IRONCORE LABS

CMK: WHAT ARCHITECTS NEED TO KNOW

A white paper on Customer Managed Keys

WHAT IS CMK?

"For many large enterprises, ownership and control of encryption keys is a baseline requirement for SaaS adoption."

- Forrester

Customer Managed Keys, or CMK, is a cloud architecture that gives customers ownership of the encryption keys that protect some or all of their data stored in SaaS applications.

CMK is known by many names:

- EKM - Enterprise Key Management
- BYOK - Bring Your Own Keys
- CSK - Customer Supplied Keys
- BYOE - Bring Your Own Encryption

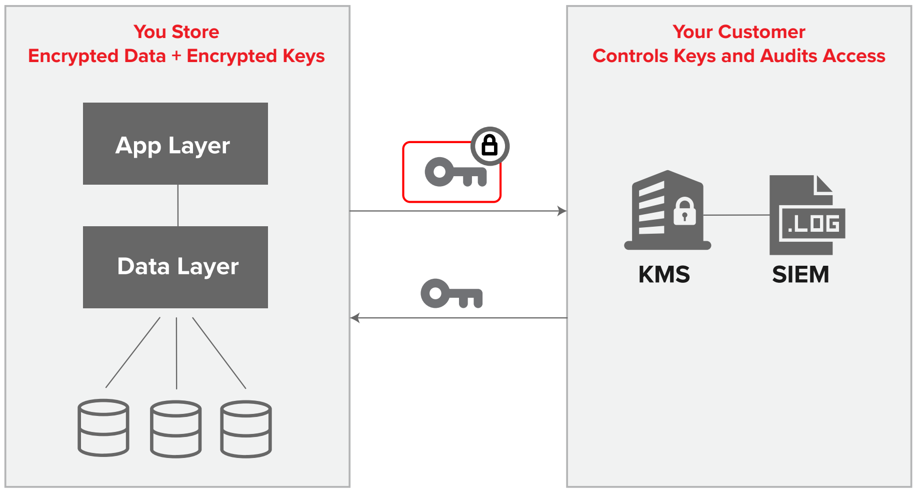
Regardless of name, CMK has these characteristics:

- Per-tenant encryption for some or all customer data.
- Your customer (tenant) manages a master key or keys needed for decryption.
- Your customer can independently monitor all data access.
- Your customer can independently revoke access at any time.

This whitepaper gives cloud, enterprise, and software architects technical context behind CMK, presents a reference architecture, and discusses implementation patterns and tradeoffs.

