



Sybase IQ による高度なセキュリティ

Sybase IQ 15.3

ドキュメント ID：DC01151-01-1530-01

改訂：2011年5月

Copyright © 2011 by Sybase, Inc. All rights reserved.

このマニュアルは Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル・ノートで特に示されないかぎり、後続のリリースにも付属します。このマニュアルの内容は予告なしに変更されることがあります。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

このマニュアルの内容を弊社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても、複製、転載、翻訳することを禁じます。

Sybase の商標は、Sybase の商標リスト (<http://www.sybase.com/detail?id=1011207>) で確認できます。Sybase およびこのリストに掲載されている商標は、米国法人 Sybase, Inc. の商標です。® は、米国における登録商標であることを示します。

このマニュアルに記載されている SAP、その他の SAP 製品、サービス、および関連するロゴは、ドイツおよびその他の国における SAP AG の商標または登録商標です。

Java および Java 関連の商標は、米国およびその他の国における Sun Microsystems, Inc. の商標または登録商標です。

Unicode と Unicode のロゴは、Unicode, Inc. の登録商標です。

このマニュアルに記載されている上記以外の社名および製品名は、当該各社の商標または登録商標の場合があります。

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

Sybase IQ による高度なセキュリティ	1
Sybase IQ での FIPS サポート	1
Sybase IQ での Kerberos 認証サポート	2
Sybase IQ でのカラムの暗号化	2
暗号化に関する用語の定義	3
暗号化カラムのデータ型	3
AES_ENCRYPT 関数 [文字列]	6
AES_DECRYPT 関数 [文字列]	7
LOAD TABLE ENCRYPTED 句	8
暗号化テキストでの文字列の比較	10
カラムの暗号化に対するデータベース・オプション	11
暗号化と復号化の例	12
索引	23

目次

Sybase IQ による高度なセキュリティ

Sybase® IQ Advanced Security オプションでは、カラムの暗号化、FIPS 認定のネットワーク暗号化技術と、データベース接続、オペレーティング・システム・ログイン、ネットワーク・ログインに対する Kerberos 認証がサポートされています。Advanced Security オプションは、別途ライセンスが必要な Sybase IQ オプションです。

Sybase IQ での FIPS サポート

Sybase IQ では、FIPS 認定の暗号化技術に対する機能強化が行われています。FIPS は、Sybase IQ でサポートされているすべてのプラットフォームでサポートされています。

Sybase IQ の FIPS サポートによる主な影響は、暗号化に非決定性を持たせることです。現在はこの動作がデフォルトになっています。非決定的アルゴリズムでは、入力値が同じでも毎回異なる出力値が得られます。したがって、文字列を暗号化するキーを使用する場合、暗号化された文字列は毎回異なります。ただしこのアルゴリズムの場合、キーを使用して非決定的結果を復号化することも可能です。この機能により、暗号化アルゴリズムの解析はさらに難しくなり、暗号化はさらに安全になります。

FIPS のサポートは、別途ライセンスが必要な Sybase IQ Advanced Security オプションに含まれています。

Sybase IQ には、RSA と FIPS の両方のセキュリティが組み込まれています。RSA の暗号化では個別のライブラリは必要ありませんが、FIPS では `dbfips11.d11` と `sbgse2.d11` の2つのオプション・ライブラリが必要です。ライブラリ `sbgse2.d11` は Certicom から提供されています。いずれのセキュリティ・モデルにも証明書が必要です。`rsaserver` 証明書は、名前が `rsaserver.crt` から `rsaserver.id` に変更されました。

