AccessMatrix™

UAS E2EE Developer Guide

j-Sprint
Trust without Boundaries

Document Version: 82.1

Date: 09 November 2015

TRADEMARK INFORMATION and DISCLAIMER

Table of Contents

1 Introduction

1.1 Purpose

This document provides guide to application integration using AccessMatrix E2EE. Application integration is usually done via the E2EE Service API available in JAVA or C# (This document only available in JAVA). Developers are to be familiar with the E2EE API.

1.2 E2EE (End to End Encryption) Implementation

AccessMatrix End-to-End Encryption Authentication Solution (AccessMatrix E2EE) implementation is based on the SafeNet's PHEFT. It is modeled as a handler using AM5's pluggable login module handler framework. It is similar for SafeNet's PSG & PSE.

Usually E2EE is implemented with users residing in the UAS repository, which could be default user store in UAS or searchable from external user store, where their encrypted passwords (or STPV) will be stored in the UAS database (refer to **1.2.1 UAS-managed persistence**); however, it is also possible to implement E2EE so that the encrypted password (or STPV) could be managed and stored by customer's application where its user repository is not managed by UAS (refer to **1.2.2 Application-managed persistence**).

Implementation Details	Application-managed persistence	UAS-managed persistence	Remark
Repository to associate user/ encrypted password	Application's database/repository	UAS database	User must be created within the default user store in UAS or searchable from external user store
Action to store STPVs	During reset/change password, application must store the current and historical STPVs into application repository	Current & historical STPVs managed by UAS	
Provider of user salt	Application must provide the user salt when calling UAS APIs	User salt is provided by UAS transparent to application	User salt is used in hashing operation to produce STPVs during UAS E2EE operations. The same salt must be provided for the same user across all operations. It must be immutable and ideally unique for each user.

Table 1.0 Implementation comparison of UAS-managed persistence and Application-managed persistence

1.2.1 UAS-managed persistence

UAS-managed persistence implementation manages and stores the encrypted user password in UAS repository. In this approach, UAS would perform password checking and validation based on the setting of the specified password policies.

Refer to below E2EE operations that are illustrated with their respective sequence diagrams, prior to each operation, the pre-authentication would be done first to prevent a session replay attack where the AM Server will generate an E2EE Session Id and the HSM box will generate a unique challenge code (a.k.a server-random number) and randomly pick a RSA public key to be used for the e2ee session:

- **2.2 Verify Password**
- **2.3 Reset Password**
- **2.4 Change Password**

1.2.2 Application-managed persistence

Application-managed persistence implementation allows applications to leverage on the E2EE implementation while still continuing to manage and store the encrypted user password in the application repository. In this approach, the user id information would not be populated inside UAS. However the enforcement of certain password policies like password expiry, bad logins, etc, needs to be enforced by the application.

Refer to below E2EE operations that are illustrated with their respective sequence diagrams, prior to each operation, the pre-authentication would be done first to prevent a session replay attack where the AM Server will generate an E2EE Session Id and the HSM box will generate a unique challenge code (a.k.a server-random number) and randomly pick a RSA public key to be used for the e2ee session:

- **2.5 Verify External User Password**
- **2.6 Reset External User Password**
- **2.7 Change External User Password**

1.3 E2EE Client Integration

E2EE client library can be loaded or integrated with different client channels (e.g. Web browser, iPhone application, Android application or BlackBerry application) to protect user clear password by having it encrypted using RSA public key to produce encrypted pin block (also known as RPIN (RSA encrypted PIN)).

Below is the required E2EE Client libraries for each application:

- **Android**
  - e2eeAndroid.jar
  - minimum API level is **android:minSdkVersion:3.0**
- **BlackBerry**
  - e2eeBlackBerry.jar
  - minimum API level is **BlackBerry OS 4.5**
- **iPhone**
  - AxMxCrypto.h
  - E2EEA.h
  - libE2EEA.a
- **.NET**
  - E2EEABC.dll

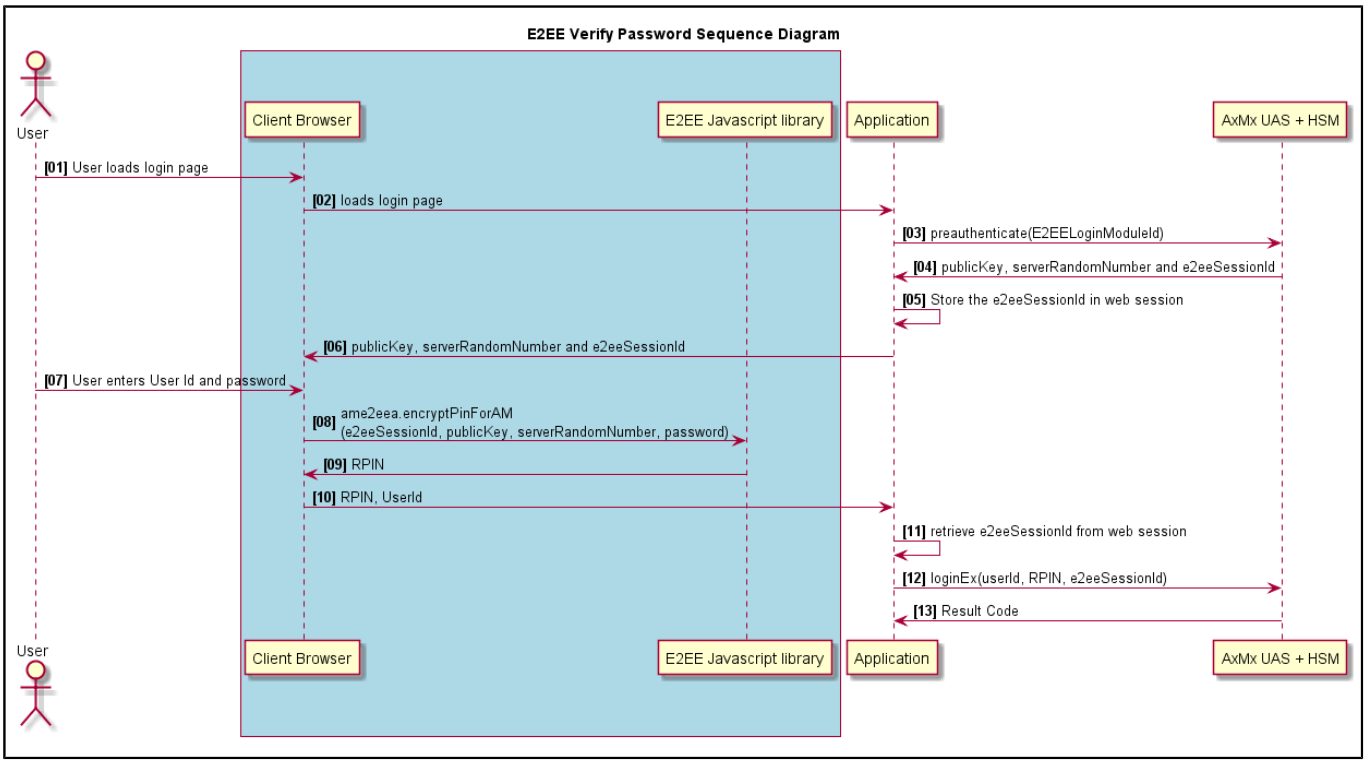
2 API Use Case and Sample Codes

2.1 Login (Pre-authentication & Authentication)

XML RPC API to be called: **preauthenticate**, **loginEx**. Sequence diagram below shows the basic flow of the API calls:

