Salesforce Shield Platform Encryption Architecture

Version 64.0, Summer ‘25

Last updated: May 16, 2025

CONTENTS

Chapter 1: About this Document 1

Chapter 2: Why Encrypt 2

Balance Data Security with Business Needs 3
Salesforce Encryption Principles 3
Encryption Components: The Big Picture 5
Before You Encrypt 5
What Gets Encrypted? 6
Major Differences Between Database Encryption and Field-Level Encryption 8

Chapter 3: How Shield Platform Encryption Works 9

Field-Level Encryption Happens in the Application Layer 10
Database Encryption Happens in the Transactional Database Layer 10
Field-Level Encryption and Database Encryption Work Together 11
Special Database Encryption Considerations 11
Data Encryption Keys (DEKs) 11
Data Encryption Keys for Database Encryption 13
Storing Encrypted Payloads for Field-Level Encryption 13
Storing Encrypted Data for Database Encryption 13
Shield Platform Encryption Process Flow for Field-Level Encryption 14
Search Encryption at Rest Process Flow 15
Database Encryption Process Flow 16
Key Management for Field-Level Encryption 18
Key Management for Database Encryption 19

Chapter 4: Data Encryption Key Derivation 20

Key Derivation Architecture 21
HSM Initialization 22
Per-Release Secret Generation and Export 22
Shield KMS Startup 24
On-Demand Tenant Secret Generation 24
Customers Can Supply Their Own Key Material with EKM, BYOK, and Cache-Only Keys 25
    External Key Management (EKM) 26
    Bring Your Own Key (BYOK) 27
    Cache-Only Keys 28
External Key Management Flow 28
BYOK Tenant Secret Flow 30
Cache-Only Key Flow 31
Encrypted Information Flow with Key Derivation 33

Chapter 5: Keys and Secrets 35

CHAPTER 1 About this Document

Welcome to the Salesforce Shield Platform Architecture Guide.

Shield Platform Ecryption is available as an add-on subscription in: **Enterprise**, **Performance**, and **Unlimited** Editions. Requires purchasing Salesforce Shield. Available in **Developer** Edition at no charge.

This guide provides information on Shield Platform Encryption, not Classic Encryption. (What's the difference?). The guide includes information on these topics.

- General encryption details, our encryption philosophy, and considerations on implementing encryption policies in your org
- Details and flow descriptions of the key management processes of each feature.
- Technical description of our key derivation architecture
- Detailed descriptions of each of the encryption features available in Shield Platform Encryption:
  - Field-level encryption (FLE)
  - Search encryption
  - Database Encryption
  - External Key Management (EKM)
  - Bring Your OwnKeys (BYOK)
  - Cache-Only Keys (CoK)

Note: This document includes information about Database Encryption. Database Encryption hasn't yet been deployed to all Hyperforce regions and instances. For product availability and purchasing information, contact your account executive.

CHAPTER 2 Why Encrypt

In this chapter ...

- Balance Data Security with Business Needs
- Salesforce Encryption Principles
- Encryption Components: The Big Picture
- Before You Encrypt
- What Gets Encrypted?
- Major Differences Between Database Encryption and Field-Level Encryption

Security and trust are major factors in every company's evaluation of public cloud services. Salesforce customers choose which business functions to run on the Salesforce Platform, which applications they can build to extend those functions, and what data they must store to enable those functions.

The 2023 ITRC Data Breach Report counted 3,205 confirmed compromises worldwide in 2023. In addition, the report estimates over 353 million victims in the same year. Compromises involving sensitive personal information remained the most common type of breach in 2023.

Customers increasingly use the Salesforce Platform to build applications that require personally identifiable information (PII) and other sensitive, confidential, or proprietary data. Standard features such as authentication and single sign-on, granular access controls, and activity monitoring give most customers enough control over when and how they protect their data. But when sensitive data is stored on the Salesforce Platform, some customers want additional layers of protection on top of our standard security measures.

Balance Data Security with Business Needs

Choosing to store PII, sensitive, confidential, or proprietary data with any third party often prompts customers to more closely investigate external regulatory and internal data compliance policies. Internal policies frequently rely on interpretation of external regulations.

As customers look at regulations through the lens of cloud-based service adoption, they typically take a pragmatic but conservative approach to data protection in the cloud. Examples of such regulations are the Payment Card Industry Data Security Standard (PCI DSS), Health Insurance Portability and Accountability Act (HIPAA)/Health Information Technology for Economic and Clinical Health Act (HITECH), General Data Protection Regulation (GDPR), and Federal Risk and Authorization Management Program (FedRAMP).

This pragmatic approach includes three requirements shared by a wide variety of customers in regulated industries such as financial services, healthcare, and life sciences, as well as manufacturing, technology, and government.

1. Encrypt sensitive data when it's stored at rest in the Salesforce cloud.
2. Support customer-controlled encryption key lifecycles.
3. Preserve application and Salesforce Platform functionality.

However, there's a tradeoff between strong security and functionality. Data encrypted at rest can make preserving Salesforce functionality difficult, if not impossible. The degree depends on where encryption and decryption occur and where the encryption keys are stored. What the business wants often differs from what security and compliance require.

Salesforce Encryption Principles

To balance security demands with customers' functional requirements, Salesforce defined a set of principles that drove our solution design and architecture decisions. We focused on the problems that we wanted to solve, clearly defined the boundaries of our solution, and identified the implications and tradeoffs of the design.

Encrypt data at rest

The Salesforce Shield Platform Encryption solution encrypts data at rest when stored on our servers, in the database, in search index files, and in the file system. To encrypt data at rest and preserve functionality, we built the encryption services natively into the Salesforce Platform.

Natively integrate encryption at rest with key Salesforce features

One of the things that makes the Lightning Platform so remarkable is that it's driven by metadata. Shield Platform Encryption uses that metadata to tell the other platform features which data is encrypted. This way, we can prevent those features from inadvertently exposing plaintext or ciphertext. And we can ensure that critical business functionality, like partial search, continues to work even when data is encrypted.

Note: If your org is running on Hyperforce, Salesforce can apply Database Encryption to the transactional database.

Use strong, flexible encryption

The Shield Platform Encryption solution provides two features that approach data encryption in different ways: Field-Level Encryption (FLE) and Database Encryption.

- The Shield Platform Encryption FLE encryption-at-rest solution runs on the application tier. It uses strong, probabilistic encryption by default. FLE uses the Advanced Encryption Standard (AES) with 256-bit keys by using Cipher Block Chaining (CBC) mode and a random initialization vector (IV). This type of encryption results in a loss of some functionality, such as filtering operations.

  However, we recognize that in some cases, business requirements depend on preserving more functionality, which can influence what data customers decide to encrypt. In those cases, we offer deterministic encryption for FLE. Deterministic encryption also uses AES 256-bit CBC keys, but it uses a field-specific static IV. Because the ciphertext for each field includes bits that associate it with a specific field, customer queries return encrypted values in that field. In this way, deterministic encryption only decreases encryption strength as much as is necessary to allow filtering.

- The Database Encryption encryption-at-rest solution runs on the transactional database tier, so all encryption operations occur within the database. It uses AES 256-bit keys and a random IV. It also uses Galois/Counter (GCM) mode instead of CBC. Further, each data fragment is encrypted by using a unique key and unique IV. No two data fragments reuse the same encryption key or IV. With Database Encryption, all transactional data is encrypted when written, and it's decrypted for every read operation. As a result, Database Encryption is transparent, even for sorting and filtering operations.

Let customers drive the key lifecycle

We built a key management framework that scales to our multitenant model and gives you complete control over the key management lifecycle. Because the encryption service is built natively into the Salesforce Platform, the encryption keys can reside in the Salesforce environment. Or, when accessible by the Salesforce Shield Key Management Service, keys can reside in an external key store. Adhering to the principle that customers must have complete control over the key lifecycle, we built key management functionality into the Setup UI and API. Customers decide when to generate, supply, rotate, import, export, and destroy keys. Customers also determine who performs these tasks. With Bring Your Own Keys (BYOK), you can generate and store key material outside of Salesforce using your own crypto libraries, enterprise key management systems, or hardware security modules. As with all administration tasks, everything is audited.

Note: You can archive and destroy FLE key material. Database Encryption key material can only be archived. You can't destroy it.

Protect keys from unauthorized access

A primary consideration when designing our key management infrastructure was making encryption keys available to the encryption service while preventing privileged Salesforce employees, such as DBAs, from inappropriately accessing them. This consideration led us to incorporate these key security concepts in Shield Platform Encryption.

- We use a hardware security module (HSM), a High Assurance Virtual Ceremony (HAVC), and the Salesforce Shield Key Management Service to generate and manage tenant cryptographic key material.
- The Salesforce Shield Key Management Service includes a primary Key Management Server (KMS), a Key Escrow Server. and secondary regional KMSs, which securely distribute key material as needed. Each regional KMS includes a key broker that securely controls the transfer of key material from the Key Escrow Server.
- Derived data encryption keys (DEKs) used to encrypt and decrypt customer data are never stored in plaintext. They're derived from multiple key material components, including a per-release secret (KDF seed), a per-release initialization vector (KDF salt), and a tenant secret. DEKs generated by a root key are wrapped by that root key and stored in the database. DEKs supplied by the customer aren't persisted, but injected directly into the encrypted key cache.
- Key material components that are persisted outside of an encrypted key cache are always wrapped and hashed or signed using HSM-generated wrapping and signing keys.
- With FLE, for final encryption and decryption of data, an encrypted key cache temporarily stores all key material. All keys in the encrypted key cache are wrapped by a cache key encryption key, which is specific to each customer.

The result is a shared key management service that secures all key material at every stage. Keys, secrets, and key components are inaccessible to Salesforce employees and, by extension, to malicious external attackers.

Encrypt the optimal amount of data

Our design gives customers control over what data they encrypt. As the administrator, you choose whether to turn on encryption for standard fields, custom fields, files, attachments, and more. You also choose which specific fields to encrypt at rest. The driving principle is to use the appropriate encryption features to preserve functionality while keeping private, sensitive, confidential, and regulated data safe.

Note: Due to platform optimizations, there's a limit of 80 encrypted fields during org migrations. If you set up your encryption strategy with 80 or more encrypted fields, you must do a phased rollout of org migrations.

Encryption Components: The Big Picture

Central to encryption are the secrets and keys that the platform uses to encrypt and decrypt data. Read about the different components involved with Shield Platform Encryption.

Throughout this guide, we refer to nearly a dozen types of secrets, materials, and keys that participate in the Shield Platform Encryption process. Before reading further, it's helpful to understand how these parts relate to each other.

The algorithm that actually encrypts your data uses a data encryption key (DEK). By default Salesforce DEKs are derived by using a special key derivation function (KDF). Because they're derived, Salesforce never writes them to disk, which means that nobody ever has direct access to them. DEKs are created when they're needed, and they're stored in a secure memory cache. Customer-supplied DEKs can be used in key derivation, or they can be used as final encryption keys.

Each DEK is derived by using up to 3 types components, or key materials. One of these key materials is a cryptographic key that Salesforce contributes, known as the KDF seed. One is a cryptographic key that you contribute, which is your tenant secret or database tenant secret. And the platform contributes the third one, which is an initialization vector (IV) or salt.

Salesforce also supports root key generated DEKs. For these DEKs, the root key is hosted in the regional Shield KMS or in an external KMS, and the generated DEK is a fragment (for derivation) or a complete encryption key (when customers opt out of derivation).

In this guide, we define the different types in these ways.

- seed — A value that's used in a KDF function to generate a key, such as our KDF seed and database tenant secret.
- secret — The term "secret" can be confusing because people often use it in place of key, seed, or salt. A tenant secret or a database tenant secret is one of the inputs to a KDF to generate an application-tier DEK.
- salt — A generic term for a random number to a KDF algorithm to generate a DEK. The KDF salt is generated once per release, and a Database Encryption salt is generated for each fragment that's encrypted.
- initialization vector — A random value produced for FLE and Database Encryption. We use IV for this term throughout this document.

Every release, Salesforce creates a new KDF seed and a new KDF salt for application tier encryption. Both are created during a High Assurance Virtual Ceremony (HAVC) by Salesforce cryptographic administrators. The KDF seed and KDF salt are used to create org-specific tenant secrets.

Every tenant can periodically create a tenant secret or a database tenant secret for rotation. Database Encryption creates a new per-fragment salt for each fragment that it encrypts.

The KDF seed, KDF salt, root key, tenant secret, root key, DEK, and database tenant secret are all secret cryptographic key components. It's not necessary for the IVs to be secret, but they must be random.

When a DEK is needed, the required components are passed to the KDF. For the application tier, a DEK needs the KDF seed, the tenant secret, and the probabilistic or deterministic IV. For Database Encryption, a DEK needs the database tenant secret and the per-fragment database encryption salt.

