12.17.2 Enterprise Encryption Usage and Examples

To use Enterprise Encryption in applications, invoke the functions that are appropriate for the operations you wish to perform. This section demonstrates how to carry out some representative tasks.

Task: Create a private/public key pair using RSA encryption.

```
-- Encryption algorithm; can be 'DSA' or 'DH' instead
SET @algo = 'RSA';
-- Minimum key length in bits; make larger for stronger keys
SET @key_len = 1024;

-- Create private key
SET @priv = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @key_len);
-- Derive corresponding public key from private key, using same algorithm
SET @pub = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv);
```

Now you can use the key pair to encrypt and decrypt data, sign and verify data, or generate symmetric keys.

Task: Use the private key to encrypt data and the public key to decrypt it. This requires that the members of the key pair be RSA keys.

```
SET @ciphertext = ASYMMETRIC_ENCRYPT(@algo, 'My secret text', @priv);
SET @cleartext = ASYMMETRIC_DECRYPT(@algo, @ciphertext, @pub);
```

Conversely, you can encrypt using the public key and decrypt using the private key.

```
SET @ciphertext = ASYMMETRIC_ENCRYPT(@algo, 'My secret text', @pub);
SET @cleartext = ASYMMETRIC_DECRYPT(@algo, @ciphertext, @priv);
```

In either case, the algorithm specified for the encryption and decryption functions must match that used to generate the keys.

Task: Generate a digest from a string.

```
-- Digest type; can be 'SHA256', 'SHA384', or 'SHA512' instead
SET @dig_type = 'SHA224';

-- Generate digest string
SET @dig = CREATE_DIGEST(@dig_type, 'My text to digest');
```

Task: Use the digest with a key pair to sign data, then verify that the signature matches the digest.

```
-- Encryption algorithm; could be 'DSA' instead; keys must
-- have been created using same algorithm
SET @algo = 'RSA';

-- Generate signature for digest and verify signature against digest
SET @sig = ASYMMETRIC_SIGN(@algo, @dig, @priv, @dig_type);
-- Verify signature against digest
```

```
SET @verf = ASYMMETRIC_VERIFY(@algo, @dig, @sig, @pub, @dig_type);
```

Task: Create a symmetric key. This requires DH private/public keys as inputs, created using a shared symmetric secret. Create the secret by passing the key length to **CREATE_DH_PARAMETERS()**, then pass the secret as the “key length” to **CREATE_ASYMMETRIC_PRIV_KEY()**.

```
-- Generate DH shared symmetric secret
SET @dhp = CREATE_DH_PARAMETERS(1024);
-- Generate DH key pairs
SET @algo = 'DH';
SET @priv1 = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @dhp);
SET @pub1 = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv1);
SET @priv2 = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @dhp);
SET @pub2 = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv2);

-- Generate symmetric key using public key of first party,
-- private key of second party
SET @sym1 = ASYMMETRIC_DERIVE(@pub1, @priv2);

-- Or use public key of second party, private key of first party
SET @sym2 = ASYMMETRIC_DERIVE(@pub2, @priv1);
```

Key string values can be created at runtime and stored into a variable or table using **SET**, **SELECT**, or **INSERT**:

```
SET @priv1 = CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024);
SELECT CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024) INTO @priv2;
INSERT INTO t (key_col) VALUES(CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024));
```

Key string values stored in files can be read using the **LOAD_FILE()** function by users who have the **FILE** privilege.

Digest and signature strings can be handled similarly.