Double layer Encryption

Content

Introduction

This function allow merchants to use a double layer encryption :

- The whole traffic is encrypted through the SSL tunnel ;
- The card object is encrypted inside the message.

Please contact our sales team to access this feature.

Encryption step

The processing takes place in 3 steps:

1. The merchant requests parameters to generate the encryption key.
2. The merchant encrypts the sensitive data on his server.
3. The merchant calls the Payline web services with the encrypted data.

How to integrate

To start this step, you must have a merchant and a merchant access key.

You must integrate Payline web services and know RSA data encryption :

- getEncryptionKey : allows you to retrieve the encryption settings to encrypt your message.

Step 1 : Call the `getEncryptionKey` to obtain the key.

Merchant calls the getEncryptionKey service on the usual endpoint.

This service can be called multiple times if needed (for instance multiple place to store the public key).
The parameters are the version and merchantKeyName. It must be set to 32.

You must integrate the Payline web services:

- The merchant performs a getEncryptionKey : retrieves the encryption parameters.
- The merchant retrieves the encryption parameters and the key.keyId from getEncryptionKey.

These data must be stored by the merchant in order to encrypt further messages.

If the merchantKeyName don't respect the format, you will receive the following error :
02204 - ERROR

Process for the call response with multiple keys (version 32)

If keypair named with the "merchantKeyName" filled in, **does not** exist **/OR/** keypair named with the "merchantKeyName" filled in exists **and** dateKeyPair > 60 :

- Generate new key pair named "merchantKeyName" and return a new public key

If keypair named with the "merchantKeyName" filled in exists and dateKeyPair **< 60 :**

- Return current public key

This service returns the key details:

- RSA public key details (alogorithm, size, exponent, ...);
- Key expiration date;
- Key ID.
- merchantKeyName;

These data must be stored by the merchant in order to encrypt further messages.

Step 2 : Encrypt the card data with the public key

Having data from step 1, merchant can instantiate a key using details provided by the getEncryptionKey service.
The merchant can then encrypt the message with the sensitive data.

The encryption function must :

- generate a public key with the parameters retrieved from the getEncryptionKeyReply : algo, modulus, exponent ;
- build the public key with the parameters: modulus and publicExponent ;
- build the Cipher is returned by the getEncryptionKeyReponse service ;
- encrypt the message with the following parameters: the message formatted with the sensitive data, the Cipher and the PublicKey.

```
cardDataToEncrypt = "CardNumber=497010000000006,
ExpDate=0220,CVX=123,OwnerBirthDate=07071977,
Password=Payline,Cardholder=John Doe"
```

- encode the message in base64.

If any data is not available, then it shall be absent in the string. For instance, if only the card and the expiration date are available :

```
cardDataToEncrypt = "CardNumber=497010000000006,
ExpDate=0220"
```

Then merchant can encrypt the cardDataToEncrypt using the publicKey. Before sending this data in any payment or authentication service, it has to be converted in base 64.

```
encryptedData = BASE64.encodeBase64(RSA.encrypt
(publicKey,cardDataToEncrypt))
```

Step 3 : Calls the Payline web services with the encrypted data.

Merchant can send `encryptedData` in web services message instead of clear data.
One very important element is the *key ID* : Monext will uncrypt data with the private key associated to this *key ID*.
Check the getEncryptionKey service to list the services using the encrypted message.

Here is an exemple of the `card` object in messages with the `encryptedData`.

Clear card data

Example encryption function

```
<ns2:card>
    <number>497993XXXXXX9978</number>
    <type>VISA</type>
    <expirationDate>1019</expirationDate>
    <cvx xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
    <ownerBirthdayDate xsi:nil="true" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"/>
    <password xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
    <cardPresent xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
  </ns2:card>
```