Root keys are special keys that are used solely to generate, wrap, and unwrap DEKs.

Wrapping keys, or key encryption keys (KEKs), are separate cryptographic keys that are used to protect key material and DEKs when they're moved or cached.

A glossary at the end of the guide defines all the different secrets and material types that we discuss.

Before You Encrypt

Before you encrypt data in Salesforce, or in any cloud service, first make sure that you're matching the right security solution to the type of threats that you face.

For example, if you're concerned about protecting against end-user or administrative account takeover attacks, it's possible that data encryption isn't an appropriate control against such a threat. Takeover attacks usually occur through social engineering and malware infection. Instead, consider malware detection and activity monitoring as ways to identify when users could have been compromised and a malicious outsider is attempting to gain access to data.

Salesforce Shield Platform Encryption protects data at rest. Don't confuse it with a control that encrypts data in transit, such as Transport Layer Security (TLS), which Salesforce provides for your org by default.

Shield Platform Encryption is best suited for:

- Protecting against data compromise due to unauthorized database access
- Bolstering compliance with regulatory requirements or internal security policies
- Satisfying contractual obligations to handle sensitive and private data on behalf of customers

The best approach is to adopt a defense-in-depth strategy that takes advantage of all the security features Salesforce offers. To learn about the available customer-controlled security capabilities, take the Security Basics Trail.

After completing a threat modeling exercise, use the outcome to inform a granular data classification. (Keep the Salesforce Security Guide handy as you work through this project.) Identify data elements that are sensitive, private, or confidential. Your best strategy is to encrypt only the most sensitive of those data elements. Doing so can help balance stronger data protection controls against the need to build and preserve critical business functionality on the Salesforce Platform.

What Gets Encrypted?

You can encrypt a variety of fields, files, and data with Shield Platform Encryption. Salesforce uses metadata to keep information in these files and fields secure while preserving the ability to perform common business tasks.

In contrast to Classic Encryption, which uses a custom field type in the Salesforce data model, Shield Platform Field-Level Encryption makes more fields, files, and data elements available for encryption with every release. For the complete list of fields that you can encrypt with Shield Platform Encryption, see What Can You Encrypt? in Salesforce Help. With Database Encryption, every database transaction is encrypted, so all fields are encrypted implicitly. Field-Level Encryption occurs before Database Encryption.

This table compares the features of Shield Platform Encryption and Classic Encryption:

Feature	Classic Encryption	Field-Level Encryption	Database Encryption
Pricing	Included in base user license	Additional fee applies	Additional fee applies
Encryption at Rest	✓	✓	✓
Native Solution (No Hardware or Software Required)	✓	✓	✓
Encryption Algorithm	128-bit Advanced Encryption Standard (AES)	256-bit Advanced Encryption Standard: AES CBC (Field-Level Encryption), AES CTC (Search Index Encryption)	256-bit Advanced Encryption Standard (AES GCM)
HSM-based Key Derivation	✗	✓	✓
Manage Encryption Keys Permission	✗	✓	✓

Feature	Classic Encryption	Field-Level Encryption	Database Encryption
Generate Keys	✓	✓	✓
Store Encryption Keys Outside of Salesforce	✗	✓	✗
Export, Import, and Destroy Keys	✓	✓	✗
Advanced Key Options	✗	EKM, BYOK, Cache-only Keys, and Root Keys	BYOK
PCI-DSS L1 Compliance	✓	✓	✓
Masking	✓	✗ No (Why Isn't my Encrypted Data Masked?)	✗ No (Why Isn't my Encrypted Data Masked?)
Mask Types and Characters	✓	✗	✗
View Encrypted Data Permission Required to Read Encrypted Field Values	✓	✗	✗
Encrypted Standard Fields	✗	✓ Limited (What Standard Fields Can You Encrypt?)	✓ All standard fields
Encrypted Attachments, Files, and Content	✗	✓	✓
Encrypted Custom Fields	Dedicated custom field type, limited to 175 characters	✓ Limited (What Custom Fields Can You Encrypt?)	✓ All custom fields
Encrypt Existing Fields for Supported Custom Field Types	✗	✓	✓
Encrypt Custom Metadata and Apex	✗	✗	✓
Search, Filters, and Queries	✗	✓ UI, Partial Search, Lookups, Certain SOSL Queries. Certain SOQL Queries with Deterministic	✓ All SOSL and SOQL Queries except on fields also encrypted with Field-Level Encryption
Sorting	✗	✗	✓ Except on fields also encrypted with Field-Level Encryption

Feature	Classic Encryption	Field-Level Encryption	Database Encryption
Encrypt the entire database including standard fields, custom fields, metadata, and Apex. (Database Encryption)	✗	✗	✓
Available in Workflow Rules and Workflow Field Updates	✗	✓	✓
Available in Approval Process Entry Criteria and Approval Step Criteria	✗	✓	✓

Major Differences Between Database Encryption and Field-Level Encryption

Database Encryption and Field-Level Encryption are different features, and each has different advantages and limitations.

- To benefit from Database Encryption, your org must be in Hyperforce.
- FLE supports Salesforce-generated keys, Bring Your Own Key (BYOK), Cache-only keys, and External Key Management (EKM) options.
- Database Encryption supports Salesforce-generated keys and BYOK.
- FLE provides the ability to archive and destroy keys.
- Database Encryption supports key archiving.
- FLE provides the ability to synchronize all of your encrypted data with the most recent encryption key by hand. Database Encryption doesn't.
- FLE has some restrictions on filtering, querying, searching, and sorting.
- Database Encryption provides full filtering, querying, searching, and sorting of the data that it encrypts.

SEE ALSO:

Special Database Encryption Considerations

CHAPTER 3 How Shield Platform Encryption Works

In this chapter ...

- Field-Level Encryption Happens in the Application Layer
- Database Encryption Happens in the Transactional Database Layer
- Field-Level Encryption and Database Encryption Work Together
- Special Database Encryption Considerations
- Data Encryption Keys (DEKs)
- Data Encryption Keys for Database Encryption
- Storing Encrypted Payloads for Field-Level Encryption
- Storing Encrypted Data for Database Encryption
- Shield Platform Encryption Process Flow for Field-Level Encryption
- Search Encryption at Rest Process Flow
- Database Encryption Process Flow
- Key Management for Field-Level Encryption
- Key Management for Database Encryption

Shield Platform Encryption meets the security requirements of customers while preserving functionality and performance in our multitenant environment. To make this happen, we provide encryption features in two separate tiers: in the application tier with FLE and in the transactional database tier with Database Encryption.

The platform's object-relational mapping model includes metadata that describes exactly which data is encrypted. Encrypted data is stored with additional information that uses strong, nondeterministic cryptography supported by the Java Cryptographic Extension (JCE). Encryption and decryption occur in the platform's application layer as application components are materialized by the runtime engine, ensuring that encrypted data doesn't persist in plaintext.

Field-Level Encryption

The platform's object-relational mapping model includes metadata that describes exactly which data is encrypted. Encrypted data is stored with additional information that uses strong, nondeterministic cryptography supported by the JCE. Encryption and decryption occur in the platform's application layer as application components are materialized by the runtime engine, ensuring that encrypted data doesn't persist in plaintext. The Shield Key Management Service derives keys on demand from key material generated by HSMs. These derived keys are never persisted.

Database Encryption

Every write operation to the transactional database is encrypted by using a database tenant secret, a per-fragment Database Encryption salt, and a per-fragment random 12-byte nonce. Database Encryption uses strong, nondeterministic cryptography supported by the JCE. Encryption and decryption occur in the platform's transactional database layer. Every database write is encrypted, and every data read is decrypted.

Encryption Keys

The Shield Key Management Service derives keys on demand from key material generated by Hardware Security Modules (HSMs). These derived keys are never persisted. For Field-Level Encryption, you can opt out of key derivation and supply a final data encryption key. You can choose to store keys in the tenant database, where they're wrapped by a tenant wrapping key. (For Database Encryption BYOK, you must provide a full data encryption key, so there is no need to opt out of derivation.) Alternatively, for FLE, you can choose to bypass storage and add the key directly to the encrypted key cache, where it's wrapped by the cache key encryption key. The architecture supports the simultaneous use of multiple encryption keys, enabling customers to quickly rotate and archive keys without losing access to their data. Rotated FLE keys are archived unless you destroy them. Database tenant secret keys are archived but can't be destroyed. With EKM, we support key material generated and wrapped by root keys generated by external KMSs.

Field-Level Encryption Happens in the Application Layer

The Lightning Platform's foundation for FLE is a metadata-driven software architecture that enables multitenant applications.

Application components, such as Salesforce objects, aren't modeled directly in our underlying relational database. Instead, when customers interact with their data in a Salesforce application, the platform's runtime engine materializes the data by using metadata stored separately in the Lightning Platform's Universal Data Dictionary (UDD).

This way, each tenant's data is kept secure in the shared database, and tenants can customize their schema in real time without affecting other tenants' data. Also, the application's code base can be patched or upgraded without breaking tenant-specific customizations. See Platform Multitenant Architecture for details.

The UDD includes metadata that determines which data is encrypted at runtime. The encryption service works in the Lightning Platform's application layer. That is, data is encrypted directly before it's stored in the database. The resulting encrypted payload is stored with metadata about the specific key used to encrypt it. For decryption, data is decrypted as it's materialized. It's then pushed up through the application pipeline and appears in plaintext to the user who requested it.

By embedding the encryption metadata in the UDD, the Shield Platform Encryption architecture gives customers the option to choose what data to encrypt. The Shield Platform Encryption engine strictly manages the flow of encrypted data from the application layer to the database and vice versa. The relevant code path is run through the UDD before data is read or stored.

Cryptographic Library and Algorithms

For FLE, Shield Platform Encryption uses the JCE to encrypt and decrypt data. Specifically, Shield Platform Encryption uses the Advanced Encryption Standard (AES- 256) in CBC mode with, by default, a random IV. Deterministic encryption uses a field-specific static IV, which is a hash of the entity ID, field ID, and key ID. This method makes the static IV unique to each customer and field per org. Upon key rotation, a new static IV is generated for each field.

KMS Encrypted Key Cache

Salesforce has taken steps to reduce access to encryption key material across the servers where encrypted keys are available for the application servers. Rather than holding data encryption keys on the application server, they're temporarily held in an encrypted key cache on a regional KMS. This feature is a central service that all application servers in a particular region can access. When the customer accesses encrypted data on the application server, the server uses a DEK for a short time to perform encryption and decryption operations. After this transaction is complete, related memory containing the DEK is freed. The DEKs are never persisted on the application server, and they're only available in the KMS cache that serves the application server's region.

Database Encryption Happens in the Transactional Database Layer

The Lightning Platform's foundation for Database Encryption is a fragment-driven architecture that supports multitenant transactional operations.

Unlike FLE, which is applied at the application tier, Database Encryption is applied at the transactional database tier. It encrypts all data in the transactional database without impeding filtering, sorting, or interfering with the many Salesforce features that rely on those actions. All transactional data, including standard fields, custom fields, custom metadata, and apex data is encrypted.

Cryptographic Library and Algorithms

For Database Encryption, Shield Platform Encryption uses the JCE to encrypt and decrypt data. Specifically, Shield Platform Encryption uses the AES- 256 in GCM mode with a random IV.

Like encryption at the field level, the secure key materials for Database Encryption are retrieved from the regional KMS. With Database Encryption, the encryption services reside within the transactional database and apply encryption at the database fragment level. That is, the smallest unit of encryption for Database Encryption is a database fragment, typically 64 KB or smaller.

Field-Level Encryption and Database Encryption Work Together

Even if you're using Database Encryption, you can still use FLE on a field-by-field basis as required for security and compliance.

FLE happens at the application tier, and Database Encryption happens at the database tier. As a result, when a user saves data, the fields configured for encryption at the application tier are encrypted first. When control passes to the database layer, the database fragment that contains the encrypted field is encrypted by Database Encryption. So, when both FLE and Database Encryption are in use, the FLE field data is encrypted twice.

Though you can use FLE and Database Encryption at the same time, their encryption processes are completely separate. The data encryption keys are derived from different key material.

Special Database Encryption Considerations

Certain aspects of transactional databases require delays before keys are used and before data can be encrypted.

- **Field-Level Encryption Happens Before Database Encryption.** FLE encryption happens before data is written to the transactional database. As a result, existing FLE limitations such as a lack of sorting still apply.
- **Database Tenant Secret Delay.** When you generate a new database tenant secret, it's listed on the Key Management page in Setup along with the rest of your org secrets, and it's stored in the regional Salesforce Shield KMS.

