Security Issues of the Digital Certificates within Public Key Infrastructures

Cristian TOMA
Economic Informatics Department,
Academy of Economic Studies, Bucharest, Romania
cristian.toma@ie.ase.ro

The paper presents the basic byte level interpretation of an X.509 v3 digital certificate according to ASN.1 DER/BER encoding. The reasons for byte level analysis are various and important. For instance, a research paper has mentioned how a PKI security may be violated by MD5 collision over information from the certificates. In order to develop further studies on the topic a serious knowledge about certificate structure is necessary.
Keywords: *digital certificates, certificates authority, ASN.1 DER/BER, PKI*

1 ASN.1 – Abstract Syntax Notation 1

ASN.1 is a sort artificial language used for describing data and data structure, instead of programs. The syntax is standardized in ISO/IEC 8824, and the coding rules are defined by ISO/IEC 8825. Both of these standards were developed from Recommendation X.409 of the CCITT. In order to get more information about ASN.1 it is recommended to consult [1]. *In ASN.1, an octet is an eight-bit unsigned integer. Bit 8 of the octet is the most significant and bit 1 is the least significant.*

The following meta-syntax and rules are used for in describing ASN.1 notation:

- **BIT** – mono-space denotes literal characters in the type and value notation; in examples, it generally denotes an octet value in hexadecimal
- **n1** – bold italics denotes a variable
- **[]** – bold square brackets indicate that a term is optional
- **{}** – bold braces group related terms
- **|** – bold vertical bar delimits alternatives with a group
- **…** – bold ellipsis indicates repeated occurrences
- **=** – bold equals sign expresses terms as sub-terms.
- Layout is not significant; multiple spaces and line breaks can be considered as a single space
- Identifiers (names of values and fields) and type references (names of types) consist of upper- and lower-case letters, digits, hyphens, and spaces; identifiers begin with lower-case letters; type references begin with upper-case letters. Comments are delimited by pairs of hyphens (--), or a pair of hyphens and a line break.

Some of the **data types** used in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
BOOLEAN	Primitive	0x01	Boolean value: yes/no
INTEGER	Primitive	0x02	Negative and positive integers
BIT STRING	Primitive	0x03	Bit sequence
OCTET STRING	Primitive	0x04	Byte sequence (1 byte=8 bits=1octet)
NULL	Primitive	0x05	A null value
OBJECT IDENTIFIER	Primitive	0x06	An object identifier, which is a sequence of integer components that identify an object such as an algorithm or attribute type
PrintableString	Primitive	0x13	An arbitrary string of printable characters
T61String	Primitive	0x14	An arbitrary string of T.61 (eight-bit) characters
IA5String	Primitive	0x16	An arbitrary string of IA5 (ASCII) characters
UTCTime	Primitive	0x17	A "coordinated universal time" or Greenwich Mean Time (GMT) value

Some of **structured types** defined in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
SEQUENCE	Constructed	0x30	An ordered collection of one or more types
SEQUENCE OF	Constructed	0x10	An ordered collection of zero or more occurrences of a given type
SET	Constructed	0x31	An unordered collection of one or more types
SET OF	Constructed	0x11	An unordered collection of zero or more occurrences of a given type
A0, A1, …	Constructed	0xAz	Where z = 0..F in hex and it represents the z-th element in SEQUENCE data type

The idea in ASN.1 is to prefix each data object with a unique label and information about its length. Users are allowed to define their own data types and nested data objects. A simple example of data type definition for a controller using ASN.1 is:

```
SC_Controller ::= SEQUENCE { //Describe a new data type for a controller
  Name IA5String,              //The name of the microchip is an ASCII string
  CPUType CPUPower,            //CPUType refers to the definition of CPUPower
  NPU BOOLEAN,                 //yes or no if coprocessor is present
  EEPROMSize INTEGER,          //the size in bytes of EEPROM
  RAMSize INTEGER,             //the size in bytes of RAM used in smart card
  ROMSize INTEGER              //the size in bytes of ROM used in smart card
}
CPUPower ::= ENUMERATED { //Definition of CPUPower as an enumeration
  8Bit (8),                    //Possible selection values for the CPUType
  16Bit (16),
  32Bit (32)
}
```