1. AuthenticationService.preauthenticate(loginModuleId, params) needs to be called to obtain the public key, server random number and e2ee session id from the server before login.
2. Application needs to store the e2ee session id in the web session. Application needs to provide this e2ee session id when calling loginEx
3. Application's login page checks user password quality as needed
4. Application's login page needs to call JavaScript function ame2eea.encryptPinForAM(e2eeSid, pubKey, randomNumber, userPassword, oaepHashAlgo) after the user has entered password, this function will generate encrypted password RPIN for login. It is best that the clear password is deleted after the encryption is done.
5. Set E2EE Authentication Realm in params and the e2ee session id obtain from preauthenticate() into ChallengeTO Object and pass it in loginEx() for login.
6. RPIN generated is passed in as password for loginEx() method.
7. AuthenticationService.loginEx(sessionToken, userId, RPIN, ChallengeTO, params) API is called to invoke login.

2.1.1 Sequence Diagram



2.1.2 XMLRPC Sample Code

E2EE Pre-authenticate (Java)

```
<%
String sessionToken ="";
AuthResultTO finalAuthRes = null;
String errorMsg = null;
int code = -1;
String loginModuleId = props.getProperty("E2EE_Login_Module_ID");
%><p>loginModuleId: <%=loginModuleId%> <%
String realmId = props.getProperty("E2EE_REALM_ID");
%><p> realmId: <%=realmId%> <%
// pre-authenticate
ChallengeTO challenge =null;
try {
Map params = new HashMap();
challenge = proxy.getAuthnSvc().preauthenticate(loginModuleId, params);
} catch (Exception e){
%>
get preAuthentication E2EE params failed!
<%
errorMsg = e.getMessage();
```



```
if (e instanceof XmlRpcException) {
XmlRpcException authEx =(XmlRpcException)e;
code = authEx.code;
if (code > -1) %>
<p> Error Code: <%=code%>
<%}%>
<p> Error Message: <%=errorMsg%>
<%}
String _e2EEPublicKey = null;
String _e2EERandomNum = null;
String _e2EESid = null;
if (challenge == null) {
%>
get preAuthentication E2EE params failed!
<% } else {
_e2EEPublicKey = (String)challenge.getAttr("params", "pubKey");
_e2EERandomNum = (String)challenge.getAttr("params", "serverRandom");
                _e2EESid       = (String)challenge.getAttr("params", "e2eeSid");
                session.setAttribute("E2EE_SID", _e2EESid);
}
%>
```

RPIN Generation (Javascript):

Refer to Appendixes 7.1 JavaScript Function for RPIN Generation

E2EE Login (Java):

```
<%
String loginModuleId = props.getProperty("E2EE_Login_Module_ID");
String realmId = props.getProperty("E2EE_REALM_ID");
String e2eeSid =(String)session.getAttribute("E2EE_SID");
String userId = (String)request.getParameter("E2EE_USER_ID");
String rpin = (String)request.getParameter("E2EE_RPIN");
Map<String, String> params = new HashMap<String, String>();
params.put("realmId", realmId);
%>
<p> userId = <%=userId%>
<p> rpin = <%=rpin%>
<p> e2eeSid = <%=e2eeSid%>
<%
ChallengeTO returnChallenge = new ChallengeTO();
returnChallenge.set("params", new HashMap());
returnChallenge.getMap("params").put("e2eeSid",e2eeSid);
try {
loginResult = proxy.getAuthnSvc().loginEx("", userId, rpin, returnChallenge, params);
}  catch (Exception e){
successful = false;
errorMsg  =e.getMessage()+"\n";
%>
errorMsg3  <%=e.getMessage()%>
<%
if (e instanceof XmlRpcException) {
XmlRpcException authEx =(XmlRpcException)e;
code = code;
}
%>
}
```

2.1.3 Parameters & Exceptions

Please refer to AuthenticationService (loginEx and preauthenticate) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

2.1.4 Responses

Please refer to AuthenticationService (loginEx and preauthenticate) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

2.2 Verify Password

XML RPC API to be called: **verifyPasswordEx**.

2.2.1 Sequence Diagram

Please refer to section 2.1.1 Sequence Diagram sequence diagram for Login (Pre-authentication & Authentication) use case.

2.2.2 XMLRPC Sample Code

RPIN Generation (Javascript):
Refer to Appendixes 7.1 JavaScript Function for RPIN Generation
E2EE Verify Password (Java)

```
HashMap parms = new HashMap();
parms.put("remoteAddr", "192.168.1.36");
AuthResultTO result = proxy. getPasswordSvc().verifyPasswordEx (sessionToken, userId, userStoreId,
loginModuleId, RPIN, challenge, parms);
```

2.2.3 Parameters & Exceptions

Please refer to Password Service (verifyPasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