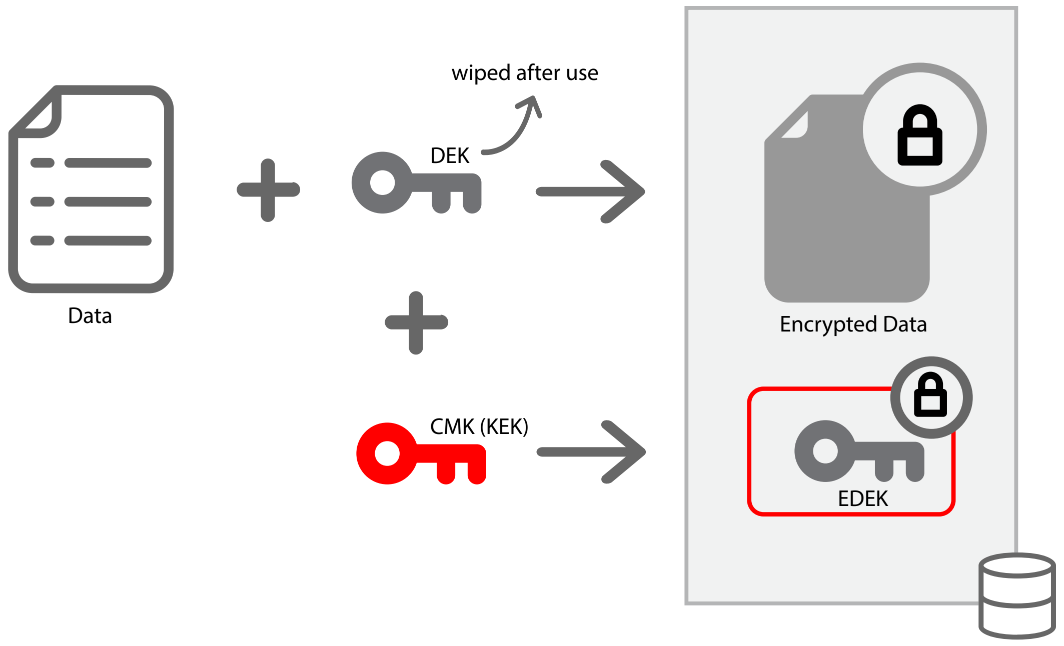
HOW CMK GENERALLY WORKS

In CMK, you encrypt sensitive customer data before you store it. When you need data access, you call your customer's infrastructure to get the decryption key. Your customer can revoke access by refusing to return the key, and they get an independent audit event on every request.


You Store
Encrypted Data + Encrypted Keys
App Layer
Data Layer
Your Customer
Controls Keys and Audits Access
.LOG
KMS
SIEM

The Envelope Encryption Pattern

Storing and encrypting data in CMK involves multiple layers of keys. The typical approach uses two layers and is referred to as "envelope encryption." In envelope encryption, you *first* encrypt data with a data encryption key or DEK. You use a *second* master key, or MK, to encrypt the DEK, producing an Encrypted DEK or EDEK.



You have access to the DEK while you are encrypting or decrypting, but you agree to wipe the key from memory after use. You never persist the DEK to storage. Instead, you store the encrypted DEK, or EDEK, alongside encrypted data. Typically you add a column to your database schema or persist the EDEK as object metadata.

The process of encrypting a key is known as key wrapping. The wrapping key is generically called a key encryption key or KEK.

In CMK, the MK needed to decrypt the EDEK is held by the customer's Key Management System (KMS). You ask the KMS to decrypt the EDEK. If the KMS grants access, it will unwrap and return a DEK which you in turn use to decrypt data. You then wipe the DEK from memory. The master key never leaves control of your customer. In a two-level hierarchy, the MK serves the role of the key encryption key.

Envelope encryption with two keys		
Definitions	plaintext	Unencrypted data
	ciphertext	Encrypted data
	DEK	The data encryption key (DEK) is the key used to encrypt plaintext data. In practice, the DEK always uses symmetric encryption.
	KEK	The DEK is wrapped (encrypted) by a key encryption key (KEK). In practice, a KEK can be symmetric or asymmetric (public/private key pair).
	EDEK	The encrypted data encryption key.
	MK	The Master Key controlled by the Customer KMS. The MK serves the role of the key encryption key. The MK never leaves the Customer KMS.

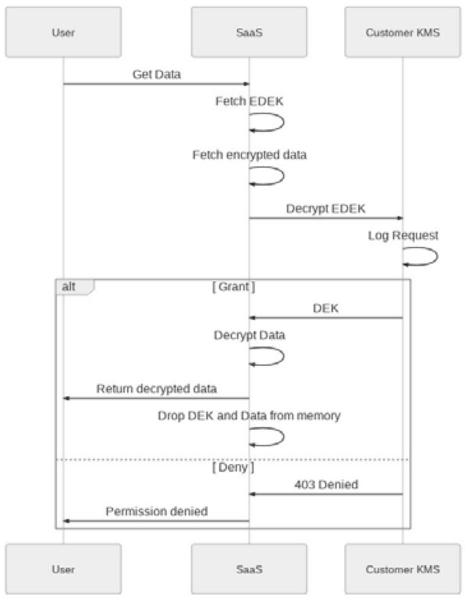
Operations	$\text{encrypt}_{\text{2-level}}$	`{ EDEK, ciphertext } = encrypt(plaintext)`
		
		`composed by...`
		
		`// The customer KMS generates a DEK and wraps it`
		`{ DEK, EDEK } = generateKeys()`
		
		`// The DEK is wiped from memory after use`
		`ciphertext = encrypt(plaintext, DEK)`

	$\text{decrypt}_{\text{2-level}}$	`plaintext = decrypt(ciphertext, EDEK)`
		
		`composed by...`
		
		`// The customer KMS unwraps the EDEK with its MK`
		`DEK = unwrap(EDEK)`
		
		`// the DEK is wiped from memory after use`
		`plaintext = decrypt(ciphertext, DEK)`

Customer Control and Independent Audit

CMK allows your customer to independently revoke and transparently monitor access to *their* data stored in *your* SaaS application. Basically, if your company shows up in the news with the term "breach," your customers want to kill access immediately. As CMK increasingly becomes labeled a "best practice," your customers are contractually (e.g. insurance) and legally (e.g. HIPAA) obligated to insist on its adoption.