An example for a particular XSS microcontroller using previous definition in ASN.1 is described in the following mode:

```
XSS SC_Controller ::= { //Specific instance-controller of SC_Controller
  Name "XS 8 Bit",             //The name of controller is XS 8 Bit
  CPUType 8,                   //this is an 8-bit CPU
  NPU true,                    //NPU present
  EEPROMSize 1024,             //The size of EEPROM size is 1024 bytes
  RAMSize 256,                 //The size of RAM size is 256 bytes
  ROMSize 8192                 //The size of ROM size is 8192 bytes
}
```

The **BER – Basic Encoding Rules** for ASN.1 are defined in the ISO/IEC 8825 standard. A BER coded data object should have:

- A label called **T – tag**
- A length field **L**
- The data content **V - value**

Figure 1 presents the BER-based TLV coding principle accordingly to ASN.1:

T - tag	L - length	V - value
1-2 bytes	1-3 bytes	n bytes

TLV Object

Fig. 1. ASN.1 BER-based TLV coding

A sample data for a particular microcontroller model is coded using the **ASN.1 BER**: 0x30 0x1C 0x16 0x08 0x58 0x53 0x20 0x38 0x20 0x42 0x69 0x74 0x0A 0x01 0x08 0x01 0x01 0xFF 0x02 0x02 0x04 0x00 0x02 0x02 0x01 0x00 0x02 0x02 0x20 0x00.

```
'30 1C'                          //Tag 0x30 for a array with a length of 28
                                 //bytes (0x1C)
'16 08 58 53 20 38 20 42 69 74'  //Tag 0x16 for an IA5String with a length of
                                 //8 bytes (0x08) with a content of "XS 8 Bit"
'0A 01 08'                       //Tag 0x0A for an enumerated data type with
                                 //a length of 1 byte (0x01) and a content of 8
'01 01 FF'      //Tag 0x01 for Boolean data type with a length of 1 byte (0x01)
                //and a content of 0xFF (true)
'02 02 04 00'  //Tag 0x02 for an integer data type with a length of 2 bytes
               //(0x02) and of 1024 bytes (0x0400)
'02 02 01 00'  //Tag 0x02 for an integer data type with a length of 2 bytes
               //(0x02) and of 256 bytes (0x0100)
'02 02 20 00' //Tag 0x02 for an integer data type with a length of 2 bytes
              //(0x02) and of 8192 bytes (0x0200)
```

When the author discuss about the start of a sequence for an X509 v3 certificates, it would be 0x30 0x82 0x01 0xC3 - SEQUENCE {…} – this means the certificate is a SEQUENCE structure (first byte with 0x30 value) with 451 bytes. The length is specified in 3 bytes 0x82 0x01 0xC3. The last hex digit (nibble = half-byte) from the first byte of the length field specifies the length in bytes of the structure => 2 bytes. The next 2 bytes actually express the length of the structure SEQUENCE, and they have the value 0x01C3 => 451 bytes.

The BER special encoding applies to the OID-s – Object Identifiers - in Internet.

In the figure 2 there is some of the ISO tree hierarchy for constructing objects in Internet structure. It is useful to imagine how would be written the OID with value 1.2.840.113549.1.1.5 ({iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}) for signature obtained from SHA-1 digest on the message and RSA algorithm applied to the digest. This encoding will be at byte level: 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x05. The first byte shows that here there is an OID – OBJECT IDENTIFIER field. The OID has 9 bytes length because of second byte in array. Because the first bit in length field is not set the length field has only one byte. The length field has value 9 which means the OID structure has the payload data in 9 bytes.

According to BER, the first two numbers of any OID (x.y) are encoded as one value using the formula (40*x)+y. The first two numbers in an OID are here 1.2. Therefore, the first two numbers of an OID are encoded as 42 or 0x2A, because (40*1)+2 = 42. After the first two numbers are encoded, the subsequent numbers in the OID are each encoded as a byte. However, a special rule is required for large numbers because one byte (eight bits) can only represent a number from 0-255. This is the case for 840 and 113549. For 840 from the OID, the first bit of the first byte should be set. The number occupies enough number of bytes till the last byte of representation is not having the first bit set. In 840 case 0x48 has the first bit NOT set. So, the formula is the last hex digit from the first byte multiplied with 2^7 = 128 (1 bit is for multiple bytes representation) and added with the value from the second byte as long as the second byte has a value less than 0x80. In 840 case, the formula is: 0x06*128 + 0x48 (form 0x86 0x48)= 768 + 72 = 840. In case of 113549 we choose the bytes 0x86 0xF7 0x0D because 0x0D is the last byte with first bit (sign bit) not set. The formula for 113549 is:

0x06*2^14 + 0x77*2^7 + 0x0d*2^0 = 6*16384 + 119*128 + 13*1 = 98304 + 15232 + 13 = 113549.