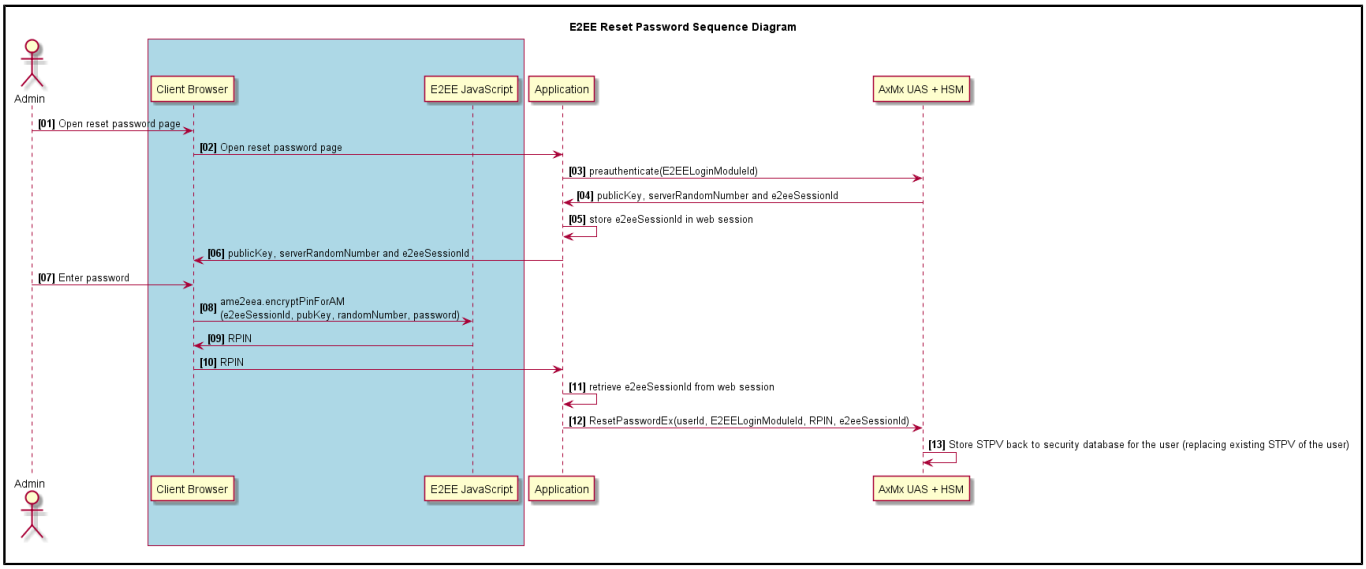
2.2.4 Responses

Please refer to Password Service (verifyPasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

2.3 Reset Password

XML-RPC API to be called: **resetPasswordEx**. Sequence for reset password scenario is similar to login. It is important to check the password quality in the reset password page as UAS is unable to check the password quality because the password is encrypted

2.3.1 Sequence Diagram



2.3.2 XMLRPC Sample Code

RPIN Generation (Javascript):
Refer to Appendixes 7.1 JavaScript Function for RPIN Generation
E2EE Reset Password (Java)

```
HashMap parms = new HashMap();
AuthResult result = null;
parms.put("remoteAddr", "192.168.1.36");
parms.put("passwordGenerationOption", "ManualEntry");
result = proxy. getPasswordSvc().resetPasswordEx (sessionToken, userId, userStoreId, loginModuleId,
 RPIN, challenge, parms);
```

2.3.3 Parameters & Exceptions

Please refer to Password Service (resetPasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

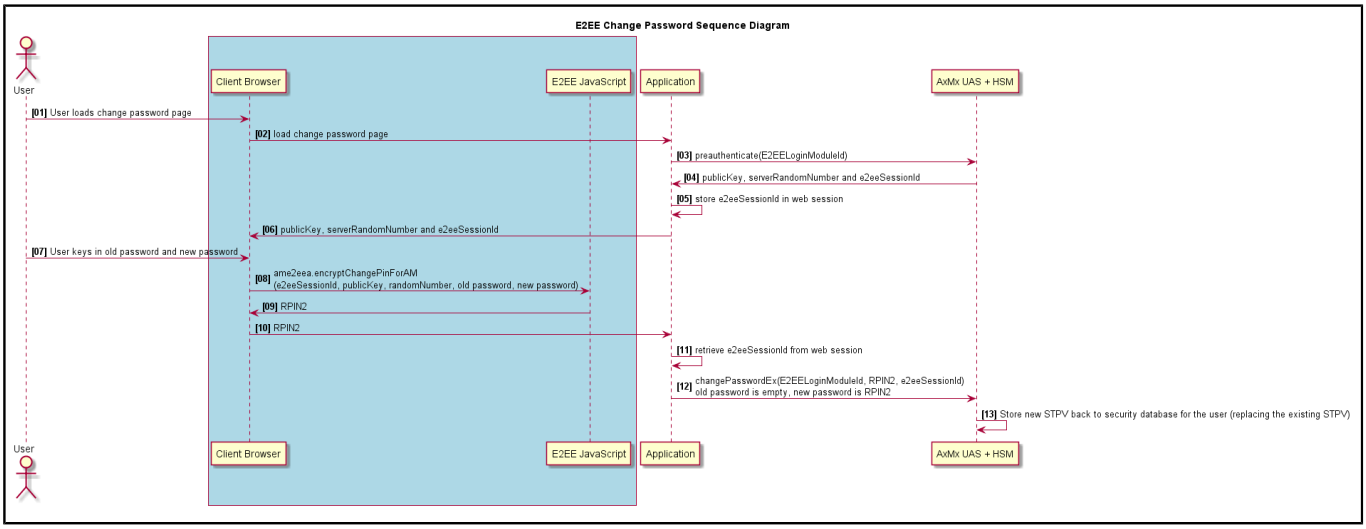
2.3.4 Responses

Please refer to Password Service (resetPasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

2.4 Change Password

XML RPC API to be called: **changePasswordEx**. Similar to login scenario, except that ame2eea.encryptChangePinForAM is called instead to encrypt both old password and new password. The value, RPIN2, should be used as the new password when calling changePasswordEx. Old password parameter of changePasswordEx should be set to empty because the user's old password is encrypted as part of RPIN2. Actual user's old password will still be verified.

2.4.1 Sequence Diagram



2.4.2 XMLRPC Sample Code

RPIN Generation (Javascript):
Refer to Appendixes 7.1 JavaScript Function for RPIN Generation
E2EE Change Password (Java)

`HashMap parms = new HashMap();`
`Parms.put("remoteAddr", "192.168.1.36");`
`AuthResultTO result = proxy. getPasswordSvc(). changePasswordEx(sessionToken, loginModuleId, RPIN, RPIN2, challenge, parms);`

Change Password RPIN Generation (Javascript):
Refer to Appendixes 7.2 JavaScript Function for Change Password RPIN Generation

2.4.3 Parameters & Exceptions

Please refer to Password Service (changePasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