  Though your database tenant secret is created, Salesforce security policy requires that your database tenant secret be backed up before it can be used for encryption. This backup policy is true whether you create a Salesforce-generated database tenant secret or supply it via BYOK. Salesforce allocates up to 24 hours for a complete database tenant secret backup. Between the time that you create or upload your database tenant secret and until the time it's used, Salesforce encrypts your transactional data by using a Salesforce-controlled temporary system database tenant secret. This system seed is used only to create DEKs for data in your org, and it becomes one of your org's permanent Database Encryption seeds.
- **Hyperforce is Required.** Database Encryption is available for Hyperforce customers with a Shield or Shield Platform Encryption license. If your org isn't running on a Hyperforce instance, Database Encryption isn't available for that org.
- **Database Encryption Seeds may not be Deleted.** An encrypted database has a large number of encrypted fragments. To decrypt a fragment's data, the database tenant secret used for encrypting it must be available. You can create a new database tenant secret when you need one, but you can't delete Database Encryption seeds as you can other types of tenant secrets. The Salesforce transactional database requires that all keys ever used for Database Encryption be preserved permanently.
- **Database Encryption supports Salesforce-generated keys or Bring Your Own Key (BYOK).** However, cache-only keys and EKM are features for FLE only.
- **Database Encryption and Field-Level Encryption IV Differences.** Database Encryption uses a random IV for each fragment, so the database is in a constant state of gradual encryption change. FLE uses a random IV for probabilistic encryption and a fixed IV for deterministic encryption.

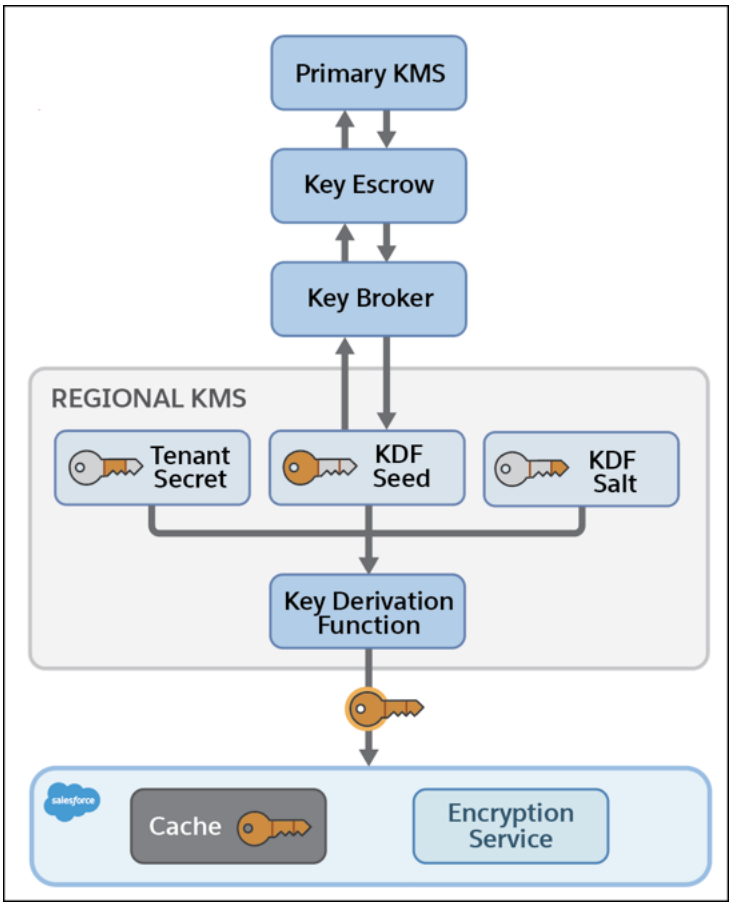
Data Encryption Keys (DEKs)

The Shield Platform Encryption services can use three types of DEK: derived, root key created, or customer supplied. All have AES-256 bit strength. Derived DEKs and DEKs supplied by the customer aren't persisted, but exist only in the encrypted key cache. Root key created DEKs are wrapped by the root key and stored in the database and unwrapped by the same root key when needed.

Derived DEKs

Customer-specific key material for use in derivation is supplied by customers or generated by customers on demand, and then stored securely wrapped in the database. Derived DEKs are generated on demand from the KDF seed and KDF salt secrets generated by the HSM.

The KDF seed and KDF salt are used as inputs to Password-Based Key Derivation Function 2 (PBKDF2) to derive the DEK. PBKDF2 is run on the Shield Key Management Service in a Salesforce data center. Data encryption keys are distributed via the Key Escrow server to the regional Shield KMS and stored in its encrypted key cache. The KDF seed and KDF salt are generated at the start of each Salesforce release and stored securely in the primary KMS. They're used to derive DEKs on demand.



Customer-Supplied DEKs

Customers who opt out of key derivation (for BYOK or Cache-Only Keys) generate the DEK outside of Salesforce. The DEKs are supplied to Shield Platform Encryption through a secure channel (TLS). Or, they can be fetched on-demand from a customer-specified external key management service.

- For BYOK, the DEKs are wrapped and stored in the database until needed. Then they're unwrapped by the regional Shield KMS or External KMS and placed in the encrypted key cache.
- For Cache-Only Keys, the DEKs are injected directly into the encrypted key cache.

Root Key Wrapped DEKs

For those services that use root keys, such as Search encryption, the customer first generates a root key in the regional Shield KMS or in an external KMS. When the root key is active, the customer triggers generation of a DEK, also within a KMS. The KMS then uses the root key to wrap the DEK. The wrapped DEK is sent back to the application or service that requires the DEK, and it's stored in the Salesforce database until needed. When needed, the DEK is sent to the issuing KMS and unwrapped using the root key. Then the unwrapped DEK is returned to the application over a secure channel and stored.

Data Encryption Keys for Database Encryption

Similar to FLE, DEKs for Database Encryption are either generated or supplied by the customer.

The database tenant secret (generated or BYOK) is used as one part of the final derived DEK. The database tenant secret is used to derive the final tenant-specific per-fragment DEK that is used for encryption and decryption in the database. Having per-fragment keys mitigates the potential concerns with AES GCM and key overuse.

The parts that contribute to the derived key are:

- Your database tenant secret (Salesforce generated or BYOK)
- A per-data-fragment randomly generated 128-bit salt

These components are supplied to a key derivation function (KDF) to generate a different DEK for every database fragment. The key derivation function is based on openssl HKDF. Unlike FLE, where the derivation happens on a key derivation server, with Database Encryption, the derivation happens within the transactional database.

Note: Due to the nature of encryption within the database, there's a delay before a new Database Encryption seed is used. See Special Database Encryption Considerations for more information.

Storing Encrypted Payloads for Field-Level Encryption

For FLE, encrypted data is stored in the database with its metadata.

The metadata includes:

- A bit that, when set, indicates the field contains ciphertext
- The ID of the customer's key material used to derive the matching encryption key
- A random or static 128-bit IV

The key material's ID is used to locate the key value and creation date. These values are stored in a Salesforce object called TenantSecret. When a user accesses or saves encrypted data, the encryption service sends a request to the regional Shield KMS. The regional Shield KMS then uses the customer's key material and corresponding KDF seed, identified by the key material's generation or upload date, to derive a DEK. The random IV is used with the encryption key to non-deterministically encrypt the data.

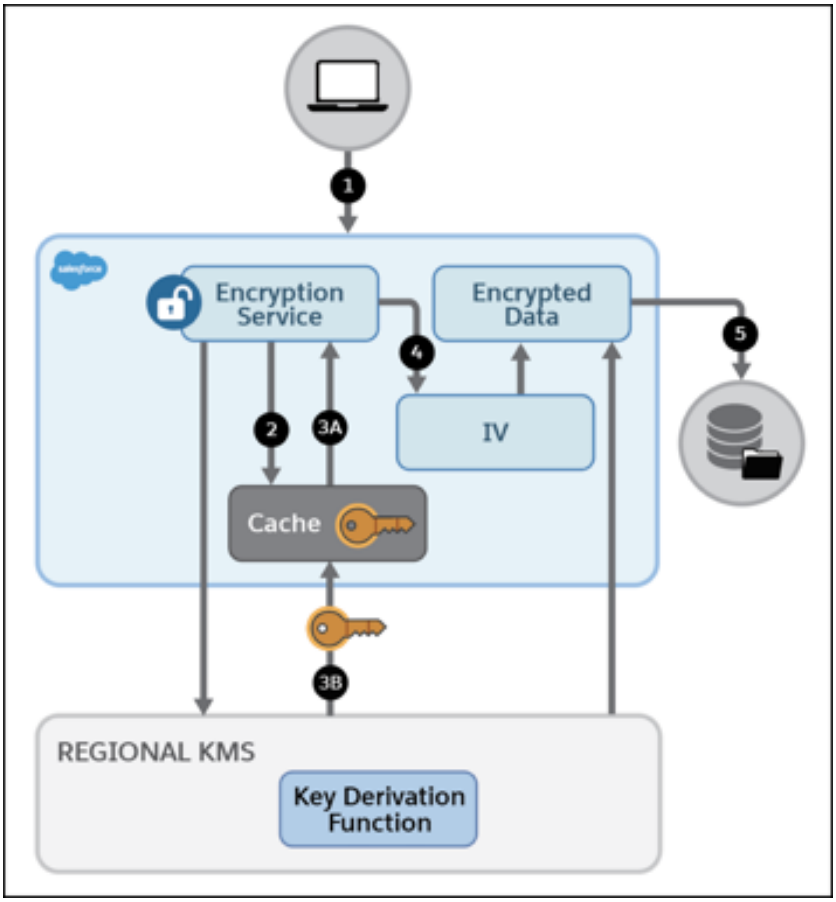
Storing Encrypted Data for Database Encryption

For Database Encryption, encrypted data is stored in a database fragment along with the IV and database tenant secret key ID.

The key material's ID is used to locate the key value and creation date. These values are stored in a Salesforce object called TenantSecret. When a user accesses or saves encrypted data, the database encryption service retrieves the fragment and then uses the key ID to get the appropriate secret from the encrypted key cache.

Shield Platform Encryption Process Flow for Field-Level Encryption

Before data is encrypted, a Salesforce administrator must enable encryption and generate or supply key material. For FLE, each field, file, attachment, and data element on which encryption is enabled, the corresponding metadata in the UDD is updated to reflect the new encryption setting.



1. When a user saves encrypted data, the runtime engine determines from metadata whether the field, file, attachment, or data element must be encrypted before storing it in the database.
2. If the decision is to encrypt, the encryption service checks for the matching data encryption key in the encrypted key cache.
3. The encryption service determines if the key exists.
   a. If yes, the encryption service retrieves the key.
   b. If no, the service sends a derivation request to the regional Shield KMS and returns it to the encryption service running on the Lightning Platform. Data moving between the regional Shield KMS and the encryption service is encrypted by the TLS protocol, which uses a certificate signed by a dedicated Salesforce authority. This certificate's private key is stored by the regional Shield KMS and the encryption service in an encrypted form. The certificate's public and private keys are rotated regularly.
4. After retrieving or deriving the key, the encryption service generates a random IV and encrypts the data by using JCE's AES-256 implementation.
5. The ciphertext is saved in the database or file storage. The IV and corresponding ID of the key material used to derive the data encryption key are saved in the database.

Search Encryption at Rest Process Flow

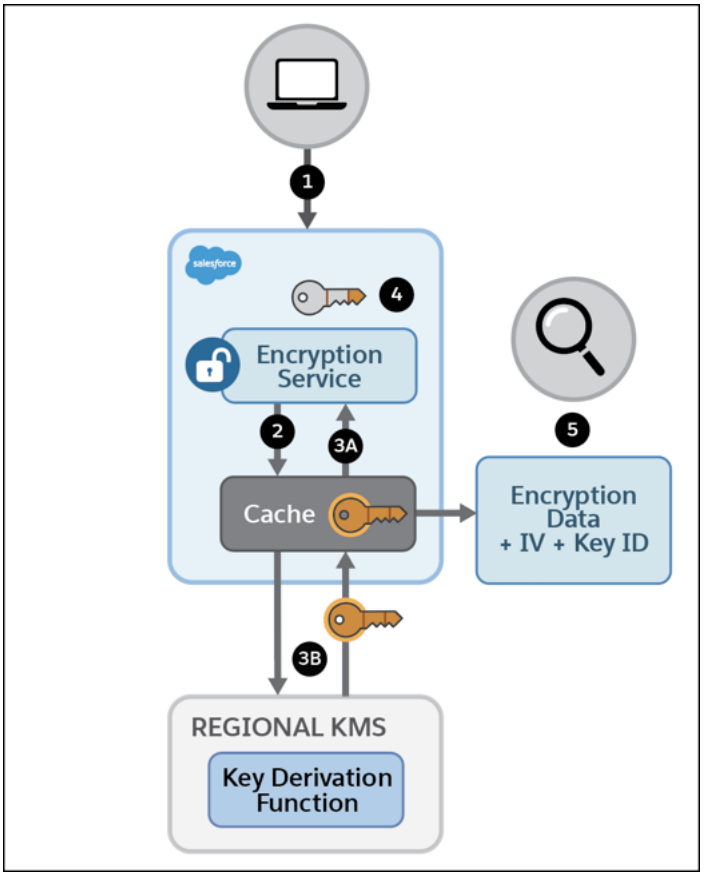
The Salesforce search engine is built on the open-source enterprise search platform software Apache Solr. The search index, which stores tokens of record data with links back to the original records stored in the database, is housed within Solr. Partitions divide the search index into segments to allow Salesforce to scale operations. Apache Lucene is used for its core library.