FIPS には、次のレジストリ設定も必要です。この設定は、Sybase IQ インストール・ユーティリティによって自動的に設定されます。

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Certicom\libs]
"expectedtag"=hex:5b,0f,4f,a6,e2,4a,ef,3b,
44,07,05,2e,b0,49,02,71,1f,d9,91,b6
```

「SQL Anywhere 11.0.1」 > 「SQL Anywhere Server」 – 「Database Administration」 > 「Security」 > 「Transport-layer security」と「SQL Anywhere 11.0.1」 > 「SQL

Anywhere Server」 – 「Database Administration」 > 「Security」 > 「Keeping your data secure」を参照してください。

Sybase IQ での Kerberos 認証サポート

Sybase IQ では Kerberos 認証がサポートされています。これは、オペレーティング・システムとネットワークの各ログインとデータベース接続の両方に対して 1 つのユーザ ID とパスワードを維持できるログイン機能です。

Kerberos クレデンシャルを使用することで、ユーザ ID やパスワードを指定せずにデータベースに接続できます。

Kerberos 認証は、別途ライセンスが必要な Sybase IQ Advanced Security オプションに含まれています。

「SQL Anywhere 11.0.1」 > 「SQL Anywhere Server – Database Administration」 > 「Starting and Connecting to Your Database」 > 「SQL Anywhere database connections」 > 「Kerberos authentication」を参照してください。

Sybase IQ でのカラムの暗号化

Sybase IQ では、ユーザ暗号化カラムがサポートされています。

Sybase IQ データベース・ファイルの強力な暗号化では、128 ビットのアルゴリズムと、セキュリティ・キーを使用します。データは判読不能で、キーがなければ事実上解読できません。サポートされるアルゴリズムは、FIPS-197 (Federal Information Processing Standard for the Advanced Encryption Standard) に準拠していません。

Sybase IQ では、**AES_ENCRYPT** 関数、**AES_DECRYPT** 関数、**LOAD TABLE ENCRYPTED** 句によってユーザ暗号化カラムをサポートしています。これらの関数をアプリケーションから呼び出すことで、カラム・データを明示的に暗号化および復号化できます。暗号化キーと復号化キーの管理は、アプリケーションで行います。

注意： この製品マニュアルで説明する Sybase IQ Advanced Security オプションの暗号化カラム・オプションの暗号化カラム機能を使用するには、正規のライセンスを取得している必要があります。

カラムの暗号化に影響を与えるデータベース・オプションがあります。この機能を使用する前に、「カラムの暗号化に対するデータベース・オプション」を参照してください。

参照：

- *カラムの暗号化に対するデータベース・オプション* (11 ページ)

暗号化に関する用語の定義

格納されているデータの暗号化について説明する場合に使用する用語の定義は、次のとおりです。

- **プレーン・テキスト** - 判読可能な元の形式のデータです。プレーン・テキストは文字データに限定されず、データを元の表現方法で記述するために使用されます。
- **暗号化テキスト** - プレーン・テキスト形式の情報の内容を保持する、判読不能な形式のデータです。
- **暗号化** - プレーン・テキストから暗号化テキストへの可逆性のある変換のことです。「暗号文化」とも呼ばれます。
- **復号化** - 暗号化テキストからプレーン・テキストへの逆変換のことです。「暗号解除」とも呼ばれます。
- **キー** - データの暗号化または復号化に使用する数値です。対称キー暗号化方式では、暗号化と復号化の両方に同じキーを使用します。非対称キー暗号化方式では、暗号化と復号化にそれぞれ異なる (ただし数学的に関連した) キーを使用します。Sybase IQ インタフェースはキーとして文字列を受け入れます。
- **Rijndael** - 「ラインダール」と読みます。さまざまなキー・サイズとブロック・サイズをサポートする暗号化アルゴリズムです。このアルゴリズムは、単純なバイト全体の操作を使用するように設計されているため、ソフトウェアで比較的簡単に実装できます。
- **AES** - Advanced Encryption Standard の略です。慎重に扱う必要があるが機密ではない電子データの保護用に FIP が承認した暗号化アルゴリズムです。AES は、ブロック・サイズとキー長を制限した Rijndael アルゴリズムを採用しています。AES は、Sybase IQ でサポートされているアルゴリズムです。

暗号化カラムのデータ型

暗号化カラムでサポートされるデータ型と、これらのデータ型の処理は、次のとおりです。

サポートされるデータ型

AES_ENCRYPT 関数の最初のパラメータには、次のサポートされるデータ型のいずれかを指定する必要があります。

CHAR	NUMERIC
VARCHAR	FLOAT

TINYINT	REAL
SMALLINT	DOUBLE
INTEGER	DECIMAL
BIGINT	DATE
BIT	TIME
BINARY	DATETIME
VARBINARY	TIMESTAMP
UNSIGNED INT	SMALLDATETIME
UNSIGNED BIGINT	

LOB データ型は、現時点では Sybase IQ のカラムの暗号化ではサポートされていません。

データ型の保持

Sybase IQ では、**AES_DECRYPT** 関数が、データ型をパラメータとして受け取るか、**CAST** 関数の中に含まれていれば、データを復号化したときにプレーン・テキストの元のデータ型に必ず戻ります。

Sybase IQ は、**CAST** のターゲットのデータ型と、元の暗号化されたデータのデータ型を比較します。この2つのデータ型が一致しない場合は、元のデータ型とターゲットのデータ型に関する詳細情報と一緒に -1001064 エラーが返されます。

たとえば、暗号化された **VARCHAR(1)** 値に対し、次の復号化文が有効であるとします。

```
SELECT AES_DECRYPT ( thecolumn, 'theKey',
VARCHAR(1) ) FROM thetable
```

データを、次の文を使用して復号しようとしたとします。

```
SELECT AES_DECRYPT ( thecolumn, 'theKey',
SMALLINT ) FROM thetable
```

この場合、次のエラーが返されます。

```
Decryption error: Incorrect CAST type smallint(5,0)
for decrypt data of type varchar(1,0).
```

このようなデータ型のチェックは、データ型が指定された場合にのみ実行されます。**CAST** もデータ型パラメータも指定されていない場合、クエリは暗号化テキストをバイナリ・データとして返します。

注意： 次の文のように、リテラル定数に対して **AES_ENCRYPT** 関数を使用したとします。

```
INSERT INTO t (cipherCol) VALUES (AES_ENCRYPT (1, 'key'))
```

この場合、1 のデータ型があいまいになることに注意してください。1 のデータ型は、TINYINT、SMALLINT、INTEGER、UNSIGNED INT、BIGINT、UNSIGNED BIGINT、またはその他のデータ型になる可能性があります。

あいまいさの問題を解決するために、次の文のように **CAST** 関数を明示的に使用することをおすすめします。