Encrypted card data

Example encryption function

```
<ns9:card>
    <ns8:encryptionKeyId>1012</ns8:encryptionKeyId>
    <ns8:
encryptedData>FXcVGLah8BVJ9yKMjOqU0sQ5qd2iGCkXjeBrJqb5fsM4rTdUyUb
uJsZZnmCHfpFbrb0haTkCokQ3DFdvpIwx2
/QavOjUUni17RHTpmik4HYaOx+uWfJYU2H2er37Wd9zHgY3DdRDe7lo4i4xOx1TLu
DexvEyNqpoSRru/+iklaidDjV74Iex2KESoJLu29zVCnoMICiYLoLR
/WpU3UBowsiBj5y0BL8UhIpn8sSS9Rw
/5Jq7IPp7wCFdNXztXq3GXByHQM9h0iayKZ1uYLQXwy3ilSLLgmAJwXTG1jin3gbn
iEl4KyfhbzBnFSMU5XRdZyY02+yLaKPU0pQnLxrhdw==</ns8:encryptedData>
    <ns8:number xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
    <ns8:type>VISA</ns8:type>
    <ns8:expirationDate xsi:nil="true" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"/>
    <ns8:cvx xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
    <ns8:ownerBirthdayDate xsi:nil="true" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"/>
    <ns8:password xsi:nil="true" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"/>
    <ns8:cardPresent xsi:nil="true" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"/>
    <ns8:cardholder xsi:nil="true" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"/>
    <ns8:token xsi:nil="true" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"/>
    <ns8:paymentData xsi:nil="true" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"/>
  </ns9:card>
```

List of Key/Value accepted

The following keys are accepted in the encrypted data:

CardNumber	card.number	497010000000006
ExpDate	card.expirationDate	0220
CVX	card.cvx	123

OwnerBirthDate	card.ownerBirthdayDate	31121980
Password	card.password	Payline01$
Cardholder	card.cardholder	Jeremy Mattio

Key renewal

A key is valid for 90 days. A new key will be issued 30 days before the previous key expiration. During this period both keys are valid and usable.
A merchant has 30 days to change the key in its systems before the old key become unusable.
A good practice is to call the `getEncryptionKey` everyday, and to start the renew process as soon as a new key ID is received by the merchant

In order to prevent misuse and not to saturate the database, the system refuses a generation of a new key, if there are more than 100 active keys simultaneously.
So, at each key draw, the system looks for the number of active key pairs (whose life date is <= 90 days) by taking the unique name of the key (disctinct merchantKeyName) .

If a merchant has more than 100 active keys at the same time, the system rejects a new key issue with code 02203 - ERROR.

Security

The key is unique per merchant.

The current key specification are :

- Algorithm : RSA
- Key size : 2048
- Cipher : RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Code example

Encryption function code example used by merchant to encrypt the message.

Example encryption function

```
public void getRSAEncodedMessage() throws Exception{
    String message = "CardNumber=497010000000006,ExpDate=0220,
CVX=123,OwnerBirthDate=,Password=Payline1,Cardholder=Alain
Durand";
    String cipherName = "RSA/ECB/OAEPWithSHA-256AndMGF1Padding";
    Cipher cipher = Cipher.getInstance(cipherName);
    String algo = "RSA";
    String modulus =
"AOLndIya3+ViAuP07V1Ky+YRuZK6zsMUsVIPP3xhFCS1CSJb9oBLmmzkMMASAn0T
Q7BrELuNbY1+9VT30ah1N0mX0BzIBqE5sojV+CkafF+LtIntpF1wUhguXlcFOXcBL
CECiMA4gBqHJMrVxnF626M1S6Wi2++WUwFgPTtHdn4B7e0RMvWqFr6uKBDq1qhdP8
iziiAn
/YPPp5ObgxuabWvPAZvRMDmgflNLYCOZB05LjZXMDvkLaYy244iY0tmVWwa7WYi2l
J7N0wK6gpDXD1WQh42AQuyVQX9i
/m3oOjH8iqdRZG8FuaoHkBtON9zhz+ZyuBAwb+zNZWBDvnp6V0E=";
    String publicExponent = "AQAB";

    final KeyFactory fact = KeyFactory.getInstance(algo);
    PublicKey publicKey =  fact.generatePublic(new
RSAPublicKeySpec(new BigInteger(Base64.decodeBase64(modulus.
getBytes())), new BigInteger(Base64.decodeBase64(publicExponent.
getBytes()))));
    System.out.println("Message encoded : " + Base64.
encodeBase64String(encrypt(cipher, publicKey, message)));
}


public static byte[] encrypt(Cipher cipher, Key key, String
message) throws Exception {
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return cipher.doFinal(message.getBytes());
}
```