life cycle. Each encryption key or group of keys needs to be governed by an individual key usage policy defining which device, group of devices, or types of application can request it, and what operations that device or application can perform — for example, encrypt, decrypt, or sign. In addition, encryption key management policy may dictate additional requirements for higher levels of authorization in the key management process to release a key after it has been requested or to recover the key in case of loss.[9]

Control(s):

- Encryption and key management policies and procedures should be developed, approved, implemented and maintained.
- Encryption and key management policies and procedures should be reviewed, changed if necessary and approved at least annually.

[9] https://www.thalesesecurity.com/faq/key-secrets-management/what-encryption-key-management-lifecycle

Risk Management

Risk management is an essential element of governance, and encryption and keys play a fundamental role in overall risk management. Encryption keys are used to protect data from unauthorized access or use by converting it into an indecipherable form. A key, together with the addition of an algorithm comprise an encryption key which can be used to scramble or unscramble data. Additionally, within encryption and key management risks must be mitigated.

Control(s):

- Encryption and key management roles and responsibilities should be defined and implemented in a manner which enforces segregation of duties.
- An encryption and key management risk program should include provisions for risk assessment, risk treatment, risk context, and monitoring and feedback.

Change Management

Change management's goal is to ensure that change in the encryption environment does not result in less or insufficient security. Additionally, all changes to cryptographic systems are recorded, and any change must be able to be rolled back to the previous state should the change result in less or insufficient security. Change management is applicable to cryptographic design, development, testing, implementation and operations. As is usual with any change management process, users must be pre-informed, consulted and educated regarding changes to ensure that the change process is smooth and that they can perform their duties efficiently and effectively after the change is implemented.

Control(s):

- Changes to encryption and key management related systems, policies, and procedures should be made in accordance with a procedure that includes determining the effects, costs, and benefits of the change.
- Changes to encryption and key management related systems, policies, and procedures should be communicated to all relevant stakeholders.
- Changes are all recorded with the possibility of the change being reversed to reach the original state.

Key Management System

Cryptographic key management systems (CKMS) are composed of individual components, software module applications and hardware security modules (HSMs) that are used to generate, establish, distribute, store, account for, suspend, revoke, or destroy cryptographic keys and metadata[10] and are used to carry out sets of key management functions or services. Key management services include the generation, destruction, revocation, distribution, and recovery of keys. Some CKMS services (e.g. certificate authority (CA)) may be provided by a third party under contract or Service Level Agreement.[11]

[10] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt2r1.pdf Page 2, Sec 1.1, Para 2, last sentence

[11] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt2r1.pdf Page 2, Sec 1.1, Para 3

Control(s):

> - Encryption protocols should be used to protect sensitive data while it is at rest, in use, or in transit.
> - The encryption algorithm used to protect data should be based on the sensitivity of the data which it encrypts.
> - An encryption and key management storage/inventory system should require keys and/or certificates to be automatically recorded.
> - Key security functions should be isolated from non-security functions via partitions or domains.

Centralized Logging

Security Logs are records of security events that can help identify attacks. However, log data is often so voluminous that it is impossible to review all of it without automated assistance. Additionally, in cloud environments applications run in containers and create logs as standard output instead of writing them to log files. This standard output needs to be transferred to a centralized log management system to be properly analyzed. Besides, operations regulations require a rapid and coordinated response to events that necessitate the collection, storage and protection of log data for forensic use on a timely basis.

Control(s):

> - A secure audit trail of unalterable logging information evidencing key transactions and administration activities should be maintained.

Metrics, Monitoring and Reporting

Metrics are measures providing a standard that allows a performance assessment. These could be technical, such as encryption time, or manual, such as risk management policy review frequency.

Monitoring refers to sufficient frequency and transparency to ensure that action can be taken to remediate poor performance or replicate good performance.

Reporting refers to reporting to the right person at the right time so that action can be taken to remediate poor performance or replicate good performance.

Control(s):

> - A monitoring and reporting system should provide transparency into the operations of the encryption and key policies, processes, procedures, and controls.

Audit

The goal of the cryptographic audit is to ensure that governance risk and compliance has been properly embedded in the key management control environment as well in the key management life cycle/phases. The encryption audit process, consisting of planning, performance, reporting and follow-up, is not unique. What is unique is the area being tested and how encryption fits within the context of the company's operation. The tests performed during an audit include, but are not limited to, tests of the controls listed in both of these sections.

Control(s):

> - Encryption and key management systems, policies, and processes should be independently audited at least annually.
> - Controls should be selected for testing using a risk-based method with the ultimate goal of determining the design and operating effectiveness of the control.

3.2 Key Management Controls

The key management life cycle/phases are the operations that govern the creation, use, storage and destruction of keys. The primary operations covered are generation, activation, distribution, deactivation, compromise, suspension, expiration, archival, recovery, rotation, revocation, replacement and destruction. Example controls appear after the explanation of each operation.

Generation

The key has been generated but has not been authorized for use. In this state the key may be used only to perform proof-of-possession or key confirmation. Symmetric keys or public/private key pairs should be generated only when required.

Control(s):

> - All keys should be generated within a secure cryptographic module which relies upon a random bit generator.

Activation

The key in this state may be used to cryptographically protect information (e.g. encrypt plaintext or generate a digital signature) and/or to cryptographically process previously protected information (e.g. decrypt ciphertext or verify a digital signature).

Symmetric or private keys should be activated only when they are required to be used. A public key is activated when it is made available or on the date indicated in its associated metadata (e.g. notBefore date in an X.509 public key certificate).

Control(s):

> - Keys should be created in a pre-activated state (not ready to be used to encrypt data). Pre-activated keys should be activated by entering the start date of the validity/crypto period). Keys created without requiring activation may be activated immediately and remain active until expired or destroyed; activation and expiration dates are recommended for better control.

