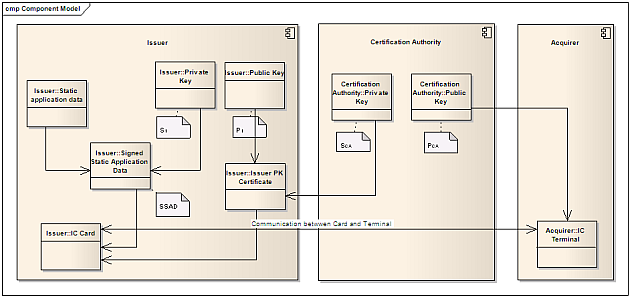
**Static Data Authentication (SDA)**

SDA ensures the authenticity of ICC data. After SDA it is sure that the data from the ICC is real and hasn't changed by anyone.  
But SDA doesn't assure the uniqueness of ICC data.

SDA is a digital signature scheme working with asymmetric cryptograhpy. Asymmetric cryptography uses a pair of keys. If you encode something with the first key, you can only decode it with the second key. The pair of keys is divided into a public and a private key. With the private key the issuer can "sign" critical data on ICC. Only the issuer knows the private key. The public key is not secret. Every terminal has the public key and can decode the signed data. If it conforms to the specification the data is authentic and has not changed.

In this figure we see the detailed process of authentication:

[](https://www.openscdp.org/scripts/tutorial/emv/sdaBig.png)

The Static application data will be signed with the Issuer Private Key (S1) and stored in Signed Application Data (SSAD). The corresponding Issuer Public Key (P1) will be stored in the Issuer PK Certificate. To verify that S1 and P1 are made by the Issuer, the Issuer PK Certificate is signed with a Certification Authority Private Key (SCA). The Certification Authority Public Key (PCA) is stored in the IC Terminal.

For SDA the terminals decrypt the Issuer PK Certificate with the PCA key. If the decryption was successful, the terminal extracts the P1 key to decrypt the SSAD. SDA was successful if the IC Terminal verifies the SSAD.

**Retrieval of Certification Authority Public Key**

For SDA we need first the Registered Application Provider Identifier (RID) to recognize whether we have a MasterCard or Visa Card because both have different public keys. To get the RID we extract the first five bytes of the Application Identifier (AID).

/\*\*

\* Get RID from EMV data model

\*

\* @type ByteString

\* @return the 5 byte RID

\*/

SDA.prototype.getRID = function() {

var aid = this.emv.cardDE[EMV.AID];

var rid = aid.[left](http://www.openscdp.org/sse4e/bytestring.html#left)(5);

return(rid);

}

The next step is to read the Certification Authority Public Key Index from the ICC. Each issuer have serveral public keys. The index reveal us which of them we need to process.

/\*\*

\* Get the Public Key Index

\*

\* @type Number

\* @return the Public Key Index

\*/

SDA.prototype.getPubKeyIndex = function() {

var index = this.emv.cardDE[0x8F];

var index = index.toUnsigned();

return(index);

}

Finally this function will return the public key:

/\*\*

\* Get the public key

\*

\*@type Key

\*@return the public key

\*/

SDA.prototype.getSchemePublicKey = function() {

var rid = this.getRID();

var index = this.getPubKeyIndex();

var key = this.schemePublicKeyTable[rid.toString(HEX)][index];

return(key);

}

**Retrieval of Issuer Public Key**

With the Certification Authority Public Key we decode the Issuer PK Certificate on the condition that both have the same length.

/\*\*

\* Decrypt the Issuer PK Certificate

\*

\* @ return the decrypted Issuer PK Certificate

\*/

SDA.prototype.decryptIssuerPKCertificate = function() {

var certificate = this.emv.cardDE[0x90];

var key = this.getSchemePublicKey();

var decryptedCertificate = [crypto](http://www.openscdp.org/sse4e/crypto.html).[decrypt](http://www.openscdp.org/sse4e/crypto.html#decrypt)(key, Crypto.RSA, certificate);

return(decryptedCertificate);

}

The certificate contains the following data:

Issuer Public Key Certificate

0000 6A 02 45 70 96 FF 12 09 00 40 E2 01 01 80 03 EC j.Ep.....@......

0010 59 32 AD A1 C4 C2 57 11 15 CC 11 C8 7B FD 9B 7E Y2....W.....{..~

0020 98 A3 33 EB F1 C1 5B A2 52 9E 03 A3 E1 A0 2D BA ..3...[.R.....-.

0030 F7 CB 7E 20 A9 9E B0 F8 F3 C0 AB 20 30 67 07 D3 ..~ ....... 0g..

0040 A4 52 02 E8 77 FC 0F B9 BC D9 0D 8E EE EE 89 BB .R..w...........

0050 12 05 C9 C8 C6 DF 12 C9 F8 D7 88 F8 62 2C BE FF ............b,..

