PUBLIC

SAP HANA Platform 2.0 SPS 04

Document Version: 1.1 – 2019-10-31

SAP HANA Client-Side Data Encryption Guide

THE BEST RUN

Content

1 SAP HANA Client-Side Data Encryption Guide

This guide provides conceptual and usage information about client-side data encryption in SAP HANA. Client-side data encryption is a column-level data encryption capability that allows you to selectively encrypt data and delivers a built-in protection of business-critical information from third-party database and cloud administrators, accidental exposure, and other data breaches.

For SAP HANA server-side data encryption services, key management, and all other security features, refer to the SAP HANA Security Guide.

1.1 Client-Side Data Encryption Information Map

This guide is intended for technical users, primarily database administrators and database application developers who want to use the SAP HANA client-side encryption capability to encrypt sensitive data and store it in the SAP HANA server.

Information Organization

Topic	Documentation
A conceptual overview of the client-side data encryption feature.	• Client-Side Data Encryption Overview [page 5] • Components of Client-Side Data Encryption [page 6]
Information on encrypted columns, column encryption keys (CEKs), and client key pairs (CKPs). It lays a conceptual foundation for the practical information provided in the subsequent chapters.	• Getting Started With Client-Side Data Encryption [page 17] • Encrypting Application Data with Client-Side Data Encryption [page 10] • Column Encryption Keys [page 11] • Client Key Pairs [page 10] • Encrypted Columns [page 12] • Putting the Pieces Together [page 26]
Information on how to use the `hdbkeystore` tool to execute encryption commands.	hdbkeystore Utility [page 22]
Supported DDL and DML operations, such as creating a key administrator, creating a client key pair, and importing and exporting column encryption keys.	Client-Side Data Encryption User Tasks [page 31]
Learn how to configure client properties for client-side data encryption.	Configuring Client Connection Properties for Client-Side Data Encryption [page 53]

2 Client-Side Data Encryption Overview

SAP HANA server provides a range of security features to ensure secured access control and protection of sensitive information, such as user roles, Single sign-on (SSO), LDAP and password-based user authentication and authorization, data encryption in the network using SSL and TLS security, and database isolation to prevent cross-tenant attacks. 9.9

SAP HANA server automatically encrypts in-flight data and the encryption for the data at rest can be enabled. For instance, network traffic is encrypted when the nameserver preprocessor of the System database communicates with the index server of a tenant database, or when data exchange occurs between the two hosts in a distributed environment.

These server-side encryption capabilities prevent unauthorized access of data for on-premise and trusted systems.

However, with more applications and databases moving into the cloud (consider database-as-a-service scenario) and third-party database administrators accessing customers' data, it's important to ensure that users' data is never compromised, and other cloud administrators cannot view the sensitive information.

Client-side data encryption provides a separation between those who own the data (and can view it) and those who manage the data (but should have no access). 9.6, 9.7

Client-side data encryption is a column-level data encryption capability managed by the client driver. With client-side data encryption, table columns containing sensitive data (credit card numbers, for instance) are encrypted using an encrypti 9.5 y that is accessible only to the client. As a result, any third-party administrator can only manage the data but can't view it.

9.5 Client-side data encryption makes encryption transparent to the applications. The column data is encrypted and decrypted only on client-driver, allowing the applications to read and write data in cleartext form. Key generation and data encryption and decryption happen on the client driver only; SAP HANA server only stores the encrypted keys and encrypted data. Data and encryption keys are never available as cleartext on the SAP HANA server.

Related Information

Getting Started With Client-Side Data Encryption [page 17]
Grant Client-Side Encryption Privileges [page 31]
Configuring Client Connection Properties for Client-Side Data Encryption [page 53]

3 Components of Client-Side Data Encryption

Client-side data encryption uses both symmetric and asymmetric encryption. This section lists the key components of client-side data encryption.

- Encryption keys – Two encryption keys are used to configure client-side encryption – Column Encryption Key (CEK) and Client Key Pair (CKP, a public-private key pair). The keys are generated by the client driver.
  - Column Encryption Keys – CEKs are symmetric keys and are used by the client driver to encrypt and decrypt the column containing sensitive data. CEKs are schema-level keys and you can create more than one CEK per schema. You can use the same CEK for all the columns in a table or use different CEK for each column. CEKs are encrypted and stored on the SAP HANA server. CEK management requires `CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN` privilege.
  - Client Key Pairs (CKPs) – CKPs are asymmetric keys used to encrypt and distribute CEKs to the clients. The public key of the CKP is stored on the SAP HANA server and in the `hdbkeystore` (a secure key store) on the client's local machine.
    The private key is stored only in the `hdbkeystore` on the client's local machine; SAP HANA server cannot access the private key. You need `CREATE CLIENTSIDE ENCRYPTION KEYPAIR` and `DROP CLIENTSIDE ENCRYPTION KEYPAIR` privileges to create or delete a CKP, respectively.
- SAP HANA Client – The following SAP HANA client utilities are used for the key management and configuration of client-side data encryption:
  - `hdbkeystore`: A utility to manage the local secure key store. Use it to view the list of local key pairs, remove key pairs from the key store, export key pairs to a file, and import key pairs from a file. `hdbkeystore` is part of the SAP HANA client installation.
  - Secure key store (`hdbkeystore.dat`): An encrypted key store file that contains the local client key pairs (CKPs). The `hdbkeystore.dat` is created automatically when you create the first CKP. The `hdbkeystore` itself is encrypted and requires a password to be provided from either a user or an application.
  - `hdbsql.exe`: Use this to access the SAP HANA database interactive terminal and run client-side encryption commands.

You can import and export both, data (tables with the encrypted columns) and the encryption keys (CEKs and CKPs).

> **Note**
>
> Client-side encryption is optional. You can use the `CLIENTSIDE ENCRYPTION ON` option to encrypt a specific column while creating a new table. To encrypt a column in an existing table, use the `ALTER TABLE` command.

Related Information

hdbkeystore Utility [page 22]

Managing Column Encryption Keys [page 34]
Managing Client Key Pairs [page 43]

4 Important Considerations When Using Client-Side Data Encryption

Review the information provided in this section before you proceed with client-side data encryption:

Creating and Altering Tables with Encrypted Columns

- Only columns in row and column tables can be altered to encrypt data.
- A table with encrypted columns must have a primary key. Note that the primary keys on the encrypted columns can be encrypted.
- For deterministic encryption, the following operations are supported on the encrypted columns:
  - Joins – encrypted with the same key
  - Indexes/primary key
  - Referential constraints on column tables – both referenced and referencing tables are encrypted with the same key.
  - GROUP BY or HAVING clause
  - Set operators (UNION/INTERSECT/EXCEPT) – columns are encrypted with the same key.
- CEKs must be located in the same schema as the tables whose columns they are encrypting.
- You cannot alter table type (row to column, or vice versa) of the tables with encrypted columns.
- Only encryption-related column alterations are supported for encrypted columns; you can run any alter column operations on the unencrypted columns as usual, without affecting encrypted columns.
- You can only alter the encryption status or encryption key for one column at a time.
- You cannot rename an encrypted column.
- Foreign keys on row tables, partition keys, and check constraints are not supported on the encrypted columns.
- You cannot include an encrypted column in the `RESET BY` query of the `CREATE` | `ALTER SEQUENCE` statements.
- Encrypted columns are not supported in SQLScript, UDFs, and stored procedures.

Querying Tables with Encrypted Columns

- The application must use prepared statements.
- Queries can perform equality look-ups on the columns using deterministic encryption, but cannot perform other operations, such as greater than or less than, pattern matching using the `LIKE` operator or arithmetical operators.
- Only basic inserts, updates, and fetches are possible on the columns using random encryption.

Supported Column Data Types

You can only encrypt the following column data types:

- BOOLEAN
- DATE, TIME, SECONDDATE, and TIMESTAMP

- TINYINT, SMALLINT, INTEGER, BIGINT, REAL, and DOUBLE
- VARBINARY
- VARCHAR and NVARCHAR
- Fixed-Decimal and Floating-point Decimal

Data Masking

You cannot configure data masking for client-side encrypted columns.

Calculation Views in Client-Side Data Encryption

With some restrictions, you can use encrypted columns in Calculation Views. For example, for a join over two columns, the columns must be encrypted using deterministic encryption with the same key. Encrypted columns are not supported in OLAP and Join views.