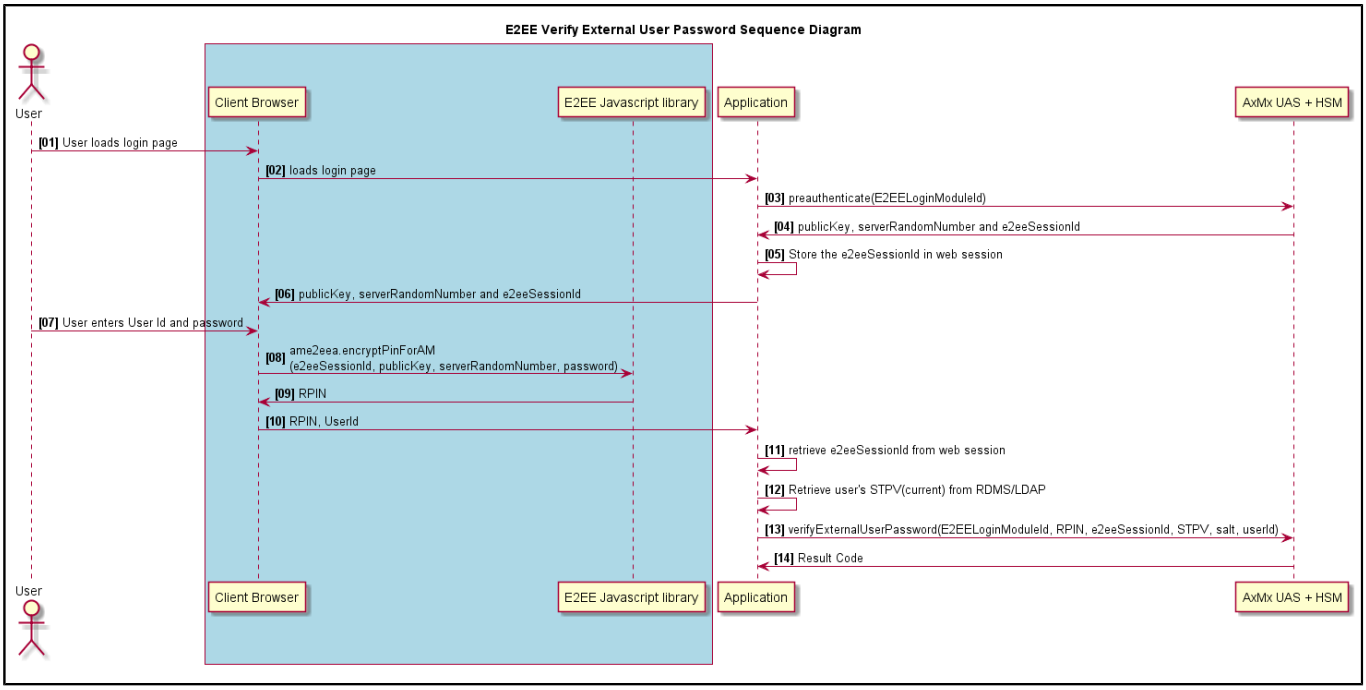
2.4.4 Responses

Please refer to Password Service (changePasswordEx) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

2.5 Verify External User Password

XML RPC API to be called: **verifyExternalUserPassword**. The verification flow is similar to UAS managed password. The difference is only that the STPV needs to be passed to UAS

2.5.1 Sequence Diagram



2.5.2 XMLRPC & SOAP Sample Code

RPIN Generation (Javascript):

Refer to Appendixes [7.1 JavaScript Function for RPIN Generation]

XMLRPC E2EE Verify External User Password

```
HashMap parms = new HashMap();
parms.put("remoteAddr", "192.168.1.36");
AuthResultTO result = proxy. getPasswordSvc().verifyExternalUserPassword
(sessionToken, externalUserId, loginModule, RPIN, salt, STPV, challenge, parms);
```

SOAP E2EE Verify External User Password

```
HashMap parms = new HashMap();
parms.put("remoteAddr", "192.168.1.36");
AuthResultTO result = proxy. getPasswordSvc().verifyExternalUserPassword
(sessionToken, externalUserId, loginModuleId, RPIN, challenge, salt, STPV, parms);
```

2.5.3 Parameters & Exceptions

Please refer to Password Service (verifyExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

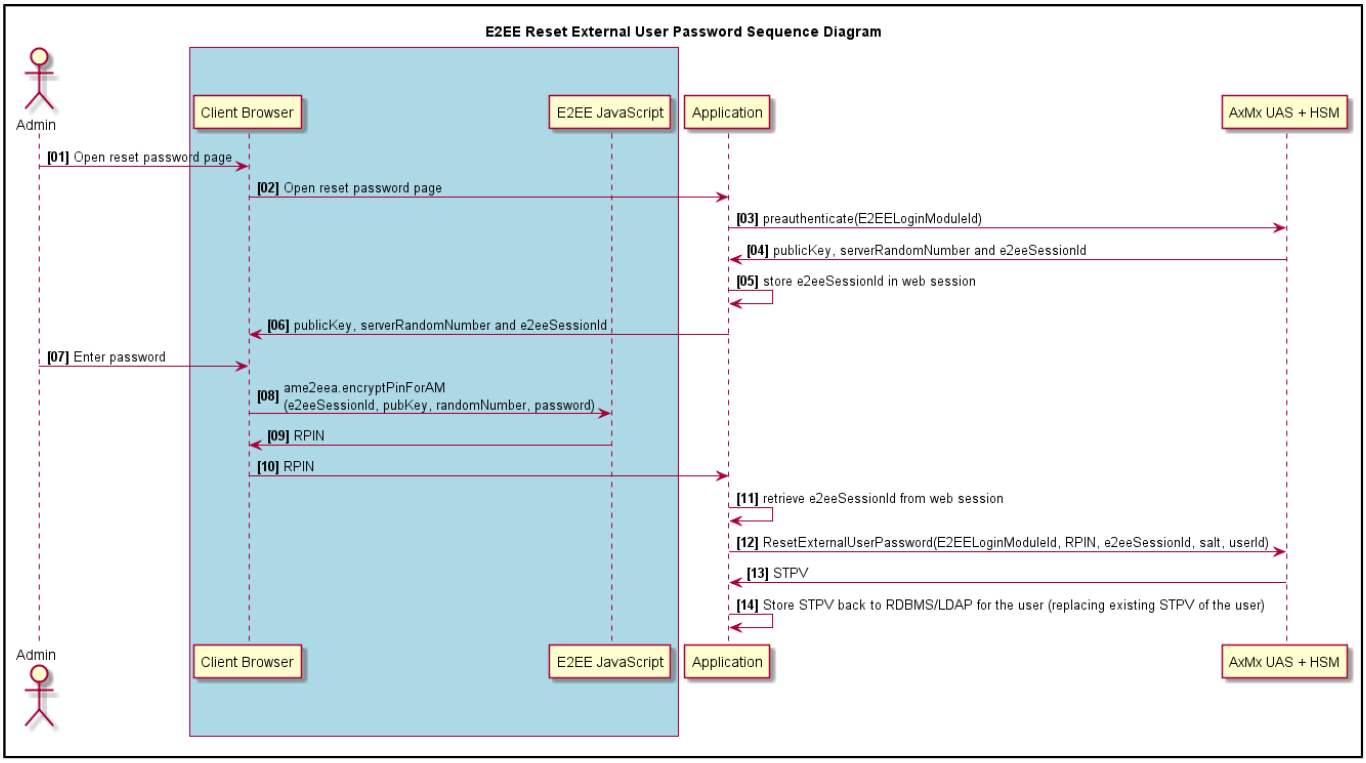
2.5.4 Password Responses

Please refer to Password Service (verifyExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

2.6 Reset External User Password

XML RPC API to be called: **resetExternalUserPassword**. The reset password flow is similar to UAS managed password. The difference is only that the STPV returned by UAS needs to be stored by application

2.6.1 Sequence Diagram



2.6.2 XMLRPC & SOAP Sample Code

RPIN Generation (Javascript):

Refer to Appendixes [7.1 JavaScript Function for RPIN Generation]

