salesforce | platform

PLATFORM ENCRYPTION ARCHITECTURE

How to protect sensitive data without locking up business functionality.

Contents

The need for encryption

There were over 63,000 security incidents and nearly 1,400 confirmed data breaches worldwide in 2013[1]. A study of 237 data breaches from the second quarter of 2014 showed that encryption was used to secure the breached data in only 4% of cases. Strong encryption and key management, the most effective tools for rendering exfiltrated data useless to attackers, were used in less than 1% of cases[2]. According to some estimates, over 3 billion records have been compromised since 2013.[3]

New security vulnerabilities, like Heartbleed, Shellshock, and POODLE, are reported on a regular basis. While Transport Layer Security (TLS) is one effective tool against data loss, researchers in the security community have demonstrated numerous techniques to compromise TLS and other encryption solutions. News reports in 2013 and 2014 confirmed that RSA accepted $10 million from the NSA to build backdoors into cryptographic libraries that are used to secure substantial chunks of Internet and corporate IT infrastructure.[4] Across the industry, there is an increasing awareness that transport security isn't enough to protect sensitive data. Additionally, attackers are targeting a wider array of targets. In the past, attacks focused on high value, financial targets – retail and point of sale, credit card processors, card issuers, and banks. Now, attackers are stealing any personally identifiable information (PII), which can enable further social engineering attacks and more significant compromises.

That's why security and trust are major factors in every company's evaluation of public cloud services. Salesforce customers in particular are choosing which business functions to run on the Salesforce1 Platform, what applications they can build to extend those functions, and what data they need to store there to enable those functions. Customers increasingly use the Salesforce1 Platform to build applications that require PII and other sensitive, confidential, or proprietary data. With this sensitive data stored on the Salesforce1 Platform, customers want additional layers of protection on top of our standard security measures. Extra features such as authentication and single sign-on, granular access controls, and advanced activity monitoring give customers control over when and how they protect their data.

1 http://www.verizonenterprise.com/DBIR/2014/reports/rp_Verizon-DBIR-2014_en_xg.pdf
2 http://www.breachlevelindex.com/pdf/Breach-Level-Index-Report-Q22014.pdf
3 http://www.breachlevelindex.com/#!home
4 http://www.reuters.com/article/2014/03/31/us-usa-security-nsa-rsa-idUSBREA2U0TY20140331

The need for encryption

Balancing 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 both external regulatory and internal data compliance policies. Internal policies frequently rely on interpretation of external regulations. As customers look at regulations such as PCIDSS and HIPAA/HITECH through the lens of cloud-based service adoption, they typically take a pragmatic but conservative approach to data protection in the cloud.

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 Salesforce1 Platform functionality.

However, if data is encrypted at rest – depending on where encryption and decryption occurs and where the encryption keys are stored – preserving Salesforce functionality becomes difficult, if not impossible. There's a tradeoff between strong security and functionality. What the business wants often differs from what security and compliance require.

The need for encryption

Principles and design goals

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

1. **Encrypt data at rest.**
   The Salesforce Platform Encryption solution encrypts data at rest when stored on our servers, both in the database and the file system. We don't address data residency or remote key management, which require off-Salesforce solutions and typically involve on-premises software and complex integrations. To encrypt data at rest and preserve functionality, we built the encryption services natively into the Salesforce1 Platform.

2. **Natively integrate encryption at rest with key Salesforce features.**
   One of the things that makes the Salesforce1 Platform so remarkable is that it is driven by metadata. 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.

3. **Use strong encryption.**
   The Platform Encryption solution uses strong, probabilistic encryption on data stored at rest. Platform Encryption uses the Advanced Encryption Standard (AES) with 256-bit keys using CBC mode, PKCS5 padding, and random initialisation vector (IV). We opted for probabilistic encryption over deterministic encryption. This type of encryption results in a loss of some functionality, such as sort and group-by operations, but we consider this a reasonable tradeoff in favor of security. However, we recognise that in some cases, business requirements depend on preserving more functionality, which might influence what data customers decide to encrypt.

4. **Enable customers to drive the key lifecycle.**
   We built a key management framework that scales to our massively multi-tenant model and gives you complete control over the key management lifecycle. Since the encryption service is built natively into the Salesforce1 Platform, the encryption keys must reside in the Salesforce environment. Adhering to the principle that customers should have complete control over the key lifecycle, we built key management functionality into the Setup UI and API such that customers decide when to generate, rotate, import, export, and destroy keys. Customers also determine who is responsible for performing these tasks. As with all administration tasks, everything is audited.

5. **Protect keys from unauthorized access.**
   A primary consideration when architecting 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 hardware security modules (HSMs)[5] into the infrastructure. Platform Encryption uses HSMs to generate cryptographic secrets used to derive organisation-specific data encryption keys. The result is a shared key management service that creates tenant-specific derived keys. The keys aren't persisted; they are therefore inaccessible to Salesforce employees and, by extension, malicious external attackers.

6. **Encrypt as little data as possible.**
   Our design gives customers control over what data they encrypt. Your organisation administrator chooses whether to turn on encryption for standard fields, custom fields, files, and attachments. You also choose which specific fields to encrypt at rest. The driving principle is to encrypt as little as possible to preserve functionality while keeping private, sensitive, confidential, and regulated data safe.

5 http://en.wikipedia.org/wiki/Hardware_security_module

The need for encryption

Before you encrypt

Before you decide to encrypt data in Salesforce – or in any cloud service – first make sure you're matching the right security solution to the type of threats you face. If, for example, you are most concerned about protecting against end-user or administrative account takeover attacks, which are usually achieved through social engineering and malware infection, data encryption may not be an appropriate control against such a threat. Consider instead malware detection and activity monitoring as ways to identify when users may have been compromised and a malicious outsider is attempting to gain access to data.

Salesforce Platform Encryption protects data at rest. It shouldn't be confused with a control that encrypts data in transit, such as Transport Layer Security[6] (which Salesforce enables by default for your org). Platform Encryption is best suited for:

- Protecting against data loss 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 adopting a defense-in-depth strategy that takes advantage of all the security features Salesforce offers. The Security Implementation Guide gives a comprehensive overview of the customer-controlled security capabilities available.

After completing a threat modeling exercise, use the outcome to inform a granular data classification. Identify data elements that are sensitive, private, or confidential. Your resulting strategy should be to encrypt only the most sensitive of those data elements. This will help to balance stronger data protection controls against the need to build and preserve critical business functionality on the Salesforce Platform or when using Sales Cloud or Service Cloud.

6 https://en.wikipedia.org/wiki/Transport_Layer_Security

What it does

Platform Encryption allows you to encrypt fields, files, and attachments stored on the Salesforce1 Platform. In contrast to Classic Encryption, which uses a custom field type in the Salesforce data model, Platform Encryption allows you to encrypt several standard fields and custom field types using metadata while preserving functionality necessary to perform common business tasks in Salesforce. Salesforce administrators can:

- Encrypt files and attachments
- Encrypt the following standard fields: Account Name, Contact Name, Mailing Address, Phone, Fax, Mobile, Home Phone, Other Phone, and Email
- Encrypt the following custom field types: Text, Long Text Area, Phone, Email, and URL
- Manage the lifecycle of data encryption keys

This table compares the features of Platform Encryption and Classic Encryption.

