TLS Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom Paris

January 28 2024

Expires: July 2024

Identity Module for TLS Version 1.3
draft-urien-tls-im-10.txt

Abstract

TLS 1.3 will be deployed in the Internet of Things ecosystem. In many IoT frameworks, TLS or DTLS protocols, based on pre-shared key (PSK), are used for device authentication. So PSK tamper resistance, is a critical market request, in order to prevent hijacking issues. If DH exchange is used with certificate bound to DH ephemeral public key, there is also a benefit to protect its signature procedure. The TLS identity module (im) MAY be based on secure element; it realizes some HKDF operations bound to PSK, and cryptographic signature if certificates are used. Secure Element form factor could be standalone chip, or embedded in SoC like eSIM.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 2024.

.

Table of Contents

1 Overview

TLS 1.3 [RFC8446] will be deployed in the Internet of Things ecosystem. In many IoT frameworks, TLS or DTLS protocols, based on pre-shared key (PSK), are used for device authentication. So PSK tamper resistance, is a critical market request, in order to prevent hijacking issues. If DH exchange is used with certificate bound to DH ephemeral public key, there is also a benefit to protect its signature procedure. The TLS identity module (im) MAY be based on secure element [ISO7816]; it realizes some HKDF [RFC5869] operations bound to PSK, and cryptographic signature if certificates are used. Secure Element form factor could be standalone chip or embedded in SOC like eSIM.

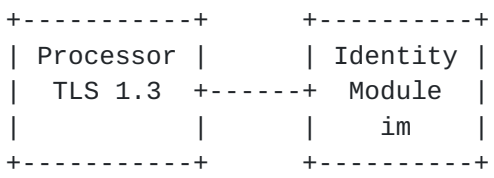

Figure 1. TLS 1.3 Identity Module (im)

The ISO7816 standards specify the binary encoding for ISO7816-4 commands and responses, refereed as Application Protocol Data Unit (APDU). APDUs can be exchanged with secure elements according to various transport protocols [GP-SPI-I2C] such as ISO7816-3 T=0, ISO7816-3 T=1, Inter Integrated Circuit (I2C) or Serial Peripheral Interface (SPI)

2 Protecting the Key Schedule for PSK

2.1 Context

According to [RFC8446] external PSKs MAY be provisioned outside of TLS.

The Early Secret (ESK) is computed according to relation:
ESK =HKDF-Extract(salt=0s,PSK) = HMAC(salt=0s,PSK)

The Binder Key (BSK) for outside provisioning is computed according to the relation:
BSK = Derive-Secret(ESK, "ext binder", "")

The Derived Secret (DSK) is computed according to the relation:
DSK= Derive-Secret(ESK, "derived", "")

The Finished External Key (FEK) is computed according to the relation:
FEK = KDF-Expand-Label(BSK, "finished", "", Hash.length)

For Derive-Secret procedures, "" is equivalent to the value hash(empty), whose size is hash-length.

2.2 Identity Module Procedures

The identity module MUST provide a "Keys Secure Generation and Storage" (KSGS) procedure, which computes and securely stores ESK, BSK and FEK keys.

The KSGS procedure MUST require administrative rights.

A set "Identity Module Key Procedures" (IMKP) of four procedures is required, in order to protect from public exposure ESK, BSK, and FEK:

- CETS: Client Early Traffic Secret
- EEMS: Early Exporter Master Secret
- HEDSK: HKDF-Extract from Derived Secret Key
- HBSK: HMAC from Binder Key Secret

These procedures MAY require user rights.

2.3 KSGS: Keys Secure Generation and Storage

The Identity module MUST provide a KSGS procedure, requiring administrative rights, which computes and securely stores ESK, BSK, DSK, and FEK. The KSGS procedure uses with a hash function (for example SHA256) identifies by an AlgoId attribute.

Input: AlgoId, salt, PSK
Output: Success or Failure

ESK, DSK, and BSK secret values are stored in the identity module.

HL16 : hash Length, 16 bits
HL8 : hash length, 8 bits
H0 : hash(empty)

ESK= HMAC(salt=0s,PSK)
DSK= HMAC(ESK,HL16||0d746c7331332064657269766564||HL8||H0||01)
BSK= HMAC(ESK,HL16||10746c733133206578742062696e646572||HL8||H0||01)
FEK= HMAC(BSK,HL16||0E746C7331332066696E69736865640001)

2.4 Identity Module Key Procedures (IMKP)

2.4.1 CETS: Client Early Traffic Secret

Input: Length, Message
Output: Client Early Traffic Secret or Failure

CETS(ClientHello) = Derive-Secret(ESK, "c e traffic", Message)
= HMAC(ESK, Length || 11746c7331332063206520747261666666963 ||
Message || 01)

Message is a hash value.

2.4.2 EEMS: Early Exporter Master Secret

Input: Length, Message
Output: Early Exporter Master Secret or Failure

EEMS(ClientHello) = Derive-Secret(ESK, "e exp master", Message)
= HMAC(ESK, Length || 12746c7331332065206578702066173746572 ||
Message || 01)

Message is a hash value

2.4.3 HEDSK: HKDF-Extract from Derived Secret Key

Input: DHE value
Output: Handshake Secret or Failure

HEDSK(DHE)= HKDF-Extract(salt=DSK,DHE) = HMAC(salt=DSK, DHE)

2.4.4 HBSK: HMAC from Binder Key Secret

Input: data
Output: HMAC(BSK, data) or Failure

HBSK(data) = HMAC(FEK, data)

Data is a hash value

3. Asymmetric Signature

The identity module MUST provide a "Generate Key" (GENKEY) procedure, in order to store or generate private asymmetric key and associated public key. This procedure MUST require administrative rights.

The procedure "Get Public Key" (GETPUB:) is required in order to read the public key value. This procedure MAY require user rights.

The procedure "Signature" (SIGN) is required in order to perform a raw signature for a digest value, computed from certificate. This procedure MAY require user rights.

The symmetric algorithm is identified by the AlgoId attribute. The key is identified by the KeyId attribute.

3.1 GENKEY

Input: AlgoId, KeyId
Output: Success or Failure

A private key is generated and stored in the identity module. A public key is computed from the private key.