SQL Commands Prerequisites

- All SQL statements containing the `CLIENTSIDE ENCRYPTION` clause must be executed on a client that supports client-side data encryption. Else, it results in the following error:

  ```
  feature not supported.
  ```

- All SQL statements containing the `CLIENTSIDE ENCRYPTION` clause must be executed on a client that is already configured with the correct path to PSE and DLL. Else, it results in the following error:

  ```
  10429 sapcrypto library was not initialized. Check that the
  sapcrypto library is in the library path and SECUDIR is set
  appropriately.
  ```

  It's applicable to `CREATE`, `ALTER` and `DROP` commands for CEKs and CKPs, and all `ALTER TABLE` statements with `CLIENTSIDE ENCRYPTION` clause.

  ```
  /* CEK commands */
  CREATE CLIENTSIDE ENCRYPTION COLUMN KEY <cek-name> ALGORITHM
  '<algorithm-name>' ENCRYPTED WITH KEYPAIR '<keypair-name>';
  DROP CLIENTSIDE ENCRYPTION COLUMN KEY <cek-name>;
  ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <cek-name> ADD
  KEYCOPY ENCRYPTED WITH KEYPAIR <keypair-name>;
  /* CKP commands */
  CREATE CLIENTSIDE ENCRYPTION KEYPAIR <ckp_name> ALGORITHM
  'RSA-OAEP-2048';
  DROP CLIENTSIDE ENCRYPTION KEYPAIR <ckp_name>;
  ALTER CLIENTSIDE ENCRYPTION KEYPAIR <ckp_name> ADD NEW
  VERSION
  ```

- All statements that access the key store must provide the key store password in the connection string.
  ODBC: -Z CLIENTSIDE_ENCRYPTION_KEYSTORE_PASSWORD=`<pwd>`
  JDBC: cseKeyStorePassword=`<pwd>`
  For example:

  ```
  hdbsql -U key_admin -Z
  CLIENTSIDE_ENCRYPTION_KEYSTORE_PASSWORD=<pwd> "CREATE
  CLIENTSIDE ENCRYPTION COLUMN KEY <cek> ENCRYPTED WITH
  KEYPAIR <ckp>"
  ```

Key Versioning Support

To use CEK and CKP versioning, upgrade the client-drivers to 2.0 SPS 04 version.

5 Encrypting Application Data with Client-Side Data Encryption

Column data is encrypted with a symmetric column encryption key (CEK) and the CEK is again encrypted using a client key pair (CKP). To access encrypted data, an application must use a client driver that supports client-side encryption, and the client driver must have access to the CEK that encrypts the column.

A key administrator can grant access to the CEK and thus the encrypted data, by first decrypting the CEK with his own private key, which is stored in his local `hdbkeystores`, and then creating a copy of the CEK.

Now the CEK copy is encrypted with the public key of the CKP of the user who needs access to data. This CEK copy for the user is stored in the SAP HANA database. When a user requests a CEK from the SAP HANA server, the server sends the CEK copy encrypted with that user's public key. The client-driver then decrypts the CEK copy using the corresponding private key of the user, and then uses this CEK to grant data access to the user.

When writing or reading encrypted data to or from the SAP HANA server, the application must use a prepared statement. When an application issues a parameterized query, the client-driver transparently collaborates with the SAP HANA server to encrypt or decrypt the column data using the key information stored in the SAP HANA server and `hdbkeystore`.

> i Note
>
> The driver encrypts bound parameter values using the CEK before passing the data to the SAP HANA server.

Related Information

Create a Table with an Encrypted Column [page 48]
Selecting Deterministic Versus Randomized Encryption [page 18]
Authorizing a New User to Access Encrypted Data [page 21]
Create a Key Administrator [page 33]

5.1 Client Key Pairs

Client key pairs (CKPs) are asymmetric keys used to distribute column encryption keys (CEKs) to clients.

CKPs generated by the client-driver are stored in the `hdbkeystore` on the local computer, along with their names and UUIDs. The key administrator must create the CKP in a secure location so that no other entities have access to the private key. The key administrator must never operate in the same cloud environment that is hosting the SAP HANA server.

Clients register their CKPs with the SAP HANA server, where they are considered database level objects, rather than schema level objects. Registration of a CKP with the server stores the public key of the CKP in the server catalog.

CKPs are not associated with a user; the client driver uses any key pair found in its `hdbkeystore`. CKPs are not shared between different database systems, so if a client accesses multiple databases that support client-side encryption, then it needs a unique CKP for each database. Also, a CKP from an existing client can be exported from its `hdbkeystore` and imported into a new client's `hdbkeystore`.

Since CKPs are identified by UUIDs, no changes are required for the clients when a database system is moved, replicated, or renamed, as long as the new system has the same CEKs, CEK copies, and public keys as the original. When connecting to the new system, CKPs are still identified by the same unique identifier.

The privileges required to create or delete a CKP are CREATE CLIENTSIDE ENCRYPTION KEYPAIR and DROP CLIENTSIDE ENCRYPTION KEYPAIR, respectively.

> **i Note**
>
> The public key value of the CKP is encoded in PEM format, using X.509 digital certificate before it’s stored in the SAP HANA server.

Related Information

Create a Client Key Pair [page 43]

Drop a Client Key Pair [page 44]

CREATE CLIENTSIDE ENCRYPTION KEYPAIR Statement (Encryption)
DROP CLIENTSIDE ENCRYPTION KEYPAIR Statement (Encryption)
GRANT Statement (Access Control)
CLIENTSIDE_ENCRYPTION_KEYPAIRS System View

5.2 Column Encryption Keys

Column encryption keys (CEKs) are symmetric keys and they are used by the client driver to encrypt and decrypt the column values, allowing the application to read and write data in cleartext form.

CEKs are encrypted by using the public part of the client key pair (CKP) and are stored persistently on the server in HANA metadata, along with other encryption information.

CEKs are decrypted on the client, using the corresponding private part of the CKP that is stored in the hdbkeystore of the client's local computer. There can be multiple copies of each CEK in the server metadata, each encrypted using a different client key pair.

CEKs are not stored persistently on the client, but can be cached in memory to improve performance. Any attempt to delete all copies of a CEK that is still in use, will fail.

i Note

RSA-OAEP-2048 encryption algorithm is used to encrypt the CEKs.

Related Information

Create a Column Encryption Key [page 34]
CREATE CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
GRANT Statement (Access Control)
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
DROP CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View
CLIENTSIDE_ENCRYPTION_KEYPAIRS System View
TABLE_COLUMNS System View
VIEW_COLUMNS System View
Import and Export Column Encryption Keys [page 42]
Recovering a Client Key Pair [page 47]

5.3 Encrypted Columns

Each encrypted column can have its own CEK, or CEKs can be shared by multiple encrypted columns. Encrypted columns are represented as a VARBINARY data type. Metadata associated with these columns includes an encrypted CEK value, the mode of encryption (random or deterministic), and the encryption status.

i Note

Encryption algorithms AES-256-CBC or ARIA-256-CBC can be used to encrypt the column.

Related Information

GRANT Statement (Access Control)
Client-Side Data Encryption
Column Encryption Keys [page 11]
Create a Column Encryption Key [page 34]
Change Column Encryption Status [page 49]

5.4 Supported DML With Client-Side Data Encryption

When querying a table with encrypted columns, there are limits on the types of queries that are supported.

Supported Expressions

Expressions referring to encrypted columns are supported if the encrypted column is referred to by direct column reference expression or alias. You cannot use an encrypted column as input parameter for either built-in or user-defined SQL functions. You cannot use implicit or explicit type casting over encrypted columns.

```
<supported_expression> :=
<deterministically_encrypted_column>
| <randomly_encrypted_column>
| <expression_without_encrypted_column>
```

Supported Predicates

Predicate expressions are supported only if they compare a deterministically encrypted column with a user-specified host variable or with a null value (=, <>, IS NULL, and IS NOT NULL). These predicate expressions can be accompanied with other supported expressions in conjunction and/or in disjunction. Any other predicates referring to encrypted columns are not supported.

```
<supported_predicates> :=
[NOT] <supported_predicate> [{AND | OR} [NOT] <supported_predicate> […] ]
 <supported_predicate> :=
  <predicate_without_encrypted_column>
  | <expression_without_encrypted_column> [NOT] IN
(<supported_subquery_without_encrypted_column_projection>)
  | [NOT] EXISTS (<supported_subquery_without_encrypted_column_projection>)
  | <d_encrypted_col> { =? | <> ? | IS NULL | IS NOT NULL | [NOT] IN (? [, ?]*) }
  | ? = <d_encrypted_col>
  | ? [NOT] IN ( <d_encrypted_col_0> [, <d_encrypted_col_list>]* )

<d_encrypted_col> := identifier for a deterministically encrypted column
```

`<d_encrypted_col_list>` is supported only if all columns in the list share the same type, encryption algorithm and column encryption key.

Both IN and EXISTS with `<supported_query>` are not supported if `<supported_query>` contains projections on any encrypted columns.

Supported SELECT and Subquery Expressions

In a SELECT statement, encrypted columns can only be used in SELECT, WHERE, and ON clauses with supported expressions or predicates. Set operators, UNION, EXCEPT, INTERSECT are supported if the columns are encrypted deterministically with the same column encryption key.

