12.13 Encryption and Compression Functions

Table 12.17 Encryption Functions

Name	Description
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`COMPRESS()`	Return result as a binary string
`DECODE()`	Decodes a string encrypted using ENCODE()
`DES_DECRYPT()`	Decrypt a string
`DES_ENCRYPT()`	Encrypt a string
`ENCODE()`	Encode a string
`ENCRYPT()`	Encrypt a string
`MD5()`	Calculate MD5 checksum
`OLD_PASSWORD()`	Return the value of the pre-4.1 implementation of PASSWORD
`PASSWORD()`	Calculate and return a password string
`SHA1()`, `SHA()`	Calculate an SHA-1 160-bit checksum
`SHA2()`	Calculate an SHA-2 checksum
`UNCOMPRESS()`	Uncompress a string compressed
`UNCOMPRESSED_LENGTH()`	Return the length of a string before compression

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a `VARBINARY` or `BLOB`binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (`CHAR`, `VARCHAR`, `TEXT`).

Some encryption functions return strings of ASCII characters: `MD5()`, `OLD_PASSWORD()`, `PASSWORD()`,`SHA()`, `SHA1()`. As of MySQL 5.5.3, their return value is a nonbinary string that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. Before 5.5.3, these functions return binary strings. The same change was made for `SHA2()` in MySQL 5.5.6.

For versions in which functions such as `MD5()` or `SHA1()` return a string of hex digits as a binary string, the return value cannot be converted to uppercase or compared in case-insensitive fashion as is. You must convert the value to a nonbinary string. See the discussion of binary string conversion inSection 12.10, “Cast Functions and Operators”.

If an application stores values from a function such as `MD5()` or `SHA1()` that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using `UNHEX()` and storing the result in a `BINARY(N)` column. Each pair of hex digits requires one byte in binary form, so the value of ***N*** depends on the length of the hex string. ***N*** is 16 for an `MD5()` value and 20 for a `SHA1()` value. For `SHA2()`, ***N*** ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a `CHAR` column is at least two times, up to eight times if the value is stored in a column that uses the `utf8` character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores `MD5()` string values in a `CHAR(32)` column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use `UNHEX()` and `BINARY(16)` instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

> **Note**
> Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead, such as `SHA2()`.

> **Caution**
> Passwords or other sensitive values supplied as arguments to encryption functions are sent in cleartext to the MySQL server unless an SSL connection is used. Also, such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(`*`crypt_str`*`,`*`key_str`*`)`

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()`.

- `AES_ENCRYPT(`*`str`*`,`*`key_str`*`)`

  `AES_ENCRYPT()` and `AES_DECRYPT()` implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. These functions implement AES with a 128-bit key length, but you can extend them to 256 bits by modifying the source. The key length is a trade off between performance and security.

`AES_ENCRYPT()` encrypts the string ***`str`*** using the key string ***`key_str`*** and returns a binary string containing the encrypted output. **`AES_DECRYPT()`** decrypts the encrypted string ***`crypt_str`*** using the key string ***`key_str`*** and returns the original cleartext string. If either function argument is **`NULL`**, the function returns **`NULL`**.

The ***`str`*** and ***`crypt_str`*** arguments can be any length, and padding is automatically added to ***`str`***so it is a multiple of a block as required by block-based algorithms such as AES. This padding is automatically removed by the **`AES_DECRYPT()`** function. The length of ***`crypt_str`*** can be calculated using this formula:

```
16 * (trunc(string_length / 16) + 1)
```

For a key length of 128 bits, the most secure way to pass a key to the ***`key_str`*** argument is to create a truly random 128-bit value and pass it as a binary value. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
```

A passphrase can be used to generate an AES key by hashing the passphrase. For example:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text', SHA2('My secret passphrase',512)));
```

Do not pass a password or passphrase directly to ***`crypt_str`***, hash it first. Previous versions of this documentation suggested the former approach, but it is no longer recommended as the examples shown here are more secure.