3.2 GETPUB

Input: KeyId
Output: Public Key Value or Failure

3.3 SIGN

Input: KeyId, DigestValue
Output: Signature Value or Failure

4 Optional Procedures

In IoT context, the computing resources needed for supporting cryptographic procedures such as elliptic curves or true random number generators can be an issue. Optional procedures facilitate TLS1.3 support in such devices.

4.1 GENDHE

Input: AlgoId, PublicKey, KeyId
Output: The DHE value

A DHE is computed according to an input public key, and an algorithm identifier.
An ephemerous public key EPK is generated, which is identified by the KeyId attribute and can be retrieved by the GETEPK procedure.

4.2 GETEPK

Input: KeyId
Output: Error or ephemerous public key

This procedure returns the ephemerous public key (EPK), identified by the KeyId attribute, previously computed by GENDHE.

4.3 RAND

Input: Number of random bytes, Nr
Output: Nr bytes

This procedure generates Nr random bytes

5 Identity Module Procedures Summary

First column: The procedure name
Second column: The procedure status Mandatory (M) or Optional (O)
for PSK or PKI (e.g. signature generation)
Third column: Input parameters
Fourth column: Output vale
Fifth column: The mode, ADMinistrator or USER

Name	Status	Comment	Input	Output	Mode
KSGS	M PSK	Compute Secrets from PSK	AlgoId Salt PSK	none	ADM
GENKEY	M PKI	Generate Private and Public Key	AlgoId KeyId	none	ADM
CETS	M PSK	Compute Early Traffic Secret	Length Message	Client Early Traffic Secret	USER
EEMS	M PSK	Compute Early Exporter Master Secret	Length Message	Early Exporter Master Secret	USER
HEDSK	M PSK	Compute Handshake Secret	DHE	Handshake Secret	USER
HBSK	M PSK	Compute HMAC For Identity Binder	Data	HMAC For Identity Binder	USER
GETPUB	M PKI	Read Public Key	KeyId	Public Key	USER
SIGN	M PKI	Compute Signature	KeyId Data	Public Key	USER
GENDHE	O	Generate Pub.Key Compute DH	AlgoID PUB.Key KeyId	DH Value	USER
GETEPK	O	Read Ephemeris Public Key	KeyId	Public Key	USER
RAND	O	Generate Random Bytes	Number of Bytes	Random Bytes	USER

6. Secure Element as Identity Module

Secure elements are defined according to [ISO7816] standards. They support hash functions (sha256, sha384, sha512) and associated HMAC procedures. They also provide DH procedures in Z/pZ* groups, and elliptic curves. Open software can be released thanks to the Javacard standards, such as JC3.04, JC3.05, JC3.1.

This section is an illustration of binary encoding rules for secure element according to the T=1 ISO7816 protocol.

An ISO7816 command (TAPDU) is a set of bytes comprising a five bytes header and an optional payload (up to 255 bytes)

The header comprises the following five bytes
- CLA, Class
- INS, Instruction code
- P1, P1 byte
- P2, P2 byte
- P3, length of the payload, or number of expected bytes

The response comprises a payload (up to 255 bytes) and a two bytes status word SW=(SW1, SW2), 9000 meaning successful operation.

6.1 Administrator mode

The [ISO7816] command VERIFY (INS=0x20) SHOULD be used to enter the administrative mode.

Tx: CLA=00 INS=20 P1=00 P2=Adm P3=PIN-Length [PIN-Value]
Rx: 9000

6.2 User Mode

The [ISO7816] command VERIFY SHOULD be used to enter the user mode

Tx: CLA=00 INS=20 P1=00 P2=User P3=PIN-Length [PIN-Value]
Rx: 9000

6.3 KSGS: Keys Secure Generation and Storage

Length= 2 + Salt-Length + PSK-Length

Tx: CLA=00 INS=TLS13 P1=AlgoId P2=KSGS P3=Length Salt-Length [Salt-Value] PSK-Length [PSK-Value]
Rx: 9000

This procedure computes and stores ESK, BSK DSK and FEK.

6.3.1 Example

```
PSK=0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

Tx: CLA=00 INS=85 P1=00 P2=0A P3=23 01 00 20
0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
Rx:9000

Sha256(empty) =
E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855

ESK= HMAC-SHA256(0,PSK)
ESK= 23499E7EDF0FBE6BAA137DF0F23BECAEF722AD19FC262855409DE8CD8B3C897

DSK= HMAC-SHA256(ESK,0020 0d746c7331332064657269766564 20
E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855 01)
DSK=E8E7AC087158FC8440E41A12989F9194783764CD5FC36564028037F2C8206E96

BSK = HMAC-SHA256(ESK,0020 10746c7331332065787420626 96e646572 20
E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855 01)
BSK=4351F8A53AA85AC394AB04C516464CAB96E9340C269632D09899537887EE651F

FEK= HMAC-256(BSK, 0020 0E746C7331332066696E6973686564 00 01)
FEK=FCA24690D17DDE3F727D29D2186A5F83E1AEBD4889A4841793139168A65BFCB0
```

6.4 CETS: Client Early Traffic Secret

```
Length = 2 + Messages-Length
Hash-Length: the hash length (2 bytes)

Tx: CLA INS=TLS13 P1=CETS P2=ESK P3=Length Hash-Length Messages-
Length [Messages]
Rx:[Client Early Traffic Secret] SW
```

6.4.1 Example

```
Tx: CLA=00 INS=85 P1=00 P2=0B P3=03 0020 00
Rx: 0738A2B6F6FAA2AF5CDD9B6F0F2B232F19B3256A5926EAC600B911F91E98D2D4
9000

Message= NULL = 0s
[Client Early Traffic Secret] =
HMAC-SHA256(ESK, 0020 11746c733133206320652074726166666963 00 01)
```

6.5 EEMS: Early Exporter Master Secret

```
Length = 2 + Messages-Length
Hash-Length: the hash length (2 bytes)
```

Tx: CLA INS=TLS13 P1=EEMS P2=ESK P3=Length Hash-Length Messages-Length [Messages]
Rx: [Early Exporter Master Secret] SW

6.5.1 Example

Tx: CLA=00 INS=85 P1=01 P2=0B P3=03 0020 00
Rx: 9B7FC6A8F854C16A301DFC566859931DB5EE9A22793142A0C67159C445E7BEAB 9000

Message= NULL = 0s
[Early Exporter Master Secret] =
HMAC-SHA256(ESK, 0020 12746c7331332065206578702065d6173746572 00 01)