Feature	Classic Encrypted Custom Fields (included in base user liscence)	Platform Encryption (additional fee applies)
Encryption at Rest	✓	✓
Native Solution (No Hardware or Software is Required)	✓	✓
Encryption Algorithm	128-bit Encryption Standard (AES)	256-bit Advanced Encryption Standard (AES)
HSM-based Key Derivation		✓
“Manage Encryption Keys” Permission		✓
Generate, Export, Import, and Destroy Keys	✓	✓
PCI-DSS L1 Compliance	✓	
Text (Encrypted) Field Type	✓ (Dedicated custom field type, limited to 175 characters)	
Masking	✓	✓
Mask Types and Characters	✓	
“View Encrypted Data” Permission is Required to Read Encrypted Field Values	✓	✓
Email Template Values Respect “View Encrypted Data” Permission		✓
Encrypted Standard Fields		✓*
Encrypted Attachments, Files and Content		✓
Encrypted Custom Short Text, Long Text Area, Phone, Email, and URL Fields		✓
Encrypt Existing Fields for Supported Custom Field Types		✓
Search (UI, Partial Search, Lookups)		✓
API Access	✓	✓
Available in Workflow Rules and Workflow Field Updates		✓
Available in Approval Process Entry Criteria and Approval Step Criteria		✓

*On the Account object, you can encrypt **Account Name.** On the Contact object, you can encrypt **Email, Fax, Home Phone, Mailing Address (Mailing street and Mailing city), Mobile, Name (First Name, Middle Name, and Last Name), Other Phone, and Phone.**

How it works

To meet the security requirements of customers while preserving functionality and performance in our multi-tenant environment, we built the encryption service directly into the Force.com platform. The platform's object-relational mapping model includes metadata that describes exactly which data is encrypted. Encrypted data is stored with additional information that enables the use of strong, nondeterministic cryptography supported by the Java Cryptographic Extension. Encryption and decryption occur in the platform's persistence layer, as application components are materialised by the runtime engine, ensuring that encrypted data isn't persisted in plaintext. Encryption keys are derived on demand from key material generated by HSMs and never persisted. Finally, the architecture supports the simultaneous use of multiple encryption keys, enabling customers to quickly rotate and archive keys without losing access to their data.

Encryption in Force.com platform's persistence layer

Force.com's foundation is a metadata-driven software architecture that enables multi-tenant 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 materialises the data using metadata stored separately in Force.com's Universal Data Dictionary (UDD). This way, each tenant's data is kept secure in the shared database, tenants can customise schema in real time without affecting other tenants' data, and the application's code base can be patched or upgraded without breaking tenant-specific customisations. See The Force.com Multitenant Architecture for details.

The UDD includes metadata that determines which data is encrypted at runtime. The encryption service works in the Force.com platform's persistence 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. In the case of decryption, data is decrypted as it's materialised. It is then pushed up through the application pipeline and appears in plaintext to the user who requested it. In the case of field data, if the user who requested the data does not have the "View Encrypted Data" permission, the data is never decrypted and appears masked to the user.

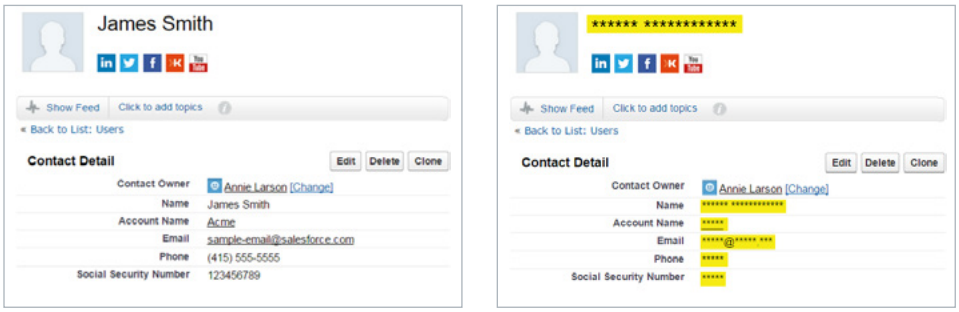

Figure 1: Encrypted field values are only decrypted when requested by users with the "View Encrypted Data" permission (left). Encrypted data appears masked to users without the permission (right).

By embedding the encryption metadata in the UDD, the Platform Encryption architecture allows customers to choose what data to encrypt. Performing the encryption work in the Platform Encryption's persistence layer enables the engine to strictly manage the flow of encrypted data from the application to the database and vice versa, since the relevant code paths run through the UDD before data is read or stored.

Cryptographic library and algorithms

Platform Encryption uses the Java Cryptography Extension (JCE), to encrypt and decrypt data. Specifically, Platform Encryption uses the Advanced Encryption Standard (AES-256) in CBC mode with randomised IV and PKCS5 for padding. The JCE class SecureRandom is used to generate the IV.