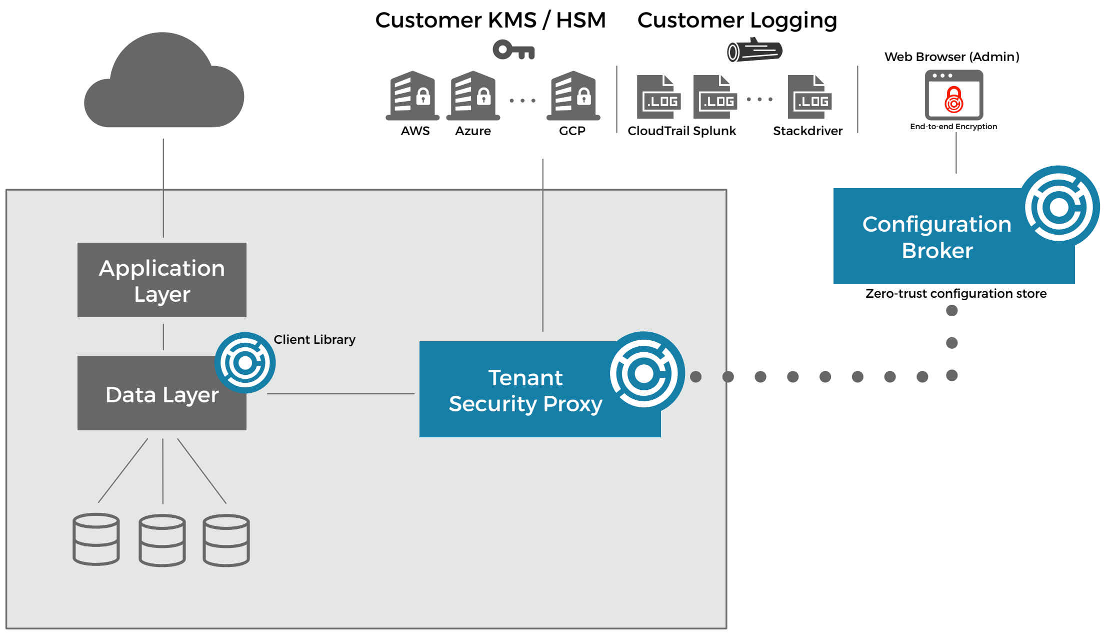
The sequence diagram below shows the decryption data flow. Notice how the log request data flow provides independent monitoring and the 403 allows revocation.



Customer data is encrypted and decrypted locally in your infrastructure using high-speed symmetric encryption operations. Only the EDEK, a small payload, goes back to your customer's infrastructure to be unwrapped. We'll see later that EDEK decryption can be further optimized.

IRONCORE CMK ARCHITECTURE

IronCore provides a turnkey CMK Implementation that you can quickly and easily integrate into your SaaS application. Even if you decide to implement your own CMK solution, the IronCore architecture can inform your technical approach.



Data Layer

CMK is a server-side approach to encryption. Generally speaking, you want to encrypt on your server as early as possible and decrypt as late as possible. In practice, the easiest way to integrate is to find persistence layers and encrypt where fetch/store activities happen. Finding the right data seam is often the most challenging aspect of integration. Look for the following:

- Anywhere fetch and store have been consolidated to make access control more reliable.
- Data transfer objects used to serialize and deserialize.
- Code generated from formal data interface definitions such as protobuf, GraphQL, or Swagger.