6.6 HEDSK: HKDF-Extract from Derived Secret Key

Tx: CLA INS=TLS13 P1=0 P2=HEDSK P3=Data-Length [Data]
Rx: [HMAC(Data,DSK)] SW

6.6.1 Example

Tx: CLA=00 INS=85 P1=00 P2=0E P3=01 00
Rx: 7092C2117D67E6AEB5C5FDF5E6D9C70FBDC69B374E914C26AB08A122483D0E73

DHE=NULL=0s
HMAC-256(DSK,DHE)= HMAC-256(DSK,0s)

6.7 HBSK: HMAC from Binder Key Secret

Tx: CLA INS=TLS13 P1=0 P2=HBSK P3=Data-Length [Data]
Rx: [HMAC(FEK,data)] SW

6.7.1 Example

Tx: CLA=00 INS=85 P1=00 P2=0C P3=01 00
Rx: 3E015D850B89C2470D4C49D4BD8E7C76F2B74175DDD85F393569315DA15480A4

Data=NULL=0s
HMAC-256(FEK,Data)= HMAC-256(DSK,0s)

6.8 Signature Procedures

6.8.1 Keys Generation

Select Identity Module Application (AID= 010203040500)
Tx: CLA=00 INS=A4 P1=04 P2=00 P3=06 01 02 03 04 05 00
Rx: 9000

```
Verify Administrator PIN (PIN= "00000000")
```

```
Tx: CLA=00 INS=20 P1=00 P2=01 P3=08 30 30 30 30 30 30 30 30
Rx: 9000

Clear Key (P2=KeyId=0)
Tx: CLA=00 INS=81 P1=00 P2=00 P3=00
Rx: 9000

Init Curve secp256r1 (P1 = idCurve, P2=KeyId)
Tx: CLA=00 INS=89 P1=00 P2=00 P3=00
Rx: 9000

GenKey (P2=KeyId)
Tx: CLA=00 INS=82 P1=00 P2=00 P3=00
Rx:9000

Read PublicKey (P2=KeyId)
Tx: CLA=00 INS=84 P1=06 P2=00 P3=00
Rx: 0041049E92726E24A548BB69ADA51103F265AA9B9F304E25971427D79EFAF471
    889CCC52FD8B05A729A400105C06AF99592535A4EDF338B5A37BB6089D3B11E7
    1B847B 9000

Read PrivateKey (P2= KeyId)
Tx: CLA=00 INS=84 P1=07 P2=00 p3=00
Rx: 00208E8793D5C399659D8A35B585534B5D9D0FAB37AD3FC7E8B43373C4BAD81E
    9000
```

6.8.2 Keys Setting

```
Select Identity Module Application (AID= 010203040500)
Tx: CLA=00 INS=A4 P1=04 P2=00 P3=06 01 02 03 04 05 00
Rx: 9000

Verify Administrator PIN (PIN= "00000000")
Tx: CLA=00 INS=20 P1=00 P2=01 P3=08 30 30 30 30 30 30 30 30
Rx: 9000

Clear Key (P2=KeyId=0)
Tx: CLA=00 INS=81 P1=00 P2=00 P3=00
Rx: 9000

Init Curve secp256r1 (P1 = idCurve, P2=KeyId)
Tx: CLA=00 INS=89 P1=00 P2=00 P3=00
Rx: 9000

Set PrivateKey (P2=KeyId)
Tx: CLA=00 INS=88 P1=07 P2=00 P3=20
2e86bdd6d3b241ddbd00999f6a0ac1cb546d2bfb55744dca40f0268ac2bf7338
Rx: 9000

Set PublicKey (P2=KeyId)
```

```
Tx: CLA=00 INS=88 P1=06 P2=00 P3=41
045c8c90d0859dd96c722a589c4b62047ff01323cc74383e0e8eb80bea4ea45e55b8
5499abd39d719885e874ed3f6327960d519ba25423c3fbdc14e6fd0cd5edee
Rx: 9000
```

6.8.3 Signature

```
Select Identity Module Application (AID= 010203040500)
Tx: CLA=00 INS=A4 P1=04 P2=00 P3=06 01 02 03 04 05 00
Rx: 9000
Verify User PIN (PIN= "0000")
CLA=00 INS=20 P1=00 P2=00 P3=04 30 30 30 30

ECDSA secp256r1 Signature (P2=KeyId)
Tx: CLA=00 INS=80 P1=00 P2=00 P3=20
0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
Rx: 0047304502206BB1B02742C90B5FEAD3EF34F87B49D2A87F846F0368D0DBB3A
    0E9D9F3ABC450022100A0178CDE84FB9ACA4662ECC68638437D46EC27B69657
    8F8080E43ACCA4B35586
```

6.9 GENDHE

```
Tx: CLA INS=GENDHE P1=AlgoId P2=KeyId P3=Key-Length [Public Key]
Rx: [DHE] SW
```

6.9.1 Example

```
Tx: CLA=00 INS=8A P1=00=Secp256r1 P2=FF P3=41
4104C4B5F7682C374AAD1C9125C2F225D343A8986C8E0A475E4003F6C98DA13F99
9E80A55E66F0644E84F7F6503615B9EC4CB7C2844AF6BE7F9091BF319B0291A2D8
Rx: 1BEE561B95F9EC99EE0E49F28E415D4F74580ACB8D9E0019BD3A7974FF3148E5
```

6.10 GETEPK

```
Tx: CLA INS=GETEPK P1=06 P2=KeyId P3=2+Key-length
Rx: [Key-Length] [Public Key] SW
```

6.10.1 Example

```
Tx: CLA=00 INS=84 P1=06 P2=FF P3=43
Rx: 004104CD8EF9695FAC953D89B9C91B994DC77F3140BDEDF54AABF63521548AB9
8031942C829FC5D958F143AA09E622E6CB190D7A91773E1794F792E1D4D7E1B84603
FF9000
```

6.11 RAND

```
Tx: CLA INS=RAND P1=00 P2=00 P3=[Number-Of-bytes]
Rx: [Number-Of-Bytes] SW
```

6.11.1 Example

```
Tx: CLA=00 INS=8B P1=00 P2=00 P3=20
Rx: 85DEE2DD24BF79D8CCB6D21C1F515CE040A2E13B8C98177822BD3B66876CD9A1
9000
```