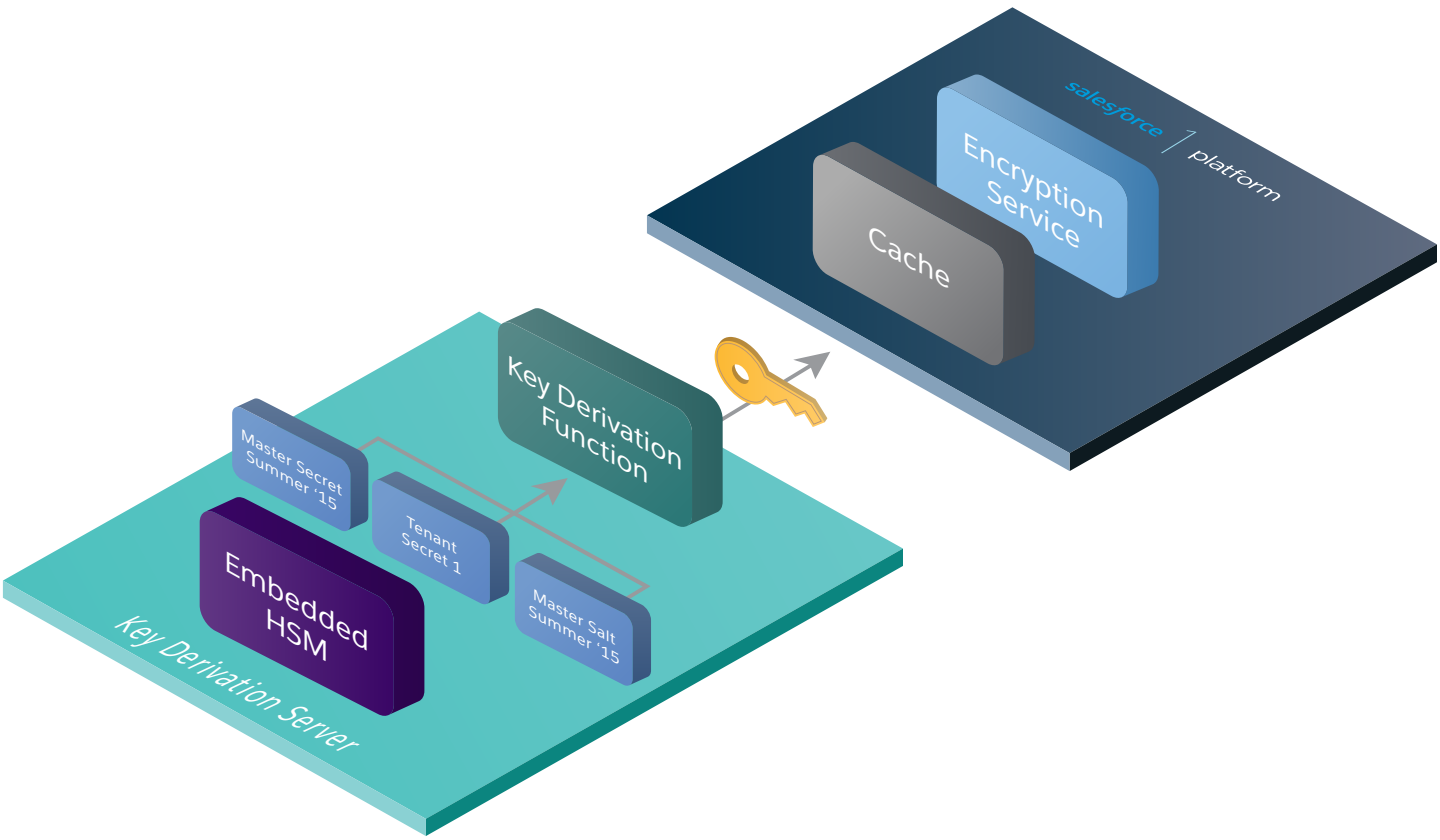

Figure 2: Deriving data encryption keys. For details, see Key Derivation Architecture.

Data encryption keys

The AES-256 keys used to encrypt customer data aren't persisted. Instead, they're derived on demand from secrets generated by logically and physically separated HSMs. The master secret is generated at the start of each Salesforce release and stored securely in Salesforce's internal file system. The customer-specific tenant secret is generated by customers on demand and stored securely in the database. These secrets, along with a master salt generated at the start of each release, are used as inputs to Password-Based Key Derivation Function 2 (PBKDF2) to derive data encryption keys. PBKDF2 is run on a key derivation server in a Salesforce data center. Once derived, data encryption keys are sent (encrypted) back to the encryption service running on the Salesforce1 Platform and stored in the cache of a platform application server until the cache is flushed.

Storing encrypted payloads

Encrypted data is stored in the database with its metadata, including:

- A bit that indicates the field contains ciphertext
- The ID of the customer's tenant secret used to derive the matching encryption key
- A randomised, 128-bit initialization vector (IV) for cryptographic use

The tenant secret ID is used to locate the tenant secret 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 a key derivation server, which uses the tenant secret and corresponding master secret, identified by the tenant secret creation date, to derive a data encryption key. The random IV is used with the encryption key to nondeterministically encrypt the data.

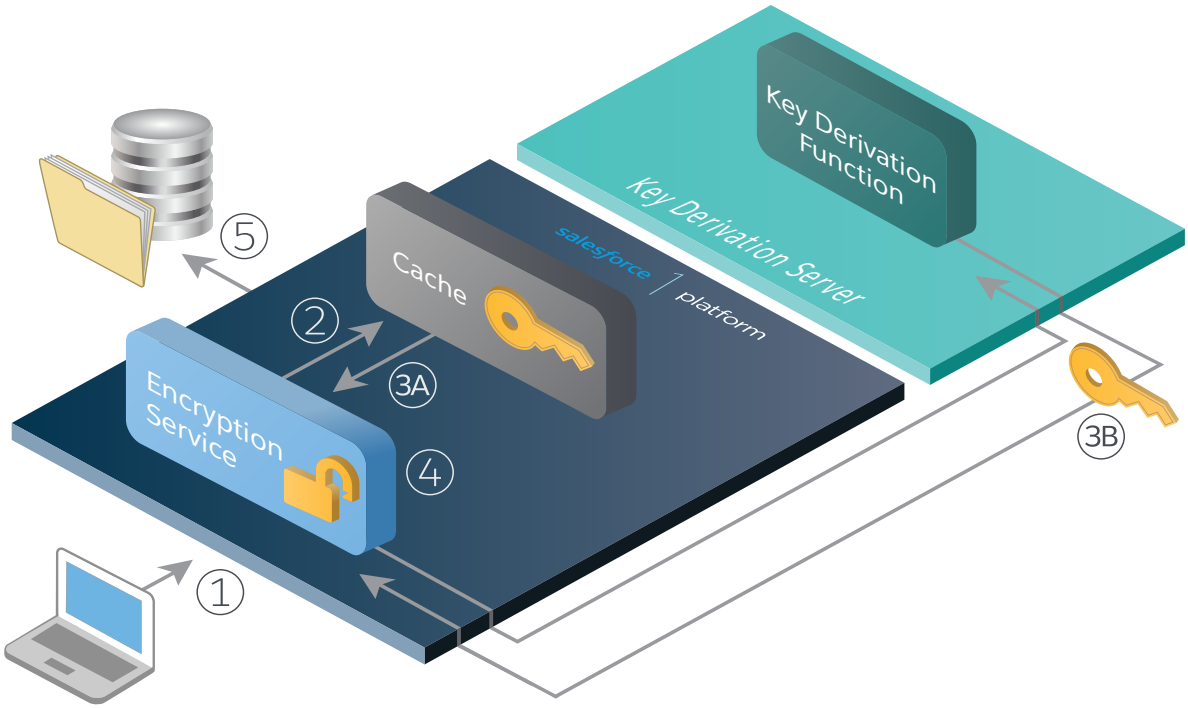

Figure 3: Process flow for encrypting data.

Platform Encryption process flow

Before data is encrypted, a Salesforce administrator must enable encryption and generate a tenant secret. For each field, file, and attachment on which encryption is enabled, the corresponding metadata in the UDD is updated to reflect the new encryption setting. Users must have the "View Encrypted Data" permission to view encrypted field data.

1. When a Salesforce user saves encrypted data, the runtime engine determines from metadata whether the field, file, or attachment should be encrypted before storing it in the database.
2. If so, the encryption service checks for the matching data encryption key in cached memory.
3. The encryption service determines if the key exists.
   a. If so, the encryption service retrieves the key.
   b. Otherwise, the service sends a derivation request to a key derivation server and returns it to the encryption service running on the Salesforce1 Platform.
4. After retrieving or deriving the key, the encryption service generates a random initialisation vector (IV) and encrypts the data using JCE's AES-256 implementation.
5. The ciphertext is saved in the database or file storage. The IV and corresponding ID of the tenant secret used to derive the data encryption key are saved in the database.

How it works

Key management

Platform Encryption allows Salesforce administrators to manage the lifecycles of their data encryption keys while protecting the keys from unauthorised access. To ensure this level of protection, data encryption keys are never persisted on disk. Instead, they're derived on demand from the master and tenant secrets.