Encrypted columns are generally supported in SELECT statement subqueries (scalar, multi-row, or derived table expression). However, encrypted columns cannot be used in SELECT clauses of scalar and multi-row/column subqueries as this results in an unsupported comparison between an encrypted column and other expressions contained in the parent query.

Predicates with deterministically encrypted column are supported on `<HAVING>` clause, like `<WHERE>` clause.

```
<supported_query> := <supported_select>
| <supported_select> UNION ALL <supported_select>
| <supported_select> UNION <supported_select>
| <supported_select> INTERSECT <supported_select>
| <supported_select> EXCEPT <supported_select>
<supported_select> :=
SELECT <supported_expression> [, <supported_expression>]*
FROM <table_expression>
[WHERE <supported_predicate>]
GROUP BY {<expression_with_deterministically_encrypted_column> |
<expression_without_encrypted_column>}
[HAVING <supported_predicate>]
[ORDER BY <expression_without_encrypted_column>]
[LIMIT <expression_without_encrypted_column>]
```

Any subquery expressions not mentioned above are not supported.

Supported Table Expressions

The following table expressions are supported with client-side encryption:

```
<supported_table_expression> :=
<table_reference>
| <table_reference> [,<table_reference>]*
| <table_reference> INNER JOIN <table_reference> ON <supported_join_predicate>
| <table_reference> LEFT OUTER JOIN <table_reference> ON
<supported_join_predicate>
| <table_reference> RIGHT OUTER JOIN <table_reference> ON
<supported_join_predicate>
| <table_reference> FULL OUTER JOIN <table_reference> ON
<supported_join_predicate>
| <supported_select>
<supported_join_predicate> :=
<supported_predicate>
| <supported_key_comparison>
```

Note that joins are supported only when the join predicate compares deterministically encrypted key columns with the same column encryption key. Also, any supported SELECT statement can be placed in a FROM clause.

Supported DML Statements

INSERT, DELETE, UPDATE, UPSERT, and MERGE INTO statements are partially supported with client-side encryption due to client-side encryption always using the client driver for encryption and decryption. All other operations not listed here are not supported.

Supported MERGE INTO statement

```
<supported_mergeinto> :=
MERGE INTO <table_reference> USING <table_reference> ON
<equal_comparision> <merge_operation_specification>
<equal_comparision> := <d_encrypted_col> { = | != }
<supported_value>
<merge_operation_specification> :=
<merge_when_clause> [...]
<merge_when_clause> :=
<when_matched_clause>
| <when_not_matched_clause>
<when_matched_clause> :=
 WHEN MATCHED [ AND <equal_comparision> ] THEN
<when_matched_specification>
 <when_matched_specification> :=
 <update_specification>
| <delete_specification>
<update_specification> := UPDATE SET <set_clause_list>
<delete_specification> := DELETE
<when_not_matched_clause> :=
WHEN NOT MATCHED [ AND <equal_comparision> ] THEN
<when_not_matched_specification>
<when_not_matched_specification> :=
 INSERT [ ( <insert_column_list>) ] VALUES <insert_value_list>)
<insert_value_list> :=
(<insert_value_element>> [ {, <insert_value_element> } … ] )
<insert_value_element> :=
<value_expression
>| <contextually_typed_value_specification>
```

Supported UPDATE Statements

```
<supported_update_statements> :=
| <supported_insert>
| <supported_delete>
| <supported_update>
| <supported_upsert>
| <supported_mergeinto>
```

Supported INSERT Statement

INSERT can be classified into two types, according to the data source: INSERT-VALUES and INSERT-SELECT.

```
<supported_insert> :=
 <supported_insert_values>
| <supported_insert_select>
<supported_insert_values> :=
INSERT <table_reference> [(<column_list>)] VALUES
(<supported_value>[,<supported_value>])
<supported_value> := ? | <value_without_encrypted_column>
<supported_insert_select> :=
INSERT <table_reference> [(<column_list>)] SELECT
<supported_expression> [, <supported_expression>] FROM
{<table_expression>}
```

The pair of columns and values to insert should have the same column encryption key.

You must use parameter values for encrypted columns as the client-driver cannot access and encrypt values from literals and expressions.

Encrypt the target column and the corresponding source column with the same column encryption key to maintain the integrity of encrypted values.

Supported DELETE Statement

Support for a DELETE statement depends on whether the predicate is supported:

```
<supported_delete> := DELETE FROM <table_reference> [WHERE
<supported_predicate>]
```

Supported UPDATE Statement

UPDATE has two different forms: UPDATE FROM WHERE and UPDATE WHERE. Similar to the DELETE statement above, they require a supported predicate in order to be used with client-side encryption. To use the SET clause with client-side encryption, you can only specify parameter values.

```
<supported_update> :=
UPDATE <table_reference> SET <supported_set_clause> WHERE
<supported_where_clause>
| UPDATE <table_reference> SET <supported_set_clause> FROM
<table_reference>[, <table_reference>]* WHERE <supported_predicate>
<supported_set_clause> := <supported_set_term> [,
<supported_set_term>]
<supported_set_term> := <column_name> = <supported_value>
<supported_value> := ? | <value_without_encrypted_column>
```

Supported UPSERT/ REPLACE Statement

The UPSERT subquery is supported. UPSERT-VALUES behaves like INSERT-VALUES.

```
<supported_upsert> :=
<supported_upsert_values>
| <supported_ upsert_select>
<supported_upsert_values> :=
UPSERT | REPLACE <table_reference> [(<column_list>)] VALUES
(<supported_value>[,<supported_value>])
<supported_value> := ? | <value_without_encrypted_column>
<supported_upsert_select> :=
UPSERT | REPLACE <table_reference> [(<column_list>)] SELECT
<supported_expression> [, <supported_expression>] FROM
{<table_expression>}
```

6 Getting Started With Client-Side Data Encryption

Setting up client-side encryption in an SAP HANA database involves deciding which sensitive columns need to be encrypted and encrypting the data in those columns.

Prerequisites

Before setting up client-side encryption, perform the following tasks:

- Use threat modeling to identify sensitive data.
- Decide whether to use deterministic or randomized encryption for your data.
- Inform application users about how client-side encryption affects your business solution.

Context

When using client-side encryption, adhere to the following best practices:

- Be selective about what you encrypt as excessive data encryption can affect functionality and performance.
- Always back up and archive your client key pairs (CKPs), column encryption keys (CEKs) and data, and store them in a safe place.
- Assign key administrator privileges to trusted users only.
- Ensure that the key administrators create their client key pairs in a secure location that is different than the cloud environment that hosts the SAP HANA server.
- Change your client key pairs (CKPs) and column encryption keys (CEKs) regularly.

Procedure

1. Create a key administrator who can create client key pairs and column encryption keys.
2. Create a client key pair. Back up the client key pair and store it in a safe place. If the client machine crashes, then the CKP can be restored to a different machine so that encrypted data can be accessed.
3. Create a column encryption key that is encrypted with the client key pair. Export the column encryption key and store in a safe place.
4. Create new tables with encrypted columns or encrypt existing data in specified database columns.
5. Enable users to access encrypted data by creating key copies of column encryption keys that are encrypted with the user's client key pair. Key copies for the new CEK must be created for the users who need access to data.

6. Rotate CEKs regularly and re-encrypt your data using the latest version of CEK.

Results

You have set up client-side encryption in an SAP HANA database.

Related Information

Create a Key Administrator [page 33]
Create an Empty Column Encryption Key [page 38]
Populate an Empty Column Encryption Key [page 39]
Create a Client Key Pair [page 43]
Recovering a Client Key Pair [page 47]
Import and Export Column Encryption Keys [page 42]
Supported DML With Client-Side Data Encryption [page 13]
Client-Side Data Encryption
Create a Table with an Encrypted Column [page 48]

6.1 Selecting Deterministic Versus Randomized Encryption

Although SAP HANA server never operates on cleartext data stored in encrypted columns, depending on the encryption type of the column, some queries on encrypted data are supported.

SAP HANA supports the following types of client-side encryption:

- Deterministic encryption
- Non-deterministic (randomized) encryption

With deterministic encryption, cleartext is always encrypted into the same ciphertext for a given cleartext and CEK; it is applicable even over different runs with the same CEK. This consistency permits operations, such as equality and inequality comparisons on the encrypted data. Deterministic encryption can result in attackers matching cleartext to encrypted values for a column, especially for the columns storing low-cardinality data like status flags, boolean values, or major classifications such as gender.

When a column is encrypted using the non-deterministic algorithm, the same cleartext yields different ciphertexts, even when encrypting the cleartext several times with the same CEK. Due to the randomness of the ciphertext, non-deterministic encryption is stronger, but limits the types of operations that can be performed with the encrypted data. No operations are possible except for basic inserts, updates, and fetches.