For the remaining encoding {…pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}, we will have only one byte for each number: 0x01 0x01 0x05.

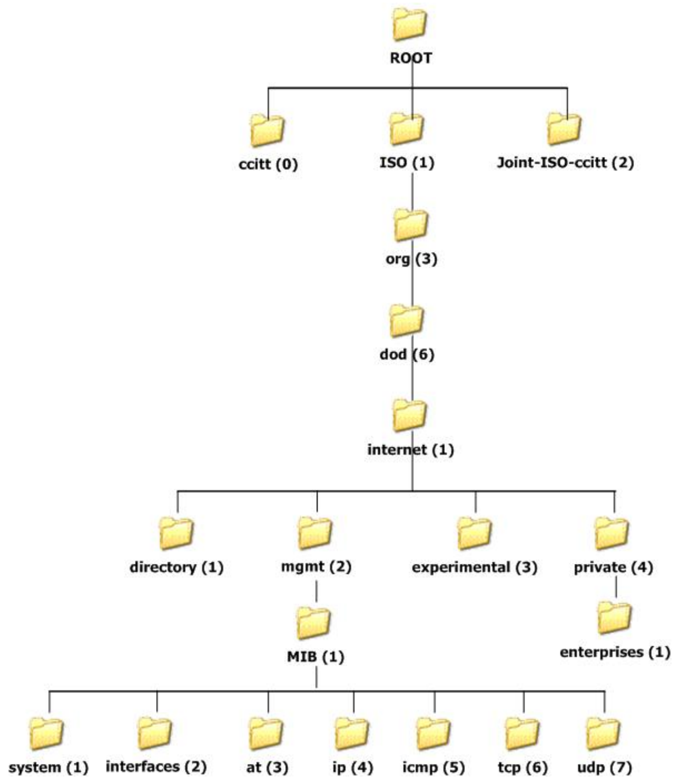

Fig. 2. ISO tree hierarchy for constructing objects in Internet structure

For checking out an OID, please visit [3]. The **DER – Distinguished Encoding Rules** form a subset of the BER and give exactly one way to represent any ASN.1 value as an octet string. DER is intended for applications in which a unique octet string encoding is needed, as is the case when a digital signature is computed on an ASN.1 value. A basic summary of the BER and DER are found in Burton Kaliski [1].
The two most significant bits of the bytes T – tag encode the class of the data object. The bits positions are presented in figure 3:

T - tag			L - length	
b7 b6 b5 b4 b3 b2 b1 b0	Meaning	b7 b6 b5 b4 b3 b2 b1 b0	Byte 1 Byte 2 Byte 3	Meaning
0 0 … … … … … …	Universal class	0 … … … … … … …	0 - 127 … …	One byte is needed
0 1 … … … … … …	Application class		0x81 128-255	Two bytes are needed
1 0 … … … … … …	Context-specific class	The value of tag code in range 31-127	0x82 256-65535	Three bytes are needed
1 1 … … … … … …	Private class			
... … 0 … … … … …	Primitive data object			
... … 1 … … … … …	Constructed data obj.			
... … … Y Y Y Y Y	Tag code (0-30)			
... … … 1 1 1 1 1	Pointer to the following byte (byte 2), which specifies the tag code			
Byte 1		**Byte 2**	**Byte 3,4,5**	

Fig. 3. Structure of T and L fields in ASN.1 [5]

In figure 3, the *universal class* indicates general data objects such as integers and character strings. The *application class* indicates that the data object belongs to a particular (e.g. electronic wallet from a smart card) or standard (e.g. ISO/IEC 7816-6) application. The classes context-specific and private are not matter of standard applications. The bit following the first two bits indicates if the tag is for a primitive or constructed data object. The five least significant bits from byte 1 are the actual label, the tag value. If the tag value is greater then 30, then all the bits from first byte have value 1 and the bits b0-b6 from second byte store a value between 31-127 ranges. The byte 3, 4 and 5 are used for marking the data length. If data has a byte length less than 127 then only the byte 3 (first byte from L) is used to store the actual length. If the length value is between128-255, then 2 bytes are used for storing the length and if the length is between 256 and 65535, then all three bytes from L field are used for storing the length value.