The master secret is generated by a master HSM at the start of each release. The master HSM is "air-gapped" from Salesforce's production network and stored securely in a bank safety deposit box. Only designated Salesforce security officers can access the safety deposit box and the master HSM stored within.

Tenant secrets are generated on demand using HSMs embedded in key derivation servers deployed in clusters to each of Salesforce's data centers. Each key derivation server has access to the release-specific secrets for every Salesforce release (e.g. Summer '15, Winter '15, Spring '16, etc). When a data encryption key is needed to encrypt or decrypt customer data, the server derives the key from the master and tenant secrets.

By controlling the lifecycle of your organisation's tenant secrets, you control the lifecycle of the derived data encryption keys. Your Salesforce administrator specifies a user to manage the tenant secrets for your organisation and assigns that user the "Manage Encryption Keys" user permission. This user permission allows the key administrator to generate, archive, export, import, and destroy tenant secrets.

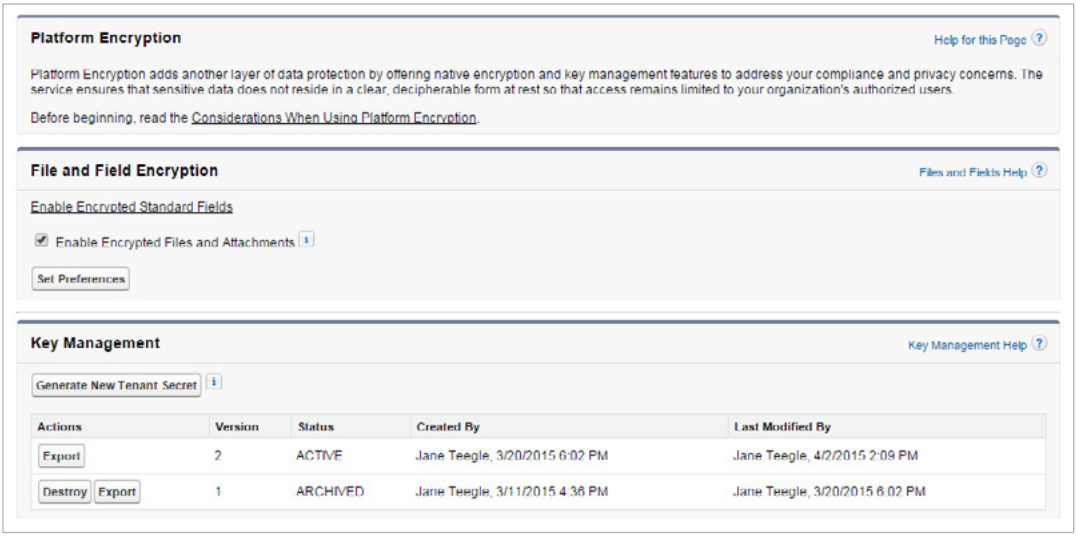

Figure 4: Platform Encryption page with Key Management features

As Figure 4 illustrates, it is possible to have more than one tenant secret in a Salesforce organisation. Only the most recent tenant secret is active, meaning only that tenant secret is used to derive the data encryption key used to encrypt data. When you generate a new tenant secret, the active secret becomes archived. Archived tenant secrets are used to decrypt data that was last encrypted when the archived tenant secret was active.

You can destroy an archived tenant secret. If you do, it is no longer possible to derive the encryption key required to decrypt the data that was encrypted using that key. You must take special care to backup and protect both archived tenant secrets and encrypted data.

Key rotation and data re-encryption

When you rotate tenant secrets by generating a new one, the resulting derived data encryption keys rotate as well. New data is encrypted and decrypted using the data encryption key derived from the new, active tenant secret. Existing data stays encrypted with the former key. Salesforce can run a background process to traverse the database and file storage, decrypt existing encrypted data, and then re-encrypt the data using the new data encryption key. This process is transparent to users and administrators and must be initiated by a Salesforce support engineer.

How it works

Key derivation architecture

Platform Encryption utilises key derivation servers to derive data encryption keys for encrypting customer data at rest. The keys are derived from fragmented secrets that are securely wrapped and stored in Salesforce's internal file system, ensuring that the keys are never persisted in their composite forms and enabling customers to control the key lifecycle. These secrets and secret-wrapping keys are initialised by a master HSM at the start of each release, or in the case of customer-driven tenant secrets, on demand in production environments by HSMs embedded in the key derivation servers.

The key derivation architecture includes the following processes:

- HSM initialisation. Before they're put to use, both the master and embedded HSMs are initialised, which includes the creation of their respective encryption key pairs.
- Per-release secret generation. At the start of each release, the master HSM is plugged into an offline laptop and used to generate the per-release secrets. The secrets are hashed and stored in Salesforce's internal file system for consumption by the embedded HSMs.
- Key derivation server startup. When a key derivation server starts up, it accesses each release's encrypted secrets in the internal file system. Then it decrypts the secrets and stores them in its cache in preparation for key derivation.
- On-demand tenant secret generation. One of the inputs into the key derivation function that creates your organisation's data encryption key is an organisation-specific, customer-managed tenant secret. Customers control the lifecycle of their data encryption keys by generating new tenant secrets. When a customer generates a new tenant secret, the request is sent to the key derivation server from the application server and authenticated. Then, an embedded HSM generates a tenant secret, which is encrypted by the key derivation server and sent back to the application server to be stored in the database.
- Key derivation in production environments. When a customer attempts to read or write encrypted data and the corresponding data encryption key isn't cached, the application server sends a derivation request to the key derivation server. The key derivation server authenticates the request and derives the key using the secrets in its cache. The key is then transmitted securely back to the application server.

HSM initialisation, secret generation, and key derivation rely on the following components:

- Master HSM (SafeNet® Luna G5). A FIPS 140-2 hardware-compliant USB device that generates per-release secrets and secret-wrapping keys, and signs the public keys of embedded HSMs. The master HSM is air-gapped from the network at all times and stored in a bank safety deposit box. Access to the master HSM is restricted to designated Salesforce security officers.
- Offline laptop. A machine that the master HSM plugs into while in use. The offline laptop exports the secrets and keys generated by the master HSM to the internal Salesforce file system.
- Embedded HSMs (SafeNet® Luna PCI-E). FIPS 140-2 hardware-compliant PCI devices that are plugged into key derivation servers in Salesforce data centers. Embedded HSMs unwrap secrets that were generated by the master HSM, encrypted, and exported to the Salesforce internal file system. They also generate tenant secrets, the customer-managed fragments of data encryption keys.
- Key derivation servers. Clusters of load-balanced servers deployed to Salesforce's production data centers that derive data encryption keys from master secrets and tenant secrets.
- 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 a key derivation server to process the request.
- Salesforce internal file system and source control. The location and source control mechanism for storing encrypted secrets and their hashes.