XMLRPC E2EE Reset External User Password

```
HashMap parms = new HashMap();
AuthResult result = null;
parms.put("remoteAddr", "192.168.1.36");
parms.put("passwordGenerationOption", "ManualEntry");
result = proxy. getPasswordSvc().resetExternalUserPassword(sessionToken, loginModule, RPIN, challenge,
salt, externalUserId, params);
```

SOAP E2EE Reset External User Password

```
HashMap parms = new HashMap();
AuthResultTO result = null;
parms.put("remoteAddr", "192.168.1.36");
parms.put("passwordGenerationOption", "ManualEntry");
result = proxy. getPasswordSvc().resetExternalUserPassword(sessionToken, externalUserId, loginModuleId,
RPIN, challenge, salt, params);
```

2.6.3 Parameters & Exceptions

Please refer to Password Service (resetExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

2.6.4 Responses

Please refer to Password Service (resetExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

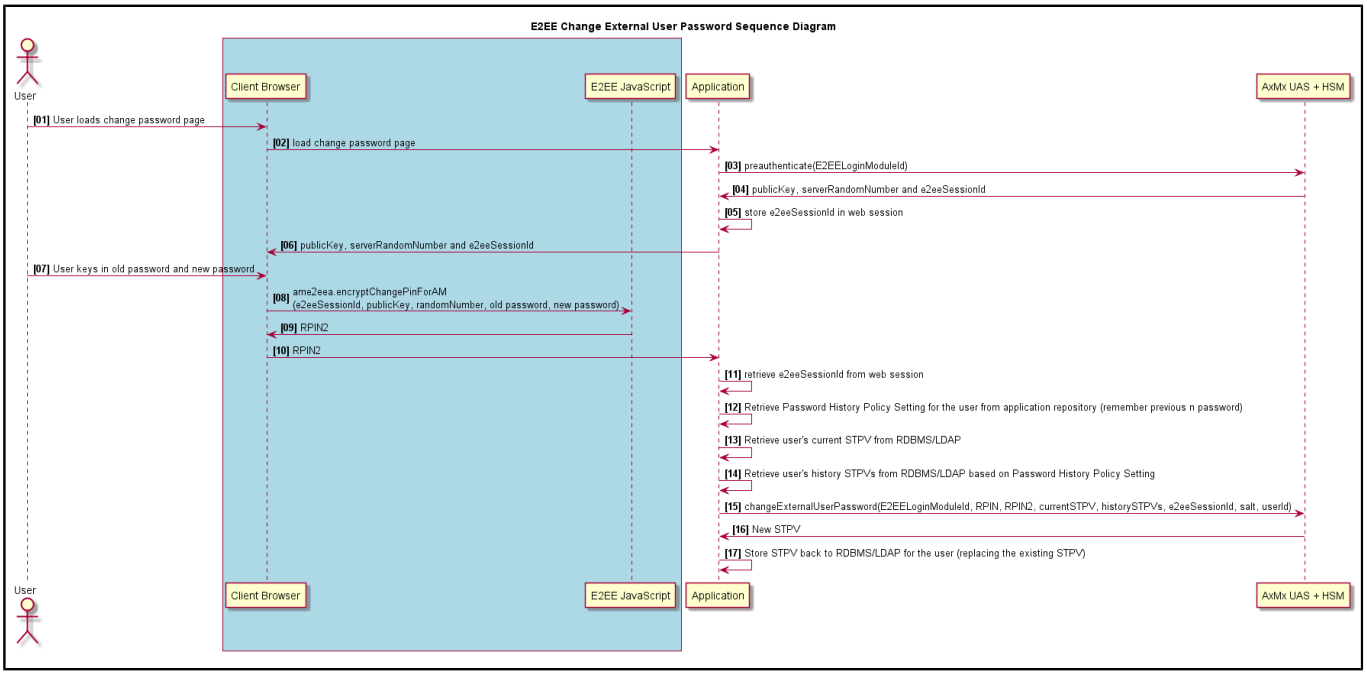
2.7 Change External User Password

XML RPC API to be called: **changeExternalUserPassword**. Similar to UAS managed password. The difference is that current STPV, historical STPVs need to be submitted to the security server. The security server will perform the following:

1. Verify that the old password is correct with the HSM.
2. Go through password histories checking with the HSM by verifying the new password against all the historical TPVs (provided by the application) with the HSM (the same HSM API used in the password verification operation will be called once against each password history)
3. Call the HSM box to perform the actual change password using the given encryption salt. The HSM box will check the new password against its password policy.

4. Return STPV, the new TPV generated ready to be stored, to the application to replace the existing STPV of the user.

2.7.1 Sequence Diagram



2.7.2 XMLRPC & SOAP Sample Code

RPIN Generation (Javascript):

Refer to Appendixes 7.1 JavaScript Function for RPIN Generation

XMLRPC E2EE Change External User Password

```
HashMap parms = new HashMap();
parms.put("remoteAddr", "192.168.1.36");
AuthResultTO result = proxy. getPasswordSvc().changeExternalUserPassword
(sessionToken, loginModule, RPIN, RPIN2, currentSTPV, historicalSTPVs, challenge, salt, externalUserId,
parms);
```

SOAP E2EE Change External User Password

```
HashMap parms = new HashMap();
parms.put("remoteAddr", "192.168.1.36");
AuthResultTO result = proxy. getPasswordSvc().changeExternalUserPassword
(sessionToken, loginModuleId, RPIN, RPIN2, currentSTPV, historicalSTPVs, challenge, salt,
externalUserId, parms);
```

Change Password RPIN Generation (Javascript):

Refer to Appendixes 7.2 JavaScript Function for Change Password RPIN Generation

2.7.3 Parameters & Exceptions

Please refer to Password Service (changeExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible parameters and exceptions.