Figure 4 presents a sample about how the first name of a person can be stored in a smart-card.

Tag	Length	Value				
0x85	0x05	0x46	0x72	0x61	0x6E	0x6B

Tag for first names — Length of the first name — First name “Frank” in ASCII

Fig. 4. TLV Encoding for the first name “Frank” [5]

Subsequent extensions to data structures can be undertaken very easily with ASN.1 since all that is necessary is to insert additional TLV-coded data objects into existing data structure. Full compatibility with the previous versions is retained as long as the previous TLV objects are not deleted. The same is true of new version of data structures in which changes have been made accordingly to the previous coding. This kind of encoding is used on the large scale in smart card industry. Figure 5 presents a basic scheme for forming constructed TLV-coded data structures from several primitive TLV-coded data objects.

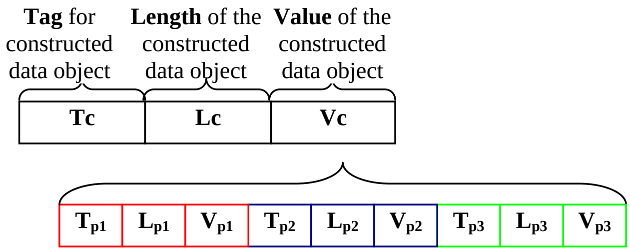

Fig. 5. Nested TLV-coded data objects [5]

The indices ‘C’ and ‘P’ stand for ‘constructed’ and ‘primitive’ in figure 5. This kind of coding has the advantage of no matter order have the TLV-coding structures the programs could be able to interpret them correct. When evaluating the TLV-coding data structure, the program compares the first tag with all tags known to it. If it finds a match, then it recognizes the first object from byte array, and it is able to “say” how long it is.

2. Digital Certificates and Certification Authorities

The infrastructures based on the cryptography with asymmetric (public) keys are essential for the viability of the message transactions and communications especially in networks and generally on the Internet. The PKI – Public Key Infrastructure consists of the multitude of services required to be ensured when the technologies of encryption with public keys are used on a large scale. These services are of technological nature, as well as legal, and their existence is necessary in order to permit the exploitation of public keys technologies at their full capacity.

The main elements of the public keys infrastructures are:

- Digital certificates
- Certification Authorities (CA)
- Management facilities (protocols) for certificates

The public keys infrastructures must ensure support for encryption functions, as well as for those for digital signature.

In the functioning of public keys systems is necessary a system for generating, circulating and certifying the users’ keys. Given a user C who intents to pose as A, and wants to fake sign as. Faker C may easily do that, generating his own pair of keys and placing the public one in the public folder, instead of the real one of A (instead of the public key of A).