Choose the encryption algorithm based on the intended use of the data and the level of security required. Use deterministic encryption for columns that are used as search parameters, for example SSN. Use randomized encryption for data that is used for basic inserts, updates, and fetches.

Related Information

Client Key Pairs [page 10]
Column Encryption Keys [page 11]
Encrypted Columns [page 12]
Manage Access to Column Encryption Keys [page 37]

6.2 Defining Environment Variables for sapgenpse and SAP Common Crypto Library

After you have installed the SAP HANA Client (`hdbclient`), in addition to the `hdbsql` utility and `hdbkeystore`, the installation directory also includes the following files that are used in client-side encryption:

- `sapgenpse.exe`: A cryptography tool to generate Personal Security Environment (PSE) files for SSL and TLS communication with the server.
- `sapcrypto.dll`: SAP Common Crypto Library that provides data encryption capabilities.

`sapgenpse.exe` and `sapcrypto.dll` can also be downloaded separately from the SAP support portal. For the latest version of the library, see SAP Downloads on the SAP ONE Support Portal and search for `COMMONCRYPTOLIB`. Download the library and extract its contents using `SAPCAR`. The `SAPCRYPTO` folder includes the following files:

- Windows
  - `sapcrypto.dll`
  - `sapgenpse.exe`
- Linux
  - `libsapcrypto.so`
  - `sapgenpse`

Defining Environment Variables

To use the SAP CommonCryptoLib (CCL) for encryption, certain environment variables must be defined. For example, `<SECUDIR>`, `<PATH>`, and `<LD_LIBRARY_PATH>`. Instead of configuring these environment variables individually, you can run the scripts provided with the SAP HANA Client installer to set these variables automatically. Use `hdbclienv.sh` for Linux, UNIX, and macOS and `hdbclienv.bat` for Windows.

Operating system	Method	Command
For Linux, UNIX, and macOS	bash	Use the `hdbclienv.sh` script. Run the following command: `. /usr/sap/hdbclient/hdbclienv.sh` For bash, the `hdbclienv.sh` script can be sourced in a start-up script, such as `~/.bashrc`. **i Note** To set the environment variables for your Operating System user on every login, source the environment script in your login script (`~/.bashrc`).
Windows	Command-prompt	Use the `hdbclienv.bat` batch file. Run the following command at the command prompt: `"C:\Program Files\SAP\hdbclient\hdbclienv"` **i Note** To set the environment variables permanently for all users, use the Windows Environment Variables system setting. If you are using client communication encryption, then the `<SECUDIR>` environment variable must contain the location of `sapcli.pse`.

For Linux, Unix, and Mac OS X, in addition to the root environment script, a helper script is also provided to launch the `hdbsql` utility, with environment script already sourced and SAP CommonCryptoLib configured. The location of the script is `/scripts/hdbsql`.

Related Information

Configuring Client Connection Properties for Client-Side Data Encryption [page 53]
Client-Side Data Encryption Overview [page 5]
Important Considerations When Using Client-Side Data Encryption [page 8]

6.3 Authorizing a New User to Access Encrypted Data

A new user first needs access to the client key pair (CKP) to decrypt the column encryption key (CEK); once the CEK is available, the user can access the encrypted table column. This section lists the steps that a key administrator would typically follow to authorize a new user to access the encrypted column.

- The user creates the CKP.
- The key administrator adds a CEK copy. In this process:
    - The client driver decrypts the key administrator's copy of the CEK, using his private key.
    - The client driver now encrypts the raw value of the CEK with the new user's public key.
    - The encrypted CEK is then stored in the SAP HANA server.
- Since only the new user has the private key of the CKP, only he can decrypt the CEK, and access the data.

> **i Note**
>
> Multiple copies of the same master CEK may exist because you may have multiple installations (clients) in your environment, with each installation having its own CKP. Each CEK copy is encrypted with the public key of the respective client key pairs (CKPs).

Other key considerations:

- When a user is no longer part of the organization – The key administrator removes the user’s CEK copy.
- When a user loses the CKP and needs to access the encrypted data – The user creates a new CKP and then requests the key administrator to add the key copy for the CEK that he needs to access.

Related Information

Important Considerations When Using Client-Side Data Encryption [page 8]
Supported DML With Client-Side Data Encryption [page 13]
Grant Client-Side Encryption Privileges [page 31]

7 hdbkeystore Utility

Use the `hdbkeystore.exe` utility to manage the client key pairs (CKPs). The CKPs are stored in the secure key store (`hdbkeystore.dat`). The secure key store is automatically created when you create the first CKP.

The secure key store uses the UUID of the key pair and is encoded as a series of name-value attribute pairs. Several commands are available for managing information stored in the secure key store.

```
hdbkeystore [OPTION]...COMMAND [PARAMETER]
```

Option	Description
-h	Displays a help message.
-p	Specifies the password used to decrypt an encrypted key store.
-v	Executes the command in verbose mode.
--version	Prints out the version information for the `hdbkeystore`.

Command	Parameter	Description
`HELP`		Prints the help message.

Command	Parameter	Description
`EXPORT`	[--uuid]<NAME>[,<NAME2>, ...]<FILENAME> [<PASSWORD>]	Exports the specified key or keys to a file. **<NAME>** The name of the key to be exported; '*' and '?' wildcards can be used to specify multiple keys. A list of key names separated by commas may be supplied and each name in the list may contain wildcards. **<FILENAME>** The file to export the key to. **<PASSWORD>** Optional password to encrypt the private key in the exported file. **<--uuid>** The keys are specified by UUID rather than by name. Wildcards are not supported for UUIDs.
`IMPORT`	<FILENAME> [<PASSWORD>]	Imports one or more keys from a file. **<FILENAME>** The file that contains the key to be imported. **<PASSWORD>** The password that the private key in the import file was encrypted with.
`LIST`		Lists all the keys in the store.

Command	Parameter	Description
`REMOVE`	[--uuid]`<NAME>`[,`<NAME2>`,...]	Removes the key `<NAME>` from the key store. **`<NAME>`** The name of the key to be removed; '*' and '?' wildcards may be used to remove all keys matching the pattern. A list of key names separated by commas may be supplied and each name in the list may contain wildcards. **`<--uuid>`** The keys are specified by UUID rather than by name. Wildcards are not supported for UUIDs.

Related Information

Import and Export Column Encryption Keys [page 42]
Create a Table with an Encrypted Column [page 48]

7.1 hdbkeystore Examples

The following examples illustrate how to use the various commands of the `hdbkeystore` utility.

Example 1: List all the keys in the key store encrypted with the password MyPassword123:

```
hdbkeystore -p MyPassword123 List
ec78bf9e-5072-0016-5398-c9e440000002  CKP1  S01
f632761a-5072-0016-5399-24e3b0000001  CKP2  S01
0a16abcf-5072-0016-5399-149b40000001  CKP3  S01
```

Example 2: Export the client keypair CKP1 to a file named ckp1:

```
hdbkeystore -p MyPassword123 Export CKP1 ckp1 MyPassword123
Exported key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002' to file
'ckp1'.
```

Example 3: Export all the keypairs in the key store to a file named ckps:

```
hdbkeystore -p MyPassword123 Export "*" ckps MyPassword123
```

```
Exported key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002' to file
'ckps'.
Exported key 'CKP2' with ID 'f632761a-5072-0016-5399-24e3b0000001' to file
'ckps'.
Exported key 'CKP3' with ID '0a16abcf-5072-0016-5399-149b40000001' to file
'ckps'.
```

Example 4: Import the client keypair CKP4 from the file ckp4:

```
hdbkeystore -p MyPassword123 Import ckp4 MyPassword123
Successfully imported key 'CKP4' with ID 'c2136d9f-5072-0016-5399-2b61e0000001'
from file 'ckp4'.
```

Example 5: Remove the client key pair CKP2 from the key store:

```
hdbkeystore -p MyPassword123 Remove CKP2
Removed key 'CKP2' with ID 'f632761a-5072-0016-5399-24e3b0000001'.
```

Example 6: Remove all the client key pairs from the key store:

```
hdbkeystore -p MyPassword123 Remove "*"
Removed key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002'.
Removed key 'CKP3' with ID '0a16abcf-5072-0016-5399-149b40000001'.
Removed key 'CKP4' with ID 'c2136d9f-5072-0016-5399-2b61e0000001'.
```

Example 7: Import the original three key pairs back into the key store from ckps:

```
hdbkeystore -p MyPassword123 Import ckps MyPassword123
Successfully imported key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002'
from file 'ckps'.
Successfully imported key 'CKP2' with ID 'f632761a-5072-0016-5399-24e3b0000001'
from file 'ckps'.
Successfully imported key 'CKP3' with ID '0a16abcf-5072-0016-5399-149b40000001'
from file 'ckps'.
```

Example 8: Change the key store password from MyPassword123 to MyNewPassword456.