Distribution

After key generation, keys, and if generated, key material such as Random Bit Generators (RBG), need to be shared between two or more entities. This sharing or transport of keys and key material may take place through manual or automated methods.

Control(s):

> - Keys should be transported using secure channels through the use of manual methods or automated key-transport.

Deactivation

Keys in the deactivated state cannot be used to apply cryptographic protection but may be used to process cryptographically protected information.

Symmetric or private keys should be deactivated when they are no longer required for applying cryptographic protection to data. Deactivation of these keys may be followed by destruction or archival. A public key is not deactivated; it may expire (e.g. at the notAfter date in an X.509 public key certificate) or may be suspended (e.g. via Certificate Revocation List (CRL) in X.509 standard) or revoked (e.g. via CRL in X.509 standard).

Control(s):

> - Keys should be deactivated at the expiration date at which time the key becomes unable to encrypt data. Should the key be created without requiring an expiration date, the key remains active indefinitely increasing the risk that it could be used improperly.

Compromise

Generally, keys are compromised when they are released to or determined by an unauthorized entity. A compromised key should not be used to apply cryptographic protection to information. However, in some cases, a compromised key or a public key that corresponds to a compromised private key of a key pair may be used to process cryptographically protected information.

Control(s):

> - Cryptographic protection applied to information of compromised keys must cease, and the compromised key must be revoked.

Revocation

If the key has been revoked (i.e. for reasons other than a compromise), then the key may continue to be used for processing.

Revocation information for public keys is securely communicated to the relying parties, via CRLs or Online Certificate Status Protocol (OCSP) responses, in the case of X.509 public key certificates. Secret keys, including private keys, should be revoked when they have been compromised or identified in CRLs or OCSP responses; then the key should be added to a compromised key list.

Control(s):

> - Key rights should be revoked only by the authorized cryptographic service administrators.

Suspension

A suspended key should not be used to apply cryptographic protection (e.g. encrypt plaintext or generate a digital signature). However, a suspended key could be used to process information that was protected prior to the suspension (e.g. decrypt ciphertext or verify a digital signature).

A suspended key or key pair may be restored to an active state at a later time or may be deactivated or destroyed, or may transition to the compromised state.

A key may be suspended from use for a variety of reasons, such as an unknown status of the key or due to the key owner being temporarily away. In the case of the public key, suspension of the companion private key is communicated to the relevant parties. This may be communicated as an "on hold" revocation reason code in a CRL and in an OCSP response.

Control(s):

> - Key transitions and the reason for the transitions, from any state to and from suspension, should be monitored, reviewed and approved.

Rotation

Key rotation refers to the retirement of an encryption key and its subsequent replacement with a new cryptographic. Key rotation reduces the amount of information encrypted with any single key, which in turn reduces the amount of data exposed should a key be compromised. Rotation and crypto-periods are two closely related concepts in that a proper rotation relies on crypto-periods that consider among other things the amount of information expected to be encrypted by a key.

Control(s):

> - Keys should be rotated by generating a new key, and marking that version as the primary one used to encrypt new data. Non-primary keys can still be used to decrypt data previously encrypted.

Replacement

Sometimes, keys may need to be replaced due to a compromise of the key or the end of the key's crypto-period. Replacement can be accomplished by a rekeying process or by a key-update process.

Rekeying is the replacement of a key with a new key that is generated in a manner that is entirely independent of the value of the old key (i.e., knowledge of the old key provides no knowledge of the value of the replaced key and vice versa). In a key-update process, the new key is usually related in some way to the old key that it replaces.

Control(s):

- Replacement keys should be generated through re-keying and not through a key update process.

Expiration

When the time a key or keys are authorized for use ends, the crypto-period is said to have expired.

Control(s):

- The validity/crypto-period assigned to keys and certificates should be commensurate with the data sensitivity and volume of protected data.

Archival

Key archival is the storing of a private key of a certificate such that it can be recovered at a future time if required.

The key is stored when it is no longer required for normal use but may be needed after the key's crypto-period. An example for secret or private keys is the possible decryption of archived data. An example for public keys is the verification of archived signed documents.

Control(s):

- Archived keys should be managed in a secure repository requiring strict access controls.

Recovery

Key recovery is the act of bringing the crypto keys back up or archived keys back to their original state. It is important to closely replicate associated key attributes to their original state to preserve the assigned usage attributes. Key information may be recovered from backups during the key's valid crypto-period or from archives if the information has been archived.

All key recovery has to be properly authorized, based on

- key recovery operational scenarios: including law enforcement key recovery, enterprise key recovery and individual key recovery
- key recovery policy:
    - when key recovery information is generated
    - when key recovery information is to be received
    - how key recovery information is to be processed/ validated
    - use of any key recovery/ escrow agents

Control(s):

> - To determine what key information needs to be preserved for possible recovery, assess the risk to operational continuity versus the risk of the keying material and the information it protects being exposed if control of the keying material is lost should be assessed.

Destruction

Keys are in the destroyed state and no longer usable. All copies of the private or symmetric key should be destroyed once it has been carefully determined that they are no longer required (e.g. for archival or reconstruction activity). Keys should be destroyed in a manner that removes all traces of the keying material so that it cannot be recovered by either physical or electronic means. Public keys may be retained or destroyed, as desired.