0060 D7 7F A0 39 D4 63 C7 89 2D 76 D1 D9 37 80 9B 1F ...9.c..-v..7...

0070 9C 83 AA D4 BE AD 95 5A 70 0D C4 A7 68 50 BD BC .......Zp...hP..

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Length** | **Description** |
| Recovered Data Header | 1 | Hex Value '6A' |
| Certificate Format | 1 | Hex Value '02' |
| Issuer Identifier | 4 | Lefmost 3-8 digits from the PAN (padded to the right with Hex 'F's) |
| Certificate Expiration Date | 2 | MMYY after which this certificate is invalid |
| Certificate Serial Number | 3 | Binary number unique to this certificate assigned by the ceritifaction authority |
| Hash Algorithm Indicator | 1 | Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme |
| Issuer Public Key Algorithm Indicator | 1 | Identifies the digital signature algorithm to be used with the Issuer Public Key |
| Issuer Public Key Length | 1 | Identifies the length of the Issuer Public Key Modulus in bytes |
| Issuer Public Key Exponent Length | 1 | Identifies the length of the Issuer Public Key Exponent in bytes |
| Issuer Public Key or Leftmost Digits of the Issuer Public Key\* | NCA - 36 | If NI <= NCA - 36, consists of the full Issuer Public Key padded to the right with NCA - 36- NI bytes of value 'BB'. If NI > NCA - 36, consists of the NCA - 36 most significant bytes of the Issuer Public Key |
| Hash Result | 20 | Hash of the Issuer Public Key and its related information |
| Recovered Data Trailer | 1 | Hex value 'BC' |
| Source: [EMV Book 2](http://www.emvco.com/specifications.aspx?id=155) |  |  |

\* NCA = Number of Bytes of Certifiacation Authority Public Key  
\* NI = Number of Bytes of Issuer Public Key Modulus

After decoding we have to proof 12 steps until we get the Issuer Public Key Modulus. The key consists of the leftmost digits of the Issuer Public Key which are stored in the certificate and the Issuer Public Key Remainder stored in an AEF file of the ICC. To get the Issuer Public Key Modulus concatenate the leftmost digits with the Issuer Public Key Remainder.

/\*\*

\* Retrieval of Issuer Public Key

\*

\* @type Key

\* @return the Issuer Public Key

\*/

SDA.prototype.retrievalIssuerPublicKey = function() {

var key = this.getSchemePublicKey();

var modulus = key.getComponent(Key.MODULUS);

var cert = this.decryptIssuerPKCertificate();

// Step 1: Issuer Public Key Certificate and Certification Authority Public Key Modulus have the same length

assert(cert.length == modulus.length);

// Step 2: The Recovered Data Trailer is equal to 'BC'

assert(cert.[byteAt](http://www.openscdp.org/sse4e/bytestring.html#byteAt)(modulus.length - 1) == 0xBC);

// Step 3: The Recovered Data Header is equal to '6A'

assert(cert.byteAt(0) == 0x6A);

// Step 4: The Certificate Format is equal to '02'

assert(cert.byteAt(1) == 0x02);

// Step 5: Concatenation of Certificate Format through Issuer Public Key or Leftmost Digits of the Issuer Public Key,

// followed by the Issuer Public Key Remainder (if present), and the Issuer Public Key Exponent

var list;

list = cert.[bytes](http://www.openscdp.org/sse4e/bytestring.html#bytes)(1, 14 + (modulus.length - 36));

var remainder = this.emv.cardDE[0x92];

var exponent = this.emv.cardDE[0x9F32];

var remex = remainder.[concat](http://www.openscdp.org/sse4e/bytestring.html#concat)(exponent);

list = list.concat(remex);

// Step 6: Generate hash from concatenation

var hashConcat = this.crypto.[digest](http://www.openscdp.org/sse4e/crypto.html#digest)(Crypto.SHA\_1, list);

// Step 7: Compare the hash result with the recovered hash result. They have to be equal

var hashCert = cert.bytes(15 + (modulus.length - 36), 20);

assert(hashCert.[equals](http://www.openscdp.org/sse4e/bytestring.html#equals)(hashConcat));

// Step 8: Verify that the Issuer Identifier matches the lefmost 3-8 PAN digits

var pan = this.emv.cardDE[0x5A];

pan = pan.left(4);

var panCert = cert.bytes(2, 4);

var panCert = panCert.toString(HEX);

var pan = pan.toString(HEX);

for(var i = 0; i < 8; i++) {

if(panCert.charAt(i) == 'F') {

var panCert = panCert.substr(0, i);

var pan = pan.substr(0, i);

}

}

assert(pan == panCert);

// Step 9: Verify that the last day of the month specified in the Certification Expiration Date is equal to or later than today's date.

