Sybase® IQ による高度なセキュリティ

ドキュメント ID：DC01151-01-1510-01
改訂：2009 年 7 月

このマニュアルでは、Sybase IQ Advanced Security オプションについて説明します。

Sybase IQ による高度なセキュリティ

Sybase IQ Advanced Security オプションには、データとユーザの両方を最も高いレベルで確実に保護する追加のセキュリティ・メカニズムが備わっています。ビジネスにとってデータが通貨のようなものである以上、貴重なデータ資産を保護することは、最優先事項の 1 つとなります。

Sybase IQ Advanced Security オプションで提供される安全なビジネス・インテリジェンス機能には、カラムの暗号化と連邦情報処理標準 (FIPS: Federal Information Processing Standards) 認定の暗号化技術のネットワーク暗号化サポート、オペレーティング・システムおよびネットワークへのログインとデータベース接続の両方に使用する Kerberos 認証などがあります。

Sybase IQ Advanced Security オプションによって強化されたセキュリティ機能は、FIPS の標準と条例に準拠しています。

Advanced Security オプションは、Sybase IQ の個別にライセンス供与されるオプションです。

Sybase IQ での FIPS サポート

Sybase IQ では、FIPS 認定の暗号化技術への機能強化が行われています。FIPS は、Sybase IQ でサポートされ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 での 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。非決定性アルゴリズムとは、同じ入力から毎回異なる出力値が生成されるアルゴリズムです。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、このアルゴリズムでは、キーを使用して非決定的な結果を復号化できます。この機能により、暗号化アルゴリズムの解析が難しくなり、暗号化が安全になります。

FIPS のサポートは、個別にライセンス供与される Sybase IQ Advanced Security オプションの一部です。

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 暗号化には個別のライブラリは必要ありませんが、FIPS には *dbfips11.dll* と *sbgse2.dll* の 2 つのライブラリがオプションで必要です。ライブラリ *sbgse2.dll* は、Certicom によって提供されています。どちらのセキュリティ・モデルにも証明書が必要です。証明書 rsaserver は、*rsaserver.crt* から *rsaserver.id* に名前が変更されました。

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¥libsb]
"expectedtag"=hex:5b,0f,4f,a6,e2,4a,ef,3b,44,07,05,2e,
b0,49,02,71,1f,d9,91,b6
```

FIPS および RSA 暗号化の使用方法の詳細については、『SQL Anywhere Server — Database Administration』の Transport-layer security (http://infocenter.sybase.com/help/topic/com.sybase.help.sqlanywhere.11.0.1/dbadmin_en11/da-transport-layer-security.html) および Keeping your data secure (http://infocenter.sybase.com/help/topic/com.sybase.help.sqlanywhere.11.0.1/dbadmin_en11/da-security.html) を参照してください。

Sybase IQ での Kerberos 認証のサポート

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・システムおよびネットワークへの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と、ユーザ ID やパスワードを指定しなくてもデータベースに接続できます。

Kerberos 認証は、個別にライセンス供与される Sybase IQ Advanced Security オプションの一部です。

Kerberos 認証の使用方法の詳細については、『SQL Anywhere Server — Database Administration』の Kerberos authentication (http://infocenter.sybase.com/help/topic/com.sybase.help.sqlanywhere.11.0.1/dbadmin_en11/da-kerberos-authentication.html) を参照してください。

Sybase IQ でのカラム暗号化

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) に準拠しています。

Sybase IQ では、AES_ENCRYPT 関数、AES_DECRYPT 関数、LOAD TABLE ENCRYPTED 句の追加によって、ユーザによるカラムの暗号化をサポートします。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。

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、「カラム暗号化のためのデータベース・オプションの設定」(22 ページ) を参照してください。

定義

格納データの暗号化について説明する場合、次の用語を使用しています。

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データ変換のことです。「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 データを暗号化または復号化するために使用される数値です。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 が認定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 がサポートするアルゴリズムです。

暗号化カラムのデータ型

この項では、暗号化カラムでサポートされるデータ型とサポートされないデータ型を示し、暗号化カラムでの元のデータ型の保護について説明します。

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、次に示す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 のカラム暗号化でサポートされていません。

データ型の保護

Sybase IQ では、プレーン・テキストのデータを復号化するときに、AES_DECRYPT 関数のパラメータとしてデータ型が指定されている場合、または CAST 関数が AES_DECRYPT 関数の中に含まれる場合でも、元のデータ型が保護されます。Sybase IQ では、CAST による変換後のデータ型と、暗号化データの元のデータ型が比較されます。この 2 つのデータ型が一致しない場合は、元のデータ型と変換後のデータ型を示す詳細情報とともに -1001064 エラーが返されます。

たとえば、暗号化された VARCHAR(1) 値に対して、有効な次の復号化文を指定したとします。

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

同じデータを復号化するために、次の文を指定します。

