🔐 Hybrid Encryption Function Documentation (PHP Version)

**🧭 Purpose :-**

The hybridEncrypt function implements Hybrid Encryption to securely transmit sensitive payloads (e.g., merchant identifiers). It combines:

* AES-256-CBC: For encrypting the actual data (fast, symmetric encryption).
* RSA Private Key: For encrypting the AES key (secure key exchange).

This ensures that even if the payload is intercepted, it remains unreadable without the corresponding decryption keys.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| mid | string | The plain-text data or merchant ID to encrypt. |
| privateKey | string | The RSA **private key** in **PEM format**. Should be passed as a string with \\n-escaped newlines. |

**⚙️ How It Works**

**1. 🔧 Private Key Formatting :-**

$privateKeyPem = str\_replace("\\n", "\n", $privateKeyPem);

* Converts escaped \\n into actual newline characters.
* Ensures the PEM format is valid when loading the key.

**2. 🔑 Load RSA Private Key:-**

$privateKey = openssl\_pkey\_get\_private($privateKeyPem);

* Loads the private key from the formatted PEM string.
* Throws an error if the key is invalid.

**3. 🔑 Generate AES Key and IV:-**

$aesKey = openssl\_random\_pseudo\_bytes(32); // 256-bit AES key

$iv = openssl\_random\_pseudo\_bytes(16); // 128-bit Initialization Vector

* AES-256 requires a 32-byte key.
* IV adds randomness for security (used once per encryption).

**4. 🔐 Encrypt Data with AES :-**

$encryptedData = openssl\_encrypt($data, 'aes-256-cbc', $aesKey, OPENSSL\_RAW\_DATA, $iv);

* The plain text ($data) is encrypted using AES-256-CBC.
* The result is binary, so it's Base64-encoded in the final response.

**5. 🔐 Encrypt AES Key with RSA Private Key :-**

openssl\_private\_encrypt($aesKey, $encryptedAesKey, $privateKey, OPENSSL\_PKCS1\_PADDING);

* The randomly generated AES key is encrypted using the RSA private key.
* Uses PKCS#1 padding for RSA encryption.
* ⚠️ *Note*: Normally, RSA public keys are used to encrypt and private keys to decrypt. Ensure this reversed use case is appropriate for your design.

**6. 📦 Return Final Payload :-**

$payload = [

'encryptedData' => base64\_encode($encryptedData),

'encryptedKey' => base64\_encode($encryptedAesKey),

'iv' => base64\_encode($iv),

];

return base64\_encode(json\_encode($payload));

* The AES-encrypted data, RSA-encrypted AES key, and IV are Base64-encoded.
* The final output is a Base64 string of a JSON-encoded bundle.

**📤 Sample Output (Decoded from Base64):-**

{

"encryptedData": "kjoz0dfEw8+U5kQ9vw==",

"encryptedKey": "PK+L8sdf8sdk==",

"iv": "bWF5QmVlU2FmZQ=="

}

**The actual return will be Base64-encoded again:-**

eyJlbmNyeXB0ZWREYXRhIjoi...== (Base64 string of JSON)

**⚠️ Error Handling :-**

If any step fails, such as:

* Invalid PEM private key
* AES encryption failure
* RSA encryption failure

Then:

error\_log("Encryption Error: " . $e->getMessage());

return false;