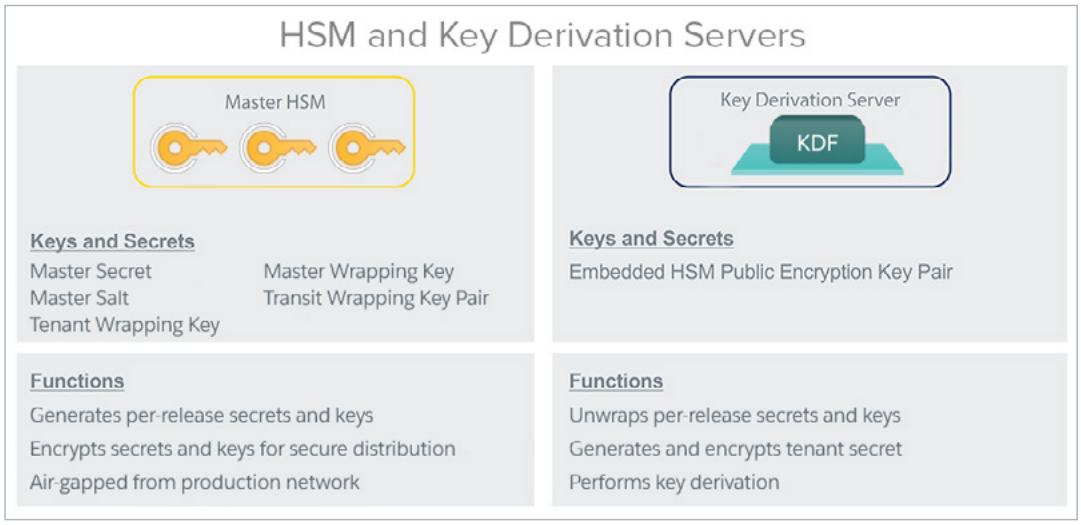

Figure 5: Key derivation architecture components

HSM Initialization

The master HSM and the embedded HSMs must be initialised before they're used. For the master HSM, the initialisation process includes creating a master HSM encryption key pair and a master HSM signing key pair. For each embedded HSM, the process includes creating an embedded HSM encryption key pair. The master HSM public signing key is used to sign and verify each embedded HSM's public encryption key. At the start of each release, the master and embedded HSM public encryption keys are used to separately encrypt a per-release master wrapping key, which is in turn used to encrypt the remainder of the per-release secrets used to derive data encryption keys. This way, each embedded HSM is able to securely access the master 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.

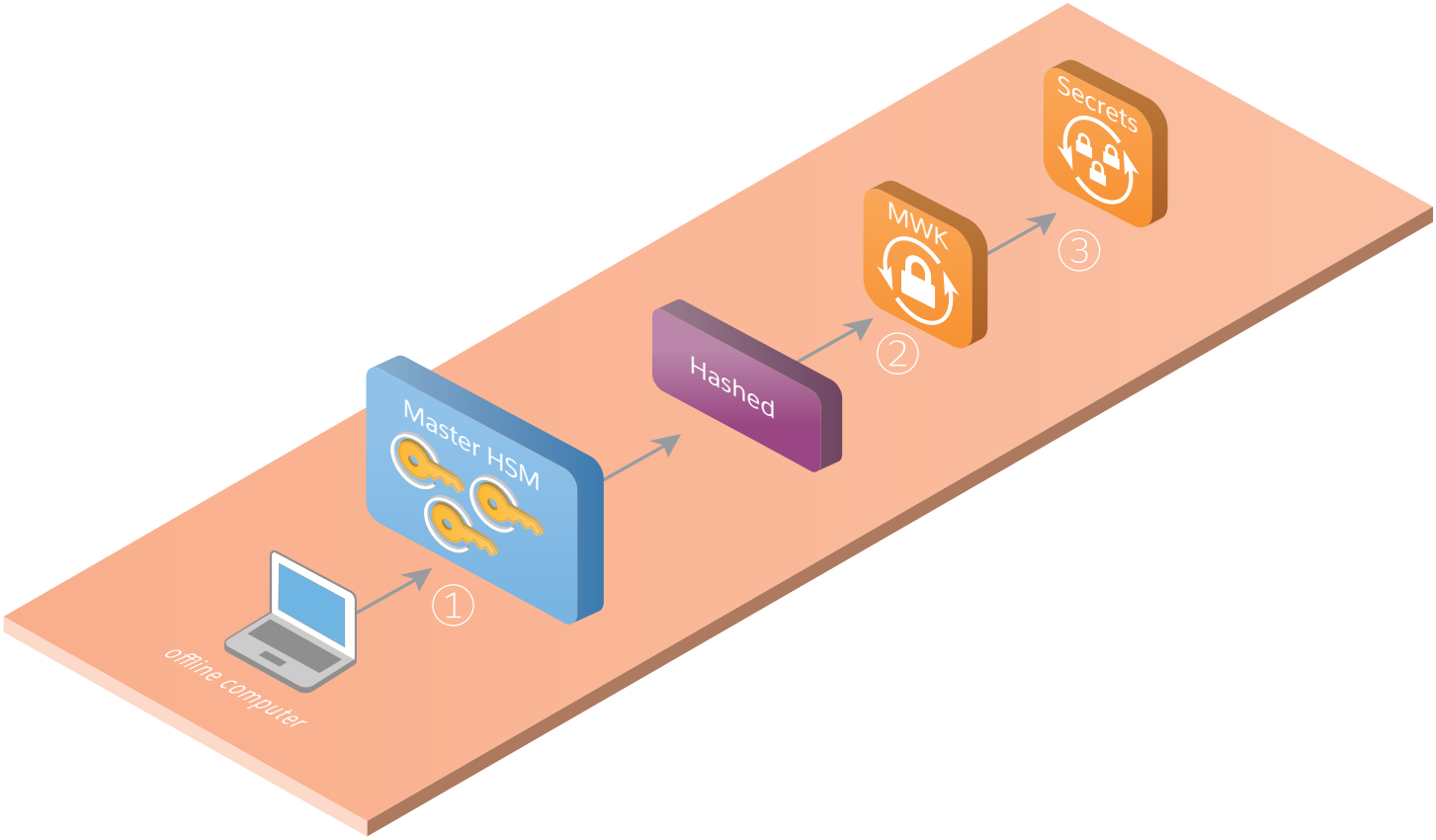

Figure 6: Per-release secret generation

Per-release secret generation

At the start of each release, the master HSM is plugged into the offline laptop and used to generate the per-release secrets and keys (on the HSM itself).

1. The master HSM generates the following secrets:

   - master secret
   - master salt
   - master wrapping key
   - tenant wrapping key
   - transit wrapping key pair

   For definitions of each secret and key, refer to the Key and Secret Glossary. Each secret is hashed using SHA-256.

2. The master wrapping key (MWK) is encrypted with the master HSM public encryption key and stored locally on the laptop along with its hash.