7. A simple Identity Module code for Javacard 3.04

An example of TLS-IM code is available at [IM-JC].

```
package im;

import javacard.framework.*;
import javacard.security.* ;
import javacardx.crypto.*  ;

public class im extends Applet
{
final static byte  INS_SIGN             = (byte)  0x80 ;
final static byte  INS_CLEAR_KEYPAIR    = (byte)  0x81 ;
final static byte  INS_GEN_KEYPAIR      = (byte)  0x82 ;
final static byte  INS_GET_KEY_PARAM    = (byte)  0x84 ;
final static byte  INS_HMAC             = (byte)  0x85 ;
final static byte  INS_GET_STATUS       = (byte)  0x87 ;
final static byte  INS_SET_KEY_PARAM    = (byte)  0x88 ;
final static byte  INS_INIT_CURVE       = (byte)  0x89 ;
final static byte  INS_SELECT           = (byte)  0xA4 ;
public final static byte INS_VERIFY     = (byte)  0x20 ;
public final static byte INS_CHANGE_PIN = (byte)  0x24 ;

public final static short N_KEYS     = (short) 16;
public final static byte[] VERSION= {(byte)1,(byte)0};

KeyPair[] ECCkp      = null ;
Signature ECCsig     = null ;
MessageDigest sha256 = null ;

short status=0  ;
byte [] DB = null                              ;
public final static short DBSIZE = (short)320 ;

private static OwnerPIN UserPin=null;

private static final  byte[]   MyPin =
{(byte)0x30,(byte)0x30,(byte)0x30,(byte)0x30,
 (byte)0xFF,(byte)0xFF,(byte)0xFF,(byte)0xFF};

private static OwnerPIN AdminPin=null;

private static final byte[]  OpPin =
{(byte)0x30,(byte)0x30,(byte)0x30,(byte)0x30,
 (byte)0x30,(byte)0x30,(byte)0x30,(byte)0x30};

private final static short SW_VERIFICATION_FAILED  = (short)0x6300;
private final static short SW_PIN_VERIFICATION_REQUIRED  =
(short)0x6380;
final static  short SW_KPUB_DEFINED    = (short)0x6401;
final static  short SW_KPRIV_DEFINED   = (short)0x6402;
```

```
final static  short SW_KPRIV_UNDEFINED = (short)0x6403;
```

```
final static  short SW_GENKEY_ERROR    = (short)0x6D10;
final static  short SW_SIGN_ERROR      = (short)0x6D20;
final static  short SW_DUMP_KEYS_PAIR  = (short)0x6D30;
final static  short SW_SET_KEY_PARAM   = (short)0x6D40;

private final static byte [] ParamA1 =
{(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x01,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x00,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,
(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,
(byte)0xff,(byte)0xfc};

private final static byte [] ParamB1 =
{(byte)0x5a,(byte)0xc6,(byte)0x35,(byte)0xd8,(byte)0xaa,(byte)0x3a,
(byte)0x93,(byte)0xe7,(byte)0xb3,(byte)0xeb,(byte)0xbd,(byte)0x55,
(byte)0x76,(byte)0x98,(byte)0x86,(byte)0xbc,(byte)0x65,(byte)0x1d,
(byte)0x06,(byte)0xb0,(byte)0xcc,(byte)0x53,(byte)0xb0,(byte)0xf6,
(byte)0x3b,(byte)0xce,(byte)0x3c,(byte)0x3e,(byte)0x27,(byte)0xd2,
(byte)0x60,(byte)0x4b};

private final static byte [] ParamField1=
{(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x01,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x00,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,
(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,
(byte)0xff,(byte)0xff};

private final static byte [] ParamG1=
{(byte)0x04,(byte)0x6b,(byte)0x17,(byte)0xd1,(byte)0xf2,(byte)0xe1,
(byte)0x2c,(byte)0x42,(byte)0x47,(byte)0xf8,(byte)0xbc,(byte)0xe6,
(byte)0xe5,(byte)0x63,(byte)0xa4,(byte)0x40,(byte)0xf2,(byte)0x77,
(byte)0x03,(byte)0x7d,(byte)0x81,(byte)0x2d,(byte)0xeb,(byte)0x33,
(byte)0xa0,(byte)0xf4,(byte)0xa1,(byte)0x39,(byte)0x45,(byte)0xd8,
(byte)0x98,(byte)0xc2,(byte)0x96,(byte)0x4f,(byte)0xe3,(byte)0x42,
(byte)0xe2,(byte)0xfe,(byte)0x1a,(byte)0x7f,(byte)0x9b,(byte)0x8e,
(byte)0xe7,(byte)0xeb,(byte)0x4a,(byte)0x7c,(byte)0x0f,(byte)0x9e,
(byte)0x16,(byte)0x2b,(byte)0xce,(byte)0x33,(byte)0x57,(byte)0x6b,
(byte)0x31,(byte)0x5e,(byte)0xce,(byte)0xcb,(byte)0xb6,(byte)0x40,
(byte)0x68,(byte)0x37,(byte)0xbf,(byte)0x51,(byte)0xf5};

private final static short ParamK1 = (short) 0x0001;

private final static byte [] ParamR1=
{(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0x00,(byte)0x00,
(byte)0x00,(byte)0x00,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,
(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xff,(byte)0xbc,(byte)0xe6,
(byte)0xfa,(byte)0xad,(byte)0xa7,(byte)0x17,(byte)0x9e,(byte)0x84,
```

```
(byte)0xf3,(byte)0xb9,(byte)0xca,(byte)0xc2,(byte)0xfc,(byte)0x63,
(byte)0x25,(byte)0x51};
```

```
private byte []   ESK = new byte[32]; // Early Secret Key
private byte []   HSK = new byte[32]; // Handshake Secret Key
private byte []  eBSK = new byte[32]; // Binder Secret Key
private byte []  rBSK = new byte[32]; // Binder Secret Key
private byte [] feBSK = new byte[32]; // Finished Binder Secret Key
private byte [] frBSK = new byte[32]; // Finished Binder Secret Key

private final static byte  EXTRACT_EARLY  =  (byte)0x0A;
private final static byte  EXPAND_EARLY   =  (byte)0x0B;
private final static byte  HMAC_EBSK      =  (byte)0x0C;
private final static byte  HMAC_RBSK      =  (byte)0x0D;
private final static byte  EXTRACT_HANDSHAKE =  (byte)0x0E;

private byte [] derived =
{(byte)0x00,(byte)32,(byte)13,(byte)'t',(byte)'l',(byte)'s',
(byte)'1',(byte)'3',(byte)' ',(byte)'d',(byte)'e',(byte)'r',
(byte)'i',(byte)'v',(byte)'e',(byte)'d',
(byte)0x20,(byte)0xE3,(byte)0xB0,(byte)0xC4,(byte)0x42,(byte)0x98,