If **`AES_DECRYPT()`** detects invalid data or incorrect padding, it returns **`NULL`**. However, it is possible for **`AES_DECRYPT()`** to return a non-**`NULL`** value (possibly garbage) if the input data or the key is invalid.

- **`COMPRESS(`*`string_to_compress`*`)`**

Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always **`NULL`**. The compressed string can be uncompressed with **`UNCOMPRESS()`**.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.

  - Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra "`.`" character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (However, use of nonbinary string data types such as `CHAR` or `VARCHAR` to store compressed strings is not recommended anyway because character set conversion may occur. Use a`VARBINARY` or `BLOB` binary string column instead.)

- `DECODE(`***`crypt_str`***`,`***`pass_str`***`)`

  Decrypts the encrypted string ***`crypt_str`*** using ***`pass_str`*** as the password. ***`crypt_str`*** should be a string returned from `ENCODE()`.

- `DES_DECRYPT(`***`crypt_str`***`[,`***`key_str`***`])`

  Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

  This function works only if MySQL has been configured with SSL support. See Section 6.3.9, "Using SSL for Secure Connections".

  If no ***`key_str`*** argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER`privilege. The key file can be specified with the `--des-key-file` server option.

  If you pass this function a ***`key_str`*** argument, that string is used as the key for decrypting the message.

  If the ***`crypt_str`*** argument does not appear to be an encrypted string, MySQL returns the given***`crypt_str`***.

- `DES_ENCRYPT(`***`str`***`[,{`***`key_num`***`|`***`key_str`***`}])`

  Encrypts the string with the given key using the Triple-DES algorithm.

  This function works only if MySQL has been configured with SSL support. See Section 6.3.9, "Using SSL for Secure Connections".

  The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given. With no argument, the first key from the DES key file is used. With a ***`key_num`*** argument, the given key number (0 to 9) from the DES key file is used. With a ***`key_str`*** argument, the given key string is used to encrypt ***`str`***.

  The key file can be specified with the `--des-key-file` server option.

  The return string is a binary string where the first character is `CHAR(128 | `***`key_num`***`)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, ***key_num*** is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each ***key_num*** value must be a number in the range from `0` to `9`. Lines in the file may be in any order. ***des_key_str*** is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE`statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
     > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCODE(`***`str`***`,`***`pass_str`***`)`

Encrypt ***str*** using ***pass_str*** as the password. The result is a binary string of the same length as***str***. To decrypt the result, use `DECODE()`.

The `ENCODE()` function should no longer be used. If you still need to use `ENCODE()`, a salt value must be used with it to reduce risk. For example:

```
ENCODE('cleartext', CONCAT('my_random_salt','my_secret_password'))
```

A new random salt value must be used whenever a password is updated.

- `ENCRYPT(`***`str`***`[,`***`salt`***`])`

Encrypts ***str*** using the Unix `crypt()` system call and returns a binary string. The ***salt*** argument must be a string with at least two characters or the result will be `NULL`. If no ***salt*** argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

`ENCRYPT()` ignores all but the first eight characters of ***str***, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

The use of **`ENCRYPT()`** with the **`ucs2`**, **`utf16`**, or **`utf32`** multibyte character sets is not recommended because the system call expects a string terminated by a zero byte.

If `crypt()` is not available on your system (as is the case with Windows), **`ENCRYPT()`** always returns**`NULL`**.

- **`MD5(`*`str`*`)`**

Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hex digits, or **`NULL`** if the argument was **`NULL`**. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

See the note regarding the MD5 algorithm at the beginning this section.

- **`OLD_PASSWORD(`*`str`*`)`**

`OLD_PASSWORD()` was added when the implementation of **`PASSWORD()`** was changed in MySQL 4.1 to improve security. **`OLD_PASSWORD()`** returns the value of the pre-4.1 implementation of **`PASSWORD()`**as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 5.5 MySQL server without locking them out. See Section 6.1.2.4, “Password Hashing in MySQL”.

As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string.

> **Note**
> Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided.