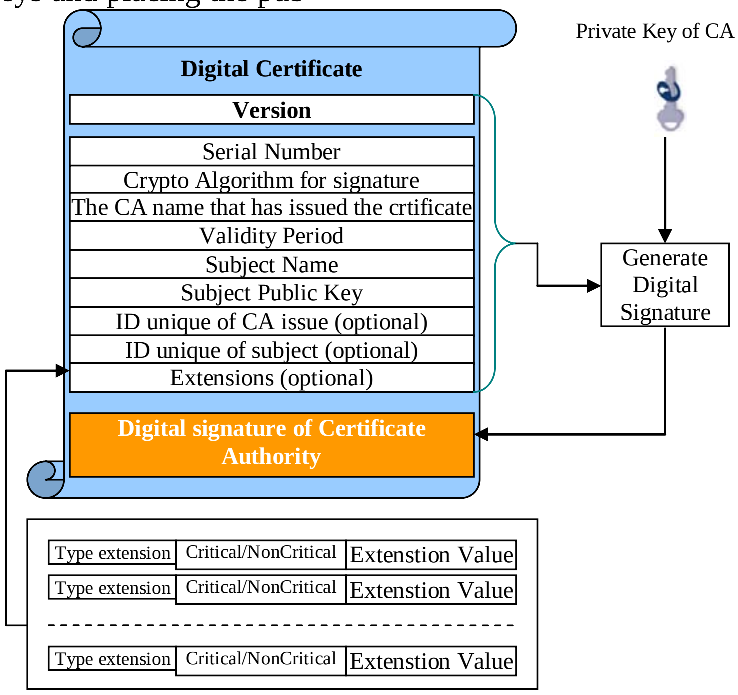

Fig. 6. X509 v3 Certificate

The documents and messages signed by C with his secret key will be verified with the public key that seems to be of A and any person will be deceived the authenticity of the messages signed on behalf of A. The main problem is thus of the total trust in public keys, those used for certifying the digital signatures. These (signatures) must be available on the network, so that any client may get the public key of a sender of a signed message. So, the technical solution exists: creating an international infrastructure, based on Certification Authorities – CA, that may allow the easy access and in a secure mode to the public keys of the entities that wish to communicate in the network or Internet. These authorities will distribute, by request, certificates of authentic keys. The most largely known and used format for digital certificates of public keys is that defined in the standard X.509 by ISO/IEC/ITU. The format X.509 for certificates has evolved in three versions. Version 3 (adopted in 1996) is the best known. Figure 6 illustrates the format of the certificate X.509 v3.

While the previous versions ensured support only for the name system X.500, X.509 v3 accepts a large variety of forms for names, including e-mail addresses and URLs.

A system based on public keys certificates presumes the existence of a Certification Authority, which issues certificates for a certain group of owners of keys pairs (public and private). Each certificate includes the value of the public key and information that uniquely identifies. The certificate's subject (who is a private person or a company, an application, a device or another entity that holds the secret key correspondent to the public key included in the certificate). The certificate represents a liaison impossible to falsify, between a public key and a certain attribute of its owner. The certificate is digitally signed by a Certification Authority (certified by the government), that so confirms the subject's identity. Once the certificates set established, an user of that public key infrastructure (PKI) may obtain the public key for any user certified by that Certification Authority, simply getting the certificate for that user extracting from it the desired public key. The ASN.1 representation of an X509.v3 certificate is the following:

```
Certificate ::= SIGNED { SEQUENCE {
version [0] Version DEFAULT v1,
serialNumber CertificateSerialNumber,
signature AlgorithmIdentifier,
issuer Name,
validity Validity,
subject Name,
subjectPublicKeyInfo SubjectPublicKeyInfo,
issuerUniqueIdentifier [1] IMPLICIT UniqueIdentifier OPTIONAL,
-- if present, version shall be v2 or v3
subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL,
-- if present, version shall be v2 or v3
extensions [3] Extensions OPTIONAL
-- If present, version shall be v3 -- } }
Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
AlgorithmIdentifier ::= SEQUENCE {
algorithm ALGORITHM.&id ({SupportedAlgorithms}),
parameters ALGORITHM.&Type ({SupportedAlgorithms}{ @algorithm}) OPTIONAL }
-- Definition of the following information object set is deferred, perhaps to standardized
-- profiles or to protocol implementation conformance statements. The set is required to
-- specify a table constraint on the parameters component of AlgorithmIdentifier.
-- SupportedAlgorithms ALGORITHM ::= { ... }
Validity ::= SEQUENCE {
notBefore Time,
notAfter Time }
SubjectPublicKeyInfo ::= SEQUENCE {
algorithm AlgorithmIdentifier,
subjectPublicKey BIT STRING }
Time ::= CHOICE {
utcTime UTCTime,
generalizedTime GeneralizedTime }
Extensions ::= SEQUENCE OF Extension
Extension ::= SEQUENCE {
extnId EXTENSION.&id ({ExtensionSet}),
critical BOOLEAN DEFAULT FALSE,
extnValue OCTET STRING
-- contains a DER encoding of a value of type &ExtnType
-- for the extension object identified by extnId -- }
ExtensionSet EXTENSION ::= { ... }
```

In figure 7 is represented an X 509 v3 certificate in hex editor and in Windows Crypto Shell program.