(byte)0xFC,(byte)0x1C,(byte)0x14,(byte)0x9A,(byte)0xFB,(byte)0xF4,
(byte)0xC8,(byte)0x99,(byte)0x6F,(byte)0xB9,(byte)0x24,(byte)0x27,
(byte)0xAE,(byte)0x41,(byte)0xE4,(byte)0x64,(byte)0x9B,(byte)0x93,
(byte)0x4C,(byte)0xA4,(byte)0x95,(byte)0x99,(byte)0x1B,(byte)0x78,
(byte)0x52,(byte)0xB8,(byte)0x55,(byte)1};

private byte [] ext_binder =
{(byte)0x00,(byte)32,(byte)16,(byte)'t',(byte)'l',(byte)'s',
(byte)'1',(byte)'3',(byte)' ',(byte)'e',(byte)'x',(byte)'t',
(byte)' ',(byte)'b',(byte)'i',(byte)'n',(byte)'d',(byte)'e',
(byte)'r',(byte)0x20,(byte)0xE3,(byte)0xB0,(byte)0xC4,(byte)0x42,
(byte)0x98,(byte)0xFC,(byte)0x1C,(byte)0x14,(byte)0x9A,(byte)0xFB,
(byte)0xF4,(byte)0xC8,(byte)0x99,(byte)0x6F,(byte)0xB9,(byte)0x24,
(byte)0x27,(byte)0xAE,(byte)0x41,(byte)0xE4,(byte)0x64,(byte)0x9B,
(byte)0x93,(byte)0x4C,(byte)0xA4,(byte)0x95,(byte)0x99,(byte)0x1B,
(byte)0x78,(byte)0x52,(byte)0xB8,(byte)0x55,(byte)0x01};

private byte [] res_binder   =
{(byte)0x00,(byte)32,(byte)16,(byte)'t',(byte)'l',(byte)'s',
(byte)'1',(byte)'3',(byte)' ',(byte)'r',(byte)'e',(byte)'s',
(byte)' ',(byte)'b',(byte)'i',(byte)'n',(byte)'d',(byte)'e',
(byte)'r',(byte)0x00,(byte)0x01};

private byte [] c_e_traffic  =
{(byte)17,(byte)'t',(byte)'l',(byte)'s',(byte)'1',(byte)'3',
(byte)' ',(byte)'c',(byte)' ',(byte)'e',(byte)' ',(byte)'t',
(byte)'r',(byte)'a',(byte)'f',(byte)'f',(byte)'i',(byte)'c'};
```

```
private byte [] c_exp_master =
{(byte)18,(byte)'t',(byte)'l',(byte)'s',(byte)'1',(byte)'3',
(byte)' ',(byte)'e',(byte)' ',(byte)'e',(byte)'x',(byte)'p',
(byte)' ',(byte)'m',(byte)'a',(byte)'s',(byte)'t',(byte)'e',
(byte)'r'};

private byte [] finished =
{(byte)0x00,(byte)32,(byte)14,(byte)'t',(byte)'l',(byte)'s',
(byte)'1',(byte)'3',(byte)' ',(byte)'f',(byte)'i',(byte)'n',
(byte)'i',(byte)'s', (byte)'h',(byte)'e',(byte)'d',(byte)0,
(byte)1};


public void process(APDU apdu) throws ISOException
{ short adr=0,len=0,index=0,readCount=0;

byte[] buffer = apdu.getBuffer() ;

byte cla = buffer[ISO7816.OFFSET_CLA];
byte ins = buffer[ISO7816.OFFSET_INS];
byte P1  = buffer[ISO7816.OFFSET_P1] ;
byte P2  = buffer[ISO7816.OFFSET_P2] ;
byte P3  = buffer[ISO7816.OFFSET_LC] ;

adr = Util.makeShort(P1,P2)       ;
len = Util.makeShort((byte)0,P3)  ;

switch (ins)
{

case INS_SELECT:
readCount = apdu.setIncomingAndReceive();
return;

case INS_GET_STATUS:
Util.arrayCopyNonAtomic(VERSION,(short)0,buffer,(short)0,(short)VERS
ION.length);
Util.setShort(buffer,(short)VERSION.length,status);
apdu.setOutgoingAndSend((short)0,(short)(2+VERSION.length));
break;

case INS_VERIFY:
readCount = apdu.setIncomingAndReceive();
if (P2 == (byte)1)
{  if (readCount != (short)8)
   ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
   verify(AdminPin,buffer) ;
   if(AdminPin.isValidated()) UserPin.resetAndUnblock();
```

```
else if (P2 == (byte)0xFF)
{  if (readCount != (short)8)
   ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
   verify(AdminPin,buffer) ;
   if(AdminPin.isValidated())
   {  UserPin.resetAndUnblock();
      UserPin.update(MyPin,(short)0,(byte)8) ;
   }
}
}
else if (P2 == (byte)0)
{  if (readCount > (short)8)
   ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
   verify(UserPin,buffer);
}
else
ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
break;


case INS_CHANGE_PIN:
readCount = apdu.setIncomingAndReceive() ;
if (readCount != (short)16)
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
buffer[4]=(byte)8;
if (P2 == (byte)1)
{  verify(AdminPin,buffer) ;
   AdminPin.update(buffer,(short)13,(byte)8);
}
else if  (P2 == (byte)0)
{  verify(UserPin,buffer)  ;
   UserPin.update(buffer,(short)13,(byte)8);
}
else
ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
break;


case INS_HMAC:


readCount = apdu.setIncomingAndReceive();
len = Util.makeShort((byte)0,buffer[(short)4]);

if (len != readCount)
ISOException.throwIt(ISO7816.SW_CORRECT_LENGTH_00);

else if ( (!AdminPin.isValidated()) && (!UserPin.isValidated()) )
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
```

```
if (P2 == (byte)2) // Compute HMAC
{  len = Util.makeShort((byte)0,buffer[(short)5]) ;
   hmac(buffer, (short)6, len, buffer, (short)(7+len),
        Util.makeShort((byte)0, buffer[(short)(6+len)]),
        sha256, buffer,(short)0,true);
   apdu.setOutgoingAndSend((short)0,(short)sha256.getLength());
}

else if (P2 == EXTRACT_EARLY)
{ len = Util.makeShort((byte)0,buffer[(short)5]); //HMAC: key-length
  hmac(buffer,(short)6,len,buffer,(short)(7+len),
        Util.makeShort((byte)0,buffer[(short)(6+len)]),
        sha256, buffer,(short)0,true);
   Util.arrayCopyNonAtomic(buffer,(short)0,ESK,(short)0,
                           (short)ESK.length);
   Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)32,
                           (short)32);


   hmac(ESK,(short)0,(short)ESK.length,
        derived,(short)0,(short)derived.length,
        sha256,
        buffer,(short)0,true);
   Util.arrayCopyNonAtomic(buffer,(short)0,HSK,(short)0,
                           (short)HSK.length);
   Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)64,
                           (short)32);


   hmac(ESK,(short)0,(short)ESK.length,
        ext_binder,(short)0,(short)ext_binder.length,