```
INSERT INTO t (cipherCol)
VALUES ( AES_ENCRYPT (CAST (1 AS UNSIGNED INTEGER), 'key'))
```

データを暗号化するとき **CAST** 関数を使用してデータ型を明示的に変換しておけば、データを復号化するとき **CAST** 関数を使用することで問題の発生を防止できます。

暗号化対象のデータがカラムのデータの場合、または暗号化されたデータが **LOAD TABLE** によって挿入された場合は、あいまいさは発生しません。

暗号化テキストに対する異なるデータ型の影響

AES_ENCRYPT の入力と同じデータ型にキャストすることで、同一の暗号化テキストを生成します。

AES_ENCRYPT 関数によって生成される暗号化テキストは、入力値とキーが同じであっても、データ型が異なれば違ったものになります。したがって、2つの暗号化テキスト・カラムに、2つの異なるデータ型を持つ暗号化した値が保持されている場合、それらのカラムをジョインして同じ結果が返されるとは限りません。

たとえば、次の行を実行したとします。

```
CREATE TABLE tablea(c1 int, c2 smallint);
INSERT INTO tablea VALUES (100,100);
```

値 **AES_ENCRYPT(c1, 'key')** は **AES_ENCRYPT(c2, 'key')** と異なり、値 **AES_ENCRYPT(c1, 'key')** は **AES_ENCRYPT(100, 'key')** と異なります。

この問題を解決するには、**AES_ENCRYPT** の入力と同じデータ型にキャストします。たとえば、次のサンプル・コードの結果は同じになります。

```
AES_ENCRYPT(c1, 'key');
AES_ENCRYPT(CAST(c2 AS INT), 'key');
AES_ENCRYPT(CAST(100 AS INT), 'key');
```

参照：

- **AES_ENCRYPT** 関数 [文字列] (6 ページ)

AES_ENCRYPT 関数 [文字列]

指定された暗号化キーを使用して、指定された値を暗号化し、VARBINARY または LONG VARBINARY を返します。

構文

```
AES_ENCRYPT( string-expression, key )
```

パラメータ

string-expression – 暗号化対象のデータです。**AES_ENCRYPT** にはバイナリ値を渡すこともできます。データベースで大文字と小文字が区別されない場合でも、このパラメータでは大文字と小文字が区別されます。

key – *string-expression* の暗号化に使用する暗号化キーです。元の値を取得するには、値を復号化するときにも同じキーを使用する必要があります。データベースで大文字と小文字が区別されない場合でも、このパラメータでは大文字と小文字が区別されます。

パスワードと同様、キーの値には推測されにくい値を選ぶことが重要です。キーの値には、長さが 16 文字以上で、大文字と小文字を含み、数字と特殊文字を使用したものを選ぶことをおすすめします。このキーは、データを復号化するとき常に必要です。

警告！ キーをなくさないでください。キーのコピーを安全な場所に保管してください。キーを失うと、暗号化したデータにまったくアクセスできなくなります。データを修復する方法もありません。

使用法

AES_ENCRYPT は、入力値の *string-expression* より最大で 31 バイト長い VARBINARY 値を返します。この関数によって返される値は暗号化テキストであり、判読できません。**AES_DECRYPT** 関数を使用して、**AES_ENCRYPT** 関数で暗号化された *string-expression* を復号化できます。*string-expression* を正常に復号化するには、データの暗号化に使用されたのと同じ暗号化キーとアルゴリズムを使用する必要があります。正しい暗号化キーを指定しないと、エラーが発生します。

暗号化した値をテーブルに格納する場合は、カラムのデータ型を VARBINARY または VARCHAR にし、長さを 32 バイト以上にして、データに対して文字セット変換が実行されないようにします (文字セット変換が実行されるとデータを復号化できなくなる場合があります)。VARBINARY カラムまたは VARCHAR カラムの長さが 32 バイトより短いと、**AES_DECRYPT** 関数はエラーを返します。

AES_ENCRYPT 関数の結果のデータ型は LONG BINARY になる可能性があります。**SELECT INTO** 文で **AES_ENCRYPT** を使用する場合は、非構造化データ分析オプショ

ンのライセンスを所有しているか、**CAST** を使用して **AES_ENCRYPT** を正しいデータ型とサイズに設定する必要があります。

詳細と使用法については、『リファレンス』の「ビルディング・ブロック、テーブル、およびプロシージャ」>「SQL 関数」>「アルファベット順の関数リスト」>「REPLACE 関数 [文字列]」を参照してください。

標準と互換性

- SQL - ISO/ANSI SQL 文法に対するベンダの拡張機能です。
- Sybase - Adaptive Server Enterprise ではサポートされていません。

参照：

- *AES_DECRYPT* 関数 [文字列] (7 ページ)
- 暗号化と復号化の例 (12 ページ)
- *LOAD TABLE ENCRYPTED* 句 (8 ページ)
- 暗号化テキストに対する異なるデータ型の影響 (5 ページ)
- 暗号化カラムのデータ型 (3 ページ)

AES_DECRYPT 関数 [文字列]

指定されたキーを使用して文字列を復号化します。デフォルトでは、VARBINARY、LONG BINARY、または元のプレーン・テキストのデータ型が返されます。

構文