3. Each other secret is encrypted with the master wrapping key and stored on the laptop with their hashes.

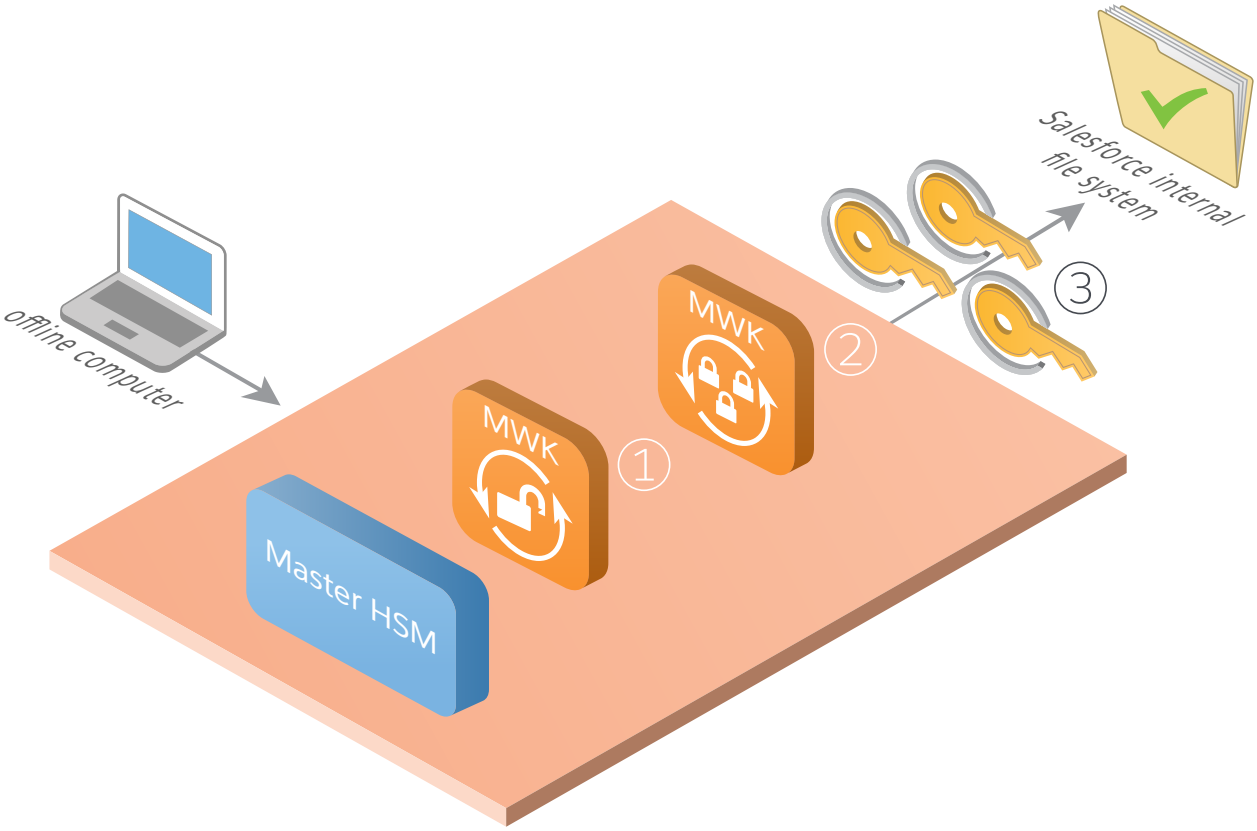

Figure 7: Per-release secret exportation

Once all the secrets are hashed and encrypted, they are checked into source control and exported to the Salesforce internal file system. Additionally, the plaintext master wrapping key is encrypted with each embedded HSM's public encryption key, checked into source control, and stored in the Salesforce internal file system. Each embedded HSM can access the per-release secrets for key derivation by first decrypting the master wrapping key, then using it to decrypt the remaining secrets.

The process for exporting the secrets includes these steps:

1. The master wrapping key is read from the file system of the offline laptop and decrypted on the master HSM.
2. For each signed and embedded HSM, the master wrapping key is encrypted with that HSM's public encryption key.
3. The encrypted secrets and their hashes are checked into source control and stored in the Salesforce internal file system.

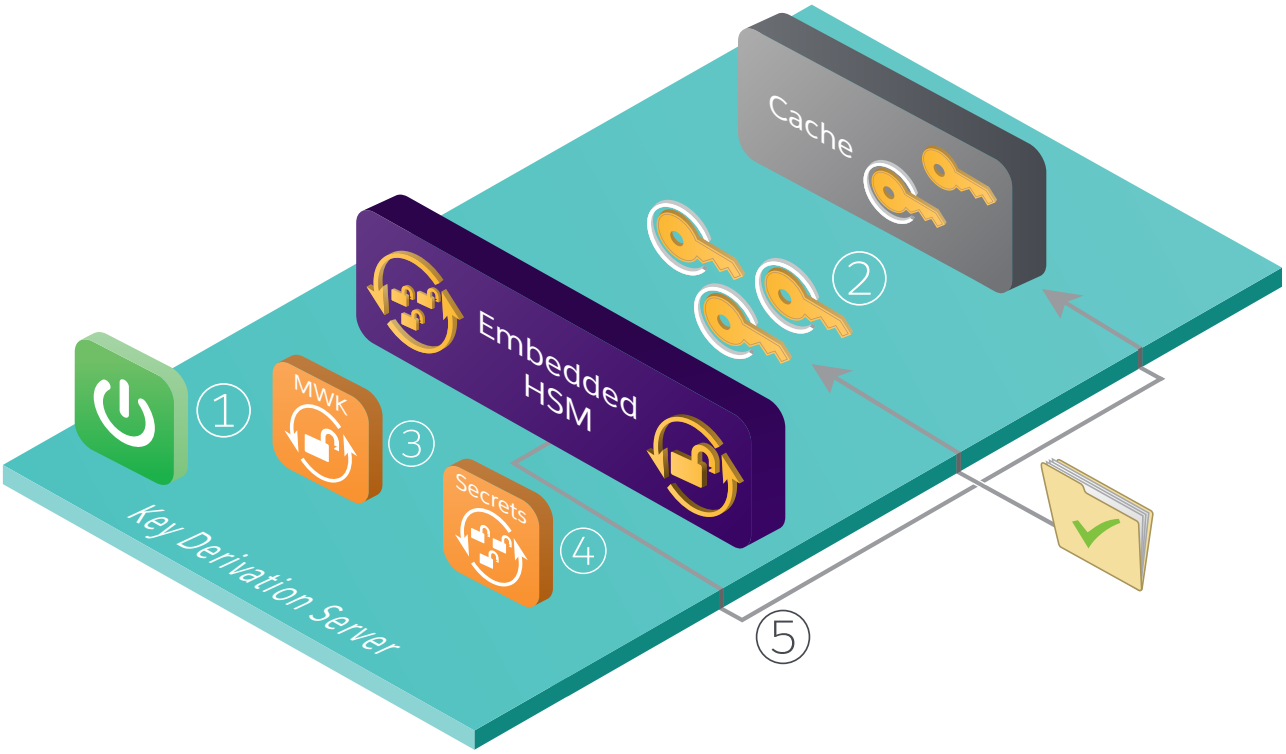

Figure 8: Key derivation server startup

Key derivation server startup

When a key derivation server starts up in a production environment, it accesses each release's encrypted secrets stored in the internal file system, decrypts and validates them, and stores them in its cache in preparation for deriving data encryption keys. The process includes these steps.

1. The key derivation server starts up.
2. The key derivation server accesses the encrypted secrets and their hashes in the appropriate folders in the internal file system.
3. The embedded HSM decrypts the master wrapping key for each release.
4. Using the master wrapping keys, the key derivation server decrypts the rest of the release secrets.
5. The key derivation server validates all the keys and secrets against their hashes and then stores them in its cache.