        sha256,
        buffer,(short)0,true);
    Util.arrayCopyNonAtomic(buffer,(short)0,eBSK,(short)0,
                            (short)eBSK.length);
    Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)96,
                            (short)32);


    hmac(ESK,(short)0,(short)ESK.length,
         res_binder,(short)0,(short)res_binder.length,
         sha256,
         buffer,(short)0,true);
    Util.arrayCopyNonAtomic(buffer,(short)0,rBSK,(short)0,
                            (short)rBSK.length);
    Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)128,
                            (short)32);
```

```
    hmac(eBSK,(short)0,(short)eBSK.length,
         finished,(short)0,(short)finished.length,
         sha256,
         buffer,(short)0,true);
    Util.arrayCopyNonAtomic(buffer,(short)0,feBSK,(short)0,
                            (short)feBSK.length);
    Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)160,
                            (short)32);


    hmac(rBSK,(short)0,(short)rBSK.length,
               finished,(short)0,(short)finished.length,
               sha256,
               buffer,(short)0,true);
    Util.arrayCopyNonAtomic(buffer,(short)0,frBSK,(short)0,
                           (short)frBSK.length);
    Util.arrayCopyNonAtomic(buffer,(short)0,buffer,(short)192,
                           (short)32);

    If (P1==(byte)0xFF)
    apdu.setOutgoingAndSend((short)32,(short)192);
    return ;
}

else if (P2 == EXPAND_EARLY)
{  len = Util.makeShort((byte)0,buffer[(short)7]);  // data length
   if (P1 == (byte)0)
   {
    Util.arrayCopyNonAtomic(buffer,(short)5,buffer,(short)0,
                                   (short)2);
    Util.arrayCopyNonAtomic(buffer,(short)7,
                            buffer,(short)(2+ c_e_traffic.length),
                            (short)(readCount-2));
    Util.arrayCopyNonAtomic(c_e_traffic,(short)0,buffer,(short)2,
                            (short)c_e_traffic.length);
    buffer[(short)(readCount + c_e_traffic.length)] = (byte)0x01;
    hmac(ESK,(short)0,(short)ESK.length,
         buffer,(short)0,(short)(readCount+c_e_traffic.length+1),
         sha256,
         buffer,(short)0,true);
         apdu.setOutgoingAndSend((short)0,(short)32);
         return;
    }
    else if (P1 == (byte)1)
    {
     Util.arrayCopyNonAtomic(buffer,(short)5,buffer,(short)0,
                             (short)2);
     Util.arrayCopyNonAtomic(buffer,(short)7,buffer,
```

```
                short)(2+ c_exp_master.length),
                (short)(readCount-2));
```

```
     Util.arrayCopyNonAtomic(c_exp_master,(short)0,buffer,(short)2,
                             (short)c_exp_master.length);
     buffer[(short)(readCount + c_exp_master.length)] = (byte)0x01;
     hmac(ESK,(short)0,(short)ESK.length,
          buffer,(short)0,(short)(readCount+c_exp_master.length+1),
          sha256,
          buffer,(short)0,true);
     apdu.setOutgoingAndSend((short)0,(short)32);
     return;
    }
    else
    ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);

else if ( P2 == HMAC_RBSK)
{  hmac(frBSK,(short)0,(short)rBSK.length,
        buffer,(short)5,readCount,
        sha256,
        buffer,(short)0,true);
        apdu.setOutgoingAndSend((short)0,(short)sha256.getLength());
}

else if (P2 == HMAC_EBSK)
{  hmac(feBSK,(short)0,(short)eBSK.length,
        buffer,(short)5,readCount,
        sha256,
        buffer,(short)0,true);
        apdu.setOutgoingAndSend((short)0,(short)sha256.getLength());
}

else if (P2 == EXTRACT_HANDSHAKE )
{  hmac(HSK,(short)0,(short)HSK.length,
        buffer,(short)5,readCount,
        sha256,
        buffer,(short)0,true);
        apdu.setOutgoingAndSend((short)0,(short)32);
}

else
ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
break;


case INS_SIGN:
readCount = apdu.setIncomingAndReceive();
if ( (!AdminPin.isValidated()) && (!UserPin.isValidated()) )
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);

index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
```

```
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if (!ECCkp[index].getPublic().isInitialized())
```

```
ISOException.throwIt(SW_KPUB_DEFINED);
if (!ECCkp[index].getPrivate().isInitialized())
ISOException.throwIt(SW_KPRIV_DEFINED);

switch (P1)
{
 case (byte)0: // RAW 256 bits
 case (byte)33:// ALG_ECDSA_SHA_256
               len= EccSign(ECCkp[index],buffer,P1) ;
               apdu.setOutgoingAndSend((short)0,len);
               break;
 default:
         ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
         break;
}
break;


case INS_CLEAR_KEYPAIR:

if ( !AdminPin.isValidated())
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if (ECCkp[index].getPublic().isInitialized())
ECCkp[index].getPublic().clearKey();
if (ECCkp[index].getPrivate().isInitialized())
ECCkp[index].getPrivate().clearKey();
break;

case INS_GEN_KEYPAIR: // Generate KeyPair

if ( !AdminPin.isValidated())
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if (ECCkp[index].getPublic().isInitialized())
ISOException.throwIt(SW_KPUB_DEFINED);
if (ECCkp[index].getPrivate().isInitialized())
ISOException.throwIt(SW_KPRIV_DEFINED);
len=this.GenECCkp(ECCkp[index]);
break;