```
AES_DECRYPT( string-expression, key [, data-type ] )
```

パラメータ

string-expression – 復号化対象の文字列。この関数には、バイナリ値を渡すこともできます。データベースで大文字と小文字が区別されない場合でも、このパラメータでは大文字と小文字が区別されます。

key – *string-expression* の復号化に必要な暗号化キーです。暗号化された元の値を取得するには、このキーは、*string-expression* の暗号化に使用されたのと同じ暗号化キーである必要があります。データベースで大文字と小文字が区別されない場合でも、このパラメータでは大文字と小文字が区別されます。

警告！ キーをなくさないでください。キーのコピーを安全な場所に保管してください。キーを失うと、暗号化したデータにまったくアクセスできなくなります。データを修復する方法もありません。

data-type – このオプション・パラメータでは、復号化する *string-expression* のデータ型を指定します。これは、元のプレーン・テキストと同じデータ型である必要があります。

AES_ENCRYPT 関数を使用してデータを挿入する際に **CAST** 文を使用していない場合は、**VARCHAR** を *data-type* として渡すことにより、**AES_DECRYPT** を使用して同じデータを表示することができます。*data-type* を **AES_DECRYPT** に渡さない場合は、**VARBINARY** データ型が返されます。

使用法

AES_DECRYPT 関数を使用して、**AES_ENCRYPT** 関数で暗号化された *string-expression* を復号化できます。データ型の指定がない場合、この関数は、入力文字列と同じバイト数の **VARBINARY** 値または **LONG VARBINARY** 値を返します。それ以外の場合は、指定したデータ型が返されます。

string-expression を正常に復号化するには、データの暗号化に使用されたのと同じ暗号化キーを使用する必要があります。暗号化キーが正しくない場合は、エラーが返されます。

例

`user_info` テーブルからユーザのパスワードを復号化します。

```
SELECT AES_DECRYPT(user_pwd, '8U3dkA', CHAR(100))
FROM user_info;
```

標準と互換性

- SQL - ISO/ANSI SQL 文法に対するベンダの拡張機能です。
- Sybase - Adaptive Server Enterprise ではサポートされていません。

参照：

- **AES_ENCRYPT** 関数 [文字列] (6 ページ)
- 暗号化と復号化の例 (12 ページ)
- **LOAD TABLE ENCRYPTED** 句 (8 ページ)
- 暗号化カラムのデータ型 (3 ページ)

LOAD TABLE ENCRYPTED 句

LOAD TABLE 文では、`column-spec` キーワード **ENCRYPTED** がサポートされています。

column-specs は、**LOAD TABLE** 文のカラム名の後ろに、次の順序で指定する必要があります。

- *format-specs*
- *null-specs*
- *encrypted-specs*

完全な構文については、『リファレンス』の「文とオプション」 > 「SQL 文」 > 「LOAD TABLE 文」を参照してください。

構文

```
ENCRYPTED(data-type 'key-string' [, 'algorithm-string' ] )
```

パラメータ

- *data-type* – **AES_ENCRYPT** 関数への入力として使用する、入力ファイル・フィールドの変換先のデータ型です。*data-type* は、**AES_DECRYPT** 関数の出力のデータ型と同じデータ型である必要があります。
- *key-string* – データの暗号化に使用する暗号化キーです。このキーは、文字列リテラルにする必要があります。元の値を取得するには、値を復号化するときにも同じキーを使用する必要があります。データベースで大文字と小文字が区別されない場合でも、このパラメータでは大文字と小文字が区別されます。パスワードと同様、キーの値には推測されにくい値を選ぶことが重要です。キーの値には、長さが 16 文字以上で、大文字と小文字を含み、数字と特殊文字を使用したものを選ぶことをおすすめします。このキーは、データを復号化するとき常に必要です。

警告！ キーをなくさないでください。キーのコピーを安全な場所に保管してください。キーを失うと、暗号化したデータにまったくアクセスできなくなります。データを修復する方法もありません。

- *algorithm-string* – データの暗号化に使用するアルゴリズムです。このパラメータはオプションですが、データの暗号化と復号化は同じアルゴリズムを使用しに行く必要があります。現時点では、サポートされているアルゴリズムは AES のみなので、これがデフォルトで使用されます。AES は、NIST (National Institute of Standards and Technology) がブロック暗号化の新しい AES (Advanced Encryption Standard) として選択したブロック暗号化アルゴリズムです。

使用法

ENCRYPTED のカラムの指定では、カラムにロードされるデータの暗号化に使用する暗号化キーと、必要に応じてアルゴリズムを指定できます。このロード先のカラムのデータ型は **VARBINARY** である必要があります。他のデータ型を指定するとエラーが返されます。

LOAD TABLE ENCRYPTED の例

```
LOAD TABLE table_name
(
  plaintext_column_name,
```

```
a_ciphertext_column_name
NULL('nil')
ENCRYPTED(varchar(6), 'tHefiRstkEy') ,
another_encrypted_column
ENCRYPTED(bigint, 'thEseconDkeY', 'AES')
)
FROM '/path/to/the/input/file'
FORMAT ascii
DELIMITED BY ';'
ROW DELIMITED BY '¥0xa'
QUOTES OFF
ESCAPES OFF
```

ここで、**LOAD TABLE** 文の入力ファイルのフォーマットは、次のとおりです。