2.7.4 Responses

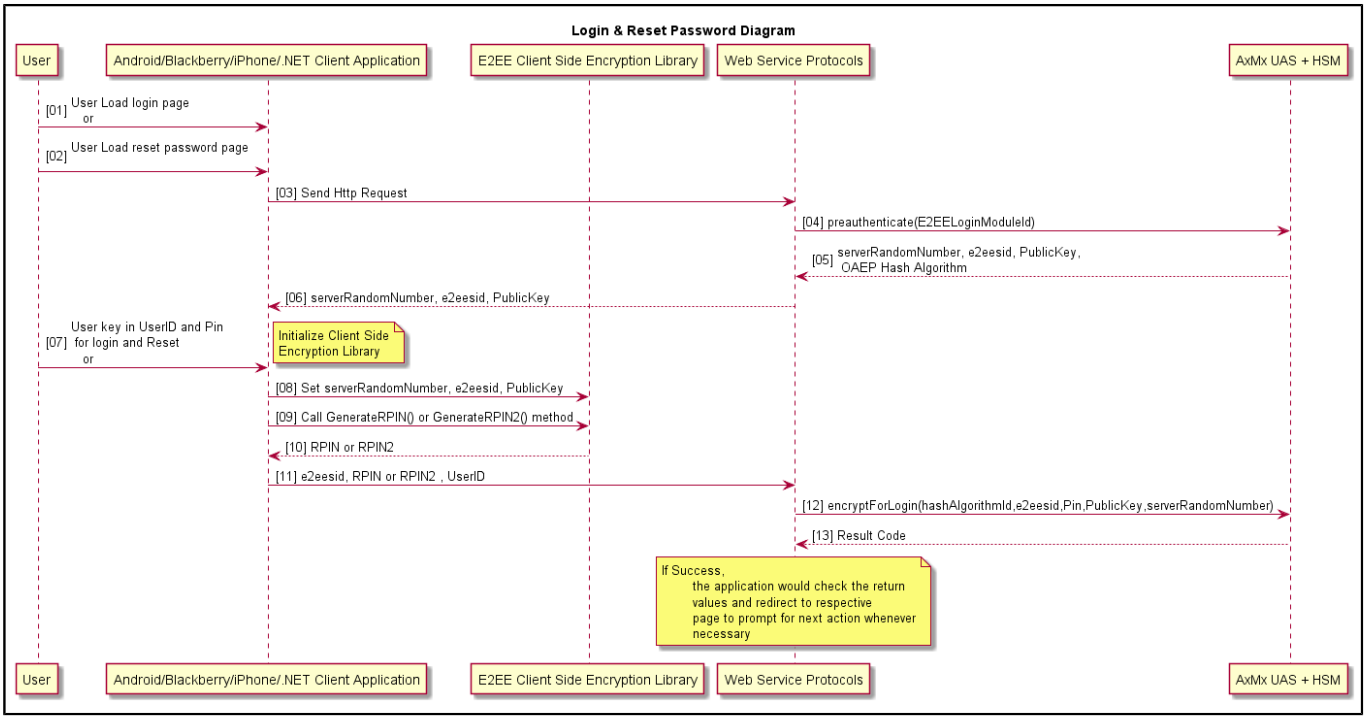
Please refer to Password Service (changeExternalUserPassword) in **"AM5 XML-RPC and SOAP API Reference"** for all possible responses.

3 API Use Case and Sample Codes for E2EE Client Application

3.1 Login / Reset Password

XMLRPC API to be called: **encryptForLogin**. Sequence diagram below shows the basic flow of the API calls:

3.1.1 Sequence Diagram



3.1.2 XMLRPC Sample Code

- For Android

E2EE encryptForLogin (Java)

```
AxMxE2EEClient e2ee = new AxMxE2EEClient();
try {
    String RPIN = e2ee.encryptForLogin(/** Default value*/ DefaultClient.ALGO_ID_3DES_SHA1, e2eesid,
pin, publicKey, serverRandom);
   int retCode = e2ee.getRetCode();
    If(retCode ==0) {
       //successful
   } else {
        If(e2ee.isInvalidPin(retCode)){
       //invalid pin
       } else {
            If(e2ee.isError(retCode))
           // is error
       }
    }
} catch (Exception e) {
// TODO Auto-generated catch block
}
```

- For BlackBerry

E2EE encryptForLogin (Java)

```
    AxMxE2EEClient e2ee = new AxMxE2EEClient();
    BouncyClient e2ee = new BouncyClient();
   try {
        String RPIN = e2ee.encryptForLogin(/** Default value*/ DefaultClient.ALGO_ID_3DES_SHA1, e2eesid,
pin, publicKey,serverRandom);
   } catch (DataValidationException e) {
   } catch (InvalidCipherTextException e) {
   } catch (Exception e) {
   }
```

- For iPhone

E2EE encryptForLogin (Java)

```
AxMxE2EEClient e2ee = new AxMxE2EEClient();
    E2EEA * e2ee = [[E2EEA alloc] init];
    NSString* RPIN = [e2ee encryptForLogin: [e2ee SHA1] session:e2eesid password:password
publicKeyValue:publicKeyValue random:serverRandom];
```

- For .NET

E2EE EncryptForLogin (Java)

```
// This example is based on E2EEANative.dll
   string RPIN = E2EEANative.Client.EncryptForLogin(
    E2EEANative.Client.ALGO_ID_SHA1,
    e2eesid, password, publicKeyValue, serverRandom);
```

3.1.3 Parameters and Exceptions

Field	Description	Type
hashAlgorithmId	7.4 Hash Algorithm Id	byte
e2eeSid	E2EE session Id generated by UAS server	String
pin	user's clear password	String
publicKey	Public Key value used to encrypt user's clear password.	String
serverRandom	HSM-generated random number	String

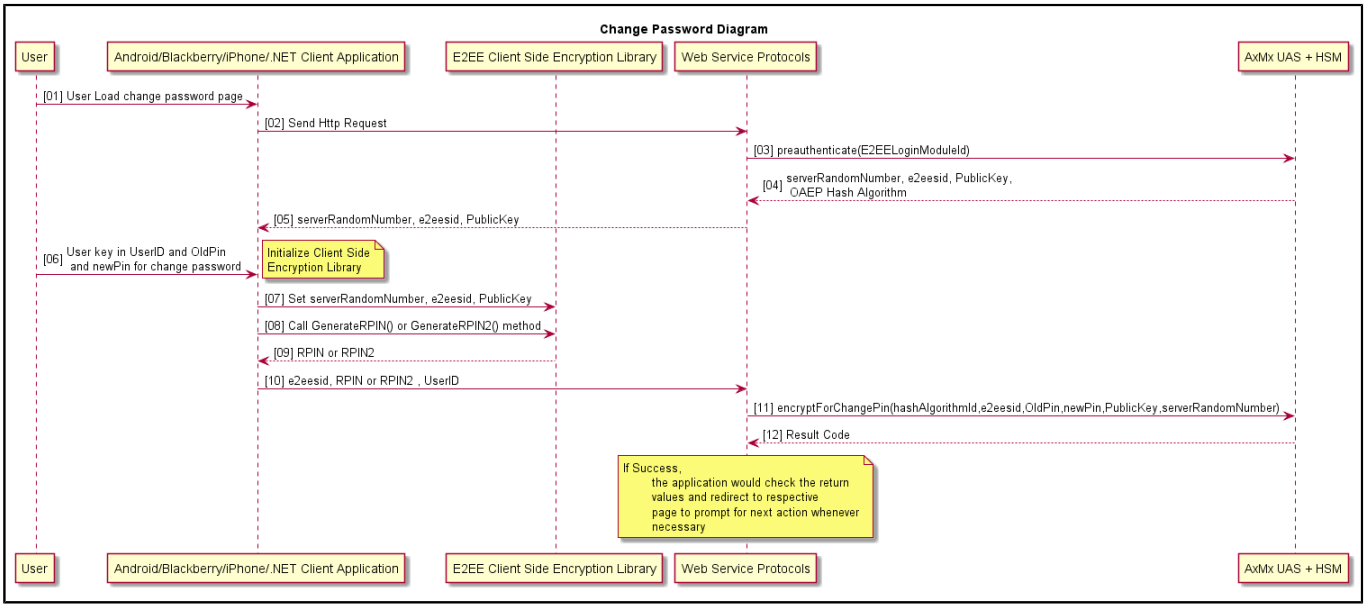
3.1.4 Responses

Field	Description	Expected Result(Success)	Result (Fail)
retCode1	Error code for RPIN	0	6 E2EE Error Codes
invalidPin	refers to RPIN2	false	true