case INS_GET_KEY_PARAM:

if ( (!AdminPin.isValidated()) && (!UserPin.isValidated())  )
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
```

```
index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
```

```
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if ( (P1 == (byte)7) && !AdminPin.isValidated())
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
if ( (P1 == (byte)6) && !ECCkp[index].getPublic().isInitialized())
ISOException.throwIt(SW_KPUB_DEFINED);
if ( (P1 == (byte)7) && !ECCkp[index].getPrivate().isInitialized())
ISOException.throwIt(SW_KPRIV_DEFINED)
try
{  switch (P1)
   {  case 0:
       len= ((ECPublicKey)ECCkp[index].getPublic())
             .getA(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       case 1:
       len= ((ECPublicKey) ECCkp[index].getPublic())
             .getB(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       case 2:
       len= ((ECPublicKey) ECCkp[index].getPublic())
             .getField(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       case 3:
       len= ((ECPublicKey)ECCkp[index].getPublic())
              .getG(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       case 4:
       len= ((ECPublicKey) ECCkp[index].getPublic()).getK();
       Util.setShort(buffer,(short)2,len);
       Util.setShort(buffer,(short)0,(short)2);
       apdu.setOutgoingAndSend((short)0,(short)4);
       break;

       case 5:
       len= ((ECPublicKey) ECCkp[index].getPublic())
             .getR(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
```

```
        break;
```

```
       case (byte)6:
       len= ((ECPublicKey) ECCkp[index].getPublic())
             .getW(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       case (byte)7:
       len= ((ECPrivateKey)ECCkp[index].getPrivate())
             .getS(buffer,(short)(2));
       Util.setShort(buffer,(short)0,len);
       apdu.setOutgoingAndSend((short)0,(short)(len+2));
       break;

       default:
       ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
       break;
   }
}
catch (CryptoException e)
{ISOException.throwIt(SW_DUMP_KEYS_PAIR);
 break;
}

break;


case INS_SET_KEY_PARAM:

readCount = apdu.setIncomingAndReceive();

if ( !AdminPin.isValidated())
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if ( (P1 == (byte)6) && ECCkp[index].getPublic().isInitialized())
ISOException.throwIt(SW_KPUB_DEFINED);
if ( (P1 == (byte)7) && ECCkp[index].getPrivate().isInitialized())
ISOException.throwIt(SW_KPRIV_DEFINED);

try
{  switch (P1)
   { case (byte)0:
     ((ECPublicKey)ECCkp[index].getPublic())
       .setA(buffer,(short)5,len);
     ((ECPrivateKey)ECCkp[index].getPrivate())
      .setA(buffer,(short)5,len);
     break;
```

```
case (byte)1:
((ECPublicKey)ECCkp[index].getPublic())
 .setB(buffer,(short)5,len);
((ECPrivateKey)ECCkp[index].getPrivate())
 .setB(buffer,(short)5,len);
break;

case (byte)2:
((ECPublicKey)ECCkp[index].getPublic())
.setFieldFP(buffer,(short)5,len) ;
((ECPrivateKey)ECCkp[index].getPrivate())
.setFieldFP(buffer,(short)5,len);
break;

case (byte)3:
((ECPublicKey)ECCkp[index].getPublic())
 .setG(buffer,(short)5,len) ;
((ECPrivateKey)ECCkp[index].getPrivate())
 .setG(buffer,(short)5,len);
break;

case (byte)4:
((ECPublicKey)ECCkp[index].getPublic())
 .setK(Util.makeShort(buffer[5],buffer[6])) ;
((ECPrivateKey)ECCkp[index].getPrivate())
 .setK(Util.makeShort(buffer[5],buffer[6]));
break;

case (byte)5:
((ECPublicKey)ECCkp[index].getPublic())
 .setR(buffer,(short)5,len);
((ECPrivateKey)ECCkp[index].getPrivate())
 .setR(buffer,(short)5,len);
break;

case (byte)6:
((ECPublicKey)ECCkp[index].getPublic())
 .setW(buffer,(short)5,len) ;
break;

case (byte)7:
((ECPrivateKey)ECCkp[index].getPrivate())
 .setS(buffer,(short)5,len);
break;

default:
ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
break;
```

```
    }
}
```

```
catch (CryptoException e)
{ISOException.throwIt(SW_SET_KEY_PARAM);
 break;
}

break;


case INS_INIT_CURVE:

if ( !AdminPin.isValidated())
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
index= Util.makeShort((byte)0,P2);
if ( (index <0) || (index >= N_KEYS))
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
if ( (P1 == (byte)6) && ECCkp[index].getPublic().isInitialized() )
ISOException.throwIt(SW_KPUB_DEFINED);
if ((P1 == (byte)7) && ECCkp[index].getPrivate().isInitialized())
ISOException.throwIt(SW_KPRIV_DEFINED);

switch((byte)P1)
{  case (byte)0:
   case (byte)1:
   (ECPublicKey)ECCkp[index].getPublic())
    .setA(ParamA1,(short)0,(short)ParamA1.length) ;
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setA(ParamA1,(short)0,(short)ParamA1.length);
   ((ECPublicKey)ECCkp[index].getPublic())
    .setB(ParamB1,(short)0,(short)ParamB1.length) ;
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setB(ParamB1,(short)0,(short)ParamB1.length);
    ((ECPublicKey)ECCkp[index].getPublic())
    .setFieldFP(ParamField1,(short)0,(short)ParamField1.length);
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setFieldFP(ParamField1,(short)0,(short)ParamField1.length);
   ((ECPublicKey)ECCkp[index].getPublic())
    .setG(ParamG1,(short)0,(short)ParamG1.length) ;
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setG(ParamG1,(short)0,(short)ParamG1.length);
