**Dynamic Data Authentication**

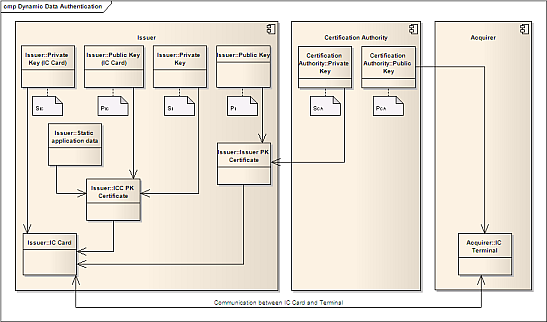
Dynamic Data Authentication (DDA) is a more securer way of authentication because it authenticates the card itself. After the authentication we can trust on the uniqueness of the card.

**Contrast between SDA and DDA**

SDA guarantees that data on cards is valid because we trust a high level certification authority which signs the data. But an attacker can record a card session and build for example a new virtuel card.

In contrast to SDA the card creates during the DDA process an own signature with their private key (SIC). This signature is different in every card session because it contains a random number generated by the terminal. The corresponding public key is stored in an ICC Public Key Certificate signed by the Certification Authority. With the public key we can proof the signature on genuineness.

**Authentication Process**

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

The dynamic authentication process is related to SDA.  
With PCA the terminal proofs the signature of the Issuer PK Certificate and extract the P1 key. The terminal proofs with P1 the signature of the ICC PK Certificate to get the PIC key. Now it is time to send an [Internal Authenticate](https://www.openscdp.org/scripts/tutorial/emv/dda.html#internalAuthenticate) command to the card that contains a random number and initiates the card to compute the signature (Signed Dynamic Application Data/SDAD). With the PIC key the terminal checks the SDAD for guineness.

**Retrieval of ICC Public Key (PIC)**

The retrieval of ICC Public Key starts after the [Retrieval of Certification Authority Public Key](https://www.openscdp.org/scripts/tutorial/emv/SDA.html#retrievalCA) and [Retrieval of Issuer Public Key](https://www.openscdp.org/scripts/tutorial/emv/SDA.html#retrievalIssuerPK) described in the chapter [SDA](https://www.openscdp.org/scripts/tutorial/emv/SDA.html).

First we have to decrypt the ICC Public Key Certificate.

DataAuthentication.prototype.retrievalICCPublicKey = function(issuerPublicKeyModulus) {

var issuerPublicKeyModulus = issuerPublicKeyModulus;

var key = new Key();

key.setType(Key.PUBLIC);

key.setComponent(Key.MODULUS, issuerPublicKeyModulus);

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

var iccCert = this.emv.cardDE[0x9F46];

The decrypted certificate contains the following data:

ICC Public Key Certificate:

0000 6A 04 45 70 96 54 32 00 43 03 FF FF 07 09 50 33 j.Ep.T2.C.....P3

0010 D6 01 01 80 03 C9 DE 79 AC 4E 62 D5 38 30 DB 3D .......y.Nb.80.=

0020 50 B8 7A 88 88 6F 74 11 D1 44 47 C8 88 46 93 F7 P.z..ot..DG..F..

0030 0B C5 B3 A2 63 E4 0D 14 7A A3 10 47 33 73 73 B9 ....c...z..G3ss.

0040 5D 0B 05 61 44 00 45 F2 D7 E5 0D C5 83 C6 04 9A ]..aD.E.........

0050 7B 37 B8 E1 48 D6 09 6A B7 8F D9 2F C3 27 01 2B {7..H..j.../.'.+

0060 6B FD 43 AB 02 76 E3 11 C7 3E 17 1E A7 C8 04 21 k.C..v...>.....!

0070 85 55 D8 0C 4F E8 D1 DC F0 7D 1C D5 61 46 57 BC .U..O....}..aFW.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Length** | **Description** |
| Recovered Data Header | 1 | Hex Value '6A' |
| Certificate Format | 1 | Hex Value '04' |
| Application PAN | 10 | 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 issuer |
| Hash Algorithm Indicator | 1 | Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme |
| ICC Public Key Algorithm Indicator | 1 | Identifies the digital signature algorithm to be used with the ICC Public Key |
| ICC Public Key Length | 1 | Identifies the length of the ICC Public Key Modulus in bytes |
| ICC Public Key Exponent Length | 1 | Identifies the length of the ICC Public Key Exponent in bytes |
| ICC Public Key or Leftmost Digits of the ICC Public Key\* | NI - 42 | If NIC <= NI - 42, consists of the full ICC Public Key padded to the right with NI - 42- NIC bytes of value 'BB'. If NIC > NI - 42, consists of the NI - 42 most significant bytes of the ICC Public Key |
| Hash Result | 20 | Hash of the ICC 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) |  |  |

With step 1 to 4 we check whether the decryption was succesful.