Control(s):

> - Once the key has been marked for deletion, there is no facility to undo this state. It is imperative to provide the user with sufficient warning prior to entering this irreversible state.
> - Software-based keys are hard to prove to have been reliably destroyed from every place of storage and use. When copies of such cryptographic keys are made, care should be taken to provide for their eventual destruction.
> - If the use case requires a signed proof of destruction, it is recommended to use FIPS 140-2 hardware, with audit logs enabled throughout the complete key lifecycle.
> - Although the keys and their associated attribute values are destroyed, the key metadata (e.g. key name, type, hash value, crypto-period, key state transition history and usage) may be retained according to organizations' data retention policies.
> - Records of their existence may not be deleted for future audit purposes. Key escrow is one such auditable use case, where the proof of destruction can be paramount to the CSP's trust relationship with their end users. In this case, the audit log will be enhanced with a timestamped signature using a secure hardware device.
> - It is possible that a compromise of the destroyed key could be determined after the key has been destroyed. It is recommended to record such a compromise state in the key state transition history table.

4. Programmatic Library Interfaces

REST (REpresentational State Transfer) and SOAP (Simple Object Access Protocol) are the two most commonly used API architectures in the industry. Although they are compared often, they inherently serve different purposes. SOAP is a standards-based web services access protocol that was originally developed by Microsoft. REST APIs leverage URIs (Uniform Resource Identifier), the HTTP protocol, and JSON for a data format. SOAP has been long the standard protocol to web service interfaces, although the industry is currently dominated by REST architecture.

Recommendations for both REST and SOAP are presented below.

4.1 REST Recommendations

REST is a resource-based software architectural style for building APIs. RESTful web services do not follow a prescribed standard except for HTTP, hence it is important to build the RESTful API in accordance with industry best practices to ease development and increase client adoption. Here are some REST- specific security best practices:

Keep it Simple / Economy of Mechanism

The design should be as simple as possible. All the component interfaces and the interactions between them should be simple enough to understand. Every time the solution is “unnecessarily” complex, it is more likely a hole will be left.

Authentication

HTTPS should always be used. By always using TLS, the authentication credentials can be simplified to a randomly generated access token that is delivered in the username field of HTTP Basic Auth. It is relatively simple to use, and a lot of security features are included/available at no additional cost. Using HTTP 2 to improve performance should be considered. Multiple requests can be sent over a single connection, thereby avoiding the complete TCP and TLS handshake overhead on later requests.

Generated API Keys should be used instead of usernames/passwords.
The API Security Key should be securely stored in a file readable only by the owner. This limits the exposure of the key while keeping the key available for use with SDKs.

Authorization

HTTP methods should be protected. It should be ensured that the incoming HTTP method is valid for the session token/API key and associated resource collection, action, and record.

Whitelist allowable methods: It is imperative to properly restrict the allowable verbs such that only the allowed verbs will work, while all others return a proper response code (e.g. 403 Forbidden).

Privileged actions and sensitive resource collections should be protected. The session token or API key should be sent along as a cookie or body parameter to ensure that privileged collections or actions are properly protected from unauthorized use.

Input Validation

Strong validation checks should be used and the request rejected immediately if validation fails.
In API response, relevant error messages and examples of correct input format should be sent to improve user experience.

Input validation failures should be logged, particularly if it is likely that client-side code is going to call web services. It must be assumed that someone who is performing hundreds of failed input validations per second is up to no good.

It is important to consider rate limiting the API to a certain number of requests per hour or day to prevent abuse.

URL validations: Common names for common input tampering attacks include forced browsing, command insertion, cross site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation.

Incoming content-types should be validated. It is necessary to always check that the Content-Type header and the content are the same type. A lack of Content-Type header or an unexpected Content-Type header should result in the server rejecting the content with a 406 Not Acceptable response.

Response types should be validated and the request (ideally with a 406 Not Acceptable response) rejected if the Accept header does not specifically contain one of the allowable types.

XML input validation: It is critical to protect against XML External Entity attacks, XML-signature wrapping, etc. (http://ws-attacks.org)

Exposure of Information in URLs

Usernames, passwords, session tokens, and API keys should not appear in the URL, as this can be captured in web server logs, which makes them easily exploitable (e.g. https://api.domain.com/user-management/users/{id}/someAction?apiKey=abcd123456789)

Use of OAuth 2.0

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. The danger of compromised credentials is mitigated by using short-lived access tokens as the credential.

Use of Clear HTTP Status Codes

HTTP defines status code. The design of REST API should not use only 200 for success or 400 for error.

Exclusive Use of Approved and Up-to-date XML Libraries

In the past few years, security researchers have discovered a large number of security vulnerabilities in XML standards and libraries such as XML eXternal Entity injection (XXE) attacks. As SOAP heavily relies on XML for transport, only approved and up-to-date XML libraries should be used to handle SOAP messages.

4.2 SOAP Recommendations

SOAP (Simple Object Access Protocol) is an XML based protocol for clients and servers to exchange information. SOAP supports OASIS and W3C recommendations. Many of the generic HTTP related security features described above, such as secure transport, are applicable here as well. Some SOAP specific security best practices are as follows:

Encryption

Encryption ensures confidentiality of the message. It guarantees that the content of the message can be read only by the intended sender and recipient. An adversary, even one who manages to intercept an encrypted message, will not be able to read the actual message content. Ideally, the whole request/response, which includes all headers and body, should be encrypted. The "WS-Security" standard makes use of the "XML Encryption" standard to allow the encryption of certain parts of the requests/responses. The "XML Encryption" standard defines multiple different algorithms for the encryption. The choice of the algorithm is a crucial part of a secure configuration because a weak cryptographic implementation drastically compromises the mathematical guarantees offered by the algorithm.

Encryption/Digital Signature Validation

It is recommended to use an offline channel for exchanging the public certificates so that servers do not have to rely on messages to extract the public key for validation. All trusted public certificates should be pre-installed on the machine's certificate manager. Only these trusted public certificates should be used for cryptographic validation operations.