// Step 10: Optional step

// Step 11: Check the Issuer Public Key Algorithm Indicator

var pkAlgorithmIndicator = cert.byteAt(12);

// Step 12: Concatenate the Leftmost Digits of the Issuer Public Key and the Issuer Public Key Remainder (if present)

// to obtain the Issuer Public Key Modulus

var leftmostDigits = cert.bytes(15, (modulus.length - 36));

var issuerPublicKeyModulus = leftmostDigits.concat(remainder);

return(issuerPublicKeyModulus);

}

**Verification of Signed Static Application Data**

We decode the Signed Static Application Data (SSAD) with the Issuer Public Key from the step before.  
The decrypted SSAD contains the following data:

Signed Static Application Data

0000 6A 03 01 D1 79 BB BB BB BB BB BB BB BB BB BB BB j...y...........

0010 BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB ................

0020 BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB ................

0030 BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB ................

0040 BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB ................

0050 BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB ................

0060 BB BB BB BB BB BB BB BB BB BB BB BC 93 56 B5 63 .............V.c

0070 68 FF C2 C9 48 E7 12 A7 C7 8A A1 3E D8 49 42 BC h...H......>.IB.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Length** | **Description** |
| Recovered Data Header | 1 | Hex Value '6A' |
| Signed Data Format | 1 | Hex Value '03' |
| Hash Algorithm Indicator | 1 | Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme |
| Data Authentication Code | 2 | Issuer-assigned code |
| Pad Pattern | NI - 26 | Pad pattern consisting of NI - 26 bytes of value 'BB' |
| Hash Result | 20 | Hash of the Static Application Data to be authenticated |
| Recovered Data Trailer | 1 | Hex value 'BC' |
| Source: [EMV Book 2](http://www.emvco.com/specifications.aspx?id=155) |  |  |

Now we have to proof 7 steps. If all steps were successful, SDA was successful:

/\*\*

\* Verification of Signed Static Application Data

\*

\* @param {Key} key the Issuer Public Key

\*/

SDA.prototype.verifySSAD = function(issuerPublicKeyModulus) {

var issuerPublicKeyModulus = issuerPublicKeyModulus;

// The Issuer Public Key consists of the Issuer Public Key Modulus from the step before

// and the Issuer Public Key Exponent read from the ICC.

var [key](http://www.openscdp.org/sse4e/key.html) = new Key();

key.[setType](http://www.openscdp.org/sse4e/key.html#setType)(Key.PUBLIC);

key.[setComponent](http://www.openscdp.org/sse4e/key.html#setComponent)(Key.MODULUS, issuerPublicKeyModulus);

key.setComponent(Key.EXPONENT, this.emv.cardDE[0x9F32]);

// Signed Static Application Data read from the ICC

var SSAD = this.emv.cardDE[0x93];

// Step 1: Signed Static Application Data and Issuer Public Key Modulus have the same length

assert(SSAD.length == issuerPublicKeyModulus.length);

// Step 2: The Recovered Data Trailer is equal to 'BC'

var decryptedSSAD = crypto.decrypt(key, Crypto.RSA, SSAD);

assert(decryptedSSAD.byteAt(decryptedSSAD.length -1) == 0xBC);

// Step 3: The Recovered Data Header is equal to '6A'

assert(decryptedSSAD.byteAt(0) == 0x6A);

// Step 4: The Signed Data Format is equal to '03'

assert(decryptedSSAD.byteAt(1) == 0x03);

// Step 5: Concatenation of Signed Data Format, Hash Algorithm Indicator, Data Authentication Code, Pad Pattern,

// the data listed by the AFL and finally the SDA Tag List

var list = decryptedSSAD.bytes(1, (decryptedSSAD.length - 22));

var daInput = this.emv.getDAInput();

var sdaTagList = this.emv.cardDE[0x9F4A];

if(typeof(sdaTagList != "undefined")) {

for(var i = 0; i < sdaTagList.length; i++) {

var tag = sdaTagList.byteAt(i);

var value = new ByteBuffer();

value = value.append(this.emv.cardDE[tag]);

}

}

list = list.concat(daInput);

if(value != 0) {

value = value.toByteString();

list = list.concat(value);

}

// Step 6: Generate hash from concatenation

var hashConcat = this.crypto.digest(Crypto.SHA\_1, list);

// Step 7: Compare recovered hash with generated hash. Store the Data Authentication Code from SSAD in tag '9F45'

var hashSSAD = decryptedSSAD.bytes(decryptedSSAD.length - 21, 20);

assert(hashConcat.equals(hashSSAD));

this.emv.cardDE[0x9F45] = decryptedSSAD.bytes(3, 2);

print("SDA was successful");

}