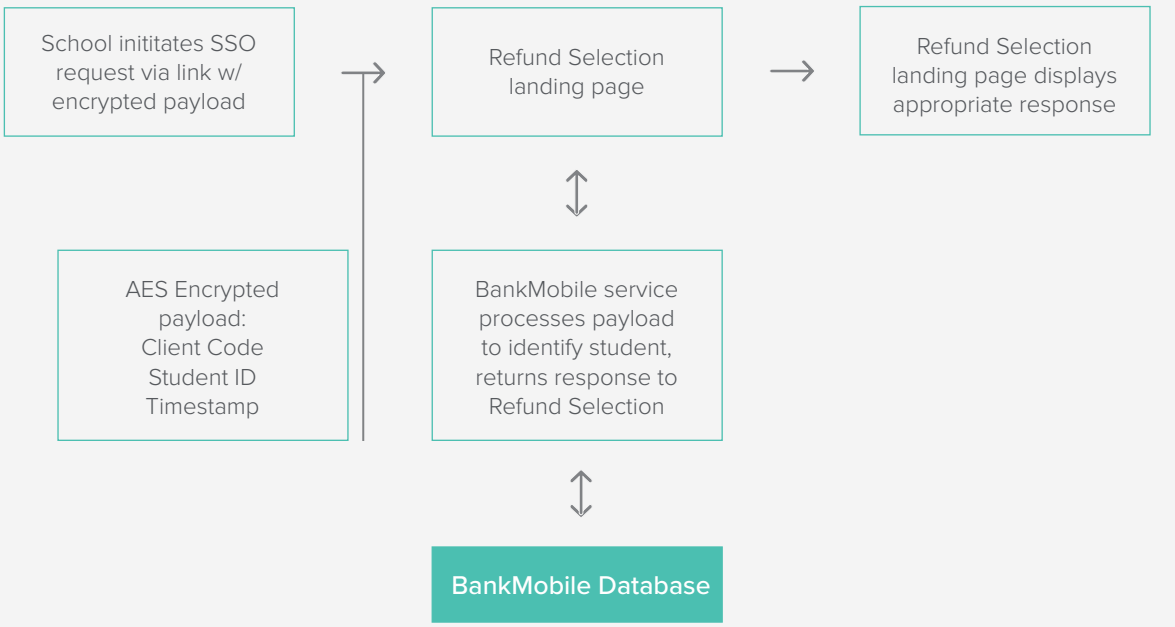
BankMobile Disbursements
Refund Selection Single Sign On

OVERVIEW

The BankMobile Disbursements Refund Selection Single Sign-On (SSO) solution will offer an integrated user authentication system for students to make their initial refund selection preferences. The intention of this service is to streamline the students experience and authenticate the student via their student portal credentials.

This service will eliminate the need for the “Personal Code” and allow for quicker access to the student refund selection process. The school will be acting as the identity provider and BankMobile Disbursements will use the data provided to deploy the solution as documented below.

The overall approach for our SSO solution is illustrated in the following diagram:



BankMobile Disbursements

As illustrated above, the school generates an SSO request link with an encrypted payload string. Students who click this link will be directed to the BankMobile Disbursements Refund Selection application, where a dedicated service processes the payload string and uses the included values to identify the student. Once having identified the student, and assuming no other errors or the expiration of the token, the Refund Selection application will display an appropriate response and "sign the user on" by directing them to the existing profile page. This allows students to circumvent the usual Refund Selection welcome page on which they are prompted to enter a Personal Code in order to identify them.

SCHOOL IMPLEMENTATION

The BankMobile Disbursements SSO solution presumes the existence of a shared key/phrase that will be used to encrypt the request payload string at the school and decrypt said string within the Refund Selection application. It is critically important that schools implementing SSO make every effort to ensure the security of the production shared key. We will be utilizing AES encryption for our initial implementation, with a key size of 128 bits.