**✅ Final PHP Function :-**

**function hybridEncrypt($data, $privateKeyPem) {  
try {  
// Replace escaped \n with actual newlines  
$privateKeyPem = str\_replace("\\n", "\n", $privateKeyPem);  
// Load the private key  
$privateKey = openssl\_pkey\_get\_private($privateKeyPem);  
if (!$privateKey) {  
throw new Exception("Invalid private key.");  
}  
// Generate AES-256 key and IV  
$aesKey = openssl\_random\_pseudo\_bytes(32); // 256 bits  
$iv = openssl\_random\_pseudo\_bytes(16); // 128 bits for AES-CBC  
// Encrypt data with AES  
$encryptedData = openssl\_encrypt($data, 'aes-256-cbc', $aesKey, OPENSSL\_RAW\_DATA, $iv);  
if ($encryptedData === false) {  
throw new Exception("AES encryption failed.");  
}  
// Encrypt AES key with RSA private key  
if (!openssl\_private\_encrypt($aesKey, $encryptedAesKey, $privateKey, OPENSSL\_PKCS1\_PADDING)) {  
throw new Exception("RSA private encryption failed.");  
}  
// Create payload  
$payload = [  
'encryptedData' => base64\_encode($encryptedData),  
'encryptedKey' => base64\_encode($encryptedAesKey),  
'iv' => base64\_encode($iv),  
];  
// Encode payload to JSON and then Base64  
$base64EncodedPayload = base64\_encode(json\_encode($payload));  
return $base64EncodedPayload;  
} catch (Exception $e) {  
error\_log("Encryption Error: " . $e->getMessage());  
return false;  
}  
}**

**$data = "62"; // pass your mid here for encrypt**

**$privateKey = `-----BEGIN PRIVATE KEY-----  
MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBAIgRQrJJa9JuoPJ0  
ybsM6HPfKPtZWei587etuusZqvIDye3vB5z1qGdoT3sKj0CT0R7YAgZhfwWRpM4A  
9sMTMRqJueME0qgziwNk9ueYXb5uk0FjI3liRhzLxjiBOjbFL7MCXwt8V38Wv6jd  
gPzWcjDkli0SiSsZ5vEGR4rfu34BAgMBAAECgYAdcL6hnMk7irxAOYy8cBqLl6li  
WP7fH/zfrTNYUwFtw+wZEzLufyik1yj7UlmUe2cv6UTm5Il7RvwFpqe9pZVl+oZB  
pDOm5ON9KQlIovk4k2OusLDr8d5ciKjXZsX7tbjVRN6uzfC9hJegcZBM0S7yRIVu  
hSKAC9J5ObDyaAtNkQJBAN7FaLt4mP3LNvnChW8EuBwCoi8JugYvg8djUWbU+49o  
orjDKfEpUMeT5ILnS68EuA2ZYTbWLMSQIkLp4vluj38CQQCcXQqGyGtYg3Zm5lyY  
qzZ6fqzYZTM2o7vFGgmOa0Chp/Uaq6EwfMKKlS5qVGU624li8cPLyyn7K4hHaCAL  
5LJ/AkAp/IsAynoLW4HBESiVBA6vOM/YjbzB7bKooc910i6XmOEScbY1/dN7yiQH  
iBB7rCgEaCQqcbN+WKwccHiXCn+BAkACEsXB9cRenxeh06okqBiH1UgrcmQUdGVX  
/rljtw5bVmPMn7v6bZ0Sw1mjNU1GU4Tl1/jVqfL7jp82+y15muW3AkAU+jKJBNFJ  
z1L0hCGUb3a/WsXa9gONzdXDIr4hqh6Qyd9z9FeI68I71xtyNBIjuzSmsPrqOx8p  
Ld0Q+z1P4M2R  
-----END PRIVATE KEY-----` // pass your here private key**

**$encryptedPayload = hybridEncrypt($data, $privateKey);**

**✅ Final Node Function :-**

**function hybridEncrypt(data, privateKey) {  
  try {  
    privateKey = privateKey.replace(/\\n/g, '\n');    const aesKey = crypto.randomBytes(32); // AES-256  
    const iv = crypto.randomBytes(16);    const cipher = crypto.createCipheriv("aes-256-cbc", aesKey, iv);  
    let encryptedData = cipher.update(data, "utf8", "base64");  
    encryptedData += cipher.final("base64");    const encryptedAesKey = crypto.privateEncrypt(  
      {  
        key: privateKey,  
        padding: crypto.constants.RSA\_PKCS1\_PADDING,  
      },  
      aesKey  
    );    const payload = {  
      encryptedData,  
      encryptedKey: encryptedAesKey.toString("base64"),  
      iv: iv.toString("base64"),  
    };    const base64EncodedPayload = Buffer.from(JSON.stringify(payload)).toString('base64');    return base64EncodedPayload;  } catch (err) {  
    console.error("Encryption Error:", err.message);  
    return false;  
  }  
}**