```
a;b;c;
d;e;f;
g;h;i;
```

参照：

- [AES_ENCRYPT 関数 \[文字列\] \(6 ページ\)](#)
- [AES_DECRYPT 関数 \[文字列\] \(7 ページ\)](#)
- [暗号化と復号化の例 \(12 ページ\)](#)
- [暗号化カラムのデータ型 \(3 ページ\)](#)

暗号化テキストでの文字列の比較

データの大文字と小文字が区別されない場合、または ISO_BINENG 以外の照合を使用している場合は、文字列の比較を実行するために暗号化テキスト・カラムを復号化する必要があります。

文字列の比較を実行する場合、多くの照合において等価な文字列と同一の文字列の違いは重要です。これは、**CREATE DATABASE** の **CASE** オプションに依存します。**CASE RESPECT** に設定され、ISO_BINENG 照合を使用するデータベースは、Sybase IQ でのデフォルトであり、等価性と同一性の問題は同様に解決されます。

同一の文字列は常に等価ですが、等価な文字列は必ずしも同一ではありません。文字列が同じバイト値を使用して表現される場合のみ、文字列は同一です。データの大文字と小文字が区別されない場合、または複数の文字が等しいものとして処理される必要がある照合を使用する場合、等価性と同一性の違いは重要です。ISO_LATIN1 はこのような照合の例です。

たとえば、大文字と小文字が区別されないデータベース内の文字列 "ABC" と文字列 "abc" は、同一ではありませんが等価です。大文字と小文字が区別されるデータベースの場合、これらは同一でも等価でもありません。

Sybase 暗号化関数によって作成される暗号化テキストは、等価性ではなく同一性を保持します。つまり、"ABC" と "abc" の暗号化テキストは決して等価ではありません。

照合または **CASE** の設定でこのような比較が許可されていない場合に暗号化テキストで等価性の比較を実行するには、アプリケーションでそのカラム内の値を、等価な値がなく、したがって同一の値もない、何らかの標準の形式に変更する必要があります。たとえば、**CASE IGNORE** と **ISO_BINENG** 照合を使用してデータベースを作成し、アプリケーションで入力値をカラムに挿入する前にその入力値すべてに **UCASE** を適用すれば、等価な値はすべて同一にもなります。

カラムの暗号化に対するデータベース・オプション

カラムの暗号化と復号化に影響する Sybase IQ データベース・オプションの設定があります。ほとんどのカラム暗号化処理で、デフォルトの設定は最適な設定ではありません。

暗号化テキストの予期しないトランケートの防止

STRING_RTRUNCATION を設定することで、暗号化テキストが予期せずにトランケートされないようにします。

暗号化関数による暗号化テキストの出力 (またはその他の文字やバイナリ文字列) が予期せずにトランケートされないようにするには、次のデータベース・オプションを設定します。

```
SET OPTION STRING_RTRUNCATION = 'ON'
```

STRING_RTRUNCATION を **ON** (デフォルト) に設定すると、ロード、挿入、更新、または **SELECT INTO** の操作で文字列がトランケートされるたびにエンジンでエラーが発生します。これは **ISO/ANSI SQL** の動作であり、推奨される方法です。

明示的なトランケートが必要な場合は、**LEFT**、**SUBSTRING**、**CAST** などの文字列式を使用します。

STRING_RTRUNCATION を **OFF** に設定すると、文字列の暗黙的なトランケートが実行されます。

また、**AES_DECRYPT** 関数も、入力された暗号化テキストのデータ長の有効性をチェックするだけでなく、テキスト出力もチェックして、復号化した後のデータ長と指定されたキーの適正さの両方を検証します (データ型の引数が指定された場合は、データ型もチェックされます)。

暗号化テキストの整合性の維持

ASE_BINARY_DISPLAY を設定することで、暗号化テキストの整合性を維持します。

```
SET OPTION ASE_BINARY_DISPLAY = 'OFF'
```

ASE_BINARY_DISPLAY を **OFF** (デフォルト) に設定すると、バイナリ・データはロー・バイナリ形式のまま変更されません。

ASE_BINARY_DISPLAY を ON に設定すると、バイナリ・データは 16 進数文字列の表示表現に変換されます。このオプションは、エンド・ユーザに対して表示するデータが必要な場合、またはデータを別の外部システムにエクスポートする必要がある (転送中にロー・バイナリが変更される可能性がある) 場合にのみ、一時的に ON に設定します。

暗号化テキストの誤用の防止

CONVERSION_MODE を設定することで、実質的に意味のない操作となる暗号化データの暗黙のデータ型変換を防止します。

CONVERSION_MODE データベース・オプションは、さまざまな操作におけるバイナリ・データ型 (BINARY、VARBINARY、LONG BINARY) とその他の非バイナリ・データ型 (BIT、TINYINT、SMALLINT、INT、UNSIGNED INT、BIGINT、UNSIGNED BIGINT、CHAR、VARCHAR、LONG VARCHAR) 間の暗黙的な変換を制限します。CONVERSION_MODE を使用することで、実質的に意味のない操作となる暗号化データの暗黙のデータ型変換を防止します。

```
SET TEMPORARY OPTION CONVERSION_MODE = 1
```

CONVERSION_MODE を 1 に設定すると、**INSERT** コマンド、**UPDATE** コマンド、クエリにおけるバイナリ・データ型からその他の非バイナリ・データ型への暗黙の変換が制限されます。バイナリ変換制限モードは、**LOAD TABLE** のデフォルト値と **CHECK** の制約にも適用されます。

CONVERSION_MODE オプションのデフォルト値 0 では、12.7 より前のバージョンの Sybase IQ でのバイナリ・データ型の暗黙の変換動作が維持されます。

『リファレンス』の「オプション」 > 「データベース・オプション」 > 「アルファベット順のオプション・リスト」 > 「CONVERSION_MODE オプション」を参照してください。

暗号化と復号化の例

コメント付きの SQL で記述した、**AES_ENCRYPT** 関数と **AES_DECRYPT** 関数の使用例を示します。

```
-- This example of aes_encrypt and aes_decrypt function use is
-- presented in three parts:
--
-- Part I: Preliminary description of target tables and users as DDL
-- Part II: Example schema changes motivated by introduction of
-- encryption
-- Part III: Use of views and stored procedures to protect encryption
-- keys
--
```



```

-- Part I: Define target tables and users

-- Assume two classes of user, represented here by the instances
-- PrivUser and NonPrivUser, assigned to groups reflecting
differing
-- privileges.