To change or reset the key store password, you need to export the contents of the key store to an encrypted file, delete the key store on the file system, and then import the exported file again specifying the new key store password.

```
hdbkeystore -p MyPassword123 Export "*" ckps MyPassword123
Exported key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002' to file
'ckps'.
Exported key 'CKP2' with ID 'f632761a-5072-0016-5399-24e3b0000001' to file
'ckps'.
Exported key 'CKP3' with ID '0a16abcf-5072-0016-5399-149b40000001' to file
'ckps'.
```

```
rm ~/.hdb/ykfl00620459a/hdbkeystore.dat
```

```
hdbkeystore -p MyNewPassword456 Import ckps MyPassword123
Successfully imported key 'CKP1' with ID 'ec78bf9e-5072-0016-5398-c9e440000002'
from file 'ckps'.
Successfully imported key 'CKP2' with ID 'f632761a-5072-0016-5399-24e3b0000001'
from file 'ckps'.
Successfully imported key 'CKP3' with ID '0a16abcf-5072-0016-5399-149b40000001'
from file 'ckps'.
```

8 Putting the Pieces Together

This section lists the series of tasks required to configure client-side data encryption involving three users – key administrator, data administrator, and user.

- Key administrator – Creates the encryption keys – CKP and CEK. The key administrator then grants access to other users to access the CEK.
- Data administrator – Responsible for creating schema, encrypting table column, and granting the privileges. The data administrator uses the CEK provided by the key administrator to encrypt the data.
- User (HR manager) – Needs access to the cleartext data and cleartext key. The user creates his own CKP.

Configuring Client-side Data Encryption

User	Description	Examples and Applicable Privileges

SYSTEM	SYSTEM user creates data admin, key admin, and the HR manager. SYSTEM user grants the following privileges to the users:
	
	• Data admin: Privilege to create schema and import and export the encryption keys.
	• Key admin: Privilege (`CREATE CLIENTSIDE ENCRYPTION KEYPAIR`) to create client-side encryption key pair.
		`/* System user creates Data Admin, Key Admin, and the User (HR manager) */`
		
		`connect system password manager;`
		
		`create user hrapp_data_admin password ClientsideEncryption123 no force_first_password_change;`
		`create user hrapp_key_admin password ClientsideEncryption123 no force_first_password_change;`
		`create user hrapp_hr_manager password ClientsideEncryption123 no force_first_password_change;`
		
		`/* grants privileges to the users */`
		
		`grant create schema to hrapp_data_admin;`
		`grant export, import to hrapp_data_admin;`
		
		`grant CREATE CLIENTSIDE ENCRYPTION KEYPAIR to hrapp_key_admin with admin option;`

User	Description	Examples and Applicable Privileges

Data admin	Data admin creates schema (`hrapp`) and grants appropriate privileges on the schema. For example, data admin grants the CEK admin privilege (`CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN`) to the key admin.	`/* Data Admin creates schema and grants privileges */`
		
		`connect hrapp_data_admin password ClientsideEncryption123;`
		`create schema hrapp;`
		
		`/* Data Admin grants CEK Admin privilege to Key Admin */`
		
		`grant` **`clientside encryption column key admin`** `on schema hrapp to hrapp_key_admin;`

Key admin	Key admin who is responsible for key management, creates the encryption keys.
	
	Key admin is granted the `CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN` privilege by the data admin to create CEK (for example, `hrapp_cek1`).
		`/* Key Admin creates the CKP and CEK */`
		
		`connect hrapp_key_admin password ClientsideEncryption123;`
		`create clientside encryption keypair` **`key_admin_ckp;`**
		
		`set schema hrapp;`
		`create clientside encryption column key` **`hrapp_cek1`** `encrypted with keypair key_admin_ckp;`

Key admin	Key admin grants permission to the HR manager to create his own CKP.	`/* Key Admin grants permission to the HR Manager to create CKP */`
		
		`grant create clientside encryption keypair to hrapp_hr_manager;`

User	Description	Examples and Applicable Privileges
Data admin	Data admin creates the table and encrypts the table column (SSN, for instance) containing the sensitive information, using the CEK (`hrapp_cek1`) created by the key admin. The SSN column is encrypted now.	`/* Data Admin creates table, encrypts column, and grants privileges */` `connect hrapp_data_admin password ClientsideEncryption123; set schema hrapp; create table Employees ( Emp_ID int primary key, Name nvarchar(32), SSN nvarchar(9) clientside encryption on with` **`hrapp_cek1`** `deterministic, Salary nvarchar(15) );` `/* Grants permissions to the HR manager */` `grant insert, select, update on Employees to hrapp_hr_manager;`
HR manager	Since the HR manager needs to access the data, he creates his own CKP (`hr_manager_ckp`). **i Note** Although the HR manager has created his CKP, he can't access the data because the data is encrypted with the key admin's CEK, which in turn, is encrypted by the key admin's CKP. To be able to access the data, the HR manager needs a copy of the CEK that is encrypted with his own CKP.	`/* HR Manager creates CKP */` `connect hrapp_hr_manager password ClientsideEncryption123; create clientside encryption keypair` **`hr_manager_ckp;`**

User	Description	Examples and Applicable Privileges
Key admin	Now the key admin grants access for the CEK to the HR manager, by adding a keycopy encrypted with the HR manager's CKP, using the `ALTER CLIENTSIDE ENCRYPTION COLUMN KEY` command.	`/* Key Admin grants access to CEK */ connect hrapp_key_admin password ClientsideEncryption123; set schema hrapp; alter clientside encryption column key hrapp_cek1 add keycopy encrypted with keypair hr_manager_ckp;`

Data admin	The data admin grants the DML permissions to the HR manager, such as `UPDATE` and `INSERT`.	`/* Data admin grants privileges to the HR manager*/`
		
		`grant insert, select, update on Employees to hrapp_hr_manager;`

HR manager	Now the HR manager has the CEK copy (encrypted with his own CKP) and the required DML permissions, so he can access the encrypted data and execute DML operations using the prepared statements.	`/* HR Manager executes DML using prepared statements */`
		
		`connect hrapp_hr_manager password ClientsideEncryption123; set schema hrapp; insert into Employees values(?,?,?,?); select * from Employees update Employees set salary = ? where SSN = ?; select * from Employees where SSN = ?;`

i Note

Though the data admin can export the CEKs and the table, and import them into a different machine, he cannot access data without the required key copy (CEK).

For information on client key pair commands and key store, refer to hdbkeystore Utility [page 22].

9 Client-Side Data Encryption User Tasks

Describes the user tasks required to set up client-side encryption.

9.1 Grant Client-Side Encryption Privileges

Grant a user the required privileges to manage client key pairs and column encryption keys, create tables that use specific column encryption keys, or export or import column encryption keys.

Prerequisites

You are logged into your database as a user who can grant the following privileges WITH ADMIN OPTION or WITH GRANT OPTION:

- CREATE CLIENTSIDE ENCRYPTION KEYPAIR system privilege
- DROP CLIENTSIDE ENCRYPTION KEYPAIR system privilege
- ALTER CLIENTSIDE ENCRYPTION KEYPAIR privilege
- CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege

For more information about granting client-side encryption privileges, see the GRANT statement in the *SAP HANA SQL and Systems Views Reference*.

Procedure

Grant a user the required privileges by executing the following statements:

Option	Description	Action
Grant the CREATE CLIENTSIDE ENCRYPTION KEYPAIR system privilege	Grants the user the ability to create client key pairs.	Execute the following GRANT statement, with the option WITH ADMIN OPTION clause if you want `<user-name>` to be able to grant the privilege to other users: `GRANT CREATE CLIENTSIDE ENCRYPTION KEYPAIR TO <user-name> [WITH ADMIN OPTION];`

Option	Description	Action
Grant the DROP CLIENTSIDE ENCRYPTION KEYPAIR system privilege	Grants the user the ability to drop client key pairs. Users who have the DROP CLIENTSIDE ENCRYPTION KEYPAIR privilege can drop key-pairs belonging to other users.	Execute the following GRANT statement, with the option WITH ADMIN OPTION clause if you want `<user-name>` to be able to grant the privilege to other users: `GRANT DROP CLIENTSIDE ENCRYPTION KEYPAIR TO <user-name> [WITH ADMIN OPTION];`
Grant the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN schema privilege	Grants the user the ability to create, alter, and drop column encryption keys.	Execute the following GRANT statement, with the option WITH GRANT OPTION clause if you want `<user-name>` to be able to grant the privilege to other users: `GRANT CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN ON SCHEMA <schema_name> TO <user-name> [WITH GRANT OPION];`
Grant the ALTER CLIENTSIDE ENCRYPTION KEYPAIR system privilege	Grants the user the ability to update client key pairs.	Execute the following GRANT statement, with the option WITH ADMIN OPTION clause if you want `<user-name>` to be able to grant the privilege to other users: `GRANT ALTER CLIENTSIDE ENCRYPTION KEYPAIR TO <user-name> [WITH ADMIN OPTION];`

Results

The user has been granted the specified privileges.