By using Shield Platform Encryption's HSM-based key derivation architecture, metadata, and configurations, Search Encryption runs automatically when Shield Platform Encryption is in use. The solution applies strong encryption on org-specific search index .fdt, .tim, and .tip file types by using an org-specific AES-256 bit encryption key. The search index is encrypted at the search index segment level, and all search index operations require index blocks to be encrypted in memory.

The only way to access the search index or the encrypted key cache is through programmatic APIs.

To use Search encryption, the Salesforce admin first enables the Encrypt Search Indexes option. Shield Platform Encryption creates the first root key, and then uses it to create and wrap a DEK. The DEK is then stored in the database.

Note: The Encrypt Search Indexes option is visible only after a tenant secret has been created.



When a User Creates or Edits Records

1. The core application determines if the search index segment must be encrypted based on metadata.
2. If the search index segment must be encrypted, the encryption service checks for the matching search encryption key ID in the encrypted key cache.
3. The encryption service determines if the key exists in the encrypted key cache.

   a. If the DEK exists in the encrypted key cache, the encryption service uses it for encryption.
   b. If not, the service sends a request to the core application, which in turn sends an authenticated derivation request to the regional Shield KMS or External KMS and returns the DEK to the core application server. Data moving between the regional Shield KMS and the encryption service is encrypted by the TLS protocol, which uses a certificate signed by a dedicated Salesforce authority. This certificate's private key is stored by the regional Shield KMS and the encryption service in an encrypted form. The certificate's public and private keys are rotated regularly.

4. After retrieving the DEK, the encryption service generates a random initialization vector (IV) and encrypts the data by using JCE's AES-256 implementation.
5. The key ID, which is the identifier of the key being used to encrypt the index segment, and IV are saved in the search index.

When a User Searches for an Encrypted Term

1. The term is passed to the search index along with the Salesforce objects to search.
2. When the search index executes the search, the encryption service opens the relevant segment of the search index in memory and reads the key ID and IV.
3. Repeats steps 3 through 5 in the previous search index encryption process.
4. The search index processes the search and returns the results to the user.

If Salesforce administrators disable encryption on a field, all index segments that were encrypted are unencrypted and the key ID is set to null. This process can take up to 7 days.

Database Encryption Process Flow

For Database Encryption, encryption and decryption happens at the fragment level.

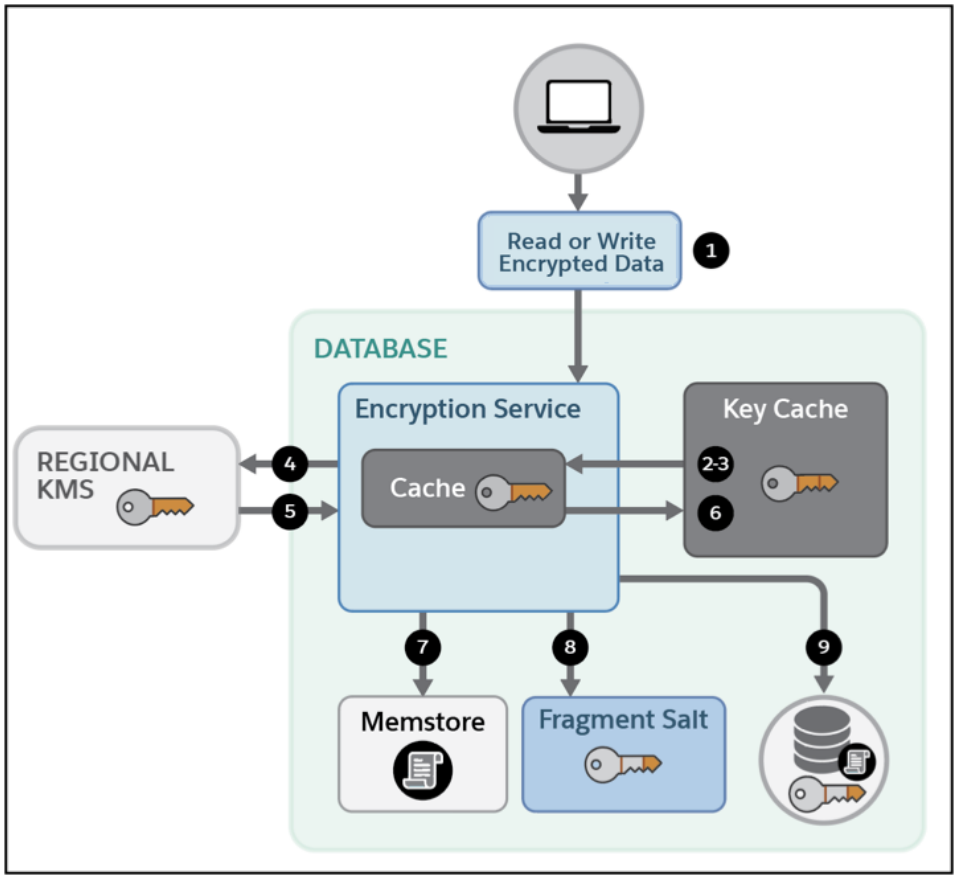
Encryption Process:

- A user saves data (1).
- If Database Encryption is active, the database encryption service attempts to retrieve the associated encryption keys from the encrypted key cache (2).
- If the key exists, the database encryption service retrieves the key (3).
- If the key isn't in the encrypted key cache:
  - The encryption service requests the database tenant secret from the regional KMS (4).
  - The regional KMS provides the database tenant secret (5).

    Note: Data moving between the regional Shield KMS and the database encryption service is encrypted by the mutual TLS (mTLS) protocol, which uses a certificate signed by a dedicated Salesforce Internal CA - certificate authority. This certificate's private key is stored in the regional KMS in an encrypted form. The certificate's public and private keys are rotated regularly.

  - The database encryption service stores the database tenant secret in the encrypted key cache (6).
- The database stages the data to be encrypted in the database memstore until it determines that it can write a database fragment (usually 64KB or smaller) (7).
- The database encryption service generates a per-fragment salt based on the database fragment being written. The service uses this per-fragment salt and the database tenant secret as parameters into a KDF to derive the DEK for that database fragment. Using this DEK, it encrypts the entire fragment by using OpenSSL AESGCM-256 implementation (8).

- The database fragment is now encrypted. The per-fragment salt and corresponding ID of the key material used to derive the data encryption key are saved alongside that fragment in the database (9).



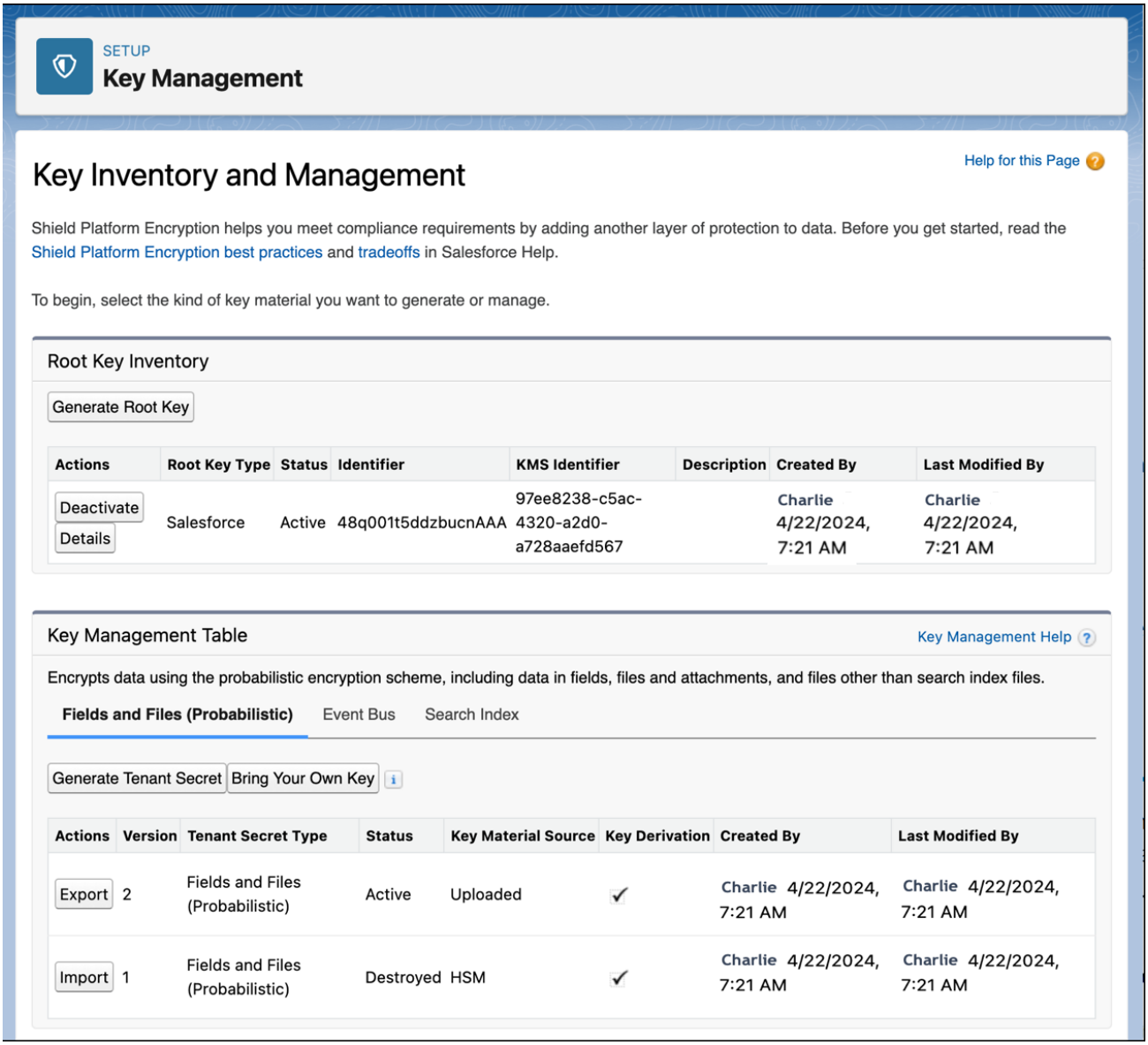
Decryption Process

Note: If you use Database Encryption for a time and then stop using it, you can still retrieve your encrypted data. If you change that data, it is rewritten to the database in a plaintext fragment.

- When an authenticated user retrieves encrypted data, the database encryption service retrieves the database fragment containing the desired data along with the per-fragment salt and ID of the key material used to encrypt the data being retrieved.
- The database encryption service checks for the matching data encryption key in the key cache.
- If the key exists, the encryption service retrieves the key.
- If the key isn't in the encrypted key cache:
  - The encryption service requests the database tenant secret from the regional KMS.
  - The regional KMS provides the database tenant secret. As when encrypting the data, data moving between the regional Shield KMS and the database encryption service is encrypted by mTLS.
  - The database encryption service stores the database tenant secret in the encrypted key cache.
- The database retrieves the database tenant secret and the per-fragment salt and supplies them to the KDF to derive the DEK.
- The data is decrypted and returned to the user.

Key Management for Field-Level Encryption

With Shield Platform Encryption, Salesforce administrators can manage the lifecycle of their data encryption keys while protecting those keys from unauthorized access. To ensure this level of protection, data encryption keys are never persisted in plain text. If a customer has opted out of key derivation, then a wrapped DEK is persisted, but must be unwrapped by the regional Shield KMS or external KMS.



The KDF seed is generated by an HSM in the primary KMS at the start of each release. Access to the HSM and generation of primary keys is done by way of a High Assurance Virtual Ceremony (HAVC), a process conducted by a ceremony administrator in cooperation with several cryptographic administrators.

Key material is generated on demand by using the HSM and the primary Shield KMS, or it's supplied by the customer using BYOK.

EKM gives customers the option for managing root keys and DEKs through their external KMS accounts. Root keys are stored in the external KMS. Shield Platform Encryption requests a wrapped DEK from the external KMS, and stores it in the Salesforce database. When the DEK is needed, Shield Platform Encryption sends the wrapped DEK to the external KMS. The external KMS unwraps the DEK and sends it back to the application where it's securely cached for a limited time. The DEK is never persisted as plaintext.