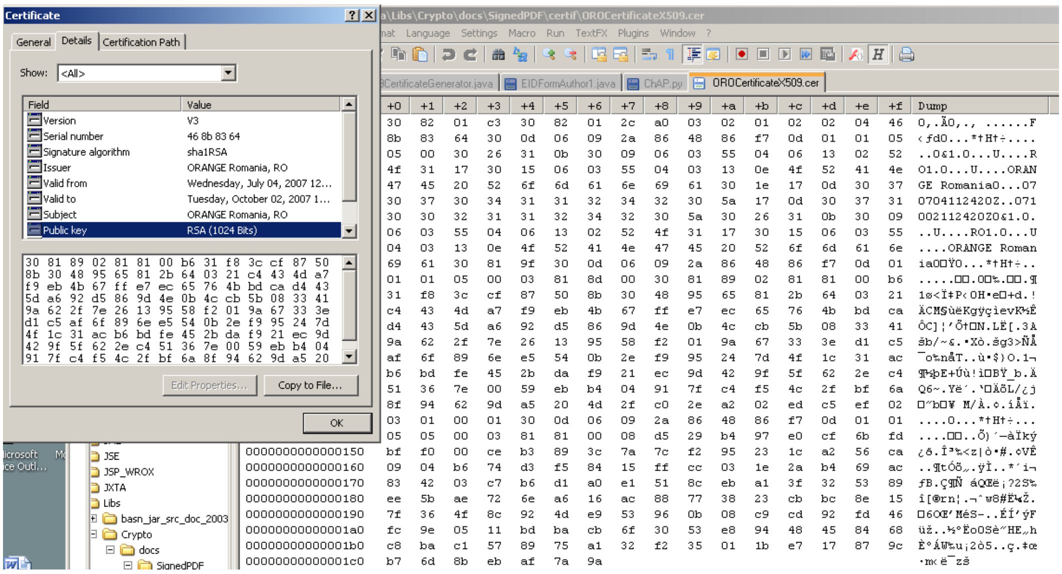

Fig. 7. X509 v3 Certificate in Hex and in MS Crypto Shell

So the interpretation of the results for a Self Signed X509 V3 Certificate in ASN.1

BER is in the following listing [4]:

Tag	Length	Value	ASN 1 Notation	Comments
30	82 01 C3		SEQUENCE {	Byte with value 0x30 is ASN 1 header, complex sequence, the the last digit from the first byte of the length bytes specifies the length in bytes of the structure => 2 bytes with 01C3 value => 451 bytes
30	82 01 2C		SEQUENCE {	ASN Complex Sequence Header with 300 bytes length, to be signed parts begin here
A0	03		[0] {	ASN 1 DER Encoding for first element from the SEQUENCE, has 3 bytes length. Tag 0x0A for an enumerated data type with a length of 3 bytes (0x03) and a content 0x02 0x01 0x02
02	01	02	INTEGER 2 }	Attribute version, 1 byte length with value 2 => X 509 version 3 (starting from 0)
02	04	46 8B 83 64	INTEGER 1183548260	Attribute Serial Number with the value 1183548260
30	0D		SEQUENCE {	Another sequence within first two sequences already opened and has 13 bytes length
06	09	2a 86 48 86 f7 0d 01 01 05	OBJECT IDENTIFIER rsaWithSha1 (1 2 840 113549 1 1 5)	Signature algorithm identifier - sha1RSA = sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)} For 840 from the OID the formula is 0x06*128 + 0x48 (form 0x86 0x48)= 768 + 72 = 840 0x06*2^14 + 0x77*2^7 + 0x0d*2^0 = 6*16384 + 119*128 + 13*1 = 98304 + 15232 + 13 = 113549
05	00		NULL }	Element NULL in ASN.1 BER with 0 bytes length. So, NULL 2 bytes and the SEQUENCE is ending.
30	26		SEQUENCE {	Another sequence within first three sequences already opened and has 38 bytes length (0x26)
31	0B		SET {	A SET with 12 bytes length
30	09		SEQUENCE {	
06	03	55 04 06	OBJECT IDENTIFIER countryName (2 5 4 6)	OID countryName {joint-iso-itu-t(2) ds(5) attributeType(4) countryName(6)}
13	02	52 4F	PrintableString 'RO'	countryName = 'RO' Printable string with 2 bytes length for issuer
			} }	Closed the SET and the SEQUENCE
31	17		SET {	A SET with 23 bytes length
30	15		SEQUENCE {	Another sequence within first three sequences already opened and has 21 bytes length (0x15)
06	03	55 04 03	OBJECT IDENTIFIER commonName (2 5 4 3)	OID commonName {joint-iso-itu-t(2) ds(5) attributeType(4) commonName(3)}
13	0E	4F 52 41 4E 47 45 20 52 6f 6d 61 6e 69 61	PrintableString 'ORANGE Romania'	commonName = 'ORANGE Romania' for issuer Printable String with 14 bytes length
			} } }	Closed the SET and the SEQUENCE and the first above SEQUENCE – the one with 38 bytes length (0x26) for issuer
30	1E		SEQUENCE {	Another sequence within first sequence already opened and has 30 bytes length (0x15)
17	0D	30 37 30 37 30 34 31 31 32 34 32 30 5a	UTCTime '070704112420Z'	A "coordinated universal time" or Greenwich Mean Time (GMT) value, for ***valid from*** YYMMDDHHmmSS format, meaning 04 July 2007 11:24:20
17	0D	30 37 31 30 30 32 31 31 32 34 32 30 5a	UTCTime '071002112420Z'	A "coordinated universal time" or Greenwich Mean Time (GMT) value, for ***valid to*** YYMMDDHHmmSS format, meaning 02 October 2007 11:24:20
			}	Closed the SEQUENCE with 0x1E length