Related Information

GRANT Statement (Access Control)

9.2 Create a Key Administrator

Create a client-side encryption key administrator who has the privileges to manage column encryption keys.

Prerequisites

You are connected to your schema as the system user.

Context

A key administrator is a user with the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege. When a key administrator creates a new column encryption key, encrypted with the public key, no other system user can access it, even other database administrators, unless the key administrator explicitly creates a copy of the column encryption key for them. Users can get access to encrypted data through copies of the column encryption key encrypted by their client key pair. If the key administrator creates a copy for a user who also has the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege, or can assign the privilege to themselves, they can also create key copies for other users. By default, the schema owner has the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege on their own schema, but cannot create a client key pair if they do not have the CREATE CLIENTSIDE ENCRYPTION KEYPAIR privilege.

Procedure

1. Open a SQL console for your schema and execute the following statements to create two new users: a data administrator and a key administrator.

```
CREATE USER <database-admin-user-name>;
CREATE USER <key-admin-user-name>;
```

2. Execute the following statement to grant `<database-admin-user-name>` the required database administration privileges:

```
GRANT CREATE SCHEMA TO <database-admin-user-name>;
```

3. Execute the following statement to grant `<key-admin-user-name>` the privileges required to create client key pairs.

```
GRANT CREATE CLIENTSIDE ENCRYPTION KEYPAIR TO <key-admin-user-name>;
```

4. The database administrator creates a schema and grants `<key-admin-user-name>` the necessary privileges to manage column encryption keys:

```
CREATE SCHEMA <new-schema>;
GRANT CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN ON SCHEMA <new-schema> TO <key-
admin-user-name>;
```

5. (Optional) To give the key administrator the ability to grant the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege to other users, use the WITH GRANT OPTION:

```
GRANT CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN ON SCHEMA <my-schema> TO <key-
admin-user-name> WITH GRANT OPTION;
```

Results

You have created an administrative user with the privileges required to create, alter, and drop column encryption keys. The key administrator uses the ALTER statement to create a key copy for a client key pair to grant access to encrypted data.

Related Information

GRANT Statement (Access Control)
CREATE USER Statement (Access Control)

9.3 Managing Column Encryption Keys

Learn how to create and manage column encryption keys for client-side encryption.

9.3.1 Create a Column Encryption Key

Create a column encryption key (CEK) to encrypt columns containing sensitive data. The CEK is encrypted with the latest version of the CKP.

Prerequisites

You are connected to a schema as the key administrator.

Procedure

Open the SQL console for your schema and execute the following statement to create a new column encryption key, encrypted with the latest version of the CKP:

```
CREATE CLIENTSIDE ENCRYPTION COLUMN KEY <key-name> ALGORITHM '<algorithm-name>'
 ENCRYPTED WITH KEYPAIR '<keypair-name>';
```

The specified key pair should belong to the key administrator creating the column encryption key.

Results

The new CEK is created and stored in the SAP HANA system catalog.

Related Information

CREATE CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View
GRANT Statement (Access Control)

9.3.2 Drop a Column Encryption Key

Delete a column encryption key.

Prerequisites

You are connected to a schema as the key administrator.

If any column is encrypted using any version of the specified key, then no key copies are dropped.

Procedure

Open a SQL console and execute the following statement to drop a new column encryption key:

```
DROP CLIENTSIDE ENCRYPTION COLUMN KEY <key-name>;
```

Results

The specified column encryption key and all of its copies are dropped.

Related Information

DROP CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View
GRANT Statement (Access Control)

9.3.3 Drop Column Encryption Key Copies

Drops the key copies for all versions of a Column Encryption Key (CEK) that is encrypted by different versions of key pair. If the key copy being dropped belongs to the key administrator, then the SAP HANA server ensures that at least one other key administrator's key copy exists for that version of CEK. If not, then the drop of key copy fails.

Context

You are connected to a schema as the key administrator.

Procedure

Open a SQL console and execute the following statement to drop the key copies for all versions of CEK encrypted by different versions of key pair:

```
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <cek_name> DROP KEY COPY ENCRYPTED WITH
KEYPAIR <keypair_name>
```

Results

The SAP HANA server removes the key copies for all versions of CEK, encrypted by different versions of key pair.

9.3.4 Manage Access to Column Encryption Keys

Grant or revoke access to a column encryption key.

Prerequisites

You are connected to a schema as the key administrator.

If you are granting access, then the user to whom access is being granted has created a key pair.

Context

When a key administrator creates a new column encryption key and encrypts it with their public key, no other user can access it, unless the key administrator explicitly creates a copy of the column encryption key for them.

Procedure

Choose one of the following options:

Option	Action
Grant access to a column encryption key	Execute the following statement to grant the owner of the specified client key pair access to the specified column encryption key: `ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <key-name> ADD KEYCOPY ENCRYPTED WITH KEYPAIR <keypair-name>;`
Revoke access to a column encryption key	Execute the following statement to remove access for the specified client key pair to the specified column encryption key : `ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <key-name> DROP KEYCOPY ENCRYPTED WITH KEYPAIR <keypair-name>;`

Results

Access to the specified column encryption key has been either granted to or revoked from the user with the specified key pair. When access is granted, server creates key copies for all versions of the CEK, encrypted with the latest version of `<keypair_name>`. When access is revoked, the server removes the CEK copies for all versions, encrypted by `<keypair_name>`.

Related Information

ALTER CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View

9.3.5 Create an Empty Column Encryption Key

Create an empty column encryption key that can be used for schema creation without requiring the creation of actual encryption keys.

Prerequisites

You are connected to a schema as a user with either the CREATE ANY privilege or the CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege.

> i Note
>
> You cannot create new versions for `HEADER ONLY` CEKs which are empty CEKs and are not yet populated.

Context

A CREATE/ALTER TABLE (on a table with no rows) statement can reference an empty column encryption key, but attempts to insert rows into the table fail.

Procedure

Open a SQL console and execute the following statement to create an empty column encryption key:

```
CREATE CLIENTSIDE ENCRYPTION COLUMN KEY <key-name> ALGORITHM '<algorithm-name>'
 HEADER ONLY;
```

Results

An empty CEK is created. Only the properties of the CEK are created along with the UUID. The CEK is not encrypted with a CKP. Once the empty CEK is altered to encrypt with a CKP, the SAP HANA server catalog is updated with the encrypted value of the CEK while the CEK UUID remains the same.

Related Information

CREATE CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View
GRANT Statement (Access Control)
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)

9.3.6 Populate an Empty Column Encryption Key

When an empty column encryption key (CEK) is populated with a client key pair (CKP), the latest version of the CKP is automatically used to encrypt the empty CEK.

Prerequisites

You are connected to a schema as the key administrator.

Procedure

Execute the following statement to populate an empty CEK that is encrypted with the latest version of the CKP:

```
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <key-name>
 ENCRYPTED WITH KEYPAIR <keypair-name>;
```

The CKP should belong to the key administrator populating the CEK so that key copies can be made. If the specified CKP does not belong to the key administrator or does not exist, then the operation fails.

Results

The SAP HANA system catalog is updated with the CEK information.

Related Information

ALTER CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
CLIENTSIDE_ENCRYPTION_COLUMN_KEYS System View

9.3.7 Rotate Column Encryption Keys

An important step to ensure the security of the cryptosystem is to periodically change client-side encryption keys. This practice decreases the risk of keys being breached. Two users are required to rotate the column encryption key (CEK). A key administrator to create a new CEK and key copies of the new CEK and an application administrator to alter the table to re-encrypt the column.

For CEK rotation, use one of the following options:

- Create a new version of an existing CEK and then re-encrypt table data using the new version. A new version of the CEK has the same name, but a different UUID and key value.
  Using a New Version of Same Column Encryption Key to Re-encrypt Data [page 40]
- Use a different CEK to encrypt column data.
  Using a Different Column Encryption Key to Re-encrypt Data [page 41]

9.3.7.1 Using a New Version of Same Column Encryption Key to Re-encrypt Data

Learn how to create a new version of the same CEK and then re-encrypt table column with this new version. After the new version of the CEK is created, there could exist more than one version of the same CEK. Note that the new version of the CEK is used for any new encryption. Any previous version of the CEK will remain in use if it was used to encrypt columns until the application administrator re-encrypts those columns with the newer version of the CEK.

Create a New Version of the CEK

On the client side, the key administrator creates a new version of the CEK which results in creation of key copies for the new version of the CEK.

```
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <[schema_name.]cek_name> ADD NEW VERSION
```

Note that the algorithm cannot be changed for the CEK while adding its new version.

Re-encrypt Column with the New Version of the CEK

The application administrator uses the TABLE_COLUMNS system view to identify which columns are encrypted with the old CEK.