BYOK gives customers more control and flexibility for managing key material via an API service. Customers can use open-source crypto libraries, their existing HSM infrastructure, or even third-party key brokering services to create and manage tenant secrets and data encryption keys outside of Salesforce. They can then give Salesforce's KMS access to that key material. Customers can revoke this access at any time.

Each regional Shield KMS has access to the release-specific secrets for every Salesforce release. By default, when a data encryption key is needed to encrypt or decrypt customer data, the regional Shield KMS derives the key from the KDF seed and the tenant secret.

Customers can opt out of key derivation on a key-by-key basis and upload a final data encryption key, or they can store their keys in external key management systems for on-demand retrieval. By controlling the lifecycle of your organization's key material, you control the lifecycle of the derived data encryption keys. Your Salesforce administrator specifies a user to manage the key material for your organization and assigns that user the Manage Encryption Keys user permission. This user permission allows the key administrator to generate, supply, archive, export, import, and destroy key material.

It's possible to have more than one active tenant secret or data encryption key in an org. You can apply specific keys to data stored in different areas of Salesforce. For example, search index files are stored separately from other Salesforce data, so customers can apply key material to specific data in those files. Only the most recent tenant secret or DEK of a given type is active. Only that key material is used to derive the DEK used to encrypt data of a specified type. When you generate or supply key material, the active secret becomes archived. Archived key material is used to decrypt data that was last encrypted when the archived key material was active.

You can destroy an archived tenant secret or data encryption key. If you destroy a tenant secret, it's no longer possible to derive the encryption key that's required to decrypt the data that was encrypted using that key. Similarly, when you destroy a customer-supplied DEK, you can't access data encrypted with that key. Take special care to back up and protect archived key material and encrypted data. After you destroy key material, it's fully removed from the persistent layer and encrypted key cache, and it can't be recovered.

Rotating Keys and Re-Encrypting Data

Generating or supplying new key material is called key rotation. When you rotate key material, any resulting derived DEKs rotate as well. New data is encrypted and decrypted by using the final DEK, which is derived by default from the new, active tenant secret. If you opt out of key derivation with BYOK, rotating your key encrypts new data with your active customer-supplied final DEK. Existing data stays encrypted with the former key material, even if that key is derived.

Customers can use the self-service background encryption service to traverse the database and file storage, decrypt existing encrypted data, and then re-encrypt the data by using the new DEK. This process is transparent to users and administrators.

Key Management for Database Encryption

Database Encryption provides you with the ability to rotate your database tenant secret after it's listed on the Key Management page. Deleting database tenant secrets isn't permitted.

When you rotate your database tenant secret, all subsequent database encryption requests use the new database tenant secret to derive the per-fragment DEKs. The previous database tenant secrets are maintained for reading old data. Updates to data previously encrypted using DEKs derived with an older database tenant secret are done by using the new seed, so in a typical database, there's a gradual migration of data encrypted with DEKs derived with the older seed to DEKs derived with the latest seed.

Along with your database tenant secret, one of the components of your final database encryption key is the salt for the database fragment or page being written. As a result, a relatively small amount of data is encrypted with an identical final database encryption key.

Currently, key rotation only affects new encryption operations. If you have a use case where you must re-encrypt your entire database, contact Salesforce support.

CHAPTER 4 Data Encryption Key Derivation

In this chapter ...

- Key Derivation Architecture
- HSM Initialization
- Per-Release Secret Generation and Export
- Shield KMS Startup
- On-Demand Tenant Secret Generation
- Customers Can Supply Their Own Key Material with EKM, BYOK, and Cache-Only Keys
- External Key Management Flow
- BYOK Tenant Secret Flow
- Cache-Only Key Flow
- Encrypted Information Flow with Key Derivation

By default, Shield Platform Encryption uses the Shield Key Management Service to derive DEKs for encrypting customer data at rest. DEKs are derived from partial key secrets that are securely wrapped and stored in the Shield KMS. Key derivation ensures that the derived keys are never persisted in their composite forms, and it enables customers to control the key lifecycle.

Key Derivation Architecture

Secrets and secret-wrapping keys used in the key derivation process are initialized by the HSM in the primary Shield KMS at the start of each release. For customer-provided tenant secrets, they're initialized on demand in production environments by the HSMs in the regional Shield KMSs.

BYOK customers can opt out of the key derivation process by uploading their own data encryption keys. EKM customers don't use derivation. Their DEKs are generated and then supplied wrapped by the external KMS. In both cases, the key materials are stored wrapped in Salesforce. Database Encryption customers using BYOK must provide a full data encryption key, so there is no need to opt out of derivation.

Customer-supplied DEKs are used to directly encrypt and decrypt data and give customers more fine-grained control over the key material used to secure their data. Because these keys are applied directly to customer data, Salesforce recommends that customers take adequate safety measures to back up and control access to these keys.

Processes in Key Derivation Architecture

HSM initialization
Before being put to use, an HSM is initialized, which includes the creation of its respective encryption key pairs.

Per-release secret generation
At the start of each release, the per-release secrets are generated within the HSM during a High Assurance Virtual Ceremony (HAVC). The secrets are stored in the primary Shield KMS for consumption by the primary Shield KMS and regional Shield KMSs.

Primary Shield KMS startup
When the primary Shield KMS starts up, it accesses each release's encrypted secrets stored in the HSM. It makes these secrets available to the Key Escrow server. Also, it decrypts the secrets and stores them in its encrypted key cache in preparation for key derivation requests.

Regional Shield KMS startup
When a regional Shield KMS starts up, a local key broker service requests secrets from the Key Escrow server. The key broker gets the key material and writes it into the regional Shield KMS. The regional Shield KMS then decrypts the secrets and stores them in its encrypted key cache in preparation for key derivation requests from app servers.

On-demand root key and DEK generation
For Shield Platform Encryption services that use root keys, such as Search, a customer generates both root keys and DEKs as needed. Root keys provide an added layer of protection for the KMS generated DEKs used by the service. The request is sent to the regional Shield KMS from the application server and authenticated. Root keys are generated either by the regional Shield KMS, or by the customer's external KMS. The root key is visible on the key management page under the Root Key inventory. With an active root key available, compatible services have their DEKs generated by the same KMS as the active root key. The DEKs are wrapped on the KMS using the active root key before being sent back to the application server to be stored in the database.

On-demand tenant secret generation
One of the inputs into the key derivation function that creates your organization's data encryption key is an org-specific, customer-managed tenant secret. Customers control the lifecycle of their DEKs by generating new tenant secrets. When a customer starts the process to generate a new tenant secret, the request is sent to the regional Shield KMS from the application server and authenticated. Then the regional Shield KMS generates and sends an encrypted tenant secret back to the application server to be stored in the database. This is the case for FLE tenant secrets and Database Encryption seeds.

Key derivation in production environments
When a customer attempts to read or write encrypted data and the corresponding data encryption key isn't cached in the app server, the application server sends a derivation request to the regional Shield KMS. The regional Shield KMS authenticates the request and derives the key using the secrets in its own encrypted key cache. The key is then transmitted securely back to the application server.

Shield Platform Encryption Architecture Components

Primary HSM (nShield® Connect HSM model XC High or XC Mid, depending on region)
A FIPS 140-2 Level 3 hardware-compliant network appliance that generates per-release secrets and secret-wrapping keys. Access to the primary HSM is restricted to designated Salesforce security officers during controlled High Assurance Virtual Ceremonies (HAVCs)

Primary Shield KMS Server
The Shield Key Management Service consists of a single Primary KMS and multiple regional KMSs. The Primary KMS is the first KMS to receive the per-release secrets. It makes those secrets available to the Key Escrow server. In addition to supplying the Key Escrow, it also services key material requests like any regional KMS server.

Key Escrow Server
The Key Escrow Server acts as a go-between for the primary Shield KMS and the regional key brokers. It obtains wrapped secrets material from the Primary KMS and hands them off to key brokers on request.

Regional Shield KMS Servers and Key Brokers
Regional Shield key management servers are deployed to the Salesforce production data centers. The regional KMSs securely provide per-release key material to the application servers. The regional KMSs also generate tenant secrets. One key broker acts as the go-between for the Key Escrow and a regional Shield KMS.

External Key Management Service
Customers who use the BYOK or Cache-only Key features must manage their key material using an external KMS.

Application Servers
Servers in production environments that run Salesforce. When a customer attempts to read or write encrypted data or generate a tenant secret, the application server communicates with the regional Shield KMS to process the request.

Database Fragment
The smallest unit of data encrypted under Database Encryption. Referred to as a page or block in some transactional databases. Database Encryption uses a unique DEK to encrypt every fragment.

Salesforce Search Index
Servers in production environments that manage Salesforce searches. When a user attempts to query encrypted data, the search index processes the request.

HSM Initialization

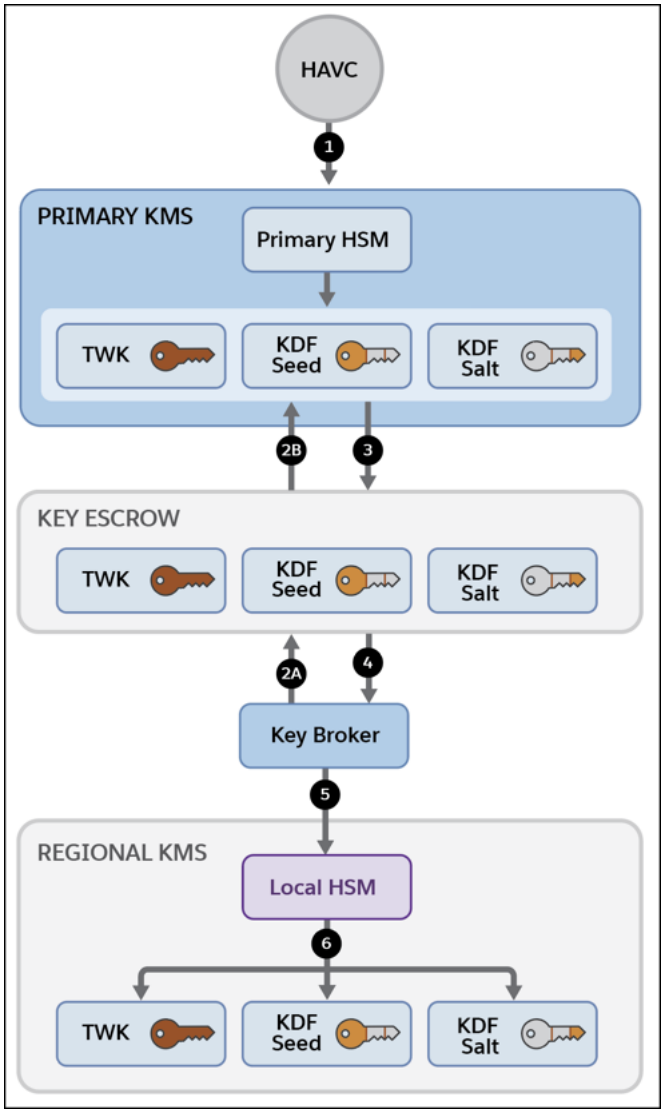
An HSM must be initialized before it's used. For the primary HSM, initialization creates a primary HSM encryption key pair and a primary HSM signing key pair. For each regional HSM, initialization creates a regional HSM encryption key pair.

The primary HSM public signing key is used to sign and verify each regional HSM's public encryption key. At the start of each release, the primary and regional HSM public encryption keys are used to separately encrypt a per-release primary wrapping key. In turn, this key is used to encrypt the remainder of the per-release secrets used to derive data encryption keys.

This way, each regional HSM is able to securely access the primary wrapping key for each release, which it uses to access the rest of the per-release secrets needed for key derivation. The private keys in each pair are accessible only inside their respective HSMs.

Per-Release Secret Generation and Export

At the start of each release, a Salesforce cryptographic security officer conducts a High Assurance Virtual Ceremony (HAVC) to generate the per-release secrets and keys. Once created, the secrets are sent to the regional Shield KMSs.



1. During the HAVC, the primary HSM generates these secrets.
   - KDF seed
   - KDF salt
   - Tenant wrapping key (TWK)

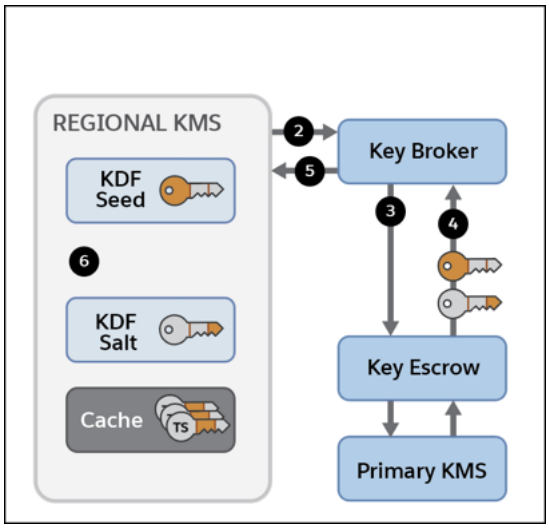
   Each secret is stored on the primary KMS.
2. A key escrow server receives requests from regional key brokers for the KDF Seed, KDF Salt, and Tenant Wrapping Key.
3. The regional Shield KSMs generates random byte data through its HSM) and sends it to the key escrow server.
4. The key escrow server retrieves the secrets from the primary Shield KMS and returns them to the key brokers.
5. The key brokers install the secrets onto the regional Shield KMSs.