- **`PASSWORD(`*`str`*`)`**

Returns a hashed password string calculated from the cleartext password ***`str`***. The return value is a nonbinary string in the connection character set (a binary string before MySQL 5.5.3), or **`NULL`** if the argument is **`NULL`**. This function is the SQL interface to the algorithm used by the server to encrypt

MySQL passwords for storage in the `mysql.user` grant table.

The `old_passwords` system variable controls the password hashing method used by the`PASSWORD()` function. It also influences password hashing performed by `CREATE USER` and `GRANT`statements that specify a password using an `IDENTIFIED BY` clause.

The following table shows the permitted values of `old_passwords`, the password hashing method for each value, and which authentication plugins use passwords hashed with each method.

Value	Password Hashing Method	Associated Authentication Plugin
`0` or `OFF`	MySQL 4.1 native hashing	`mysql_native_password`
`1` or `ON`	Pre-4.1 (“old”) hashing	`mysql_old_password`

If `old_passwords=1`, `PASSWORD(str)` returns the same value as `OLD_PASSWORD(str)`. The latter function is not affected by the value of `old_passwords`.

```
mysql> SET old_passwords = 0;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------------------------------+------------------------+
| PASSWORD('mypass')                        | OLD_PASSWORD('mypass') |
+-------------------------------------------+------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 | 6f8c114b58f2ce9e       |
+-------------------------------------------+------------------------+

mysql> SET old_passwords = 1;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+--------------------+------------------------+
| PASSWORD('mypass') | OLD_PASSWORD('mypass') |
+--------------------+------------------------+
| 6f8c114b58f2ce9e   | 6f8c114b58f2ce9e       |
+--------------------+------------------------+
```

Encryption performed by `PASSWORD()` is one-way (not reversible). It is not the same type of encryption as used for Unix passwords; for that, use `ENCRYPT()`.

> **Note**
> `PASSWORD()` is used by the authentication system in MySQL Server; you should *not*use it in your own applications. For that purpose, consider `MD5()` or `SHA2()` instead. Also see RFC 2195, section 2 (Challenge-Response Authentication Mechanism (CRAM)), for more information about handling passwords and authentication securely in your applications.

> **Note**
> Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided.

Caution

Statements that invoke `PASSWORD()` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about password logging in the server logs, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, seeSection 4.5.1.3, “mysql Logging”.

- `SHA1(`*`str`*`)`, `SHA(`*`str`*`)`

  Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hex digits, or `NULL` if the argument was `NULL`. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. You can also use `SHA1()` as a cryptographic function for storing passwords. `SHA()` is synonymous with `SHA1()`.

  As of MySQL 5.5.3, the return value is a nonbinary string in the connection character set. Before 5.5.3, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

  ```
  mysql> SELECT SHA1('abc');
          -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
  ```

  `SHA1()` can be considered a cryptographically more secure equivalent of `MD5()`. However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- `SHA2(`*`str`*`, `*`hash_length`*`)`

  Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the cleartext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is `NULL` or the hash length is not one of the permitted values, the return value is`NULL`. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

  As of MySQL 5.5.6, the return value is a nonbinary string in the connection character set. Before 5.5.6, the return value is a binary string; see the notes at the beginning of this section about using the value as a nonbinary string.

  ```
  mysql> SELECT SHA2('abc', 224);
          -> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
  ```

  This function works only if MySQL has been configured with SSL support. See Section 6.3.9, “Using SSL for Secure Connections”.

`SHA2()` can be considered cryptographically more secure than `MD5()` or `SHA1()`.

`SHA2()` was added in MySQL 5.5.5.

- `UNCOMPRESS(`*`string_to_uncompress`*`)`

  Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

  ```
  mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
          -> 'any string'
  mysql> SELECT UNCOMPRESS('any string');
          -> NULL
  ```

- `UNCOMPRESSED_LENGTH(`*`compressed_string`*`)`

  Returns the length that the compressed string had before being compressed.

  ```
  mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
          -> 30
  ```