Demande d'une clef de chiffrement

Contenu

- 3D Secure
- 3D Secure - Personnaliser le nom du marchand
- Actualisation automatique des cartes
- Bascule à la source - Tests d'intégration sur les appels API Webservices
- Choix de la marque
- Demande d'une clef de chiffrement
- Déliassage
- Fonctionnalités avancées
- La gestion des codes retour
- Marque blanche

Le principe

Vous pouvez demander des paramètres de chiffrement à Payline afin d'encoder vos données sensibles pour vos appels web services.

Veuillez contacter notre équipe commerciale pour accéder à cette fonctionnalité.

Les étapes de chiffrement

Le traitement se déroule en 3 étapes :

1. Le marchand demande des paramètres pour générer la clef de chiffrement.
2. Le marchand encrypte les données sensibles sur son serveur.
3. Le marchand appelle les web services de Payline avec les données encryptées.

Comment intégrer ces paramètres de chiffrement

Pour commencer cette étape, vous devez avoir un commerçant et une clef d'accès marchand.

Vous devez intégrer les web services Payline et connaitre le chiffrement de données RSA :

- getEncryptionKey : permet de récupérer les paramètres de chiffrement pour chiffrer votre message.

Étape 1 : Appeler le getEncryptionKey pour obtenir la clé.

Le marchand appelle un getEncryptionKey : permet de récupérer les paramètres de chiffrement.
Le marchand récupère les paramètres de chiffrement dans l'object key du service getEncryptionKeyReponse

Ce service peut être appelé plusieurs fois si nécessaire (par exemple, plusieurs endroits pour stocker la clé publique).
Les paramètres sont version et merchantKeyName. La version du web service doit être défini à 32.

Vous devez intégrer les services Web de Payline :

- Le commerçant effectue un getEncryptionKey : récupère les paramètres de cryptage.
- Le commerçant récupère les paramètres de cryptage dans l'object key à partir de getEncryptionKey.

Ces données doivent être stockées par le commerçant afin de pouvoir crypter d'autres messages.

Si le merchantKeyName ne respecte pas le format, vous recevrez l'erreur avec le code 02204 - ERROR.

Processus pour la réponse à l'appel avec des clés multiples (version 32)

Si la paire de clés nommée avec le "merchantKeyName" renseigné, n'existe pas /OU/ la paire de clés nommée avec le "merchantKeyName" renseigné existe et dateKeyPair > 60 :

- Générer une nouvelle paire de clés nommée "merchantKeyName" et retourner une nouvelle clé publique.

Si la paire de clés nommée avec le "merchantKeyName" renseigné existe et dateKeyPair < 60 :

- Retourner la clé publique actuelle

Ce service retourne les détails de la clé :

- Détails de la clé publique RSA (alogorithme, taille, exposant, ...) ;
- Date d'expiration de la clé ;
- ID de la clé.
- merchantKeyName.

Ces données doivent être stockées par le commerçant afin de pouvoir crypter d'autres messages.

Étape 2 : Cryptage des données de la carte avec la clé publique

Avec les données de l'étape 1, le commerçant peut instancier une clé en utilisant les détails fournis par le service getEncryptionKey.
Le commerçant peut alors chiffrer le message avec les données sensibles.

La fonction de cryptage doit :

- générer une clé publique avec les paramètres récupérés dans le getEncryptionKeyReponse : algo, module, exposant ;
- construire la clé publique avec les paramètres : modulus et publicExponent ;
- construire le Cipher retourné par le service getEncryptionKeyReponse ;
- crypter le message avec les paramètres suivants : le message formaté avec les données sensibles, le Cipher et la PublicKey.

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

- coder le message en base64.

Si une donnée n'est pas disponible, elle doit être absente de la chaîne. Par exemple, si seules la carte et la date d'expiration sont disponibles :

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

Le commerçant peut alors chiffrer les données de la carte à l'aide de la clé publique. Avant d'envoyer ces données dans un service de paiement ou d'authentification, elles doivent être converties en base 64.

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

Étape 3 : Appel des services web de Payline avec les données cryptées.

Le marchand peut alors appeler les web services Payline en transmettant le message encrypté dans la balise encryptionData et la clef ID du service getEncryptionKeyReponse d ans la balise encryptionKeyId.
Consulter le wbs getEncryptionKey pour lister les wbs utilisant le message encrypté.

Voici un exemple de l'objet carte dans les messages avec le code encryptedData :

Données de la carte en clair

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>
```


Liste des clés/valeurs acceptées

Les clés suivantes sont acceptées dans les données cryptées :

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

Renouvellement de la clé

Une clé est valable pendant 90 jours. Une nouvelle clé sera émise 30 jours avant l'expiration de la clé précédente. Pendant cette période, les deux clés sont valables et utilisables.
Un commerçant a 30 jours pour changer la clé dans ses systèmes avant que l'ancienne clé ne devienne inutilisable.
Une bonne pratique consiste à appeler la fonction getEncryptionKey tous les jours et à lancer le processus de renouvellement dès qu'un nouvel identifiant de clé est reçu par le commerçant.

Afin d'éviter les abus et de ne pas saturer la base de données, le système refuse la génération d'une nouvelle clé s'il y a plus de 100 clés actives simultanément.
Ainsi, à chaque tirage de clé, le système recherche le nombre de paires de clés actives (dont la date de vie est <= 90 jours) en prenant le nom unique de la clé (disctinct merchantKeyName)
Si le nombre actuel de clés est supérieur ou égal à 100, alors le système renvoie l'erreur suivante et le tirage de clé n'est pas effectué : 02203 - ERROR.

Sécurité

La clé est unique par marchand.

Les spécifications actuelles de la clé sont :

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

Exemple de code

Exemple de code de fonction de cryptage utilisé par le commerçant pour crypter le message.

Example d'une fonction d'encryption

```
   public void getRSAEncodedMessage() throws Exception{
       String message = "CardNumber=497010000000006,ExpDate=0220,
CVX=123,OwnerBirthDate=,Password=Payline,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());
   }
```

Pages associées

- Demande d'une clef de chiffrement
- Webservice - getEncryptionKeyRequest