When you have identified one or more data seams, use the IronCore SDK API calls:

```
// Request a key from the KMS and use it to encrypt the document
EncryptedDocument encryptedResults = client.encrypt(documentMap, metadata).get();
```

```
// Decrypt the document back to plaintext
PlaintextDocument decryptedResults = client.decrypt(recreated, metadata).get();
```

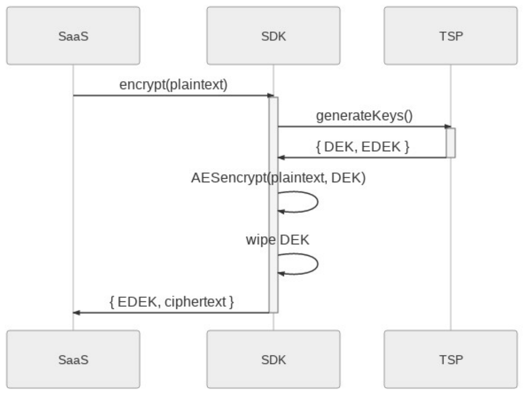
Writing on an iMessage vulnerability...

"There is an important lesson in this: security is hard. Apple Computer has one of the best security teams on the planet. This feature was not tossed out in a day; it was designed and implemented with a lot of thought and care. If this team could make a mistake like this, imagine how bad a security feature is when implemented by a team without this kind of expertise.

Getting security right is hard for the best teams in the world. It's impossible for most teams."

- Bruce Schneier

A core design principle of IronCore's service is to *remove decisions from individual developers*. Getting security and cryptography right is notoriously error-prone. The IronCore API is simple. Encryption policy is metadata-driven and separated as a cross-cutting concern. In this way policy enforcement is architected directly into your platform and access rules are defined, tested and audited by security specialists.



The IronCore SDK is embedded in your data layer and hides the details of envelope encryption.

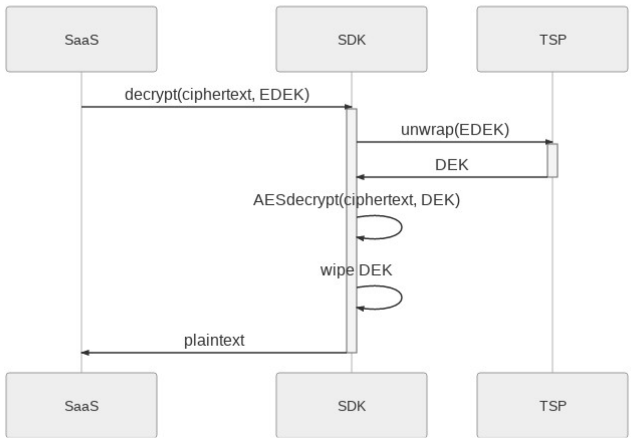
The SDK gets a generated DEK and EDEK from the Tenant Security Proxy (TSP, discussed below). The SDK encrypts plaintext to ciphertext with the DEK. The DEK is wiped from memory and the EDEK and ciphertext are returned to the data layer of your SaaS application.

"Data privacy concerns are causing significant sales cycle delays for up to 65 percent of businesses worldwide."

- Cisco Privacy Study

On decrypt, the SDK makes a request to the Tenant Security Proxy to unwrap the EDEK. The SDK decrypts the ciphertext and wipes the DEK from memory. The plaintext is returned to the data layer of your application.

IronCore provides a batch API for parallel encryption and decryption operations. The encryption server is horizontally scalable and has an optional key leasing mode discussed below. This enables highly performant batch operations.



Metadata

CMK gives your customers the ability to independently monitor data access. The metadata passed on encrypt and decrypt operations enriches the information included in this audit trail.

```
// Create metadata used to associate this document to a tenant, name the document, and
// identify the service or user making the call
DocumentMetadata metadata = new DocumentMetadata(TENANT_ID, "serviceOrUserId", "document label");
```

Default metadata properties define who (tenant and user), what (data label or classification), and where (end user IP address or data center location). You can augment metadata with custom properties for additional audit, policy and control.

Audit trails are logged to your customer's preferred Security and Incident Event Management (SIEM) system. In addition to the log events produced by your customer's KMS, IronCore's implementation streams enriched events and telemetry to your customer's SIEM infrastructure.