After all the secrets are in the regional Shield KMSs, they're used for encryption services to tenant orgs in the region.

For definitions of each secret and key, refer to the Keys and Secrets section.

Shield KMS Startup

When the regional Shield KMS starts up in a production environment, it accesses each release's encrypted secrets stored in the regional KMS. It then stores them in the encrypted key cache in preparation for deriving data encryption keys.

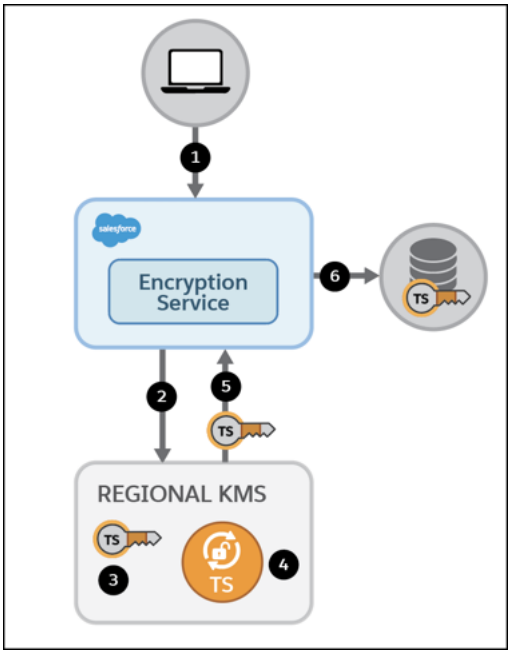


The process includes these steps.

1. The regional Shield KMS starts up.
2. After the basic Shield KMS processes run, the server notifies the local key broker service that it's ready to accept key material.
3. The key broker service communicates with the Key Escrow server to request current release secrets.
4. The Key Escrow server gets the release secrets from the primary Shield KMS and securely passes them to the key broker service.
5. The key broker server writes the secrets into the regional Shield KMS.
6. The regional Shield KMS validates all the keys and secrets against their hashes and then stores them in the encrypted key cache. It's then ready for encryption service.

On-Demand Tenant Secret Generation

Customers can generate or upload key material every 24 hours in their production or Developer Edition orgs, and every 4 hours in sandbox orgs. Field-Level Encryption key material can be destroyed at any time. When a customer generates new key material, all future data is encrypted with the new final data encryption key (DEK). This DEK is derived by default. Customers can opt out of derivation and supply their own DEK.



The on-demand FLE tenant secret or database tenant secret is generated by the regional Shield KMS.

The process of generating one of these tenant secrets includes these steps.

1. An admin attempts to generate a new tenant secret by using the UI or API.
2. The encryption service sends an authenticated request to the regional Shield KMS.
3. The regional Shield KMS generates the tenant secret (TS).
4. The regional Shield KMS encrypts the tenant secret with the per-release tenant wrapping key.
5. The regional Shield KMS sends the encrypted tenant secret back to the encryption service running on the Lightning Platform. In the case of Database Encryption, the database tenant secret is sent directly to the transactional database.
6. The encryption service stores the encrypted tenant secret securely in the database. The encrypted tenant secret is used to derive the org's specific data encryption keys on demand. This encrypted tenant secret can only be decrypted in the regional Shield KMS.

Customers Can Supply Their Own Key Material with EKM, BYOK, and Cache-Only Keys

Customers can supply their own key material by using External Key Management (EKM), Bring Your Own Key (BYOK), or Cache-Only Keys. Each offers customers different levels of control over key material. Further, each has different setup requirements. With BYOK, customers can upload tenant secrets and data encryption keys outside of Salesforce by using their own crypto libraries, enterprise key management system, or hardware security module. With EKM and Cache-Only Keys, customers can supply their own DEKs.

If required, customer supplied key material can be uploaded once every 4 hours in sandbox orgs and every 24 hours otherwise. For more information, see Rotate Your Encryption Key Material in Salesforce Help. Key material can be destroyed declaratively or programmatically by the customer any time.

The process for generating and encrypting customer-supplied key material varies depending on whether customers use a crypto service, HSM, or key brokering service. However, all customer-supplied key material must meet the same basic requirements before it can be uploaded to Salesforce. Users need the Manage Encryption Keys permissions to upload and rotate key material and the Manage Certificates permission to manage certificates.

After they're uploaded, customer-supplied tenant secrets work with the Salesforce key management systems just like Salesforce-generated tenant secrets. By default, when customer-supplied tenant secrets are uploaded, all subsequent data is encrypted with the key derived from the current primary secret and the new customer-supplied tenant secret. This org-specific derived data encryption key is never persisted on disk.

External Key Management (EKM)
EKM provides the ability to set up and configure keys within supported public cloud key management services controlled by the customer, for use by Salesforce as permitted by the customer. After the wrapped DEKs are in place, they're used like any other Shield Platform Encryption DEK. When they're needed, Shield Platform Encryption sends the wrapped DEK to the external KMS via TLS and requests that it unwrap the DEK. The unwrapped DEK is returned to Shield Platform Encryption via TLS and placed in the encrypted key cache. The DEK is cached for a limited time and never persisted as plain text.

Bring Your Own Key (BYOK)
With BYOK, customers can bring key material from outside of Salesforce. They generate key material by using their own crypto libraries, enterprise key management system, or hardware security module. Customers can encrypt it with a self-signed or certificate authority (CA) certificate's public key. They upload the keys to Shield Platform Encryption. They can revoke access on demand via the Key Management tooling in Setup or programmatically via the API.

Cache-Only Keys
Customers can create and store a DEK outside of Salesforce and use Cache-Only Keys to apply that DEK to data in Salesforce. Customers can use an on-premises key service, host their own cloud-based key service, or use a cloud-based key brokering vendor. Root keys and named principals are supported. DEKs are fetched on demand over a secure channel that the customer configures. Salesforce-generated DEKs are wrapped with a cache key encryption key and placed directly in the encrypted key cache for encrypt and decrypt operations. DEKs generated by an external KMS are wrapped by the generating root key, and they're unwrapped by that same root key when needed.

External Key Management (EKM)

EKM provides the ability to set up and configure keys within supported public cloud key management services controlled by the customer, for use by Salesforce as permitted by the customer. After the wrapped DEKs are in place, they're used like any other Shield Platform Encryption DEK. When they're needed, Shield Platform Encryption sends the wrapped DEK to the external KMS via TLS and requests that it unwrap the DEK. The unwrapped DEK is returned to Shield Platform Encryption via TLS and placed in the encrypted key cache. The DEK is cached for a limited time and never persisted as plain text.

Note: Salesforce currently supports Amazon Web Services (AWS) KMS as one of the external KMS options. We plan to expand the supported list of external KMS providers over time.

With EKM, we introduce the logical concept of a root key. A root key tracks the key that the customer admin manages on the external KMS. Customers apply Salesforce generated root key policies within the Shield Platform Encryption UI. They then add these policies to their external KMS account. The policies enable Shield Platform Encryption to do two things.

- Request that the root key on the external KMS create a wrapped DEK.
- Request unwrapping of a wrapped DEK by the root key on the external KMS.

Example: External AWS KMS Example

A company wants to take advantage of its investment in AWS KMS key management. It creates a root key specially for the purpose of Shield Platform Encryption operations.

In Salesforce, they generate a unique AWS KMS Key Policy to permit the Shield Platform Encryption service to perform the minimum required operations.

- Once their Salesforce admin vets the policy and applies it on the AWS KMS Key that they own, the setup is complete. Salesforce creates the first EKM based DEK, and an active root key appears under the Root Key Inventory of the Key Management Page in Salesforce setup.
- With an active Root key available, the customer can trigger DEK generation. This action requires a Generate DEK request from Salesforce to the External AWS KMS.
- The AWS KMS generates a symmetric AES 256 key and wraps the key by using the root key. The only copy of the generated and wrapped DEK is returned to Salesforce to be persisted securely.
- When the DEK is needed, Salesforce Shield Platform Encryption invokes the Decrypt operation on the external AWS KMS via the Salesforce Regional KMS (the authorized principal). As part of the Decrypt request, Salesforce passes the wrapped DEK, the necessary encryption context, and the AWS KMS Key Reference back to the external AWS KMS. The plaintext DEK is then returned to Salesforce by way of a TLS connection. Finally, the key is passed to the specific internal service that needs the key via mTLS.

Bring Your Own Key (BYOK)

With BYOK, customers can bring key material from outside of Salesforce. They generate key material by using their own crypto libraries, enterprise key management system, or hardware security module. Customers can encrypt it with a self-signed or certificate authority (CA) certificate's public key. They upload the keys to Shield Platform Encryption. They can revoke access on demand via the Key Management tooling in Setup or programmatically via the API.

Customers can opt out of the key derivation process on a key-by-key basis. When customers opt out of derivation, they upload their own DEK, which is used to directly encrypt and decrypt data. Before they're stored in the Salesforce database, customer-supplied DEKs are wrapped, either with a tenant wrapping key or with the external root key that created them. When needed, Shield Platform Encryption unwraps the keys on the regional Shield KMS and places them in the encrypted key cache for a limited time.

Customer-supplied key material can be uploaded once every 24 hours in production and Developer Edition orgs, and every 4 hours in sandbox orgs. FLE key material can be destroyed declaratively or programmatically by the customer at any time.

Field-Level Encryption BYOK

For FLE, by default, when customer-supplied tenant secrets are uploaded, all subsequent data is encrypted by using the new material. They're encrypted either with the key derived from the current primary secret and the new customer-supplied tenant secret or the new DEK alone. This org-specific derived DEK is never persisted on disk.

Customers can opt out of the key derivation process on a key-by-key basis. When customers opt out of derivation, they upload their own DEK, which is used to directly encrypt and decrypt data. Before they're stored in the Salesforce database, customer-supplied DEKs are wrapped, either with a tenant wrapping key or the external root key that created them. When needed, Shield Platform Encryption unwraps them on the regional Shield KMS and places them in the encrypted key cache for a limited time.

Alternatively, Field-Level Encryption customers can create and store their DEKs outside of Salesforce and use the Cache-Only Key Service to use those DEKs for Shield Platform Encryption. Customers can use an on-premises key service, host their own cloud-based key service, or use a cloud-based key brokering vendor. DEKs are fetched on demand over a secure channel that the customer configures. These DEKs are wrapped with a cache key encryption key and placed in the encrypted key cache for encrypt and decrypt operations. Because cache-only keys bypass the key derivation process, they're used to directly encrypt and decrypt your data.

Subsequent encryption and decryption requests go through the encrypted key cache until the cache-only key is revoked or rotated, or the cache is flushed. After the cache is flushed, the Cache-Only Key Service fetches key material from your specified key service. The cache is regularly flushed every 72 hours. Certain Salesforce operations flush the cache on average every 24 hours. Destroying a DEK invalidates the corresponding DEK that's stored in the cache.

Database Encryption BYOK

BYOKs for Database Encryption are always Database Encryption seeds. Full DEK upload isn't supported for Database Encryption. All Database Encryption uses derived DEKs, so customers can only upload a database tenant secret. Then, unique DEKs are derived for every database fragment within the Salesforce database, as opposed to FLE DEKs, which are derived in the KMS.

Per Salesforce key backup policy, new database tenant secret material isn't used for new DEKs for up to 24 hours. During this time a unique Salesforce provided Database Encryption seed is used. This seed becomes a permanent addition to the customer's key archive.

Cache-Only Keys

Customers can create and store a DEK outside of Salesforce and use Cache-Only Keys to apply that DEK to data in Salesforce. Customers can use an on-premises key service, host their own cloud-based key service, or use a cloud-based key brokering vendor. Root keys and named principals are supported. DEKs are fetched on demand over a secure channel that the customer configures. Salesforce-generated DEKs are wrapped with a cache key encryption key and placed directly in the encrypted key cache for encrypt and decrypt operations. DEKs generated by an external KMS are wrapped by the generating root key, and they're unwrapped by that same root key when needed.

Because cache-only keys bypass the key derivation process, they're used to directly encrypt and decrypt your data. Subsequent encryption and decryption requests go through the encrypted key cache until the cache-only key is revoked or rotated, or the cache is flushed. After the cache is flushed, cache-only keys fetches key material from your specified key service. The cache is regularly flushed every 72 hours.