3.2 Change Password

XMLRPC API to be called: **encryptForChangePin**. Sequence diagram below shows the basic flow of the API calls:

3.2.1 Sequence Diagram



3.2.2 XMLRPC Sample Code

- For Android

E2EE encryptForChangePin (Java)

```
    AxMxE2EEClient e2ee = new AxMxE2EEClient();
   try {
        String RPIN = e2ee.encryptForChangePin(/** Default value*/ DefaultClient.ALGO_ID_3DES_SHA1,
e2eesid,
oldPin, newPin, publicKey, serverRandom);
```



```
      int retCode = e2ee.getRetCode();
       If(retCode ==0) {
         //successful
      } else {

          If(e2ee.isInvalidPin(retCode)){
      //invalid pin
         } else {
            If(e2ee.isError(retCode))
           // is error
         }
      }
   } catch (Exception e) {
      // TODO Auto-generated catch block
   }
```

- For BlackBerry

E2EE encryptForChangePin (Java)

```
     AxMxE2EEClient e2ee = new AxMxE2EEClient();
     BouncyClient e2ee = new BouncyClient();
    try {
          String RPIN = e2ee.encryptForChangePin(/** Default value*/ DefaultClient.ALGO_ID_3DES_SHA1,
e2eesid, oldPin, newPin, publicKey, serverRandom);
    } catch (DataValidationException e) {
    } catch (InvalidCipherTextException e) {
    } catch (Exception e) {
    }
```

- For iPhone

E2EE EncryptForChangePin (Java)

```
AxMxE2EEClient e2ee = new AxMxE2EEClient();
    E2EEA* e2ee = [[E2EEA alloc] init];
    NSString* RPIN = [e2ee encryptForChangePin:[e2ee SHA1] session:e2eesid oldPassword:oldPin
newPassword:newPin publicKeyValue:publicKeyValue random:serverRandom];
```

- For .NET

E2EE EncryptForChangePin (Java)

```
// This example is based on E2EEABC.dll
   string RPIN = E2EEABC.Client.EncryptForChangePin(E2EEABC.Client.ALGO_ID_SHA1,e2eesid,oldPassword,
          newPassword,publicKeyValue, serverRandom);
```

3.2.3 Parameters and Exceptions

Field	Description	Type
hashAlgorithmId	7.4 Hash Algorithm Id	byte
e2eeSid	E2EE session Id generated by UAS server	String
oldPin	User's old password in clear text	String
newPin	User's new password in clear text	String
publicKey	Public Key value used to encrypt user's clear password.	String
serverRandom	HSM-generated random number	String

3.2.4 Responses

Field	Description	Expected Result(Success)	Result (Fail)
retCode1	Error code for RPIN	0	6 E2EE Error Codes
invalidPin1	refers to RPIN	false	true
invalidPin	refers to RPIN2	false	true

4 AM5 XMLRPC Errors

4.1 Authentication Fault Codes

Please refer to Authentication Faults in **"AM5 XML-RPC and SOAP API Reference"** for all possible authentication faults.

4.2 Query Fault Codes

Please refer to Query Faults in **"AM5 XML-RPC and SOAP API Reference"** for all possible query faults.

4.3 System (Runtime) Fault Codes

Please refer to System (Runtime) Faults in **"AM5 XML-RPC and SOAP API Reference"** for all possible query faults.

5 SafeNet PHEFT HSM Native Errors

Definition in <eftapibase.h>	Error Code (Hexadecimal)	Error Code (Decimal)	Message From ErrToString() function
EFT_NO_ERROR	0x00	0	No error
EFT_DES_FAULT	0x01	1	DES Fault
EFT_PM_NOT_ENABLED	0x02	2	Function Not enabled
EFT_INCORRECT_MSG_LEN	0x03	3	Incorrect Message Length
EFT_INVALID_DATA	0x04	4	Invalid Data
EFT_INVALID_KEY_INDEX	0x05	5	Invalid Key Index
EFT_INVALID_PIN_FMT_SPEC	0x06	6	Invalid PIN Format Specifier
EFT_PIN_FORMAT_ERROR	0x07	7	PIN Format Error
EFT_VERIFICATION_FAILED	0x08	8	Verification Failure
EFT_TAMPERED	0x09	9	ESM Tampered
EFT_UNITIALISED_KEY	0x0A	10	Uninitialised Key
EFT_PIN_CHECK_LEN_ERROR	0x0B	11	PIN Check Length Error
EFT_INCONSISTENT_REQ_FIELDS	0x0C	12	Inconsistent Request Fields
EFT_INVALID_VISA_PIN_VER_KEY_IND	0x0D	15	Invalid VISA PIN Verification Key Index
	0x10	16	Internal Error
	0x1a	26	Duplicate Key or Record
EFT_INVALID_KEY_SPECIFIER_LENGTH	0x20	32	Invalid Key Specifier Length
EFT_UNSUPPORTED_KEY_SPECIFIER	0x21	33	Unsupported Key Specifier
EFT_INVALID_KEY_SPECIFIER_CONTENT	0x22	34	Invalid Key Specifier Content
EFT_INVALID_KEY_SPECIFIER_FORMAT	0x23	35	Invalid Key Specifier Format
EFT_INVALID_FUNCTION_MODIFIER	0x24	36	Invalid Function Modifier
	0x30	48	Invalid Admin Signature
	0x31	49	Unknown Error 49
	0x32	50	No Admin Session
	0x36	54	Device Error
	0x39	57	Public Key Pair Unavailable
	0x3a	58	Public Key Pair Generating
	0x3b	59	RSA Cipher Error
	0x40	64	Unsupported SEED Key
	0x70	112	Invalid PIN Block Flag
	0x71	113	Invalid PIN Block Random Flag
	0x72	114	Invalid PIN Block Delimiter
	0x73	115	Invalid PIN Block RB
	0x74	116	Invalid PIN Block RandNum
	0x75	117	Invalid PIN Block RA
	0x76	118	Invalid PIN
	0x77	119	Invalid PIN Length
	0x7f	127	Invalid Print Token
	0x80	128	OAEP Decode Error
	0x81	129	OAEP Invalid Header Byte
	0x82	130	OAEP Invalid PIN Block
	0x83	131	OAEP Invalid Random Number
EFT_CL_ERROR_OFFSET	0x100	256	Unknown Error 256
	0x101	257	ESM not found
	0x102	258	Initialisation failed
	0x103	259	Tcp host address error
	0x104	260	Cannot create TCP socket
	0x105	261	Cannot set TCP socket options
	0x106	262	Cannot connect on TCP socket
	0x107	263	Error closing TCP socket
	0x108	264	Error sending on TCP socket
	0x109	265	Timeout sending on TCP socket
	0x10a	266	Error receiving on TCP socket