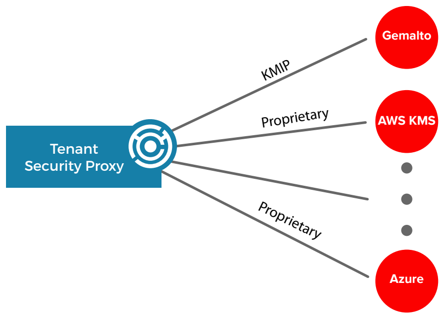
Joins, Filters and Search

CMK does not impact joins as primary and foreign keys should not be encrypted.

Sorting and filtering of encrypted fields can be performed in-memory with no security implications. If in-memory is not possible or practical, an order-revealing or order-preserving encryption scheme may be appropriate.

IronCore works with customers to determine what is needed and appropriate while explaining security trade-offs.

Tenant Security Proxy



The Tenant Security Proxy (TSP) facilitates the encryption and decryption of DEKs and EDEKs. It communicates with your customer's KMS or Hardware Security Module (HSM) to provide control and auditing.

With IronCore, you integrate once and immediately gain support for all popular KMS configurations. The TSP integrates with key management systems and hardware security modules from Gemalto, AWS, Azure, GCP and other vendors.

Leveraging IronCore makes if practical for your SaaS application to implement CMK without the integration complexity and ongoing technical debt associated with supporting, testing and auditing new and updated customer key management infrastructure.

In some configurations, the TSP performs encrypt and decrypt operations in the service using keys leased from the customer's KMS. In other configurations, the customer's KMS performs all of the wrapping and unwrapping.

The Tenant Security Proxy has no persisted state. It is a multi-tenant Docker container that runs in your infrastructure and scales horizontally. It configures itself by calling the Configuration Broker (discussed below) on startup and as needed. IronCore encrypts the configuration used to access tenant KMS systems with end-to-end encryption. Not even administrators can access this information. In a do-it-yourself (DIY) scenario, you need to safeguard KMS secrets carefully.

IronCore's code is independently audited by the NCC Group, a firm that specializes in cryptography services. IronCore transparently publishes security and trust information in our Trust Center where we discuss our SOC2 certification, privacy measures, and other

"Firms should consider greater use of encryption and BYOK, especially for the cloud...to both address growing compliance mandates and also to move to industry best practices."

- 451 Research

industry standards.

The IronCore Tenant Security Proxy generates audit events indicating every access of data, what was accessed, by whom, and from where. IronCore feeds these events to your customer's Security and Incident Event Management (SIEM) system so your customer's security and compliance teams can use their existing monitoring and reporting systems rather than needing to adopt and look at yet another dashboard. A wide variety of SIEMs are supported. IronCore audit events bolster the logs generated directly from your customer's KMS with richer details needed for data maps, investigations, and compliance reporting. Audit events and telemetry information are also available to your application.

Improving Performance and Availability with Key Leasing

The CMK pattern uses a remote call to your customer's network every time data is encrypted or decrypted. Remote calls introduce latency whenever sensitive data is accessed. The reliance on your customer's network to be online and reachable introduces uptime risks that threaten the availability terms of your service level agreements (SLAs).

Key leasing is an alternate approach supported by IronCore's Tenant Security Proxy. Key-leasing relies on three layers of keys, in which the customer's KMS delegates limited control to the IronCore Tenant Security Proxy.

Key leasing uses three layers of keys		
Definitions	LEK	The lease encryption key (LEK) is delegated by the KMS to serve as a local key encryption key.
	MK	The LEK is encrypted (wrapped) by a Master Key (MK). In key leasing, the envelope encryption pattern is applied twice. The MK is the key encryption key for the LEK and the LEK is the key encryption key for the DEK. The MK never leaves the KMS.
	ELEK	The encrypted lease encryption key (ELEK).