SOAPAction

The SOAPAction parameter leverages an HTTP request header to represent the function call in the SOAP Body of the request. Firewalls often use this parameter to determine whether a request from a certain source should be accepted or not. This design allows firewalls to make decisions without parsing the whole XML document. Some web services ignore this parameter and simply execute the function included in the Body of the request. This discrepancy allows attackers to change the HTTP request header to bypass all firewall restrictions and force the web service to execute a disallowed function. There are other implementations that execute the function referenced by the SOAPAction parameter without checking the actual function call in the request body. This enables attackers to change the executed function even if the request body is protected by encryption/digital signature. If the SOAPAction parameter is not being used, then it is highly recommended that this feature be disabled.

WS-Addressing

This standard allows the addition of addressing information to the header of SOAP messages. Adversaries may abuse these elements to reach web services or other servers that are not directly accessible to them. It is recommended to use this feature only if it is absolutely necessary. Otherwise, this feature should be disabled explicitly.

5. API Practical Considerations

This section identifies the important API entities and describes the three most important API relationships. It then addresses how these relationships and technology are managed in order to achieve secure encryption.

5.1 API Relationships in Practice

5.1.1 The Interface Triangle

From an API perspective, there are three entities that need to interact with each other as shown in the diagram below.

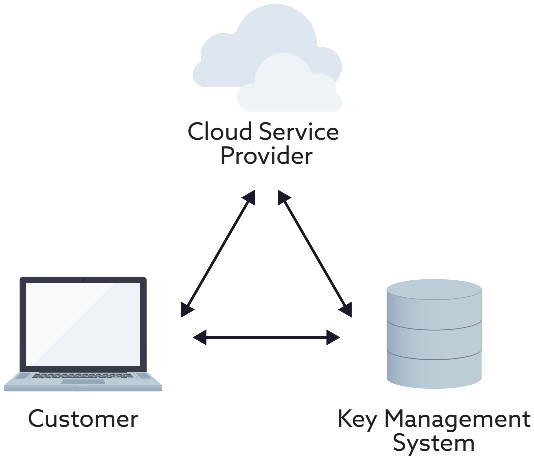

Figure 6: The Interface Triangle

5.1.2 Relationships

Customer / Cloud Provider
In the customer/cloud provider relationship, the customer uses APIs to consume cloud services from the cloud provider.

Requirements: Typically, a rich, broad API, covers a lot of functionality. The clients of the API are typically Graphical User Interfaces (GUIs), Command Line Interfaces (CLIs), or scripts. The API usually prioritizes ease of use and integration over latency or performance. Even though these APIs are cloud provider specific, they are usually similar to how customers manage other non-KMS cloud resources offered by the same provider. These APIs are generally accessible over the internet.

Customer / KMS
In this relationship, customers use KMS management APIs to manage keys. Managing keys entails setting access control policies, enforcing security and crypto policies, managing key lifecycle and rotation, etc.

Requirements: The requirements for this leg are similar to those of the customer / cloud provider interface. For added security protection, many KMS providers restrict the use of these APIs only to a handful of trusted machines inside an isolated private network. Some KMS providers also enforce physical access by exposing these APIs only locally.

Cloud Provider / KMS
The cloud provider uses a narrow interface to the KMS to either fetch or use the customer keys as needed to encrypt/decrypt the customer's data assets while in the cloud.

Requirements: These APIs need to be low latency to ensure that cloud providers can maintain their SLAs. These interfaces may be binary in order to minimize latency and processing overhead.

5.2 Practical API Management

This section provides a summary of the characteristics of some existing key management APIs and focuses on Key Management Interoperability Protocol (KMIP), Network-Attached Encryption - XML (NAE-XML), and KeySecure REST. Even though there are a few standardized protocols in the industry, rarely any of the interfaces described above are interoperable across different cloud service and KMS providers in practice.

Security

Description
Security is concerned with how the API is protected.

KMIP
Mutual Transport Layer Security (TLS) authentication (mTLS)

NAE-XML
Transport Layer Security (TLS). mTLS is typical, but client certificates are optional.

KeySecure REST
TLS. mTLS is typical, but client certificates are optional.

Authentication

Description
This category deals with how entities authenticate to the API and describes what types of entities and credentials are supported. It explores/determines if authentication is at the connection level, per request, or both.

KMIP
KMIP specification requires client certificates for connection-level security. The client certificate must be valid. Once the connection is established, each request may include headers with additional login credentials. Therefore, the KMS can choose to authenticate the actor from the TLS certificates or may require additional credentials in the request headers. KMIP request headers support both username/password credentials, as well as an "attestation" of the client's identity. This "attestation" method is very flexible and open: it could support a challenge and response scheme or could be used for the client to send any type of assertion, like a Json Web Token (JWT), access token, or SAML assertion.

NAE-XML
NAE-XML is a proprietary XML login API. The client connects to port and can issue some commands (like version info) before issuing a login command. Other commands cannot be issued until after the login command. Credentials are usually the TLS certificates, but username/password is also supported. The authentication context is tied to the TCP connection, not to individual requests. There is a timeout to close the connection after X minutes of no activity. Client entities can be either humans or machines, similar to KeySecure REST, but NAE-XML does not distinguish between them.

KeySecure REST
Securing the API with OAUTH 2.0.
Credentials are usually the TLS certificates, but username/password is also supported. The client must first acquire an access token from a /tokens endpoint and then use this access token in the Authorization header with bearer scheme in every subsequent request. Authentication is performed by the KMS for each request, by examining the bearer token. Tokens are JWTs with a five-minute expiration. Clients are also issued a refresh token with a longer expiration, which they can use to acquire additional access tokens. Client entities can be either users or machines. There are differences in the way users and machines are provisioned, and users typically have username/password credentials where machines have certificates. But the access token endpoint and the actual KMS requests are identical. The refresh tokens are typically configured to expire after X minutes of inactivity.

Authorization

Description
This category entails how operations are authorized.