-- The initial state reflects the schema prior to the introduction
-- of encryption.

-- Set up the starting context: There are two tables with a common
key.
-- Some columns contain sensitive data, the remaining columns do
not.
-- The usual join column for these tables is sensitiveA.
-- There is a key and a unique index.

grant connect to PrivUser identified by 'verytrusted' ;
grant connect to NonPrivUser identified by 'lesstrusted' ;

grant connect to high_privileges_group ;
grant group to high_privileges_group ;
grant membership in group high_privileges_group to PrivUser ;

grant connect to low_privileges_group ;
grant group to low_privileges_group ;
grant membership in group low_privileges_group to NonPrivUser ;

create table DBA.first_table
    (sensitiveA char(16) primary key
    ,sensitiveB numeric(10,0)
    ,publicC varchar(255)
    ,publicD date
    ) ;

-- There is an implicit unique HG (HighGroup) index enforcing the
primary key.

create table second_table
    (sensitiveA char(16)
    ,publicP integer
    ,publicQ tinyint
    ,publicR varchar(64)
    ) ;

create hg index second_A_HG on second_table ( sensitiveA ) ;

-- TRUSTED users can see the sensitive columns.

grant select ( sensitiveA, sensitiveB, publicC, publicD )
    on DBA.first_table to PrivUser ;
grant select ( sensitiveA, publicP, publicQ, publicR )
    on DBA.second_table to PrivUser ;

-- Non-TRUSTED users in existing schema need to see sensitiveA to

```

```

be
--   able to do joins, even though they should not see sensitiveB.

grant select ( sensitiveA, publicC, publicD )
  on DBA.first_table to NonPrivUser ;
grant select ( sensitiveA, publicP, publicQ, publicR )
  on DBA.second_table to NonPrivUser ;

--   Non-TRUSTED users can execute queries such as

select I.publicC, 3*II.publicQ+1
from DBA.first_table I, DBA.second_table II
where I.sensitiveA = II.sensitiveA and I.publicD IN
( '2006-01-11' ) ;

--   and

select count(*)
from DBA.first_table I, DBA.second_table II
where I.sensitiveA = II.sensitiveA and SUBSTR(I.sensitiveA,4,3)
BETWEEN '345' AND '456' ;

--   But only TRUSTED users can execute the query

select I.sensitiveB, 3*II.publicQ+1
from DBA.first_table I, DBA.second_table II
where I.sensitiveA = II.sensitiveA and I.publicD IN
( '2006-01-11' ) ;

--   Part II: Change the schema in preparation for encryption
--
--   The DBA introduces encryption as follows:
--
--   For applicable tables, the DBA changes the schema, adjusts
access
--   permissions, and updates existing data. The encryption
--   keys used are hidden in a subsequent step.

--   DataLength comparison for length of varbinary encryption result
--   (units are Bytes):
--
--   PlainText CipherText      Corresponding Numeric Precisions
--
--           0           16
--          1 - 16          32      numeric(1,0)   - numeric(20,0)
--         17 - 32          48      numeric(21,0)  - numeric(52,0)
--         33 - 48          64      numeric(53,0)  - numeric(84,0)
--         49 - 64          80      numeric(85,0)  - numeric(116,0)
--         65 - 80          96      numeric(117,0) - numeric(128,0)
--         81 - 96         112
--         97 - 112        128
--        113 - 128        144
--        129 - 144        160
--        145 - 160        176

```

```

--      161 - 176      192
--      177 - 192      208
--      193 - 208      224
--      209 - 224      240

--      The integer data types tinyint, small int, integer, and bigint
--      are varbinary(32) ciphertext.

--      The exact relationship is
--      DATALENGTH(ciphertext) =
--      (((DATALENGTH(plaintext)+ 15) / 16) + 1) * 16

--      For the first table, the DBA chooses to preserve both the
plaintext and
--      ciphertext forms. This is not typical and should only be done if
the
--      database files are also encrypted.

--      Take away NonPrivUser's access to column sensitiveA and transfer
--      access to the ciphertext version.

--      Put a unique index on the ciphertext column. The ciphertext
--      itself is indexed.

--      NonPrivUser can select the ciphertext and use it.