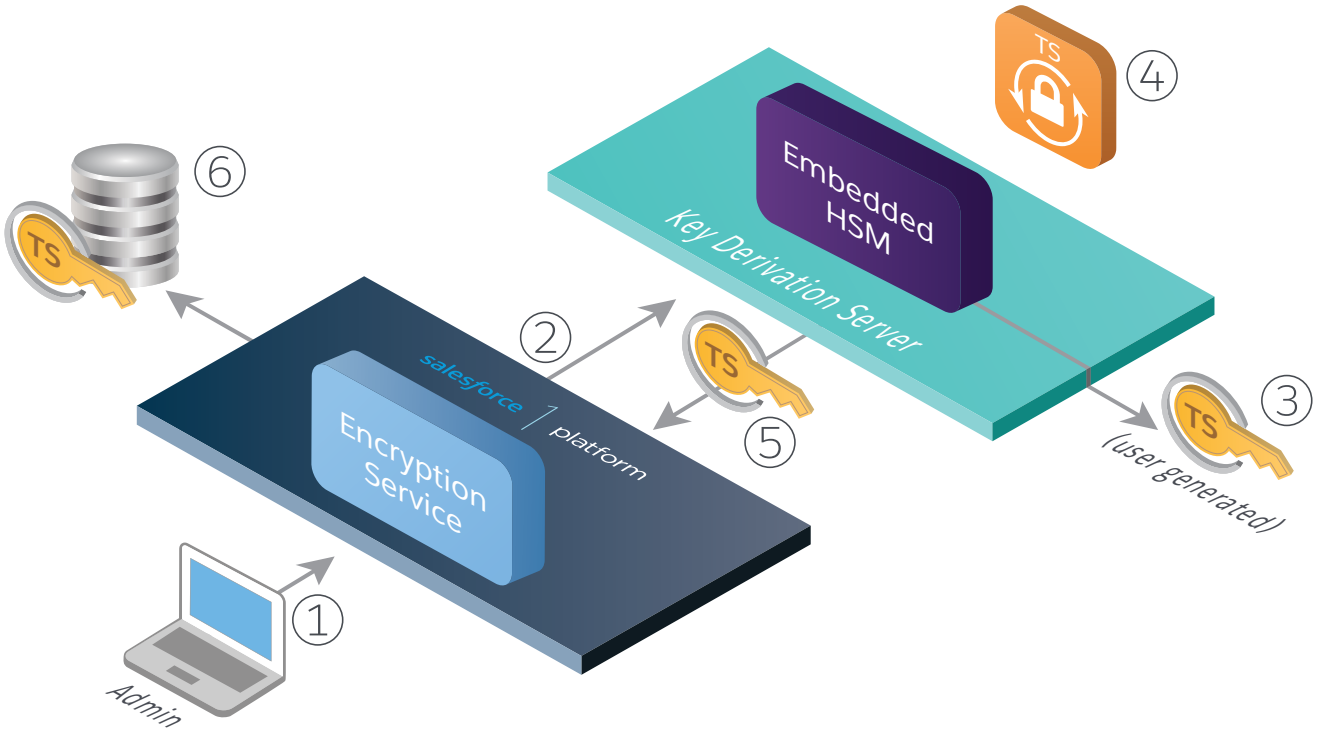

Figure 9: On-demand tenant secret generation

On-Demand Tenant secret generation

Customers can generate tenant secrets every 24 hours in their production or Developer Edition organisations and every four hours in their sandbox organisations. Tenant secrets can be destroyed at any time. When a customer generates a new tenant secret, all future data is encrypted with the key derived from the current master secret and the new tenant secret. The tenant secret is generated by an embedded HSM connected to a key derivation server. The process of generating a tenant secret includes these steps:

1. An admin attempts to generate a new tenant secret using the UI or API.
2. The encryption service sends an authenticated request to a key derivation server.
3. The embedded HSM generates the tenant secret.
4. The key derivation server encrypts the tenant secret with the per-release tenant wrapping key.
5. The key derivation server sends the encrypted tenant secret back to the encryption service running on the Salesforce1 Platform..
6. The encryption service stores the encrypted tenant secret in the database.