Certain Salesforce operations flush the cache on average every 24 hours. Destroying a data encryption key invalidates the corresponding data encryption key that's stored in the cache.

External Key Management Flow

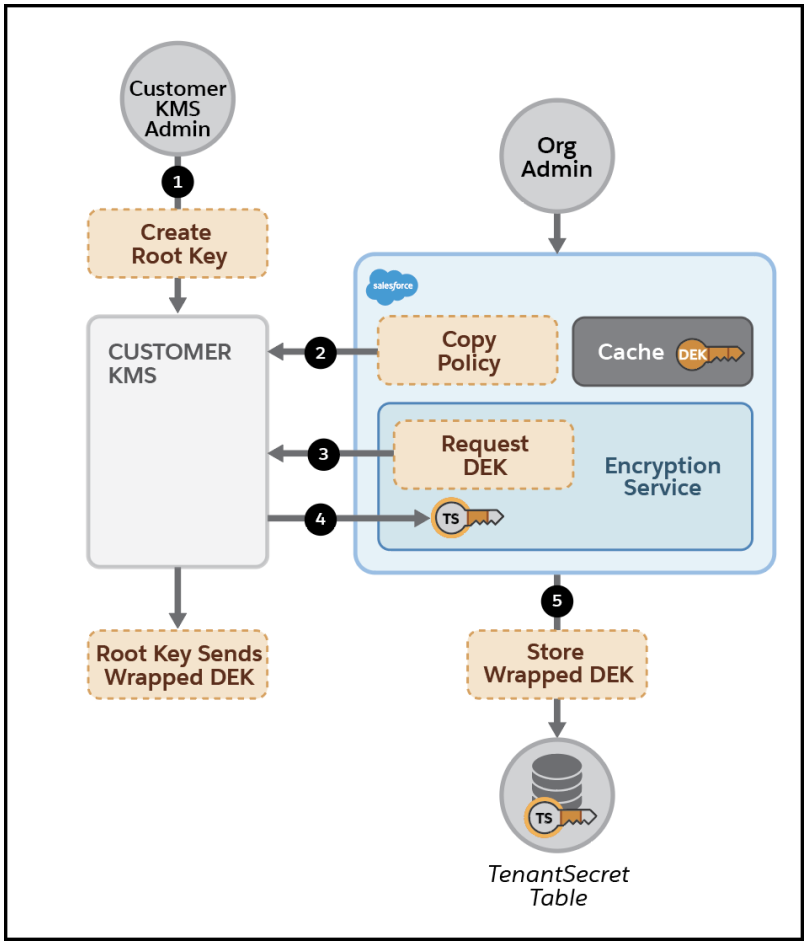
For EKM, Shield Platform Encryption relies on the customer's external KMS to generate and secure the DEKs used by the Shield Platform encryption service. These DEKs reside with the Shield Platform encrypted key cache in a wrapped state. When encryption or decryption operations are needed, the appropriate DEK is unwrapped by the external KMS. The customer key service then sends the unwrapped DEK securely back to the Shield Platform encryption service, which stores it in the Salesforce encrypted key cache. Encryption and decryption operations then use the cached DEKs.

The process begins when you create a root key in the customer KMS. You create a policy that gives Salesforce's regional Shield KMS some important permissions.

- Permission to request the customer key service to generate and wrap a DEK by using the root key
- Permission to request the customer key service to unwrap the DEK by using the customer root key

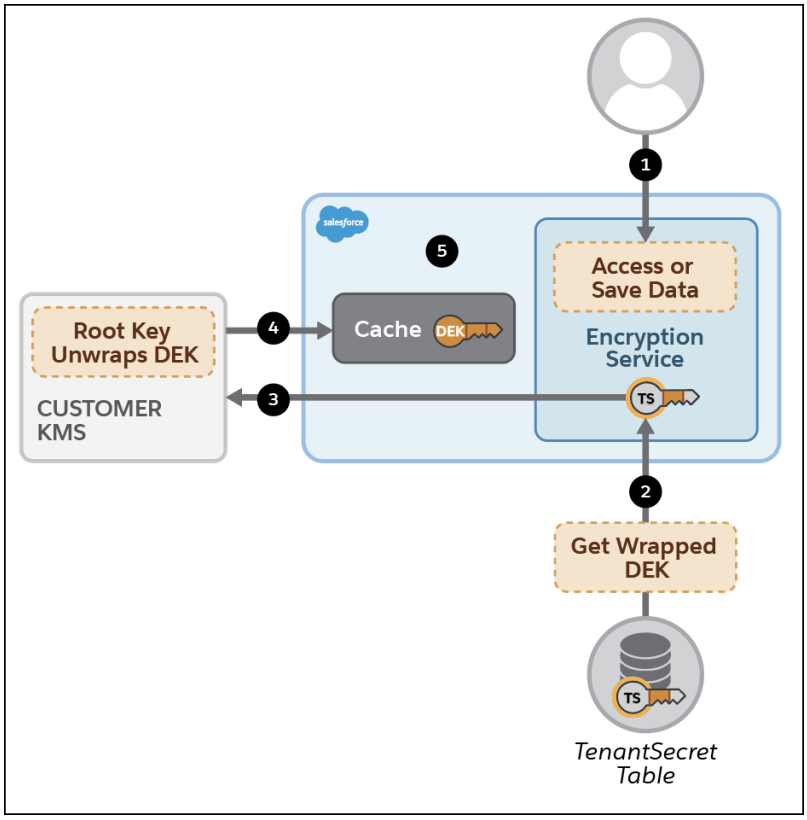
You use this policy to create an EKM DEK in Setup. Then the Shield Platform encryption service requests the customer KMS to generate a DEK and wrap it by using the root key. The customer KMS creates a DEK, wraps it, and sends it to the Shield Platform encryption service over TLS. This copy is the only copy of the DEK that exists. Shield Platform Encryption stores the DEK wrapped by the root key in the database. Here's the process, step by step.

1. The customer KMS admin creates a root key.
2. The Salesforce admin creates a key policy and copies it to the customer KMS.
3. With the policy in place, the Salesforce encryption service requests a DEK for local storage.
4. The customer KMS uses the root key to wrap the new DEK, which it sends back via TLS.
5. The encryption service stores the wrapped DEK in the Salesforce database.



When the Shield Platform encryption service detects encryption operations that require the EKM DEK, it checks its encrypted key cache for it. If the unwrapped DEK isn't present in the cache, the Shield Platform encryption service requests that the key service unwrap the DEK. The key service unwraps the DEK and sends it back to the Shield Platform encryption service over TLS. Then the Shield Platform encryption service adds the unwrapped key to the encrypted key cache.

1. A user accesses or saves encrypted data.
2. The Shield Platform encryption service gets the DEK from the TenantSecret table.
3. The encryption service sends the wrapped key to the customer KMS over a secure channel to be unwrapped.
4. The customer KMS uses the root key to unwrap the DEK and sends it back to the encryption service.
5. The encryption service stores the unwrapped key in the encrypted key cache for immediate use.



If the unwrapped DEK is present in the cache, the Shield Platform encryption service uses it for encryption and decryption of customer data.

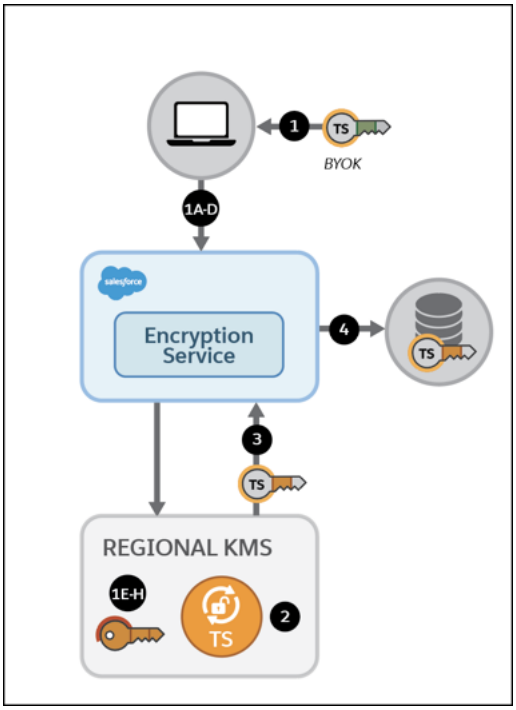
Because EKM DEKs bypass the key-derivation process, they're used to directly encrypt and decrypt your data.

The Shield KMS enhanced cache controls ensure that key material is stored securely while in the cache. The Shield KMS encrypts the fetched key material with an org-specific AES 256-bit cache encryption key and stores the encrypted key material in the cache for encrypt and decrypt operations. HSM-protected keys secure the cache encryption key in the cache, and the cache encryption key is rotated along with key lifecycle events such as key destruction and rotation. Once the encryption key is in the cache, encryption and decryption requests go through the encrypted key cache.

The enhanced cache controls provide a single source of truth for key material that's used to encrypt and decrypt your data. Keys are unwrapped by the customer KMS and stored in the key cache until the DEK is revoked or rotated or until the cache is flushed. After the cache is flushed, the EKM service again fetches the DEK from your specified key service. The cache is flushed regularly every 72 hours. Certain Salesforce operations flush the cache, on average, every 24 hours. Destroying a DEK invalidates the corresponding DEK that's stored in the cache.

BYOK Tenant Secret Flow

To supply your own tenant secret, you create the secret, sign it with a BYOK-compatible certificate, and encode it with base64 encoding.

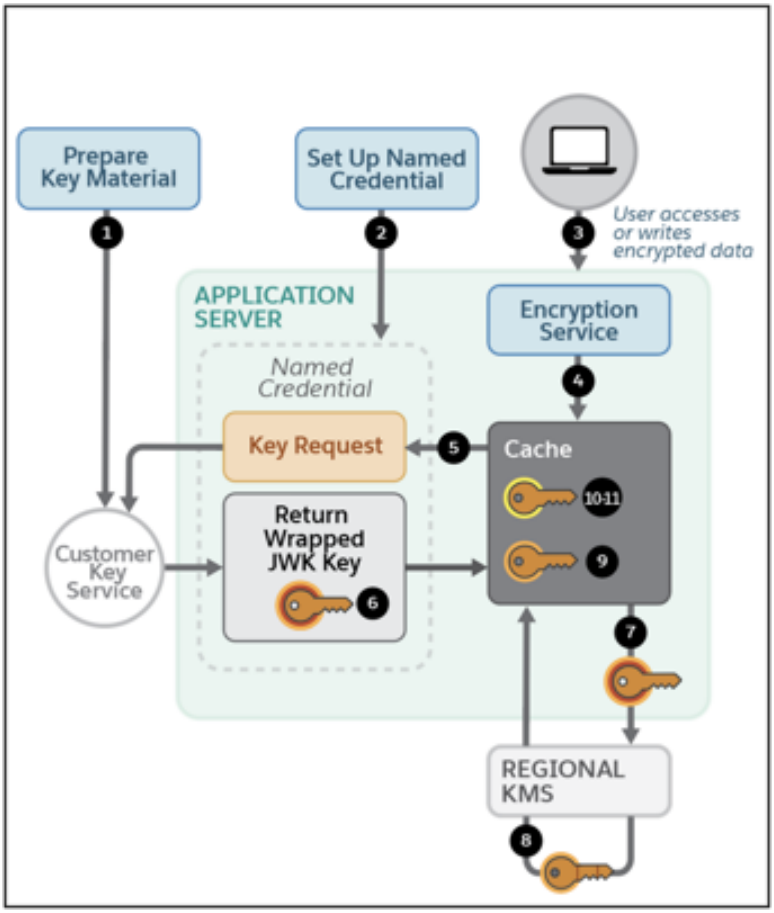


1. Users prepare their tenant secret for upload.
   a. The user generates a BYOK-compatible certificate declaratively or programmatically, where the certificate's private key is encrypted with an org-specific derived data encryption key. The user issues or creates a self-signed or certificate authority (CA)-signed certificate. The user then downloads this certificate.
   b. The user generates a 256-bit tenant secret by using the method of their choice, encrypts it with the public key from their BYOK-compatible certificate, and then encodes the encrypted tenant secret to base64.
   c. The user calculates an SHA-256 hash of the plaintext tenant secret and then encodes this hash to base64.
   d. The user uploads the encrypted tenant secret and hashed plaintext tenant secret files to Salesforce.
   e. The application server then passes the encrypted tenant secret and hashed plaintext tenant secret files to the regional Shield KMS.
   f. If customers don't opt out of derivation, the regional Shield KMS creates the BYOK-derived encryption key to unwrap the certificate's private key.
   g. The customer's uploaded tenant secret is decrypted by using the BYOK certificate's private key.
   h. The tenant secret is then hashed by using SHA-256 and compared to the SHA-256 hash provided by the customer.
2. If the hashes match, the regional Shield KMS encrypts the tenant secret with the per-release tenant wrapping key.
3. The regional Shield KMS sends the encrypted tenant secret back to the encryption service running on the Lightning Platform.
4. The encryption service stores the encrypted tenant secret securely in the database. The encrypted tenant secret is used to derive the org's specific data encryption keys on demand. This encrypted tenant secret can only be decrypted in the regional Shield KMS.