--      PrivUser can still select either form (without paying decrypt
costs).

      revoke select ( sensitiveA ) on DBA.first_table from
NonPrivUser ;
      alter table DBA.first_table add encryptedA varbinary(32) ;
      grant select ( encryptedA ) on DBA.first_table to PrivUser ;
      grant select ( encryptedA ) on DBA.first_table to NonPrivUser ;
      create unique hg index first_A_unique on first_table
( encryptedA ) ;
      update DBA.first_table
      set encryptedA = aes_encrypt(sensitiveA, 'seCr3t')
      where encryptedA is null ;
      commit

--      Now change column sensitiveB.

      alter table DBA.first_table add encryptedB varbinary(32) ;
      grant select ( encryptedB ) on DBA.first_table to PrivUser ;
      create unique hg index first_B_unique on first_table
( encryptedB ) ;
      update DBA.first_table
      set encryptedB = aes_encrypt(sensitiveB,
      'givethiskeytonoone') where encryptedB is null ;
      commit

--      For the second table, the DBA chooses to keep only the
ciphertext.
--      This is more typical and encrypting the database files is not
required.

```

```

    revoke select ( sensitiveA ) on DBA.second_table from
NonPrivUser ;
    revoke select ( sensitiveA ) on DBA.second_table from PrivUser ;
    alter table DBA.second_table add encryptedA varbinary(32) ;
    grant select ( encryptedA ) on DBA.second_table to PrivUser ;
    grant select ( encryptedA ) on DBA.second_table to NonPrivUser ;
    create unique hg index second_A_unique on second_table
( encryptedA ) ;
    update DBA.second_table
        set encryptedA = aes_encrypt(sensitiveA, 'seCr3t')
        where encryptedA is null ;
    commit
    alter table DBA.second_table drop sensitiveA ;

-- The following types of queries are permitted at this point,
before
-- changes are made for key protection:

-- Non-TRUSTED users can equi-join on ciphertext; they can also
select
-- the binary, but have no way to interpret it.

    select I.publicC, 3*II.publicQ+1
    from DBA.first_table I, DBA.second_table II
    where I.encryptedA = II.encryptedA and I.publicD IN
( '2006-01-11' ) ;

-- Ciphertext-only access rules out general predicates and
expressions.
-- The following query does not return meaningful results.
--
-- NOTE: These four predicates can be used on the varbinary
containing
-- ciphertext:
--     = (equality)
--     <> (inequality)
--     IS NULL
--     IS NOT NULL

select count(*)
from DBA.first_table I, DBA.second_table II
where I.encryptedA = II.encryptedA and SUBSTR(I.encryptedA,4,3)
    BETWEEN '345' AND '456' ;

-- The TRUSTED user still has access to the plaintext columns that
-- were retained. Therefore, this user does not need to call
-- aes_decrypt and does not need the key.

select count(*)
from DBA.first_table I, DBA.second_table II
where I.encryptedA = II.encryptedA and SUBSTR(I.sensitiveA,4,3)
    BETWEEN '345' AND '456' ;

-- Part III: Protect the encryption keys

```

```

-- This section illustrates how to grant access to the plaintext,
but
-- still protect the keys.

-- For the first table, the DBA elected to retain the plaintext
columns.
-- Therefore, the following view has the same capabilities as the
trusted
-- user above.
-- Assume group_member is being used for additional access control.

-- NOTE: In this example, NonPrivUser still has access to the
ciphertext
-- encrypted in the base table.

create view DBA.a_first_view (sensitiveA, publicC, publicD)
as
select
  IF group_member('high_privileges_group',user_name()) = 1
    THEN sensitiveA
    ELSE NULL
  ENDIF,
  publicC,
  publicD
from first_table ;

grant select on DBA.a_first_view to PrivUser ;
grant select on DBA.a_first_view to NonPrivUser ;

-- For the second table, the DBA did not keep the plaintext.
-- Therefore, aes_decrypt calls must be used in the view.
-- IMPORTANT: Hide the view definition with ALTER VIEW, so that no
one
-- can discover the key.

create view DBA.a_second_view
(sensitiveA,publicP,publicQ,publicR)
as
select
  IF group_member('high_privileges_group',user_name()) = 1
    THEN aes_decrypt(encryptedA,'seCr3t', char(16))
    ELSE NULL
  ENDIF,
  publicP,
  publicQ,
  publicR
from second_table ;

alter view DBA.a_second_view set hidden ;
grant select on DBA.a_second_view to PrivUser ;
grant select on DBA.a_second_view to NonPrivUser ;

-- Likewise, the key used for loading can be protected in a stored
procedure.
-- By hiding the procedure (just as the view is hidden), no-one can

```

```

see
-- the keys.

create procedure load_first_proc(@inputFileName varchar(255),
                                @colDelim varchar(4) default '$',
                                @rowDelim varchar(4) default '¥n')
begin
    execute immediate with quotes
        'load table DBA.second_table
        (encryptedA encrypted(char(16),' ||
        '|| 'seCr3t' || '|| '|| '),publicP,publicQ,publicR)
        ' from ' || '|| @inputFileName || '|| '||
        ' delimited by ' || '|| @colDelim || '|| '||
        ' row delimited by ' || '|| @rowDelim || '|| '||
        ' quotes off escapes off' ;
    end
;

alter procedure DBA.load_first_proc set hidden ;

-- Call the load procedure using the following syntax:

call load_first_proc('/dev/null', '$', '¥n') ;

-- Below is a comparison of several techniques for protecting the
-- encryption keys by using user-defined functions (UDFs), other
views,
-- or both. The first and the last alternatives offer maximum
performance.

-- The second_table is secured as defined earlier.