   ((ECPublicKey)ECCkp[index].getPublic())
    .setK(ParamK1) ;
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setK(ParamK1);
   ((ECPublicKey)ECCkp[index].getPublic())
    .setR(ParamR1,(short)0,(short)ParamR1.length) ;
   ((ECPrivateKey)ECCkp[index].getPrivate())
    .setR(ParamR1,(short)0,(short)ParamR1.length);
    break;
```

```
   default:
   ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
   break;
  }
break;

default:
ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}

}

public short EccSign(KeyPair ECCkeyPair, byte [] buf, byte mode)
{ short len,sLen=(short)0;
  len= Util.makeShort((byte)0,buf[4]);
  Util.arrayCopy(buf,(short)5,buf,(short)2,len) // Sign
  try
  {  if (mode == (byte)0)// default
      { ECCsig.init(ECCkeyPair.getPrivate(),Signature.MODE_SIGN);
        sLen = ECCsig.signPreComputedHash(buf,(short)2,len buf,
             (short)(2+len));
      }
     else
      { ECCsig.init(ECCkeyPair.getPrivate(),Signature.MODE_SIGN);
        sLen = ECCsig.sign(buf, (short)2, len, buf, (short)(2+len));
      }
   }
  catch (CryptoException e)
  {ISOException.throwIt(SW_SIGN_ERROR);
   return (short)0;
  }

Util.arrayCopy(buf,(short)(2+len),buf,(short)2,sLen);
Util.setShort(buf,(short)0,sLen);
return(short)(sLen+2);
}

public short GenECCkp(KeyPair ECCkeyPair)
{ short len;
  try
  { ECCkeyPair.genKeyPair(); }
  catch (CryptoException e)
  { ISOException.throwIt(SW_GENKEY_ERROR);
    return (short)0;
  }
return 0;
}
```

```
public void verify(OwnerPIN pin,byte [] buffer) throws ISOException
  {short i,x;
   x = Util.makeShort((byte)0,buffer[4]);
   for(i=x;i<(short)8;i=(short)(i+1))
   buffer[(short)(5+i)]=(byte)0xFF;
   if ( pin.check(buffer, (short)5,(byte)8) == false )
   ISOException.throwIt((short)((short)SW_VERIFICATION_FAILED |
                                (short)pin.getTriesRemaining()));
  }

public static final short DB_off = (short)0 ;

public void hmac
( byte []  k,short k_off, short lk, // Secret key
  byte []  d,short d_off,short ld,  // data
  MessageDigest md,
  byte out[], short out_off, boolean init)
{
short i,DIGESTSIZE, DIGESTSIZE2=(short)64,BLOCKSIZE=(short)128;
DIGESTSIZE=(short)md.getLength();
if (md.getAlgorithm() == md.ALG_SHA_512)
{ DIGESTSIZE2= (short)64; BLOCKSIZE  = (short)128; }
else if (md.getAlgorithm() == md.ALG_SHA_256)
{ DIGESTSIZE2= (short)32; BLOCKSIZE = (short)64;}

if (init)
{ if (lk > (short)BLOCKSIZE )
  { md.reset();
    md.doFinal(k,k_off,lk,k,k_off);
    lk = DIGESTSIZE ;
  }
  for (i = 0 ; i < lk ; i=(short)(i+1))
  DB[(short)(i+DB_off+BLOCKSIZE+DIGESTSIZE2)] =
  (byte)(k[(short)(i+k_off)] ^ (byte)0x36) ;
  Util.arrayFillNonAtomic (
  DB,(short)(BLOCKSIZE+DIGESTSIZE2+lk+DB_off),
  (short)(BLOCKSIZE-lk),(byte)0x36);
  for (i = 0 ; i < lk ; i=(short)(i+1))
  DB[(short)(i+DB_off)] = (byte)(k[(short)(i+k_off)] ^ (byte)0x5C);
  Util.arrayFillNonAtomic(DB,(short)(lk+DB_off),
                          (short)(BLOCKSIZE-lk),(byte)0x5C);
}
  md.reset();
  md.update(DB,(short)(DB_off+BLOCKSIZE+DIGESTSIZE2),BLOCKSIZE);
  md.doFinal(d, d_off,ld,DB,(short)(DB_off+BLOCKSIZE));
  md.reset();
  md.doFinal(DB,DB_off,(short)(DIGESTSIZE+BLOCKSIZE),out,out_off);
}
```

```
protected im(byte[] bArray,short bOffset,byte bLength)
{ init();
  register();
}


public void init()
{ short i=0;
  status = (short)0;
  ECCkp  = new KeyPair[N_KEYS];
  UserPin   = new OwnerPIN((byte)3,(byte)8);  // 3 tries, 4=Max Size
  AdminPin  = new OwnerPIN((byte)10,(byte)8); // 10 tries 8=Max Size
  UserPin.update(MyPin,(short)0,(byte)8) ;
  AdminPin.update(OpPin,(short)0,(byte)8);
  for(i=0;i<N_KEYS;i++)
  {
    try{
        ECCkp[i] = new
        KeyPair(KeyPair.ALG_EC_FP,KeyBuilder.LENGTH_EC_FP_256);
        status =(short)(status + (short)1);
       }
    catch (CryptoException e){}
  }
  try {
       ECCsig =
       Signature.getInstance(Signature.ALG_ECDSA_SHA_256, false);
       status =(short)(status | (short)0x0100);
      }
  catch (CryptoException e){}
  try {
       sha256 =
       MessageDigest.getInstance(MessageDigest.ALG_SHA_256, false);
       status =(short)(status | (short)0x2000);
      }
  catch (CryptoException e){}
  DB = JCSystem.makeTransientByteArray(DBSIZE,
                                       JCSystem.CLEAR_ON_DESELECT);
}

public static void install(byte[] bArray, short bOffset,
                           byte bLength )
{ new im(bArray,bOffset,bLength);}


public boolean select()
{ if (UserPin.isValidated()) UserPin.reset();
  if (AdminPin.isValidated()) AdminPin.reset();
  return true;
}
```

8 IANA Considerations

This draft does not require any action from IANA.

9 Security Considerations

This entire document is about security.

10 References

10.1 Normative References

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, https://www.rfc-editor.org/info/rfc8446.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

[GP-SPI-I2C] GlobalPlatform Technology, APDU Transport over SPI/I2C Version 0.0.0.39, July 2019

10.2 Informative References

[IM-JC] https://github.com/purien/TLS-SE/blob/master/im/im.java

11 Authors' Addresses

Pascal Urien
Telecom Paris
19 place Marguerite Perey
91120 Palaiseau Phone: NA
France Email: Pascal.Urien@telecom-paris.fr