KMIP
Out of scope

NAE-XML
Keys have owners, who have complete access to keys. Key owners can grant groups of other users limited access to use the key.

KeySecure REST
This is a very flexible ABAC model. Out of the box, it works just like NAE-XML, with key owners and group permissions.

Rotation

Description
Rotation concerns how keys are rotated or versioned as well as how clients know which version is current.

KMIP
There is no native concept of key “versions”. Each key is a standalone object. However, keys can have “links” to other keys, where each link has some semantic meaning. Rotating, or “replacing” a key in KMIP involves generating a new key, moving the “names” of the prior key to the new key, and creating links between the two keys, one link of type “replacedObject” pointing from the new key to the old, and a link of type “replacementObject” from the old to the new. Clients performing protect or signing operations, which are only interested in the latest version of a key, refer to the key by one of its “names.” Since those names are moved from the old key to the new key, the names will always refer to the newest version of the key. (Decrypt and verify operations refer to keys by an ID, which never moves).

NAE-XML
NAE-XML supports both versioned and unversioned keys. Unversioned keys do not support rotation. For versioned keys, a key can have multiple versions. In a sense, a "key" really refers to a set of key versions. Keys have a name, so referring to a key by name always resolves to the latest version of the key. Clients can also refer to a key by name and version (for decrypt/verify ops).

KeySecure REST
KeySecure REST is similar to NAE-XML. Keys have a single "name" and a version sequence number. When a key is rotated, a new key is created with the same "name" and a new version sequence number. Keys also have 0-n "aliases" (which have semantics similar to names). Aliases are moved from the old version to the new version (similar to KMIP). There are also "links," with semantics identical to KMIP, and "replacementObject" and "replacedObject" links are auto created. In all other respects, key versions are independent objects with their own properties and attributes. Rotating a key is like cloning the latest version of a key, but after the clone, the new version is an independent object (like KMIP). Clients wanting the latest version of key refer to the key by "name" or by one of its aliases.

Identifiers

Description
This category is concerned with how keys are identified, how many identifiers the keys have, how the key versions are identified and what the semantics of each identifier are.

KMIP
KMIP is a Unique Identifier, a completely opaque string, globally unique (across tenants, never reused, ideally across KMS). It is generally generated by the KMS name, 0-n opaque string identifiers chosen by the client. It is moved to the newer key on rotation. Another option is shortUniqueIdentifier. New in KMIP 2.0, this is a shorter identifier, which is otherwise the same semantically as the uniqueIdentifier. It is suggested to be a hash of the uniqueIdentifier.

NAE-XML
Key name; this is an immutable string identifier selected by the client, or generated on the server, and refers to all versions of a key. NAE-XML is unique only within a single tenant/account/domain. If the key is deleted, the name may be reused.

KeySecure REST
ID, MUID, UUID, and UniqueIdentifier: are all opaque string identifiers, generated by the KMS. They all have the same semantics but have slightly different formats to conform the needs to various clients.
name; The name is a unique string, same semantics as the NAE-XML name.
Aliases: Aliases are 0-n unique string identifiers, assigned by clients, may be reused, unique only within a single domain, copied to the newest key on rotation. They are similar semantically to KMIP "name" attribute.

Attribute/Exportability

Description
This category concerns whether the API has a notion that a key is exportable, has ever been exportable and has ever been exported.

KMIP
KMIP has several attributes to describe exportability characteristics of a key:

- One such attribute is Fresh: this is true if the object has never been exported (not yet been served to a client)
- (specifications say by the "Get" operation, and do not mention the "Export" operation, which is very similar. It is not clear if Export should also trigger flipping this flag from True to False.)
- Another attribute is Extractable, which is true if the object can be exported.
- The 'Never Extractable' attribute is true if the object has never been extractable.
- The next attribute is 'Sensitive', which is true, when the key can only be exported if wrapped. Unwrapped export is not allowed.
- The final attribute is 'Always Sensitive', which is true if the object was Always Sensitive.

NAE-XML
Out of Scope

KeySecure REST
The REST API has similar semantics to KMIP, but the attribute names are different ("unexportable," "neverExported," "neverExportable"). Attributes equivalent to Sensitive and Always Sensitive are on the roadmap.

Export

Description
This category entails how keys are exported, if they are wrapped and if the wrapping is optional.

KMIP
KMIP supports two operations for the export: "Get" and "Export." The documentation is not clear regarding when it is appropriate to use each one. "Get" returns only the key material (not the attributes) and seems targeted at clients who want to use the material. "Export" includes the material and all the attributes and seems targeted at migration of a key from one KMS to another. Both commands allow optionally passing wrapping parameters. The wrapping parameters define the format of the wrapping, UniqueIdentifiers of the keys to use for wrapping and HMAC-ing (there must be another key already present in the KMS).

KeySecure REST
The REST API supports a POST /keys/:id/export endpoint, which exports the key and all the attributes. The export parameters may include wrapping instructions, which may either be the "name" of a symmetric key (already present in the KMS) or may explicitly include an RSA public key.

Lifecycle

Description
This category is concerned with the state lifecycle of a key.

KMIP
The lifecycle for KMIP keys is described here: https://docs.oasis-open.org/kmip/kmip-spec/v2.0/cs01/kmip-spec-v2.0-cs01.html#_Toc6497510

Lifecycle states are Pre-Active, Active, Deactivated, Compromised, Destroyed, and DestroyedCompromised.

Lifecycle flows only in one direction. One cannot return to any prior states. It does not have a notion of temporary suspension. KMIP also has a notion of "archived," but it is a simple binary state and orthogonal to the lifecycle.