-- Alternative 1:
-- This baseline approach relies on restricting access to the
entire view.

create view
DBA.second_baseline_view(sensitiveA,publicP,publicQ,publicR)
as
select
    IF group_member('high_privileges_group',user_name()) = 1
        THEN aes_decrypt(encryptedA,'seCr3t', char(16))
        ELSE NULL
    ENDIF,
    publicP,
    publicQ,
    publicR
from DBA.second_table ;

alter view DBA.second_baseline_view set hidden ;
grant select on DBA.second_baseline_view to NonPrivUser ;
grant select on DBA.second_baseline_view to PrivUser ;

```

```

-- Alternative 2:
-- Place the encryption function invocation within a user-defined
-- function (UDF).
-- Hide the definition of the UDF. Restrict the UDF permissions.
-- Use the UDF in a view that handles the remainder of the security
-- and business logic.
-- Note: The view itself does not need to be hidden.

create function DBA.second_decrypt_function(IN datum
varbinary(32))
    RETURNS char(16) DETERMINISTIC
    BEGIN
        RETURN aes_decrypt(datum,'seCr3t', char(16));
    END ;

grant execute on DBA.second_decrypt_function to PrivUser ;
alter function DBA.second_decrypt_function set hidden ;

create view
DBA.second_decrypt_view(sensitiveA,publicP,publicQ,publicR)
as
    select
        IF group_member('high_privileges_group',user_name())
= 1
            THEN second_decrypt_function(encryptedA)
            ELSE NULL
        ENDIF,
        publicP,
        publicQ,
        publicR
    from DBA.second_table ;

grant select on DBA.second_decrypt_view to NonPrivUser ;
grant select on DBA.second_decrypt_view to PrivUser ;

-- Alternative 3:
-- Sequester only the key selection in a user-defined function.
-- This function could be extended to support selection of any
-- number of keys.
-- This UDF is also hidden and has restricted execute privileges.
-- Note: Any view that uses this UDF therefore does not compromise
-- the key values.

create function DBA.second_key_function()
    RETURNS varchar(32) DETERMINISTIC
    BEGIN
        return 'seCr3t' ;
    END

grant execute on DBA.second_key_function to PrivUser ;
alter function DBA.second_key_function set hidden ;

create view

```

Sybase IQ による高度なセキュリティ

```
DBA.second_key_view(sensitiveA,publicP,publicQ,publicR)
    as
        select
            IF
group_member('high_privileges_group',user_name()) = 1
                THEN
aes_decrypt(encryptedA,second_key_function(),
            char(16))
            ELSE NULL
            ENDIF,
            publicP,
            publicQ,
            publicR
        from DBA.second_table ;

grant select on DBA.second_key_view to NonPrivUser ;
grant select on DBA.second_key_view to PrivUser ;

-- Alternative 4:
-- The recommended alternative is to separate the security logic
-- from the business logic by dividing the concerns into two views.
-- Only the security logic view needs to be hidden.
-- Note: The performance of this approach is similar to that of the
first
-- alternative.

        create view

DBA.second_SecurityLogic_view(sensitiveA,publicP,publicQ,publicR)
    as
        select
            IF group_member('high_privileges_group',user_name())
= 1
                THEN aes_decrypt(encryptedA,'seCr3t', char(16))
            ELSE NULL
            ENDIF,
            publicP,
            publicQ,
            publicR
        from DBA.second_table ;

alter view DBA.second_SecurityLogic_view set hidden ;

        create view

DBA.second_BusinessLogic_view(sensitiveA,publicP,publicQ,publicR)
    as
        select
            sensitiveA,
            publicP,
            publicQ,
            publicR
        from DBA.second_SecurityLogic_view ;

grant select on DBA.second_BusinessLogic_view to NonPrivUser ;
```



```
grant select on DBA.second_BusinessLogic_view to PrivUser ;  
-- End of encryption example
```

参照：

- *AES_ENCRYPT* 関数 [文字列] (6 ページ)
- *AES_DECRYPT* 関数 [文字列] (7 ページ)
- *LOAD TABLE ENCRYPTED* 句 (8 ページ)

索引

A

Advanced Security オプション
Sybase IQ 用 1

AES

定義 3

AES_DECRYPT 関数

SQL 構文 7

AES_ENCRYPT 関数

SQL 構文 6

ASE_BINARY_DISPLAY

データベース・オプション 11

暗号化の整合性 11

C

CONVERSION_MODE

データベース・オプション 12

暗号化テキストの保護 12

F

FIPS

Sybase IQ でのサポート 1

暗号化アルゴリズム 2

K

Kerberos 認証 2

L

LOAD TABLE

ENCRYPTED 句 8

ENCRYPTED 句の例 9

R

Rijndael 3

RSA サポート 1

S

SQL 関数

AES_DECRYPT 関数 7

AES_ENCRYPT 関数 6

STRING_RTRUNCATION

データベース・オプション 11

暗号化テキストの保護 11

お

オプション

ASE_BINARY_DISPLAY 11

CONVERSION_MODE 12

STRING_RTRUNCATION 11

カラムの暗号化用 11

カラムの復号化用 11

か

カラムの暗号化 2

き

キー

定義 3

せ

セキュリティ

FIPS サポート 1

Kerberos 認証 2

RSA サポート 1

Sybase IQ Advanced Security オプション 1

て

データベース・オプション

ASE_BINARY_DISPLAY 11

CONVERSION_MODE 12

STRING_RTRUNCATION 11

カラムの暗号化用 11

カラムの復号化用 11

データ型

暗号化カラムのサポート 3

元の型の保持 3, 4

索引

ふ

プレーン・テキスト 3