After the new version of the CEK has been created successfully, the application administrator who has the `ALTER`, `SELECT`, and `UPDATE` permissions on the table and key copies of the old and new versions of the CEK, can run the following `ALTER TABLE` command to re-encrypt column with the new version of the CEK.

If the application administrator does not have either the specified permissions on the `<schema_name>` or both key copies, the re-encryption fails.

```
ALTER TABLE <table_name> ALTER (<col_name> ALTER CLIENTSIDE ENCRYPTION ACTIVATE
NEW VERSION)
```

The schema owners can alter tables in their own schemas as they also have `SELECT` and `UPDATE` permissions on the tables in their respective schemas.

Drop Old Versions of the CEK

This step is optional. After all the columns using the old version of the CEK are re-encrypted with the new version, the old CEK versions can be dropped from the catalog by the key administrator. This results in dropping of key copies for the old versions of the CEK.

```
ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <schema_name.cek_name> DROP OLD VERSIONS
```

9.3.7.2 Using a Different Column Encryption Key to Re-encrypt Data

Create a new Column Encryption Key (CEK) and encrypt table column using this key.

- Key administrator, a user with the `CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN` privilege:
    - Creates a new CEK.

      ```
      CREATE CLIENTSIDE ENCRYPTION COLUMN KEY <new-column-encryption-key-name>
      ALGORITHM '<algorithm-name>'
      ENCRYPTED WITH KEYPAIR '<keypair-name>';
      ```

    - Uses the CLIENTSIDE_ENCRYPTION_COLUMN_KEYS system view to identify the key copies for the old CEK.
    - Creates copies for the new column encryption key for the specified CKP (`keypair-name1`).

      ```
      ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <new-column-encryption-key-name>
      ADD KEYCOPY ENCRYPTED WITH KEYPAIR <keypair-name1>;
      ```

- Application administrator on the client side:

    - Alters tables to re-encrypt them with the new CEK.

      ```
      ALTER TABLE <table-name>
      ALTER (<column-name> ALTER CLIENTSIDE ENCRYPTION WITH <new-column-
      encryption-key-name> RANDOM);
      ```
- Key administrator:
    - Uses the TABLE_COLUMNS system view to check if any old CEK is being used.
    - If not used, drops the old CEK. The CEK copies are dropped automatically.

      ```
      DROP CLIENTSIDE ENCRYPTION COLUMN KEY <old-column-encryption-key-name>;
      ```

9.3.8 Import and Export Column Encryption Keys

Prerequisites

For exporting a column encryption key (CEK), you must have the following privileges:

- EXPORT system privilege
- USAGE privilege on all CEKs being exported

For importing a CEK, you must have the following privileges:

- IMPORT system privilege
- CREATE CLIENTSIDE ENCRYPTION KEYPAIR system privilege
- CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN schema privilege

Importing encrypted data into an existing table requires the UUIDs of the CEKs of the exported data to match the UUIDs of the CEKs of existing table columns.

Context

Exporting a CEK is equivalent to exporting all CEK copies of all versions of the CEK. The export of a CEK copy includes an export of the public key of all the relevant client key pairs.

Procedure

Choose one of the following options:

Option	Action
Export a CEK	Execute the following statement to export a CEK: `EXPORT CLIENTSIDE ENCRYPTION COLUMN KEY <column_encryption_key_name> AS CSV INTO '<filepath>';`
Import a CEK	Execute the following statement to import a CEK: `IMPORT CLIENTSIDE ENCRYPTION COLUMN KEY <column_encryption_key_name> FROM '<filepath>';`

Related Information

IMPORT Statement (Data Import Export)
EXPORT Statement (Data Import Export)
GRANT Statement (Access Control)

9.4 Managing Client Key Pairs

Learn how to create and manage client key pairs (CKPs) for client-side encryption.

9.4.1 Create a Client Key Pair

Create a client key pair (CKPs).

Prerequisites

- You are connected to your schema as the key administrator or as a user with the CREATE CLIENTSIDE ENCRYPTION KEYPAIR privilege.
- You have set the local key store password for your connection using the `cseKeyStorePassword` connection property.

Context

Client Key Pairs (CKPs) are generated by the client-driver. CKPs are asymmetric keys used to distribute column encryption keys to clients.

Procedure

Open a SQL console for your schema and execute the following statement to create a new client key pair:

```
CREATE CLIENTSIDE ENCRYPTION KEYPAIR <key-name> ALGORITHM 'RSA-OAEP-2048';
```

Results

The new client key pair is created with the specified key name and the algorithm RSA-OAEP-2048. The client key pair is stored, along with its key name and UUID, in the hdbkeystore. The public part of the client key pair is stored in the SAP HANA system catalog. CKPs are named database level objects. CKPs are not shared between database systems. If a client accesses multiple databases that support client-side encryption, then it needs a unique CKP for each one.

Since the key administrator's private key is stored in the hdbkeystore on a local computer, all subsequent operations must be performed on that computer unless the key administrator exports the key pair from the hdbkeystore and imports them to a new computer or generates a new key pair for the new computer and creates new key copies of CEKs encrypted with the new key pair.

Related Information

CREATE CLIENTSIDE ENCRYPTION KEYPAIR Statement (Encryption)
CLIENTSIDE_ENCRYPTION_KEYPAIRS System View
Configuring Client Connection Properties for Client-Side Data Encryption [page 53]

9.4.2 Drop a Client Key Pair

Delete a client key pair (CKP).

Prerequisites

You must have the DROP CLIENTSIDE ENCRYPTION KEYPAIR privilege to drop other users' key pairs.

Context

Dropping a CKP drops all the CEK copies encrypted with the CKP. But if the CEK copy being dropped belongs to the key administrator, then the SAP HANA server ensures that at least one other key administrator CEK copy exists for that CEK. If there is no other key administrator key copy for that CEK, then the drop of CKP fails.

Procedure

Open a SQL console for your schema and execute the following statement to drop a client key pair:

```
DROP CLIENTSIDE ENCRYPTION KEYPAIR <keypair-name>;
```

Results

The client key pair and every column encryption key assigned to it are dropped.

Related Information

DROP CLIENTSIDE ENCRYPTION KEYPAIR Statement (Encryption)
CLIENTSIDE_ENCRYPTION_KEYPAIRS System View

9.4.3 Drop Old Versions of a Client Key Pair

Delete old versions of a client key pair (CKP).

Context

After all the CEK copies have been encrypted with the new version of the CKP, the old versions of the CKPs can be dropped; it also deletes the CEK copies encrypted with the old versions of CKP.

Procedure

Open SQL console and execute the following statement to delete the old versions of the CKP:

```
ALTER CLIENTSIDE ENCRYPTION KEYPAIR <ckp_name> DROP OLD VERSIONS
```

9.4.4 Rotate Client Key Pairs

Periodic rotation of client key pairs reduces the probability of encryption keys being compromised and helps you safeguard sensitive data against potential attacks.

This section describes how to configure client key pair (CKP) rotation.

- Create a new version of an existing CKP and use it to re-encrypt CEK. Note that new CEK copies encrypted with the new version of the CKP are automatically created.
  Using a New Version of Same Client Key Pair to Re-encrypt Column Encryption Key [page 46]
- Use a different CKP to re-encrypt CEK.
  Using a Different Client Key Pair to Re-encrypt Column Encryption Key [page 46]

9.4.4.1 Using a New Version of Same Client Key Pair to Re-encrypt Column Encryption Key

Learn how to create a new version of a client key pair and re-encrypt column encryption keys using this new version.

Create a New Version of the Client Key Pair

The key administrator creates a new version of the CKP. This will result in automatic creation of CEK copies for the new version of CKP. Algorithm cannot be changed while adding new versions of CKP.

CKP owners who have access to the `hdbkeystore` and have `ALTER CLIENTSIDE ENCRYPTION KEYPAIR` privilege, can add new version for the CKPs.

```
ALTER CLIENTSIDE ENCRYPTION KEYPAIR <ckp_name> ADD NEW VERSION
```

If multiple versions of a CEK are encrypted by the old version of the CKP, then key copies will be created for all the versions.

Key administrator drops old versions of CKP; it drops the CEK copies automatically. This step is optional.

```
ALTER CLIENTSIDE ENCRYPTION KEYPAIR DROP OLD VERSIONS
```

9.4.4.2 Using a Different Client Key Pair to Re-encrypt Column Encryption Key

The steps to re-encrypt a CEK with a different client key pair (CKP) are as follows:

- Key administrator or any other user who has CREATE CLIENTSIDE ENCRYPTION KEYPAIR privilege:

    - Creates a new CKP.

      ```
      CREATE CLIENTSIDE ENCRYPTION KEYPAIR <keypair-name> ALGORITHM 'RSA-
      OAEP-2048';
      ```
- Key administrator:
    - Uses the CLIENTSIDE_ENCRYPTION_COLUMN_KEYS system view to identify the CEK copies encrypted with the old CKP.
    - Adds CEK copies encrypted by the new CKP.