KMIP also has a Process Start Date and Protect Stop Date. These dates further constrain when a key can first be used to decrypt/verify (Process Start) and when a key can last be used to encrypt/sign (Protect Stop Date). These dates have constraints on their values which are related to the lifecycle states (e.g. Protect Stop Date must not be after the Deactivation Date, although it can be prior or equal to the Deactivation Date).

The state of a key cannot be directly changed by a client. Transitions from one state in the lifecycle to another occur as the result of either passing a date in another attribute (e.g. Activation Date), or as the result of some operation which causes a state change (e.g. Revoke).

NAE-XML
NAE-XML supports four lifecycle states: Active, Restricted, Retired, and Wiped. Active means that all operations are possible; Restricted means that only decrypt or verify operations are allowed; and retired means that no operations are allowed. Wiped means that the key material is destroyed (it is like an irreversible version of Retired). Active, Restricted, and Retired can be freely changed to any other state. These states apply only to individual versions of versioned keys. (Non-versioned keys do not seem to have a lifecycle).

State changes between Active, Restricted, and Retired occur when a client directly sets the state attribute. Wiped is a side effect of destroying a key.

KeySecure REST
This is the same as KMIP.

Attributes

Description
This category focuses on how the API handles attributes of keys.

KMIP
KMIP supports a large set of standard attributes. A key does not need to define all these attributes, but any that are defined and tightly constrained by the specification. KMIP clients and servers may optionally define any number of additional custom attributes. Attributes must be defined in terms of the type system of TTLV (the native binary encoding format KMIP uses).

NAE-XML
Keys have a fixed set of built-in attributes which are always present. Most are read-only (managed solely by the server). Clients can add a finite number of additional custom attributes.

KeySecure REST
This is similar to KMIP. Keys have a "meta" attribute, which is a JSON document that the client can freely set. Clients can add any attributes or nested JSON documents to "meta."

6. Conclusion

The adoption of cloud services can help organizations realize many advantages including technological agility, elastic scale, speed to market, and lowered capital expenditures. Cloud services also present challenges, particularly in the realm of data privacy and security. When adopting cloud services, it is crucial for organizations to establish a very clear understanding of their data privacy expectations and obligations.[12] Before thinking in technological terms — implementation details and cybersecurity concepts — an organization must first be able to unambiguously answer the question "What is/are the outcome(s) we expect to achieve?" This question can lead to a wide spectrum of answers including the following:

- The data privacy risks of using the cloud service must be fully documented, clearly understood, and accepted by the organization's legal and risk management teams.
- Data will be no less secure than it was prior to adoption of the cloud service.
- Privacy obligations are supported by the cloud vendor certifications, and any change to the vendor's certifications provides adequate notice for an orderly exit from the service.
- Data will not be at risk of exposure (in readable/intelligible form) to cloud service human personnel.
- Data cannot be released by the cloud vendor to any third party in readable/intelligible form.

It should be noted again that none of the above examples refer to technology, controls, encryption, etc. Each of the above statements can allow for various technological means of meeting the desired outcome, and implementations may or may not involve encryption, keys, and key management systems. Understanding the organization's obligations and goals for data privacy and security is the precursor to choosing technological solutions and implementations, including the use of encryption. A great deal of human energy and time has been wasted implementing encryption where the outcome did not deliver the expected data privacy or security, leading to embarrassment, data loss, audit failures, and job insecurity. Establishing clear business and data privacy and security expectations can avoid these unpleasant outcomes.

[12] These outcomes also need to be balanced with other considerations when using cloud services: elastic scale, high performance, and geographic footprint.

Once encryption is established as a required or recommended piece of a technology architecture, it is then crucial to understand the dynamics of encryption key generation, distribution, handling and destruction. Because encryption is a piece of so many technology architectures, the use of keys and key management systems tends to become a highly complex ecosystem within many organizations. Some organizations then need help in determining which forms of key management systems are appropriate for the given business and technical use case requirements that aim to meet the organization's data protection, compliance, and privacy expectations.

Once an organization has chosen to use encryption with cloud services it is once again necessary to revisit the expected outcomes for data privacy and security. The dynamics of encryption key usage play a fundamental role in the data privacy outcomes achieved. This was illustrated in section 2.1.1, The Meaning of BYOK where it was shown that the use of so-called BYOK (Bring Your Own Key) and similar models with cloud services does not achieve the outcome that is typically expected. Most organizations seeking to use these so-called BYOK models are expecting that the cloud service provider cannot be forced to turn over the customer's data to a third party such as a court or law enforcement entity. However, most vendor implementations of so-called BYOK models do not actually prevent that outcome because the cloud service has use of the data encryption keys and can therefore produce unencrypted data for export if required. The term BYOK (and its variants) focus, incorrectly, on the generation of keys rather than the use of keys. Decryption of data is orthogonal to how a key was generated; so long as the cloud service can use a key that will decrypt data, the technology does not care at all where or how the decryption key (or its parent or parent's parent, etc.) was generated. Therefore, to evaluate expectations of data privacy and security, one must examine how keys can be used: what entities can use them, under what circumstances, for what purposes, and under what constraints. All of these inputs are needed in determining the operational model and parameters of any key management systems supporting the encryption solution in use.

An additional aspect of encryption keys and their supporting key management systems that is sometimes overlooked is the analysis of threats and their relative importance. As all cybersecurity experts know, security is not free, and prioritizing threats is necessary to achieve the best balance of resource expenditures and induced friction. For example, some encryption keys are useful only for destruction of data — witness the case of a typical disk/volume encryption technology, where the key is controlling the availability of data, but not its security. The key itself is necessary to decrypt the disk's contents but is only indirectly providing data confidentiality (a PIN or human-factor artifact is normally used to "unlock" the encryption key for use by the operating system). In this type of encryption system, theft of the key is effectively a non-event and thus not really a threat that should have any priority. However, destruction of the key could render the entire contents of the disk permanently lost. Therefore, the highest priority (and perhaps only) threat that needs to be addressed is ensuring that the destruction of the key (and escrowed copies) is very difficult without proper authorization and supporting procedures. This type of dynamic has obvious implications for the supporting key management system and therefore upon the choice of KMS.