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

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

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

var decryptedICC = crypto.decrypt(key, Crypto.RSA, iccCert);

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

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

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

//Step 4: The Certificate Format is equal to '04'

assert(decryptedICC.byteAt(1) == 0x04);

Step 5 implements the concatenation which is necessary to apply the hash algorithm in the next step.  
For the concatenation we need from the ICC Public Key Certificate:

 Certificate Format

 Application PAN

 Certificate Expiration Date

 Certificate Serial Number

 Hash Algorithm Indicator

 ICC Public Key Algorithm Indicator

 ICC Public Key Lnegth

 ICC Public Key Exponent Length

 ICC Public Key or Leftmost Digits of the ICC Public Key

Also we need

 ICC Public Key Remainder (if present)

 ICC Public Key Exponent

 Data located by the [AFL](https://www.openscdp.org/scripts/tutorial/emv/Read%20Application%20Data.html)

 SDA Tag List

// Step 5: Concatenation

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

var remainder = this.emv.cardDE[0x9F48];

var exponent = this.emv.cardDE[0x9F47];

var remex = remainder.concat(exponent);

list = list.concat(remex);

var daInput = this.emv.getDAInput();

list = list.concat(daInput);

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

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

var value = new ByteBuffer();

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

var tag = sdaTagList.byteAt(i);

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

}

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

var hashICC = decryptedICC.bytes(decryptedICC.length - 21, 20);

assert(hashConcat.equals(hashICC));

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

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

var panCert = decryptedICC.bytes(2, 10);

var panCert = panCert.toString(HEX);

var pan = pan.toString(HEX);

for(var i = 0; i < 20; 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.

// Not proved, so you can test your expired cards.

// Step 10: Check the ICC Public Key Algorithm Indicator

var pkAlgorithmIndicator = decryptedICC.byteAt(18);

If step 5 to 10 were successful we can start concatenating the ICC Public Key (PIC).

// Step 11: Concatenate the Leftmost Digits of the ICC Public Key

//and the ICC Public Key Remainder (if present) to obtain the ICC Public Key Modulus

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

var leftmostDigits = decryptedICC.bytes(21, (modulus.length - 42));

var iccPublicKeyModulus = leftmostDigits.concat(remainder);

return(iccPublicKeyModulus)

}

**Dynamic Data Authentication**

**Internal Authenticate**

The Internal Authenticate command initiates the card to sign with their Privat Key (SIC) Dynamic Application Data and a random number generated by the terminal. The card will return the Signed Dynamic Application Data.

Structure of the Internal Authenticate command:

|  |  |
| --- | --- |
| **Code** | **Value** |
| CLA | '00' |
| INS | '88' |
| P1 | '00' |
| P2 | '00' |
| Lc | Length of authentication-related data |
| Data | Authentication-related data (random number) |
| Le | '00' |

DataAuthentication.prototype.dynamicDataAuthentication = function(iccPublicKeyModulus){

var iccPublicKeyModulus = iccPublicKeyModulus;

var Data = crypto.generateRandom(4);

var internalAuthenticate = card.sendApdu(0x00, 0x88, 0x00, 0x00, Data, 0x00);

var asn = new ASN1(internalAuthenticate);

var tag = asn.find(0x9F4B);

var SDAD = tag.value;

**Signed Dynamic Application Data**

Now we decode the Signed Dynamic Application Data

var picKey = new Key();

picKey.setType(Key.PUBLIC);

picKey.setComponent(Key.MODULUS, iccPublicKeyModulus);

picKey.setComponent(Key.EXPONENT, this.emv.cardDE[0x9F47]);

var decryptedSDAD = crypto.decrypt(picKey, Crypto.RSA, SDAD);

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Length** | **Description** |
| Recovered Data Header | 1 | Hex value '6A' |
| Signed Data Format | 1 | Hex value '05' |
| Hash Algorithm Indicator | 1 | Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme |
| ICC Dynamic Data Length | 1 | Identifies the length of the ICC Dynamic Data in bytes |
| ICC Dynamic Data Length | LDD | Dynamic data generated by and/or stored in the ICC |
| Pad Pattern | NIC - LDD - 25 | (NIC - LDD - 25) padding bytes of value 'BB' |
| Hash Result | 20 | Hash of the Dynamic Application Data and its related infromation |
| Recovered Data Trailer | 1 | Hex value'BC' |
| Source: [EMV Book 2](http://www.emvco.com/specifications.aspx?id=155) |  |  |

Decrypted SDAD:

0000 6A 05 01 09 08 0C 1E 38 91 B9 1B E7 62 BB BB BB j......8....b...

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 33 C2 28 11 71 ...........3.(.q

0070 05 9A D4 2A 9A B1 EF 66 93 E8 86 7B 30 AF CF BC ...\*...f...{0...

If the following 7 steps were successful, DDA was successful.

//Step 1: SDAD and ICC Public Key Modulus have the same length

assert(SDAD.length == iccPublicKeyModulus.length);

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

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

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

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

//Step 4: The Signed Data Format is equal to '05'

assert(decryptedSDAD.byteAt(1) == 0x05);

//Step 5: Concatenation of Signed Data Format, Hash Algorithm Indicator,

// ICC Dynamic Data Length, ICC Dynamic Data, Pad Pattern, random number

var LDD = decryptedSDAD.byteAt(3);

var list = decryptedSDAD.bytes(1, 3 + LDD + decryptedSDAD.length - LDD - 25);

list = list.concat(Data);

//Step 6: Genereate hash from concatenation

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

//Step 7: Compare recovered hash with generated hash

var hashSDAD = decryptedSDAD.bytes(decryptedSDAD.length - 21, 20);

assert(hashConcat.equals(hashSDAD));

print("DDA was successful");

}