      ```
      ALTER CLIENTSIDE ENCRYPTION COLUMN KEY <schema_name.cek_name> ADD KEY COPY
      ENCRYPTED WITH KEYPAIR <keypair_name>
      ```
    - Drops old CKP; it automatically drops key copies. This step is optional.

      ```
      DROP CLIENTSIDE ENCRYPTION KEYPAIR <keypair-name>;
      ```

9.4.5 Recovering a Client Key Pair

Implementing the following best practices ensures that you can recover your client key pair if the machine on which it is stored crashes, or if the key administrator is unavailable.

Back Up Your Client Key Pairs to a Secondary Secure Key Store

In case the machine storing your client key pairs crashes, or must be offline for a period of time, ensure that your client key pairs are always accessible by implementing the following best practice:

1. Export your key pair(s) from the secure key store (hdbkeystore).
2. Save the exported key pair(s) in a safe location.
3. Import the key pair(s) into the new secure key store (hdbkeystore) once your machine has recovered.

Ensure Key Administrator Redundancy

Ensure that you have a secondary key administrator in case your primary key administrator is unavailable.

1. The secondary key administrator creates a key pair, exports the key pair from the secure key store (hdbkeystore) and saves the key pair in a safe place.
2. The primary key administrator assigns a "recovery" key copy of each of the column encryption keys they manage to the secondary key administrator.
3. When necessary, ensure that the secondary key administrator is granted CLIENTSIDE ENCRYPTION COLUMN KEY ADMIN privilege either by the primary key administrator, or, if the key administrator is unavailable, by another privileged user.

Related Information

hdbkeystore Utility [page 22]
GRANT Statement (Access Control)

9.5 Create a Table with an Encrypted Column

Create a table that encrypts a column using client-side encryption.

Prerequisites

You are a user who has been granted the USAGE privilege on the specified column encryption key.

Context

Both the table and the column encryption key must exist in the same schema.

Procedure

Open a SQL console for the schema where the table is being created and execute a statement similar to the following to encrypt a column:

```
CREATE TABLE MyTable (ID INT, Name NVARCHAR(32)
 CLIENTSIDE ENCRYPTION ON WITH myCEK RANDOM);
```

In the example above the column `Name` is encrypted using the column encryption key `myCEK`.

The type of encryption used is RANDOM, meaning that similar cleartext is encrypted to different ciphertext. NULL values are replaced with a special value and encrypted to hide the fact that the column is NULL for random encryption.

Results

The table is created and the specified column is encrypted using the specified column encryption key.

Related Information

CREATE TABLE Statement (Data Definition)
GRANT Statement (Access Control)

9.6 Change Column Encryption Status

Change an existing encrypted column to unencrypted and vice versa.

Prerequisites

You are a user who has been assigned key copy of the column encryption key used to encrypt or decrypt the data.

Procedure

Choose one of the following options:

Option	Action
Change an existing column from unencrypted to encrypted	Open a SQL console for the schema where the table exists and execute the following statement: `ALTER TABLE <table-name> ALTER (<column-name> ALTER CLIENTSIDE ENCRYPTION ON WITH <column-encryption-key> [ RANDOM | DETERMINISTIC ]);` For RANDOM encryption, similar cleartext is encrypted to different ciphertext. NULL values are replaced with a special value and encrypted to hide the fact that the column is NULL. Alternatively, you can use DETERMINISTIC encryption, which means that any cleartext value is encrypted into the same ciphertext value.
Change an existing column from encrypted to unencrypted	Open a SQL console for the schema where the table exists and execute the following statement: `ALTER TABLE <table-name> ALTER (<column-name> ALTER CLIENTSIDE ENCRYPTION OFF);`

Option	Action
Add an encrypted column to an existing table	Open a SQL console for the schema where the table exists and execute the following statement: `ALTER TABLE <table_name> ADD (<column_name> <data_type> CLIENTSIDE ENCRYPTION ON WITH <key_name> RANDOM);`
Cancel client-side encryption for a table	Open a SQL console for the schema where the table exists and execute the following statement: `ALTER TABLE <table_name> CANCEL CLIENTSIDE ENCRYPTION;` Cancel client-side encryption if you want to roll back your transaction. The only way to stop an encryption/decryption operation is for an administrator to kill the session or for the session owner to interrupt the SQL statement. If the operation is interrupted, you can cancel it by executing the above statement to roll back your transaction.
Continue client-side encryption for a table	Open a SQL console for the schema where the table exists and execute the following statement: `ALTER TABLE <table_name> CONTINUE CLIENTSIDE ENCRYPTION;` Continue client-side encryption if it has been interrupted and you want to restart the operation from where the encryption operation was interrupted without re-encrypting the whole column.

Results

You have encrypted or decrypted the specified column.

Related Information

ALTER CLIENTSIDE ENCRYPTION COLUMN KEY Statement (Encryption)
ALTER TABLE Statement (Data Definition)
Abandon a Long-Running ALTER TABLE Operation [page 51]

9.7 Abandon a Long-Running ALTER TABLE Operation

Abandon a long-running operation and then revert the table to its pre-encrypted state, or continue encrypting the table.

Context

All pending transactions on a connection where an ALTER TABLE statement is executed are committed before starting the ALTER TABLE operation.

Procedure

1. An ALTER TABLE operation can fail if it encounters an error. Alternately, you can explicitly abandon a long-running ALTER TABLE operation by choosing one of the following actions:

Option	Action
Explicitly issue a cancel session request by executing a statement similar to the following:	Execute the following statement: `ALTER SYSTEM CANCEL [WORK IN] SESSION <connection_id>;`
Issue a cancel request via the client.	The request depends on the client being used. For example, use `cancel()` for JDBC or `SQLCancel()` for ODBC.

2. Once the operation has been abandoned, choose to either cancel or continue client-side encryption for the table:

Option	Action
Cancel client-side encryption	Execute the following statement: `ALTER TABLE <table_name> CANCEL CLIENTSIDE ENCRYPTION;`
Continue client-side encryption:	Execute the following statement: `ALTER TABLE <table_name> CONTINUE CLIENTSIDE ENCRYPTION;`

Results

You have abandoned a long-running query and either canceled or continued encryption on the specified table.

Related Information

ALTER TABLE Statement (Data Definition)
ALTER SYSTEM CANCEL [WORK IN] SESSION Statement (System Management)

10 Configuring Client Connection Properties for Client-Side Data Encryption

Client-side data encryption is supported on JDBC and ODBC (SQLDBC) client drivers. To use client-side encryption, ensure that your interface meets the following requirements:

JDBC client

When connecting to an SAP HANA database using JDBC, use the following connection property.

Property	Value	Default	Description
`cseKeyStorePassword`	String		Provides the password for the local key store. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number. This value is only used locally by the client and is not passed on to the database.

deferredPrepare Connection Property	The deferredPrepare connection property must be set to DISABLED (the default) to use client-side encryption.
JDK Version	If you are using JDK 7, then use update 134 or later. Using JDK 7 versions with an earlier update results in a key generation error.
JDK Policy Files	Update the JDK policy files to support unlimited strength encryption. Failure to update the policy files results in a decryption error. Update the policy files as follows: • For JDK 6, download the files here: http://www.oracle.com/technetwork/java/javase/downloads/jce-6-download-429243.html • For JDK 7, download the files here: http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html • For JDK 8, update 150 and earlier, download the files here: http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

- For JDK 8, update 151 and later, and JDK 9, edit the file `conf/security/java.security` and uncomment the line `crypto.policy=unlimited`.
- For JDK 10.0, the `crypto.policy` property, in the `java.security` file of the JRE is already set to unlimited (`crypto.policy=unlimited`).

SQLDBC or ODBC client

When connecting to an SAP HANA database using ODBC, you need to use the following connection property:

Property	Value	Default	Description
`cseKeyStorePassword`	String		Provides the password for the local key store. Passwords must be at least eight characters and have at least one upper case character, one lower case character, and one number. This value is only used locally by the client and is not passed on to the database.

For SQLDBC-based clients, you must use CommonCryptoLib to use client-side encryption. Check the `/client` folder of the installation package to see if `COMMONCRYPTOLIB.TGZ` is present. If not, you can download CommonCryptoLib separately. For instructions on how to download CommonCryptoLib, see *Download and Install SAP Common Crypto Library* in the *SAP HANA Client Installation and Update Guide*.

i Note

To use CEK and CKP versioning, upgrade the client-drivers to 2.0 SPS 04 version.

Related Information

Download and Install SAP Common Crypto Library
Components of Client-Side Data Encryption [page 6]
Important Considerations When Using Client-Side Data Encryption [page 8]

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

THE BEST RUN