```
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 またはデータ型パラメータがない場合、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消するために、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に CAST 関数を使用してデータ型を明示的に変換すると、データを復号化するときに CAST 関数を使用する際の問題を防止できます。

暗号化する対象のデータがカラムのデータである場合、または暗号化したデータが 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。

AES_ENCRYPT 関数 [文字列]

機能

指定された値を指定された暗号化キーを使用して暗号化し、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 暗号化するデータです。サポートされるデータ型については、「暗号化カラムのデータ型」(5 ページ) を参照してください。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* を暗号化するため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。

パスワードと同様に、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に常に必要です。

警告！ キーをなくさないでください。キーは安全な場所に格納してください。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、VARBINARY 値を返します。これは、入力された *string-expression* より長さが最大で 31 バイト増えた値です。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_ENCRYPT 関数を使用して暗号化された *string-expression* を復号化するには、AES_DECRYPT 関数を使用します。*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、データに対して文字セット変換が実行されないように、カラムのデータ型を VARBINARY または VARCHAR にし、32 バイト以上にする必要があります (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 または VARCHAR のカラムの長さが 32 バイトより小さい場合、AES_DECRYPT 関数はエラーを返します。

AES_ENCRYPT 関数の結果データ型には LONG VARBINARY を指定できます。SELECT INTO 文で AES_ENCRYPT を使用する場合は、ラージ・オブジェクト管理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。

追加の詳細と使用法については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「REPLACE 関数 [文字列]」を参照してください。

標準と互換性

- **SQL92** ベンダの拡張機能
- **SQL99** 主要な SQL に含まれない、SQL/foundation の機能
- **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]」(8 ページ)

「LOAD TABLE ENCRYPTED 句」(9 ページ)

例

AES_ENCRYPT 関数の使用例については、「暗号化と復号化の例」(12 ページ) を参照してください。

AES_DECRYPT 関数 [文字列]

機能

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

構文

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

パラメータ

string-expression 復号化する文字列。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* を復号化するため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。キーは安全な場所に格納してください。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 このオプションのパラメータでは、復号化された *string-expression* のデータ型を指定します。このデータ型は、元のプレーン・テキストのデータ型と同じ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 文を使用しない場合は、*data-type* として VARCHAR を渡すことにより、AES_DECRYPT 関数を使用して同じデータを表示できます。*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_ENCRYPT 関数を使用して暗号化された *string-expression* を復号化するには、AES_DECRYPT 関数を使用します。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。

標準と互換性

- **SQL92** ベンダの拡張機能
- **SQL99** ベンダの拡張機能
- **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]」(7 ページ)
- 「暗号化と復号化の例」(12 ページ)
- 「LOAD TABLE ENCRYPTED 句」(9 ページ)

例

次の例では、user_info テーブルからユーザのパスワードを復号化しています。

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

LOAD TABLE ENCRYPTED 句

LOAD TABLE 文は、カラム指定キーワードである ENCRYPTED をサポートしています。*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*

詳細については、「例」(11 ページ) を参照してください。

完全な構文については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の「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変換される入力ファイル・フィールドのデータ型です。サポートされるデータ型については、「暗号化カラムのデータ型」(5 ページ) を参照してください。*data-type* は、AES_DECRYPT 関数の出力のデータ型と同じデータ型にする必要があります。詳細については、「AES_DECRYPT 関数 [文字列]」(8 ページ) を参照してください。

key-string データを暗号化するため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。

パスワードと同様に、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に常に必要です。

警告！ キーをなくさないでください。キーは安全な場所に格納してください。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 データを暗号化するため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て行う必要があります。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 カラム指定によって、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を指定できます。オプションでアルゴリズムも指定できます。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。

参照

- 「AES_ENCRYPT 関数 [文字列]」(7 ページ)
- 「AES_DECRYPT 関数 [文字列]」(8 ページ)
- 「暗号化と復号化の例」(12 ページ)

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;
```

暗号化カラムの使用

この項では、暗号化カラムの使用方法について説明し、例をいくつか示します。

暗号化された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あり、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。ISO1LATIN1 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 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比較を実行するには、アプリケーションでそのカラムの値を標準的な形式に変更し、同一の値ではない等価な値が生成されないように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、カラムに挿入する前にアプリケーションですべての入力値に UCASE を適用する場合、等価な値はすべて同一の値となります。

暗号化と復号化の例

例 1

次の AES_ENCRYPT 関数と AES_DECRYPT 関数の使用例は、コメント付きの SQL で記述されています。