	0x10b	267	TCP window full error
	0x10c	268	API timer thread error
	0x10d	269	API timer thread terminate error
	0x10e	270	API heartbeat thread error
	0x10f	271	API receive thread error
	0x110	272	Receive queue empty
	0x111	273	Error translating error code to string
	0x112	274	Timeout waiting to receive
	0x113	275	ESM not responding
	0x114	276	Callback function not registered
	0x115	277	Error opening serial comms port
	0x116	278	Error getting serial comms port state
	0x117	279	Error setting serial comms port state
	0x118	280	Error setting serial comms port mask
	0x119	281	Error setting serial comms port timeouts
	0x11a	282	Error reading from serial comms port
	0x11b	283	Error sending to serial comms port
	0x11c	284	Cannot find ethernet adapter
	0x11d	285	Cannot set ethernet packet filter settings
	0x11e	286	Error sending to ethernet interface
	0x11f	287	Error reading from ethernet interface
	0x120	288	Next ESM found
	0x121	289	Function Parameter Error
	0x122	290	Error Reading ESM
	0x123	291	Error Writing to ESM
EFT_PARAM_ERROR	0x125	293	Parameter Error
EFT_MEMORY_ERROR	0x126	294	Memory Allocation Error
EFT_CL_ERROR	0x127	295	Cryptolink Error
EFT_INTERNAL_ERROR	0x128	296	Internal Error
EFT_INVALID_BUFFER_LENGTH	0x129	297	Invalid Buffer Length
EFT_MAX_MSG_SIZE_EXCEEDED	0x12A	298	Maximum Message Length Exceeded
EFT_NO_SCRIPT_RETURNED_DATA	0x12B	299	Expected Response missing from Test Script
EFT_BAD_RETURNED_DATA	0x12C	300	Returned Data Doesn't Match Expected Response

6 E2EE Error Codes

Error Code	Description
-1	Invalid hex character
0	No error
1	Invalid hex character
10	Invalid PIN length
11	Invalid PIN string
16	Invalid PIN string
20	Invalid PIN block
21	Invalid random number length
22	Invalid random number
30	Invalid pin message
31	Invalid pin message length
40	Invalid encoded message length
41	Invalid RSA key length
42	Invalid RSA key error
43	Public key not found error

7 Appendixes

7.1 JavaScript Function for RPIN Generation

```
<script language="javascript" src="ame2eea.js"></script>
<script>
function onFormSubmit(form, userId, password, RPIN) {
// Additional validation if required
if (password.value.length < 4 || password.value.length > 14){
alert("The PIN is too short or too long");
return false;
}
// Encryption parameters
var e2eeSid = '<%=_e2EESid%>';
var pubKey = '<%=_e2EEPublicKey%>';
var randomNumber = '<%=_e2EERandomNum%>';
var oaepHashAlgo = 'SHA-1';
// Encrypt (get RPIN)
// ame2eea.encryptPinForAM(e2eeSid, publicKey, randomNumber, pin, oaepHashAlgo)
RPIN.value = ame2eea.encryptPinForAM(e2eeSid, pubKey, randomNumber, password.value, oaepHashAlgo);
password.value = ""; // clear unencrypted pin
return true;
</script>
}
```

7.2 JavaScript Function for Change Password RPIN Generation

```
<script language="javascript" src="ame2eea.js"></script>
<script>
HashMap parms = new HashMap();
function onFormSubmit(form, flduserid, fldclearpin1, fldclearpin2, fldclearpin3, fldoldpwd, fldnewpwd) {
// Additional validation if required
if (fldclearpin1.value == fldclearpin2.value) {
alert("New password must be different from old password");
return false;
}
if (fldclearpin2.value != fldclearpin3.value) {
alert("New password not match with retype new password");
return false;
}
if (fldclearpin1.value.length < 4 || fldclearpin1.value.length > 14 || fldclearpin2.value.length < 4
                          || fldclearpin2.value.length > 14) {
alert ('The old PIN or new PIN is too short or too long');
return false;
}
var regex=/^[0-9A-Za-z]+$/;
if (!regex.test(fldclearpin2.value)) {
alert("The new PIN contains illegal character. Please use alphanumeric ONLY.");
return false;
}
// Encryption parameters
var e2eeSid = '<%=_e2EESid%>';
var pubKey = '<%=_e2EEPublicKey%>';
var randomNumber = '<%=_e2EERandomNum%>';
var oaepHashAlgo = 'SHA-1';
// Encrypt (get RPIN)
// ame2eea.encryptChangePinForAM(e2eeSid, publicKey, randomNumber, oldPin, newPin, oaepHashAlgo)
fldnewpwd.value = ame2eea.encryptChangePinForAM(e2eeSid, pubKey, randomNumber, fldclearpin1.value,
                                        fldclearpin2.value, oaepHashAlgo);
fldoldpwd.value = "";
fldclearpin1.value = ""; // clear unencrypted pin
fldclearpin2.value = "";
fldclearpin3.value = "";
return true;
}</script>
```

7.3 Terms and Defination

Term	Defination
RPIN	HPIN encrypted by Pk
E2EELoginModuleId	configured in UAS Admin Console, that corresponds to a KTPV Index
KTPV	Symmetric Key in HSM that is used to protect HPIN (i.e. to create SPIN). Only one KTPV Index can be configured for all users.

SPIN or TPV(Transformed PIN Value)	HPIN encrypted by KTPV
STPV(Store TPV)	SPIN or TPV ready to be stored.
salt	Salt to be used for password encryption. The same salt need to be passed back upon calling.

7.4 Hash Algorithm Id

- **For Android / Blackberry / .NET**

Hash Algorithm Id constant class	Constant of the reference class	Value type	Meaning
AxMxE2EEClient	ALGO_ID_3DES_SHA1	byte	SHA 1 algorithm
	ALGO_ID_3DES_SHA224		SHA 224 algorithm
	ALGO_ID_3DES_SHA256		SHA 256 algorithm
	ALGO_ID_3DES_SHA384		SHA 384 algorithm
	ALGO_ID_3DES_SHA512		SHA 512 algorithm

- **For iPhone**

Method	Value type	Meaning
SHA1	Unsigned char	SHA 1 algorithm
SHA256		SHA 256 algorithm
SHA512		SHA 512 algorithm