Cache-Only Key Flow

With Field-Level Encryption, you can insert a final data encryption key into the encrypted key cache. You supply a cache-only key as an encrypted JWE JSON file, wrapped with your content encryption key.

Note: Customer-supplied cache-only keys are available for application tier applications, such as FLE, Search Index Encryption, Event bus, and so on. They're not supported for Database Encryption.



1. Users prepare their key material. Cache-only key material is wrapped in a JSON format.
   a. The user generates a 256-bit AES data encryption key by using a cryptographically secure method.
   b. The user generates and downloads a BYOK-compatible certificate.
   c. The user creates a JWE protected header. The JWE protected header is a JSON object with three claims: the algorithm used to encrypt the content encryption key, the algorithm used to encrypt the DEK, and the unique ID of the cache-only key.
   d. The user encodes the JWE protected header as BASE64URL(UTF8(JWE Protected Header)).
   e. The user encrypts the content encryption key with the public key from the BYOK certificate by using the RSAES-OAEP algorithm. The user then encodes this encrypted content encryption key as BASE64URL(Encrypted CEK).
   f. The user generates an initialization vector for use as input to the DEK's AES wrapping and then encodes it in base64url.
   g. The user wraps the DEK with the content encryption key.
      i. Encode the JWE header as ASCII(BASE64URL(UTF8(JWE Protected Header))).
      ii. Reform authenticated encryption on the DEK with the AES GCM algorithm. Use the content encryption key as the encryption key, the initialization vector (the bytes, not the base64URL encoded version), and the Additional Authenticated Data value, requesting a 128-bit Authentication Tag output.
      iii. Encode the resulting ciphertext as BASE64URL(Ciphertext).

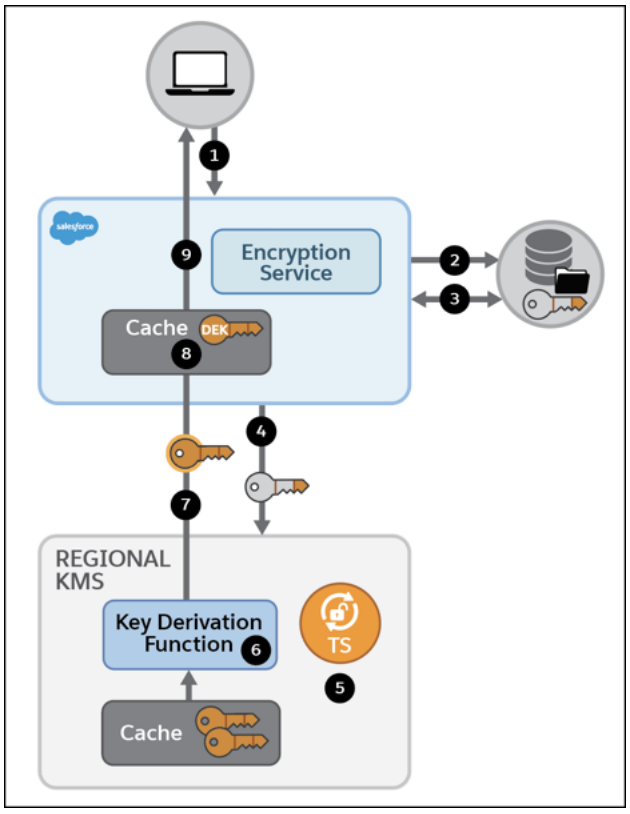
   iv. Encode the Authentication Tag as BASE64URL(Authentication Tag).

h. The user assembles the JWE as a compact serialization of all the preceding values, concatenating values separated by a period.

2. Users prepare a Salesforce named credential to serve as a secure callout connection, specifying their external key service as the endpoint. (Named principals are also supported.)
3. The user accesses or writes encrypted data.
4. The encryption service requests key material from the encrypted key cache for a DEK.
5. If no DEK is returned, the encryption service initiates a request to the endpoint specified in the customer-created named credential.
6. The customer's key service returns a content encryption key wrapped in a key encryption key, formatted as a JSON Web Encryption structure (JWE).
7. The encrypted key cache validates the response and sends the content encryption key to the regional Shield KMS to be unwrapped.
8. The regional Shield KMS unwraps the content encryption key with the HSM-generated certificate's key encryption key.
9. The regional Shield KMS returns the unwrapped content encryption key to the encrypted key cache over a TLS-secured channel.
10. The DEK is unwrapped with the content encryption key.
11. The DEK is wrapped with a cache key encrypting key and placed in the encrypted key cache for encrypt and decrypt operations.

Encrypted Information Flow with Key Derivation

When attempting to read or write encrypted data, if the DEK isn't already in the encrypted key cache, the encryption service transmits the request to the regional Shield KMS to retrieve it. For content encrypted at the application level, such as FLE, the KDF derives a DEK by using the KDF secret, KDF salt, and customer tenant secret. The derived data encryption key is returned to the encryption service.



The process for deriving the data encryption key during a decryption request includes these steps. (Decryption is nearly identical.)

1. A user attempts to read encrypted data.
2. The Lightning Platform queries the data from the storage engine.
3. Based on metadata stored with the encrypted data, the encryption service retrieves the appropriate encrypted tenant secret from the database.
4. The encryption service sends an authenticated request for the derived key to the regional Shield KMS. The request includes the encrypted tenant secret.

   Data moving between the regional Shield KMS and the encryption service is encrypted by the TLS protocol, which uses a certificate signed by a dedicated Salesforce authority. This certificate's private key is stored by the regional Shield KMS and the encryption service in an encrypted form. The certificate's public and private keys are rotated regularly.
5. The regional Shield KMS decrypts the tenant secret with the appropriate tenant wrapping key in the encrypted key cache.
6. The regional Shield KMS derives the requested data encryption key by using the appropriate KDF seed, KDF salt, and tenant secret as inputs for the key derivation function (PBKDF2WithHmacSHA256).
7. The regional Shield KMS sends the encrypted data encryption key back to the encryption service.
8. The encryption service decrypts the data encryption key. The data encryption key is encrypted with a cache key encryption key and stored in the encrypted key cache.
9. Using the data encryption key, the encryption service decrypts the customer data and returns it to the user.

PBKDF2 Inputs

Data encryption keys are derived by using PBKDF2 with these values as inputs.

- PRF—HmacSHA256
- Password—KDF seed XOR tenant secret
- Salt—KDF salt
- c—15,000
- dkLen—256

CHAPTER 5 Keys and Secrets

Shield Platform Encryption uses a hierarchy of secrets, key materials, and fragments to ensure the highest encryption service.

Glossary

For a high-level explanation of how the most important key components go together, check out Encryption Components: The Big Picture on page 5.

Cache Key Encrypting Key

Function

Org-specific key used to encrypt derived and customer-supplied data encryption keys in the encrypted key cache

Type

AES-256 key

How it's generated

Generated when a data encryption key is generated, rotated, or destroyed

Where it's stored

Encrypted with the tenant wrapping key and stored in the database

Content Encryption Key (CEK)

Function

Unique key that wraps the data encryption key supplied by a customer's specified key service

Type

AES-256 key

How it's generated

Generated by customer

Where it's stored

As part of the customer-supplied data encryption key, encrypted with the cache encryption key and stored in the encrypted key cache

Data Encryption Key (DEK)

Function

Org-specific key used to encrypt customer data. The key used for encryption and decryption.

Type

AES-256 key

How it's generated

Generated on the Shield KMS with PBKDF2 for Field-Level Encryption, HKDF for Database Encryption, or generated by the customer

Where it's stored

Never persisted on disk in any form. Customer supplied or derived on demand DEKs are stored in the encrypted key cache.

Regional HSM Encryption Key Pair

Function

Used to encrypt and decrypt data that can only be accessed on the regional KMS

Type

4096-bit RSA key pair

How it's generated

Generated upon initialization of the regional HSM

Where it's stored

Public key is signed by the primary HSM and stored in the Salesforce internal file system. The private key can't be accessed outside of the regional KMS.

Primary HSM Encryption Key Pair

Function

Used to encrypt and decrypt data that can only be accessed on the primary HSM

Type

4096-bit RSA key pair

How it's generated

Generated upon initialization of the primary HSM

Where it's stored

Public key is stored in the Salesforce internal file system. A private key can't be accessed outside of the primary HSM.

Primary HSM Signing Key Pair

Function

Used to verify the public keys of regional HSMs

Type
4096-bit RSA key pair

How it's generated
Generated upon initialization of the primary HSM

Where it's stored
Signing key pairs can't be accessed outside of the primary HSM

KDF Seed

Function
Also known as primary secret. Used in conjunction with organization tenant secrets to derive data encryption keys

Type
256-bit value

How it's generated
Generated once each release by the primary HSM

Where it's stored
Stored in the primary KMS

KDF Salt

Function
Also known as primary salt. Used as input to PBKDF2 to derive data encryption keys

Type
256-bit value

How it's generated
Generated once each release by the primary HSM

Where it's stored
Stored in the primary KMS

Primary Wrapping Key

Function
Used to encrypt the KDF seed, KDF salt, tenant wrapping key, and transit wrapping private key before they're stored in the Salesforce internal file system

Type
AES-256 key

How it's generated
Generated once each release by the primary HSM

Where it's stored
Encrypted with each regional HSM's public encryption key and the primary HSM's public encryption key and stored in the Salesforce internal file system

Root Key

Function
A special 256-bit key that's used to wrap and unwrap DEKs for EKM, Search, and BYOK encryption.

Type
256-bit key

How it's generated
Shield Platform Encryption root keys are generated on the Shield KMS. External root keys are generated by the customer KMS admin on the external KMS.

Where it's stored
Root keys are stored in a key management server, either a Salesforce KMS or an external KMS

Search Index Root Key

Function
Wrap and unwrap Search Index DEKs

Type
256-bit value

How it's generated
Generated by the customer when Search encryption is activated

Where it's stored
Stored in the regional Shield KMS

Search Index Data Encryption Key

Function
Used to encrypt and decrypt the search index segments.

Type
AES-256 key

How it's generated
Generated on the Shield KMS and wrapped with the customer's Search encryption root key.

Where it's stored
Stored in the Search service database, wrapped by the customer's Search encryption root key
Unwrapped on demand and stored in the encrypted key cache.

Tenant Secret

Function
Combined with the KDF seed to derive a unique data encryption key

Type
256-bit value

How it's generated
Generated on customer demand by the HSM on the regional Shield KMS, or uploaded by the customer

Where it's stored
Encrypted with the tenant wrapping key, sent from the regional Shield KMS to an application server on the Lightning Platform, and stored in the database

Field-Level Encryption Initialization Vector (IV)

Function
Used as input to PBKDF2 to encrypt field data

Type
128-bit value

How it's generated
Generated per encrypted field. This static IV is a hash of a field's entity ID, field ID, and key ID, making it unique to each customer and field per org. Upon key rotation, a new static IV is generated for each field. When data is encrypted deterministically, the application server computes the unique static IV for each field, and then uses the static IV to generate ciphertext.

Where it's stored
Encrypted with the tenant wrapping key, sent from the regional Shield KMS to an application server on the Lightning Platform, and stored in the database

Tenant Wrapping Key

Function
Used to encrypt tenant secrets before they're stored in the database

Type
AES-256 key

How it's generated
Generated once each release by the primary HSM

Where it's stored
Encrypted with the primary wrapping key and stored in the Salesforce internal file system

Database Tenant Secret

Function
Cryptographic key used to encrypt data stored in the database

Type
AES-256 key

How it's generated
Generated on demand by the user, or supplied as a Database Encryption BYOK

Where it's stored
The database tenant secret is cached, not stored, in the Salesforce Key Management Server (KMS) cache

Temporary System Database Tenant Secret

Function

Salesforce managed cryptographic key used to encrypt content stored in the database when a database tenant secret is being backed up

Type

AES-256 key

How it's generated

Generated whenever a new database tenant secret is created by the user

Where it's stored

Stored wrapped by the primary wrapping key in the Salesforce Key Management Server (KMS). When the Salesforce database is bootstrapped this key is stored unwrapped within the database.

Database Fragment Salt

Function

Used as input to OpenSSL HKDF derivation function, which derives the Database Encryption DEKs

Type

256-bit value

How it's generated

Generated for each transactional database fragment, which is the smallest unit of data encrypted under Database Encryption

Where it's stored

Encrypted with database tenant secret and stored in the fragment

SEE ALSO:

Get Started with Shield Platform Encryption

Use Trailhead to learn how you can go the extra mile and encrypt your data