```
--    この aes_encrypt 関数と aes_decrypt 関数の使用例は、3 つの部分に分かれています。
--
-- 第 1 部：ターゲット・テーブルとユーザを DDL として事前に定義する
-- 第 2 部：暗号化の導入に伴うスキーマの変更の例
-- 第 3 部：ビューおよびストアド・プロシージャを使用した暗号化キーの保護
--


--    第 1 部：ターゲット・テーブルとユーザの定義

--    ここで例に示されている、PrivUser と NonPrivUser の 2 つのクラスのユーザは、異なる
--    権限を反映するグループに割り当てられているものとします。

--    初期状態は、暗号化の導入以前のスキーマを反映したものです。

--    開始時のコンテキストを次のように設定します。共通キーを使用する 2 つのテーブルがあります。
```

```
--    機密データが含まれているカラムがありますが、それ以外のカラムには含まれていません。
--    これらのテーブルの通常のジョイン・カラムは sensitiveA です。
--    キーとユニーク・インデックスがあります。

    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
                )  ;


--    プライマリ・キーを強制的に適用する暗黙的な HG (HighGroup) ユニーク・インデックスがあ
ります。

    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 ユーザは機密情報のカラムを表示できます。

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

--    既存のスキーマにおける TRUSTED ユーザ以外のユーザは、sensitiveB を表示してはならな
--    い場合でも、ジョインを行うために sensitiveA を表示する必要があります。

    grant select ( sensitiveA, publicC, publicD )
```

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

--   TRUSTED ユーザ以外のユーザは、次のようなクエリを実行できます。

   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' ) ;

--   および

   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' ;

--   次のクエリは TRUSTED ユーザのみが実行できます

   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' ) ;



--    第 2 部：暗号化に備えたスキーマの変更
--
--    DBA は次のように暗号化を導入します。
--
--    DBA は、適用可能なテーブルに対してスキーマを変更し、アクセス・パーミッションを調整して、
--    既存のデータを更新します。使用される暗号化キーは、それ以降の手順で非表示になります。


--    varbinary の暗号化テキストの長さに関する DataLength の比較（単位はバイト）：
--
--    プレーン・テキスト 暗号化テキスト    対応する数値精度
--
--             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

--    整数データ型である tinyint、small int、integer、bigint は、varbinary(32)
--    暗号化テキストです。

--    正確な関係は次のとおりです。
--    DATALENGTH(ciphertext) =
--    (((DATALENGTH(plaintext)+ 15) / 16) + 1) * 16

--    最初のテーブルには、DBA はプレーン・テキストと暗号化テキストの両方の形式を保持するよう
--    指定しています。これは、通常の指定方法ではありません。データベース・ファイルも暗号化する
--    場合にのみ行ってください。

--    NonPrivUser によるカラム sensitiveA へのアクセスを暗号化テキスト・バージョンへのア
--    クセスに切り替えます。

--    暗号化テキストのカラムにユニーク・インデックスを挿入します。暗号化テキスト自体にインデッ
--    クスが作成されます。

--    NonPrivUser は暗号化テキストを選択して使用できます。

--    PrivUser は、どちらの形式も選択でき、復号化にコストをかけずに済みます。

    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

--    この時点で、カラム sensitiveB を変更します。

    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

--   2 番目のテーブルには、DBA は暗号化テキストのみを保持するよう指定しています。
--   これは通常の指定方法であり、データベース・ファイルを暗号化する必要はありません。

   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 ;

--   キーを保護するために変更を行う前に、この時点で、次のタイプのクエリを使用できます。

--   TRUSTED ユーザ以外のユーザは、暗号化テキストに対して等価ジョインを実行できます。バイナ
--   リを選択することもできますが、解釈できません。


   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' ) ;

--   暗号化テキストのみにアクセスを制限すると、一般的な述部と式が除外されます。
--   次のクエリは意味のある結果を返しません。
--
--   注意：暗号化テキストを含んでいる varbinary に対して、次の 4 つの述部を使用できます。
--  = (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' ;

--   TRUSTED ユーザは、保持されたプレーン・テキストのカラムにアクセスできます。したがって、
--   TRUSTED ユーザは aes_decrypt を呼び出す必要がなく、キーは必要ありません。

   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' ;


--    第 3 部 : 暗号化キーの保護

--    この項では、プレーン・テキストへのアクセス・パーミッションを付与する一方で、キーを保護す
--    る方法を示します。

--    最初のテーブルには、DBA はプレーン・テキストのカラムを保持するよう指定しています。
--    したがって、次のビューには前述の trusted ユーザと同じ機能があります。
--    追加のアクセス制御のために group_member が使用されているものとします。

--    注意 : この例では、NonPrivUser はベース・テーブルで暗号化されたテキストにアクセスでき
--    ます。

   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 ;

--    2 番目のテーブルには、DBA はプレーン・テキストを保持していません。
--    したがって、ビュー内で aes_decrypt 呼び出しを使用する必要があります。
--    重要 : ALTER VIEW を使用してビューの定義を非表示にします。そうすることで、誰もキーを
--    検出できなくなります。

   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 ;

--  同様に、ロードに使用されるキーをストアド・プロシージャで保護できます。
--  ビューを非表示にする場合と同様に、プロシージャを非表示にすると、誰もキーを表示できなくな
--  ります。

    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 ;

--  次の構文を使用して、ロード・プロシージャを呼び出します。

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

--  次の項では、ユーザ定義関数 (UDF)、他のビュー、またはその両方を使用して暗号化キーを
--  保護する方法を比較しています。最初の選択肢と最後の選択肢で最高のパフォーマンスが実現
--  されます。

--  second_table は、前に定義したように保護されます。


--  選択肢 1:
--  この基本的なアプローチでは、ビュー全体へのアクセスを制限する方法を使用しています。

     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 ;


--  選択肢 2：
--  ユーザ定義関数 (UDF) に暗号化関数の呼び出しを挿入します。
--  UDF の定義を非表示にします。UDF のパーミッションを制限します。
--  その他のセキュリティとビジネス・ロジックを処理する UDF をビューで使用します。
--  注意：ビュー自体を非表示にする必要はありません。

    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 ;

--  選択肢 3：
--  ユーザ定義関数でキーの選択のみを分離します。
--  この関数を拡張すると、キーをいくつでも選択できます。
```

```
--    この UDF は、非表示にもなり、実行権限が制限されています。
--    注意：したがって、この UDF を使用するビューからキーの値が漏れることはありません。

    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 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 ;


--    選択肢 4：
--    考慮事項を 2 つのビューに分けて、ビジネス・ロジックからセキュリティ・ロジックを分離する
--    アプローチをおすすめします。
--    この場合、セキュリティ・ロジックのビューのみを非表示にする必要があります。
--    注意：このアプローチのパフォーマンスは、最初の選択肢のパフォーマンスとほぼ同じです。

    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 ;
```

```
-- 暗号化の使用例の終わり
```