CSA recommends that organizations — regardless of any cloud adoption planned, underway, or established — document the capabilities of existing key management systems in terms that can be evaluated for potential use in achieving business and privacy/security outcomes. This establishes the foundation for evaluating the potential for using a KMS with any cloud service. The same evaluation can and should be done with cloud KMS services.

In summary, key management systems, including hardware security modules and other cryptographic tools, are used to meet compliance, data control requirements and security benefits in service to business needs. This document illustrated the use of four KMS patterns with cloud services: Cloud Native Key Management System, External Key Origination, Cloud Service Using External Key Management System and Multi-Cloud Key Management Systems. They were defined, their attributes and challenges were enumerated, and recommendations for their use were given. Next, key management control was covered in two sections: The Control Environment and Controls themselves where the control necessary to ensure compliance, data control and security were reviewed. After this, the document reviewed and made recommendations concerning the use of the two most used API architectures and finished with practical API considerations.

The KMS patterns, controls and API architectures and practical considerations are to be used as a guide by organizations to help them achieve their key management objectives, which are in service to the data privacy and security expectations of the organization.

References

[NISTIR 7956]	NISTIR 7956. Cryptographic Key Management Issues and Challenges in Cloud Services, September 2013. https://csrc.nist.gov/publications/detail/nistir/7956/final
[NIST SP 800-57]	NIST SP 800-57 Part 1 Rev. 4. Recommendation for Key Management, Part 1: General, January 2016. https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final
[NIST SP 800 57] Part 2	NIST SP 800-57 Part 2 Rev. 1. Recommendation for Key Management: Part 2: Best Practices for Key Management Organizations, May 2019. https://csrc.nist.gov/publications/detail/sp/800-57-part-2/rev-1/final
[NIST SP 800-57] Part 3	NIST SP 800-57 Part 3 Rev. 1. Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance, January 2015. https://csrc.nist.gov/publications/detail/sp/800-57-part-3/rev-1/final
[NIST SP 800-57] Part 1	NIST SP 800-57 Part 1 Rev. 5. Recommendation for Key Management: Part 1 – General. May 2020. https://doi.org/10.6028/NIST.SP.800-57pt1r5
[NIST SP 800-52]	NIST SP 800-52 Rev. 2, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, August 2019. https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final
[NIST SP 800-133]	NIST SP 800-133 Rev. 1. Recommendation for Cryptographic Key Generation, July 2019. https://csrc.nist.gov/publications/detail/sp/800-133/rev-1/final
[OpenAPI Initiative]	OpenAPI Initiative (OAI) v3.0. https://github.com/OAI/OpenAPI-Specification
[OWASP]	OWASP Key Management Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html
[CSA's Security Guidance]	Security Guidance for Critical Areas of Focus in Cloud Computing v4.0. https://cloudsecurityalliance.org/artifacts/security-guidance-v4/
[OASIS Key Mgmt Interoperability Control]	OASIS Key Management Interoperability Protocol (KMIP). https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=kmip

Appendix A: Acronyms

Selected acronyms and abbreviations used in this paper are defined below.

AI	Artificial Intelligence
API	Application Programming Interface
BYOK	Bring Your Own Key
CC	Common Criteria
CISO	Chief Information Security Officer
CLI	Command Line Interface
CMK	Customer Master Keys
CRL	Certificate Revocation List
CKMS	Crypto Key Management System
CSA	Cloud Security Alliance
CSP	Cloud Service Provider
CYOK	Control Your Own Key
DEK	Data Encryption Keys
EKM	External Key Manager
FIPS	Federal Information Processing Standard
GRC	Governance Risk and Compliance
GUI	Graphical User Interface
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HYOK	Hold Your Own Key
IAM/IdAM	Identity and Access Management
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
KMS	Key Management System

KMIP	Key Management Interoperability Protocol
KYOK	Keep Your Own Key
ML	Machine Learning
MCKMS	Multi-Cloud Key Management System
MTLS	Mutual Transport Layer Security (TLS) Authentication
NAE-XML	Network-Attached Encryption (NAE) -eXtensible Markup Language (XML) (NAE-XML)
NIST	National Institute of Standards and Technology
NISTIR	National Institute of Standards and Technology Interagency Report
OAUTH	Open Authorization
OCSP	Online Certificate Status Protocol
OS	Operating System
PaaS	Platform as a Service
PCI DSS	Payment Card Industry Data Security Standard
PKI	Public Key Infrastructure
RBAC	Role-Based Access Control
RDS	Relational Database Service
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SaaS	Software as a Service
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
TPM	Trusted Platform Module
TLS	Transport Layer Security
mTLS	Mutual Transport Layer Security
TTLV	Tag, Type, Length, Value
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

UOID	Unique Object Identifier
WS-Security	Web Services Security
XML	Extensible Markup Language
XXE	XML External Entity

Appendix B: Big 5 CSP KMS Comparison

Presented below is an overview of four key public cloud service providers' KMS Offerings.

