

**RemotePIN**

**Browser/Mobile SDK**

**Version 2.0**

**27th April 2017**

**Confidential**

**Copyright © 2017 Salt Group Pty Ltd**

**Melbourne**

**+61 3 9866 4400**

**Contents**

1 Introduction 3

2 SDK Overview 3

2.1 Browser - JavaScript 3

2.2 Android SDK 3

2.3 iOS SDK 5

# Introduction

The RemotePIN architecture involves end user browser/mobile to capture the PIN then pass the encrypted PIN block into the backend Internet Banking application. For this purpose, Salt has provided a browser/mobile side JavaScript/SDK to generate the encrypted Intermediate PIN-Block (eIPB). The following sections describe the JavaScript and mobile SDK information in detail.

# SDK Overview

## Browser - JavaScript

The eIPB generation JavaScript provided by Salt is in format of a compressed .js file. The following sample illustrates the JavaScript method signature for the encrypted PIN-Block generation function.

//@param publicKeyInfo, a String object- Public Key information

//@param pin, a String object- user entered PIN to generate the IPB

//@param panReference, a String object- The PAN Reference

//return a String object - Encrypted Intermediate PIN-Block in Base64 encoded

function getEncryptedIPB(publicKeyInfo, pin, panReference)

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| publicKeyInfo | This is a Java String object which contains the RSA Public key details for the IPB encryption. The Browser application is required to retrieve the publicKeyInfo string object by calling the Internet banking (IB) application. Upon request IB retrieves the publicKeyInfo string by calling the Salt Echidna Server SOAP call 'getPublicKeyInfo'. |
| pin | User entered PIN passed in as a String parameter. |
| panReference | The PAN Reference passed in as a String parameter.  PAN Reference,   1. Can be a valid hexadecimal String. 2. Can also be null or empty. 3. Accepts any length. If length is greater than 12, the check digit will be excluded and the 12 rightmost characters will be taken as the PAN Reference. |
| Return - String | Return String contains the eIPB in base64 encoded format |

## Android SDK

The Salt provided Android SDK is in format of a .jar file, which contains compiled java classes. Following is the method in the SDK that generates the encrypted PIN-Block.

The method to retrieve the eIPB is in the au.com.saltgroup.remotepin.sdk.RemotePin

class within the provided SDK library.

/\*\*

\* Encrypts the provided pin with a provided public key under ISO-0

\* PIN-Block format/PKCS#1. This method uses the list of active cryptography

\* {@link Provider}s available to the system. If there is a need to add new

\* provider, then it can be done by calling

\* {@link #addProvider(java.security.Provider)}.

\*

\* Note: ISO-0 defines that pin digits must be 4bit values. This means that

\* valid pins can be from 0x0 to 0x9. Values out of this range will result in

\* throwing a {@link RemotePinException}.

\*

\* @param publicKeyInfo A string containing the Base64 representation of

\* the public Key in its primary encoding format.

\* @param pin user PIN.

\* @param panReference The PAN Reference

\* @return Encrypted IPB in base64 encoded format.

\* @throws RemotePinException if any error happens during the operation or the

\* arguments contain invalid data. The Exception contains the detailed

\* message describing the error and also contains the cause of the

\* exception.

\*/

**public** String getEncryptedIPB(String publicKeyInfo, **char**[] pin, **char**[] panReference) **throws** RemotePinException

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| publicKeyInfo | This is a Java String type parameter which contains the RSA Public key details for the IPB encryption. The Android application is required to retrieve the publickKeyInfo string by calling the Internet banking (IB) application. Upon request, IB retrieves the publickKeyInfo string by calling the Salt Echidna Server SOAP call 'getPublickKeyInfo'. |
| pin | User entered PIN passed in as a char[] parameter. |
| panReference | The PAN Reference passed in as a char [] parameter.  PAN Reference,   1. Can be a hexadecimal character which is from 0x0 to 0xF. 2. Can also be null or empty char [] 3. Accepts any length. If length is greater than 12, the check digit will be excluded and the 12 rightmost characters will be taken as the PAN Reference. |
| Return - String | Return String contains the encrypted IPB in base64 encoded format |

**Usage example:**

String publicKeyString = "MIIBIjANBgkqhkiG9..............................";

char[] pin = "1234".toCharArray();

char[] panReference = "12345678901234".toCharArray();

RemotePin remotePin = new RemotePin();

String encryptedIPB = remotePin.getEncryptedIPB(publicKeyString, pin);

In case that the user prefers a specific Security provider, it can be done by calling the addProvider() method. This method accepts a Java Security Provider and adds it to the Security providers of the system with the highest priority in the provider’s chain.

/\*\*

\* Adds a new provider, at the first position. The position is the preference

\* order in which providers are searched for requested algorithms.

\*

\* First, if there is a security manager, its <code>checkSecurityAccess</code>

\* method is called with the string

\* <code>"insertProvider."+provider.getName()</code> to see if it's OK to add

\* a new provider. If the default implementation of

\* <code>checkSecurityAccess</code> is used (i.e., that method is not

\* overridden), then this will result in a call to the security manager's

\* <code>checkPermission</code> method with a

\* <code>SecurityPermission("insertProvider."+provider.getName())</code>

\* permission.

\*

\* @param provider the provider to be used for encryption.

\* @throws RemotePinException with a cause of SecurityException if a security

\* manager exists and its <code>{@link

\* SecurityManager#checkSecurityAccess}</code> method denies access to

\* add a new provider

\*/

**public** void addProvider(Provider provider) **throws** RemotePinException

## iOS SDK

The iOS version of the SDK provided by Salt will be in .a file format. The SDK file will includes compiled Objective-C classes. Following is the method signature of the encrypt PIN-Block.

+(NSString\*) getEncryptedIPB:(NSString\*)publicKeyInfo andPIN:(unsigned char\*)pin andPANReference:(unsigned char\*) panReference;

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| publicKeyInfo | This is a NSString\* type parameter which contains the RSA Public key details for the IPB encryption. The iOS application is required to retrieve the publickKeyInfo data by calling the Internet banking (IB) application. Upon request, IB retrieves the publickKeyInfo details by calling the Salt Echidna Server SOAP call 'getPublickKeyInfo'. |
| pin | User entered PIN passed in as a char\* parameter. |
| panReference | The PAN Reference passed in as a char\* parameter.  PAN Reference,   1. Can be a hexadecimal character which is from 0x0 to 0xF. 2. Can also be null or empty char\* 3. Accepts any length. If length is greater than 12, the check digit will be excluded and the 12 rightmost characters will be taken as the PAN Reference. |
| Return - NSString\* | Return NSString\* contains the encrypted IPB in base64 encoded format. |

**Usage example:**

#import "RemotePin.h"

.....

..

NSString\* pin = @"1234";

NSString\* panReference = @"12345678901234";

NSString\* publicKeyStr = @"MIIBIjANBgkqhki..................";

NSString\* encryptedString = [RemotePin getEncryptedIPB:publicKeyStr andPIN:(unsigned char \*)[pin UTF8String] andPANReference: (unsigned char \*)[panReference UTF8String]];