Key leasing uses three layers of keys		

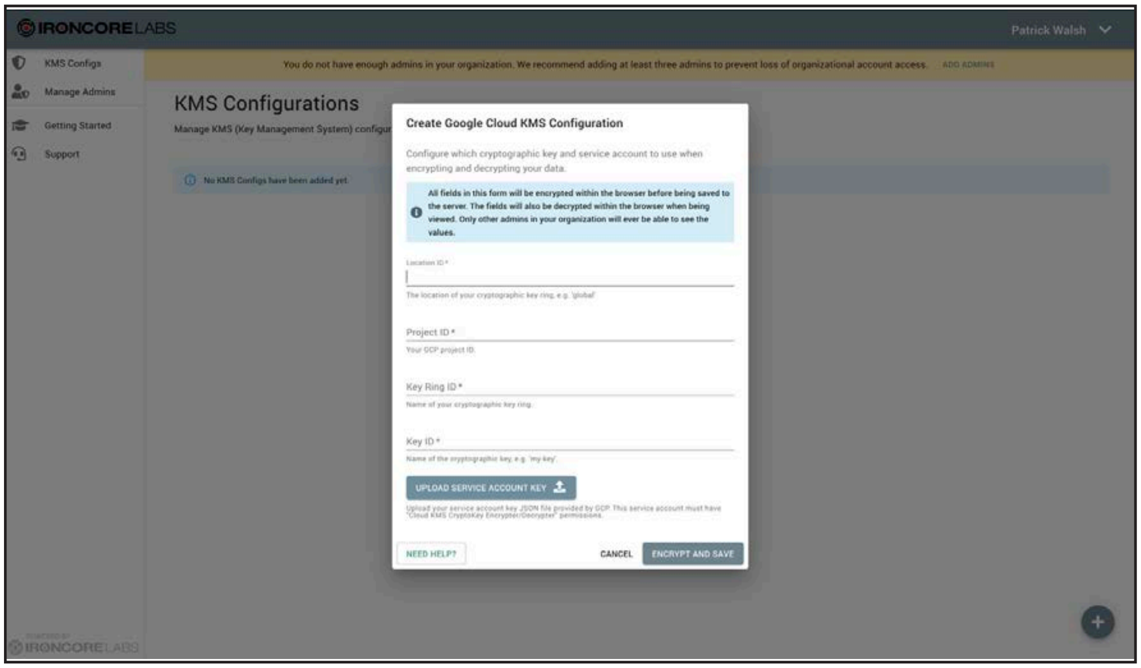
Operations	lease / refresh	`// TSP calls KMS, caches LEK with short Time-To-Live`
		`{ LEK, ELEK } = leaseKeys()`

	encrypt$_{3\text{-level}}$	`{ EDEK, ciphertext } = encrypt(plaintext)`
		
		`composed by...`
		
		`DEK = generateKey()    // TSP`
		`EDEK = wrap(DEK, LEK)  // TSP encrypts DEK with LEK`
		`ciphertext = encrypt(plaintext, DEK) // SDK`

	decrypt$_{3\text{-level}}$	`plaintext = decrypt(ciphertext, EDEK)`
		
		`composed by...`
		
		`DEK = unwrap(EDEK, LEK) // TSP decrypts EDEK with LEK`
		`plaintext = decrypt(ciphertext, DEK) // SDK`

You can implement key leasing as part of a DIY implementation, but it requires more trust from your customer. Using a third-party encryption service, even if run locally, mitigates that issue.

Customers typically embrace key leasing for better performance and availability. Availability is improved because keys are leased asynchronously and decryption does not need to wait for a lease renewal. By default, IronCore renews leases every 5 minutes but will hold a key for up to an hour before wiping the leased key if the remote key server remains unavailable. See the sequence diagram in Appendix A for additional details.



You do not have enough admins in your organization. We recommend adding at least three admins to prevent loss of organizational account access.

KMS Configurations

Manage KMS (Key Management System) configur

No KMS Configs have been added yet.

Create Google Cloud KMS Configuration

Configure which cryptographic key and service account to use when encrypting and decrypting your data.

All fields in this form will be encrypted within the browser before being saved to the server. The fields will also be decrypted within the browser when being viewed. Only other admins in your organization will ever be able to see the values.

Location ID *

The location of your cryptographic key ring, e.g. 'global'

Project ID *

Your GCP project ID.

Key Ring ID *

Name of your cryptographic key ring.

Key ID *

Name of the cryptographic key, e.g. 'my key'.

Upload your service account key JSON file provided by GCP. This service account must have "Cloud KMS CryptoKey Encrypter/Decrypter" permissions.

Sample CMK Offerings:

Slack EKM
Salesforce Shield
Box KeySafe
Microsoft BYOK

If you implement key-leasing, your customer's KMS can no longer independently log data access events. IronCore's key-leasing implementation streams rich audit events to your customer's SIEM infrastructure in this mode.