The request payload string will have the following format: clientCode&studentid&timestamp

The constituent parts of the payload string are defined as follows:

clientCode: School identifier provided by BankMobile Disbursements

studentid: School's identifier for the student, which is also the ID1 value that is configured in the Customer Import process. This ID is unique per student at the school.

timestamp: Capture the current timestamp and convert to UTC timezone, formatted as MM/dd/yyyy HH:mm:ss.

The above request payload shall be AES encrypted, then used as a parameter to the incoming HTTPS request URL to the Refund Selection SSO landing page. The parameter should be appended to the SSO landing base URL with a parameter name of "token". Additionally, a parameter named "clientcode" should be appended to the end of the URL following the token parameter. The clientcode parameter value is the same as that used in the payload string prior to encryption. Note this is an HTTP GET request,

www.refundselection.com/#/landing?token=ENCRYPTED PAYLOAD&clientcode=CLIENT_CODE

ENCRYPTION SCHEME

Prior to implementation, BankMobile shall provide a 32-character random hexadecimal string to be used as the shared key between systems for encryption and decryption of the token payload. Further technical details of the AES encryption scheme are as follows:

Encryption Algorithm: AES

Key Size: 128 bit

Mode: ECB

Padding: PKCS5

PAYLOAD STRING & REQUEST URL EXAMPLE

Given the following values:

clientCode: some_university
studentid: 12345678
shared key: 0123456789ABCDEF0123456789ABCDEF

Capture the timestamp:

Determine current timestamp: 01/09/2017 12:14:15
Convert to UTC timezone formatted as “MM/dd/yyyy HH:mm:ss”: 01/09/2017 17:14:15

Concatenate clientCode, studentid and timestamp, using ampersand symbol as separator:

some_university&12345678&01/09/2017 17:14:15

AES encrypt the resulting string using the shared key, then represent it as a string of hexadecimal digits for use as the token parameter value:

www.refundselection.com/#landing?token=298C3A4DE0FA95D02CAA07F24F7F234BC78EB18F98B DD2 CBC20AC18FB9F39AF84B0CE8A37773DEBEDC9D74AB1C2D8E5D&clientcode=some_universitySCHOOL

REFUND SELECTION ERROR HANDLING

The following is a list of use cases we will handle for this implementation:

1. Inability to decrypt the payload, or missing value in payload: Prompt the user about configuration error and ask them to reach out to a school administrator to learn about the status of their refund.
2. clientCode does not match a configured client school: Prompt the user about configuration error and ask them to reach out to a school administrator to learn about the status of their refund.
3. studentid does not match a student at client school: Prompt the user about configuration error and ask them to reach out to a school administrator to learn about the status of their refund.
4. expiration is past: Prompt the user about session timeout and ask them to log back in to school’s portal and try again
5. Student is identified but in an improper state: Prompt user to call Care
6. Student is identified but already in an active state: Prompt user to log in
7. Core banking services are not available: Prompt the user about system being unavailable and ask them to try again later.

AES TOKEN ENCRYPTION SAMPLE CODE

Test Class

```
package com.higherone.service.sso;

import org.junit.Test;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;

import static org.junit.Assert.assertEquals;

/**
 * <pre>
 * Test class to exercise the SSOAESExample class.
 * </pre>
*/
public class TestSSOAESExample {
    private static final String SHARED_KEY = "0123456789ABCDEF0123456789ABCDEF";

    @Test
    public void testSSOAESEncryptThenDecrypt() {
        String clientcode = "schoolcode";
        String studentid = "12345678";
        Calendar timestampCal = new GregorianCalendar(2017, 0, 9, 17, 29, 15);
        TimeZone utc = TimeZone.getTimeZone("UTC");
        SimpleDateFormat smp = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
        smp.setTimeZone(utc);
        String token = SSOAESExample.buildTokenString(clientcode, studentid,
smp.format(timestampCal.getTime()));
        String encrypted = SSOAESExample.aesEncrypt(SHARED_KEY, token);
        String decrypted = SSOAESExample.aesDecrypt(SHARED_KEY, encrypted);
        assertEquals("Token string prior to encryption should equal decrypted token
string", token, decrypted);
        System.out.println("Original token string: " + token);
        System.out.println("Encrypted token string: " + encrypted);
        System.out.println("Decrypted token string: " + decrypted);
    }
}
```