例 2

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、次の文を実行したとします。

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

AES_ENCRYPT(c1, 'key') と AES_ENCRYPT(c2,'key') の値は同一ではなく、AES_ENCRYPT(c1,'key') と AES_ENCRYPT(100,'key') の値は同一ではありません。

この問題を解決するには、AES_ENCRYPT の入力を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');
```

カラム暗号化のためのデータベース・オプションの設定

Sybase IQ の特定のデータベース・オプション設定は、カラムの暗号化と復号化に影響を与えます。AES_ENCRYPT 関数または AES_DECRYPT 関数を使用する前に、この項で説明するオプションを確認してください。ほとんどのカラム暗号化処理に対して、デフォルト設定は最適な設定ではありません。

暗号化テキスト・データの間違いによるトランケートの防止

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列やバイナリ文字列) が誤って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場合は、エンジンで必ずエラーが発生します。これは ANSI/ISO SQL92 の動作であり、推奨される方法です。

明示的なトランケーションが必要な場合は、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ションが実行されます。

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し、復号化した後のデータ長と指定されたキーの正しさの両方を検証するために、出力テキストをチェックします (データ型を持つ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

暗号化テキストの整合性を維持するには、次のデータベース・オプションを設定します。

```
SET OPTION 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 データベース・オプションは、さまざまな操作においてバイナリ・データ型 (BINARY、VARBINARY、LONG BINARY) と非バイナリ・データ型 (BIT、TINYINT、SMALLINT、INT、UNSIGNED INT、BIGINT、UNSIGNED BIGINT、CHAR、VARCHAR、LONG VARCHAR) の間で行われる暗黙の変換を制限します。CONVERSION_MODE を使用して、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できます。

```
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、およびクエリでのバイナリ・データ型から非バイナリ・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 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 は、12.7 より前のバージョンの Sybase IQ でのバイナリ・データ型の暗黙的な変換動作を維持します。

詳細については、『リファレンス：文とオプション』の「第 2 章 データベース・オプション」の「CONVERSION_MODE オプション」を参照してください。