Configuration Broker

The IronCore configuration broker lets your customer configure the many settings that are part of their security policy, key management infrastructure, and SIEM system. It is branded with your logo as a SaaS provider and allows your customers to self-serve.

The IronCore Configuration Broker is in a zero trust position and uses end-to-end encryption. IronCore encrypts sensitive information, such as KMS access credentials, in the browser of your customer's administrator, and the configuration can only be decrypted by the Tenant Security Proxy instances at point of use. The encrypted configuration is stored by IronCore for distribution to Tenant Security Proxy instances. IronCore cannot read the sensitive information it brokers.

In a DIY implementation, you should be careful with how you handle your customer's sensitive configuration information and have your code independently audited.

Note that as its own SaaS application, the Configuration Broker is continually updated with new integrations.

Native KMS Management

Key lifecycle management is the concept that keys are "born," serve a useful lifetime, and are eventually archived or revoked. Your customer's security organization has strict policies governing key lifecycles.

Once your customer initially configures CMK via the configuration broker, they perform all key lifecycle management from their native KMS or HSM. IronCore ties encrypted information back to the KMS and key that was used to encrypt the data, and automatically tracks key rotations. Your customers prefer to maintain control using their KMS, and it reduces your technical support costs.

A Note on Trust Models and End-to-End Encryption

Trust is a characteristic of a security architecture. CMK is a trust-but-verify model, in which you encrypt and decrypt data on the server-side. You can access data, but your customer has control over encryption keys.

End-to-end encryption uses a stronger zero-trust model. By default, services do not have access to encrypted data. You typically encrypt and decrypt on the client or from a cloud service with privileged identity.

IronCore provides end-to-end encryption options in addition to CMK, and the approaches can be mixed. We recommend that SaaS vendors begin with server-side CMK and selectively and iteratively provide client-side protections for the most sensitive data. For example, it's common for SaaS applications to provide CMK protection of records and fields and end-to-end encryption of sensitive, opaque file attachments.

Iterative Approaches

Most commercial implementations of data privacy and security are delivered iteratively. For example, when Salesforce launched their Shield Encryption Platform, only custom fields and manual key management were supported. Over several years they added attachments and selected standard fields. With each release, the list of standard fields has grown. In Winter 2019 they are extending encrypted fields to applications listed on AppExchange, and adding a "cache-only key service" which provides true CMK support.

"Buyers of cloud service and mobile devices should demand that providers offer them the option of managing their own encryption keys."

- Gartner

To get started in your own system, look to the sensitivity and shape of your data. Sensitivity is a data classification such as restricted, private or public. Shape is the format of your persisted data - files or buckets, database records and fields, log files, search indices or message queues.

Don't start out encrypting all data in all shapes. Identify the most critical and sensitive data stored in easily supported systems (e.g., files or buckets). Add more data elements and additional persistence systems over time. Ship progress to immediately shorten sales cycles and create market differentiation.