IMPLEMENTATION

```
package com.higherone.service.sso;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
```

```
import javax.crypto.spec.SecretKeySpec;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * <pre>
 * Utility functions for BankMobile Refund Selection SSO (SSO).
 * </pre>
*/
public class SSOAESExample {
    private static final String CIPHER_TYPE_AES = "AES/ECB/PKCS5Padding";
    private static final String ENCODING_TYPE_AES = "AES";


    /**
     * AES encrypt message string using sharedKey.
     * @param sharedKey - This is the shared encryption key between systems
     * @param token - The token to encrypt
    */
    protected static String aesEncrypt(String sharedKey, String token) {
        String encryptedString = null;
        try {
            SecretKey key = new SecretKeySpec(asBinary(sharedKey), ENCODING_TYPE_AES);
            Cipher cipher = Cipher.getInstance(CIPHER_TYPE_AES);
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] encrypted = cipher.doFinal(token.getBytes());
            encryptedString =  asHex(encrypted).toUpperCase();
        } catch
(NoSuchAlgorithmException|NoSuchPaddingException|InvalidKeyException|IllegalBlockSizeE
xception|BadPaddingException e) {
            e.printStackTrace();
        }
        return encryptedString;
    }
    /**
     * AES decrypt message string using sharedKey.
     * @param sharedKey - This is the shared encryption key between systems
     * @param token - The token to deccrypt
     */
    protected static String aesDecrypt(String sharedKey, String token) {
        String decryptedString = null;
        try {
            SecretKey key = new SecretKeySpec(asBinary(sharedKey), ENCODING_TYPE_AES);
            Cipher cipher = Cipher.getInstance(CIPHER_TYPE_AES);
            cipher.init(Cipher.DECRYPT_MODE, key);
            byte[] decrypted = cipher.doFinal(asBinary(token));
            decryptedString = new String(decrypted);
        } catch
(NoSuchAlgorithmException|NoSuchPaddingException|InvalidKeyException|IllegalBlockSizeE
xception|BadPaddingException e) {
            e.printStackTrace();
        }
        return decryptedString;

    }
```

```
    /**
     * Utility function to convert bytes in to hex strings
     * @param buf
     * @return - hex string
     */
    private static String asHex( byte buf[] )
    {
        StringBuffer strbuf = new StringBuffer(buf.length * 2);
        for (int i = 0; i < buf.length; i++)  {
            if (((int) buf[i] & 0xff) < 0x10) {
                strbuf.append("0");
            }
            strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
        }
        return strbuf.toString();
    }
    /**
     * Convert hex string into corresponding bytes.
     * @param hexChars
     * @return bytes
     */
    private static byte[] asBinary( String hexChars )
    {
        // Find the length in bytes to be made. Each hexChar is 4 bits.
        int numBytes = hexChars.length() / 2;
        byte[] binary = new byte[ numBytes ];
        for ( int x=0; x<numBytes; x++ )
        {
            String temp = "0x" + hexChars.charAt( x*2 ) + hexChars.charAt( x*2+1 );
            binary[ x ] = Integer.decode( temp ).byteValue();
        }
        return binary;
    }

    protected static String buildTokenString(String clientCode, String studentID,
String timestamp) {
        StringBuffer tokenString = new StringBuffer();
        tokenString.append(clientCode);
        tokenString.append("&");
        tokenString.append(studentID);
        tokenString.append("&");
        tokenString.append(timestamp);
        return tokenString.toString();
    }
}
```