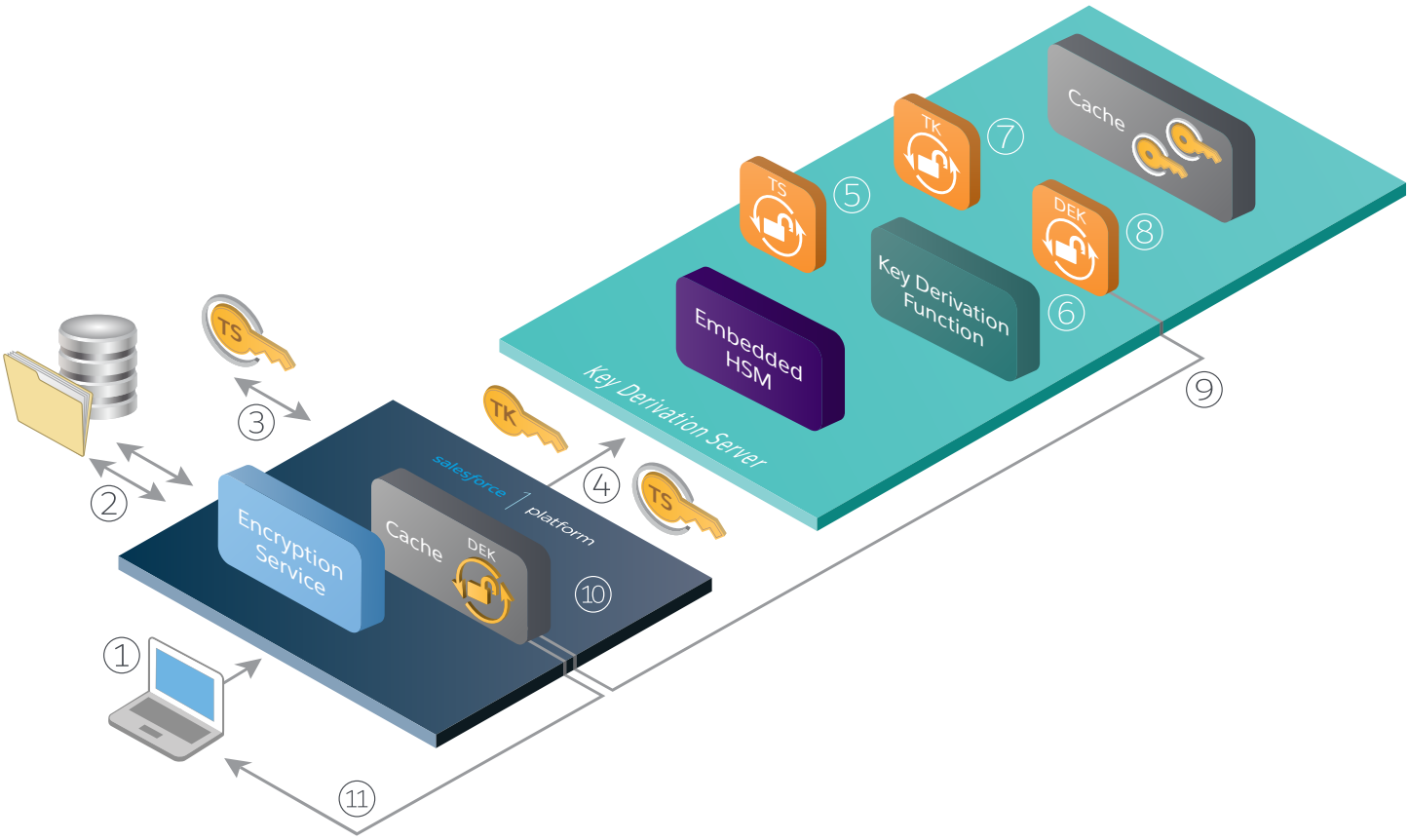

Figure 10: Key derivation in production

Key derivation

When a customer attempts to read or write encrypted data, the encryption service transmits the request to a key derivation server to retrieve the appropriate data encryption key (unless the key is already in the application server cache). Once the data encryption key is returned to the encryption service, the key is stored in memory for future requests until the cache is flushed. The process for deriving the data encryption key during a decrypt request includes these steps (note: encryption is nearly identical):

1. A user attempts to read encrypted data.
2. The Salesforce1 Platform queries the data from the storage engine. This could be the database or file storage.
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 a key derivation server. The request includes the following information:
   - The encrypted tenant secret
   - A unique transit key (TK) that's generated on the a Salesforce1 Platform application server each time it boots up. The transit key is used to encrypt the derived data encryption key before it's sent back to the encryption service. The transit key is itself encrypted by the encryption service with the transit wrapping public key, which is half of the transit wrapping key pair generated by the master HSM each release.
5. The key derivation server decrypts the tenant secret with the appropriate tenant wrapping key in its cache.
6. The key derivation server derives the requested data encryption key using the appropriate master secret, master salt, and tenant secret as inputs to the key derivation function (PBKDF2WithHmacSHA256).
7. The key derivation server decrypts the application server's transit key with the transit wrapping private key.
8. The key derivation server encrypts the data encryption key (DEK) with the transit key. This ensures that the data encryption key isn't transmitted in the clear.
9. The key derivation server sends the encrypted data encryption key back to the encryption service.
10. The encryption service decrypts the data encryption key and caches it.
11. 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 using PBKDF2 with the following values as inputs.

- *PRF*–HmacSHA256
- *Password*–master secret XOR tenant secret
- *Salt*–master salt
- *c*–15,000
- *dkLen*–256

Glossary

These are the keys and secrets used in the key derivation architecture.

Data encryption key

FUNCTION:	Organisation-specific key used to encrypt customer data (the "final" key)
TYPE:	AES-256 key
HOW IT'S GENERATED:	Generated on a key derivation server with PBKDF2
WHERE IT'S STORED:	Never persisted on disk in any form. Derived on demand and stored in the cache of an application server on the Salesforce1 Platform.

Embedded HSM encryption key pair

FUNCTION:	Used to encrypt and decrypt data that can only be accessed on the embedded HSM
TYPE:	4096-bit RSA key pair
HOW IT'S GENERATED:	Generated once, upon initialisation of embedded HSM[7]
WHERE IT'S STORED:	Public key is signed by master HSM and stored in the Salesforce internal file system. Private key cannot be accessed outside of embedded HSM.

Master HSM encryption key pair

FUNCTION:	Used to encrypt and decrypt data that can only be accessed on the master HSM
TYPE:	4096-bit RSA key pair
HOW IT'S GENERATED:	Generated once, upon initialisation of master HSM[8]
WHERE IT'S STORED:	Public key is stored in the Salesforce internal file system. Private key cannot be accessed outside of master HSM.

Master HSM signing key pair

FUNCTION:	Used to verify the public keys of embedded HSMs
TYPE:	4096-bit RSA key pair
HOW IT'S GENERATED:	Generated once, upon initialisation of master HSM[8]
WHERE IT'S STORED:	Signing key pair cannot be accessed outside of master HSM

7 http://www.safenet-inc.com/data-encryption/hardware-security-modules-hsms/luna-hsms-key-management/luna-pci-e/#tab2

8 http://www.safenet-inc.com/data-encryption/hardware-security-modules-hsms/luna-hsms-key-management/luna-G5-usb-attached-hsm/#tab2

Master salt

FUNCTION:	Used as input to PBKDF2 to derive data encryption keys
TYPE:	256-bit value
HOW IT'S GENERATED:	Generated once each release by the master HSM[7]
WHERE IT'S STORED:	Encrypted with master wrapping key and stored in the Salesforce internal file system

Master secret

FUNCTION:	Used in conjunction with organisation tenant secrets to derive data encryption keys
TYPE:	256-bit value
HOW IT'S GENERATED:	Generated once each release by the master HSM[7]
WHERE IT'S STORED:	Encrypted with master wrapping key and stored in the Salesforce internal file system

Master wrapping key

FUNCTION:	Used to encrypt the master secret, master salt, tenant wrapping key, and transit wrapping private key before they are stored in the Salesforce internal file system
TYPE:	AES-256 key
HOW IT'S GENERATED:	Generated once each release by the master HSM[7]
WHERE IT'S STORED:	Encrypted with each embedded HSM's public encryption key and the master HSM's public encryption key and stored in the Salesforce internal file system

Tenant secret

FUNCTION:	Combined with the master secret to derive a unique data encryption key
TYPE:	256-bit value
HOW IT'S GENERATED:	Generated on customer demand by an embedded HSM on a key derivation server
WHERE IT'S STORED:	Encrypted with the tenant wrapping key, sent from the key derivation server to an application server on the Salesforce1 Platform, and stored in the database

Tenant wrapping key

FUNCTION:	Used to encrypt tenant secrets before they are stored in the database
TYPE:	AES-256 key
HOW IT'S GENERATED:	Generated once each release by the master HSM[7]
WHERE IT'S STORED:	Encrypted with the master wrapping key and stored in the Salesforce internal file system

Transit key

FUNCTION:	Used to encrypt derived data encryption keys before they are sent from the key derivation server to the an application server on the Salesforce1 Platform
TYPE:	AES-256 key
HOW IT'S GENERATED:	Generated on each application server upon startup using the JCE class SecureRandom, and sent to the key derivation server upon a key derivation request
WHERE IT'S STORED:	Stored in application server memory. Never persisted on disk.

Transit wrapping key pair

FUNCTION:	Used to encrypt the transit key before it's sent from the application server to a key derivation server
TYPE:	4096-bit RSA key pair
HOW IT'S GENERATED:	Generated once each release by the master HSM[7]
WHERE IT'S STORED:	Public key is stored in the Salesforce internal file system. Private key is encrypted with master wrapping key and stored on the Salesforce internal file system.

Get Started with Platform Encryption

To see how Platform Encryption can help your company, contact your account executive or call **1800-667-638 (AU), 0800-450-064 (NZ)** today.

CONTACT ME