Alibaba Cloud	
Alibaba has one managed service that is relevant to this document.	
Alibaba Cloud KMS	Alibaba Cloud Key Management Service (KMS) provides secure and compliant key management and cryptography services to help customers/users/organizations encrypt and protect sensitive data assets. KMS is integrated with a wide range of Alibaba Cloud services to allow customers/users/organizations to encrypt data across the cloud and to control its distributed environment. KMS provides key usage logs via ActionTrail, supports custom key rotation, and provides HSMs that have passed FIPS 140-2 Level 3 or other relevant validation, to help customers/users/organizations meet their regulatory and compliance needs.
AWS	
AWS offers two managed services that are relevant to this document.	
KMS	AWS Key Management Service (AWS KMS) is a managed service that makes it easy for customers to create and control the encryption keys used to encrypt the data. It uses hardware security modules that have been validated under FIPS 140-2 or are in the process of being validated. AWS KMS allows users to manage customer master keys (CMK) and use them for various cryptographic operations. A CMK is a logical representation of a master key. CMKs are used for managing data encryption keys (DEK) that are used for encrypting data, including large amounts of data and other data encryption keys.
CloudHSM	AWS CloudHSM is a cloud-based hardware security module (HSM) that manages encryption keys on the AWS Cloud using FIPS 140-2 Level 3 validated HSMs. This service essentially provides a CloudHSM Cluster. Clusters can contain multiple HSM instances spread across multiple Availability Zones in a cloud region. HSM instances in a cluster are automatically synchronized and load-balanced. Customers receive dedicated, single-tenant access to each HSM instance in their cluster. Each HSM instance appears as a network resource in their virtual cloud network. AWS provides HSM cluster management APIs.
Azure	
Azure has two managed services that are relevant to this document.	

Azure Dedicated HSM	Azure Dedicated HSM is an Azure service that provides cryptographic key storage in Azure using dedicated HSMs. It is the ideal solution for customers who require FIPS 140-2 Level 3-validated Thales Luna devices and complete and exclusive control of the HSM appliance. It is also suited for applications which need Common Criteria EAL 4+, NITES, or Brazil ITE and needs cryptography other than RSA and ECC. HSM devices are deployed globally across several Azure regions. They can be easily provisioned as a pair of devices and configured for high availability. HSM devices can also be provisioned across regions to assure against regional-level failover. Microsoft delivers the Dedicated HSM service by using the SafeNet Luna Network HSM 7 (Model A790) appliance from Gemalto.
Azure Key Vault	Azure Key Vault helps customers with secrets management, key management and certificate management. Secrets and keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules (HSMs). The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. Authentication is done via Azure Active Directory. Authorization may be done via role-based access control (RBAC) or key vault access policy. RBAC is used when users deal with the management of the vaults, and key vault access policy is used when they are attempting to access data stored in a vault. This service has native integration with a large number of other Azure services.
Google Cloud	
Google has two managed services that are relevant to this document.	
GCP Cloud KMS	GCP Cloud KMS is a scalable cloud-hosted key management service that lets customers manage cryptographic keys for their cloud services the same way they do on premises. It supports AES256, RSA 2048, RSA 3072, RSA 4096, EC P256, and EC P384 cryptographic keys. Cloud KMS is integrated with Cloud IAM and Cloud Audit Logging so that customers can manage permissions on individual keys and monitor how these are used. Cloud Key Management Service stores cryptographic keys in a hierarchical structure designed for useful and elegant access control management. Cloud KMS has a built-in 24-hour delay for key material destruction in order to prevent accidental or malicious data loss. Cloud KMS can also generate HSM-backed keys with GCP Cloud HSM.

GCP Cloud EKM	GCP Cloud External Key Manager (EKM) is in early access and allows to encrypt data with encryption keys that are stored and managed in a third-party key management system. Cloud EKM lets customers/users encrypt data in BigQuery and Compute Engine with encryption keys that are stored and managed in a third-party key management system that is deployed outside Google's infrastructure. External Key Manager allows customers/users/organizations to maintain separation between data at rest and encryption keys while still leveraging the power of cloud for compute and analytics.
IBM	
IBM has two managed services that are relevant to this document.	
IBM Cloud HSM	IBM Cloud Hardware Security Module (HSM) Luna 7.0 from Thales protects the cryptographic infrastructure of some of the most security-conscious organizations in the world by securely managing, processing and storing cryptographic keys inside a tamper-resistant, tamper-evident device. With IBM Cloud HSM 7.0, customers/users/organizations can solve complex security, compliance, data sovereignty and control challenges associated with migrating and running workloads on the cloud.
IBM Key Protect	IBM Key Protect is a cloud-based security service that provides life cycle management for encryption keys that are used in IBM Cloud services and non-IBM cloud applications. Key Protect provides roots of trust, backed by a FIPS-140-2 Level 3 hardware security module (HSM). It supports importing the customer's root of trust encryption keys into the service. It can generate, store, and manage customer keys with a secure, application-friendly, cloud-based key management solution for encryption keys. When keys are deleted, they can never be recovered, and any data that is encrypted under those keys cannot be recovered. All programmatic interfaces are secured by TLS and mutual authentication.

Appendix C: Key State Lifecycle Layered View

Minimal standards:

This document proposes that the CSPs follow each of the states in the key state lifecycle. Recognizing that not all application developers will find it useful to move through the key states. This document proposes a two-tiered key state transition. Implementing five Tier 1 state models will be mandatory for all keys, while Tier 2 sub-states/ reasons will be optional.

Tier 1	Tier 2
1. Generated	
2. Suspended: includes sub-states	
	a. Distribution
	b. Replacement
	c. Rotation
3. Activated	
4. Deactivated: includes reasons such as	
	a. Compromised
	b. Revoked
	c. Expired
	d. Archived
5. Destroyed	

The advantage of this view is that a developer who wants to do a quick key state check can simply work with these five values (and state transitions) and still meet the key lifecycle management criteria for logging/ auditing.

Also, a particular CSP may want to limit their actions based on these Level 1 states, e.g. if the key is in a deactivated state to only allow decrypt/ verify/ MAC verify/ unwrap operations. However, if an auditor wants to explore the reasons for the deactivated state, there is an option for them to delve into Level 2 details about why a key is currently deactivated.