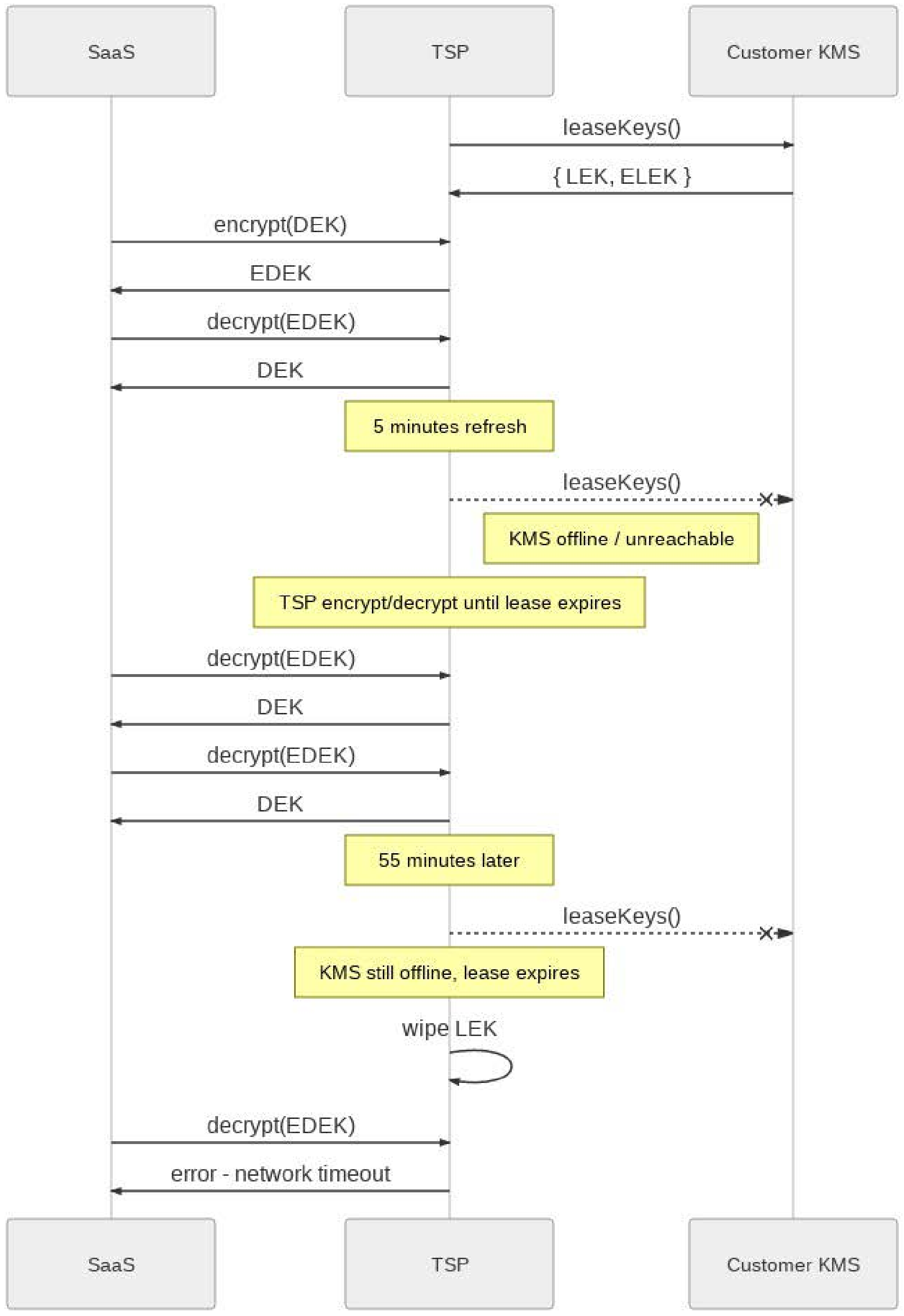
CONCLUSION

For many large enterprises, Customer Managed Keys (CMK) are becoming a baseline requirement for cloud software. IronCore provides a turnkey solution that is quickly and easily integrated to get you to market faster with security that is a competitive differentiator.

With IronCore, you integrate once to support many KMSes. Your customers get a secure interface to manage their KMS configuration without providing that data to you. For more sensitive data, you can encrypt end-to-end on the same platform. The developer experience is simple and continuously improved. NCC Group audits critical security code. Policy driven controls and rich logging are supported out of the box. Performance and availability are optimized.

Most importantly, integrating IronCore accelerates your roadmap and eliminates complex security technical debt. Cloud security is evolving rapidly in response to new government regulation and increasing system complexity. IronCore tracks these changes and keeps you current with the latest, most secure options for you and your customers.

APPENDIX A: KEY LEASING AND AVAILABILITY



IRONCORE LABS

ABOUT IRONCORE

We are a data privacy platform for application layer encryption and customer managed keys (CMK). We enable software developers and businesses to rapidly build enterprise applications with strong data control. Data owners decide who can access their data, monitor how it's used, when, where, and by whom, and can revoke that access at any time. We are the fastest and easiest way to control data in multi-cloud and SaaS environments.

IronCore Labs
1750 30th Street #500
Boulder, CO 80301, USA

Inquiries
Email: info@ironcorelabs.com
Phone: +1.415.968.9607

CONNECT WITH US

- blog.ironcorelabs.com
- linkedin.com/company/ironcore-labs
- twitter.com/ironcorelabs
- ironcorelabs.com

1.0.1