30	26		SEQUENCE {	Another sequence within first sequence already opened and has 38 bytes length (0x26)
31	0B		SET {	A SET with 11 bytes length
30	09		SEQUENCE {	Another sequence within first two sequences and one set already opened and has 09 bytes length
06	03	55 04 06	OBJECT IDENTIFIER countryName (2 5 4 6)	OID countryName {joint-iso-itu-t(2) ds(5) attributeType(4) countryName(6)}
13	02	52 4F	PrintableString 'RO'	countryName = 'RO' Printable string with 2 bytes length for subject
			} }	End of SEQUENCE and SET
31	17		SET {	A SET with 23 bytes length
30	15		SEQUENCE {	Another sequence within first three sequences already opened and has 21 bytes length (0x15)
06	03	55 04 03	OBJECT IDENTIFIER commonName (2 5 4 3)	OID commonName {joint-iso-itu-t(2) ds(5) attributeType(4) commonName(3)}
13	0E	4F 52 41 4E 47 45 20 52 6f 6d 61 6e 69 61	PrintableString 'ORANGE Romania'	commonName = 'ORANGE Romania', for subject Printable String with 14 bytes length
			} } }	Closed the SET and the SEQUENCE and the first above SEQUENCE – the one with 38 bytes length (0x26) for subject
30	81 9F		SEQUENCE {	Another sequence within first sequence already opened and has 159 bytes length (0x9F)
30	0D		SEQUENCE {	Another sequence within first two sequences already opened and has 13 bytes length
06	09	2a 86 48 86 F7 0D 01 01 01	OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)	OBJECT IDENTIFIER rsaEncryption ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) rsaEncryption(1)}
05	00		NULL }	Element NULL in ASN.1 BER with 0 bytes length. So, NULL 2 bytes and the SEQUENCE is ending.
03	81 8D	00 30 81 89 02 81 81 00 b6 31 f8 3c cf 87 50 8b 30 48 95 65 81 2b 64 03 21 c4 43 4d a7 f9 eb 4b 67 ff e7 ec 65 76 4b bd ca d4 43 5d a6 92 d5 86 9d 4e 0b 4c cb 5b 08 33 41 9a 62 2f 7e 26 13 95 58 f2 01 9a 67 33 3e d1 c5 af 6f 89 6e e5 54 0b 2e f9 95 24 7d 4f 1c 31 ac b6 bd fe 45 2b da f9 21 ec 9d 42 9f 5f 62 2e c4 51 36 7e 00 59 eb b4 04 91 7f c4 f5 4c 2f bf 6a 8f 94 62 9d a5 20 4d 2f c0 2e a2 02 ed c5 ef 02 03 01 00 01	BIT STRING 0 unused bits, encapsulates {	BIT STRING with 141 bytes length for RSA Public Key. The second byte is the sequence (0x30), the third and fourth byte are the length of RSA public key, 0x0189. The fith byte specifies an INTEGER of 0x181 bytes length. This is the length of the RSA modulus. After RSA modulus, comes the RSA public exponent. In this case, the RSA public exponent is 0x01 0x01 0x01 (the last 3 bytes).
			}	End BIT STRING
30	0D		SEQUENCE {	Another sequence within first two sequences already opened and has 13 bytes length
06	09	2a 86 48 86 f7 0d 01 01 05	OBJECT IDENTIFIER rsaWithSha1 (1 2 840 113549 1 1 5)	Signature algorithm identifier - sha1RSA = sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}
05	00		NULL }	Element NULL in ASN.1 BER with 0 bytes length. So, NULL 2 bytes and the SEQUENCE is ending.
03	81 81	00 08 d5 29 b4 97 e0 cf 6b fd bf f0 00 ce b3	BIT STRING 0 unused bits, encapsulates {	BIT STRING with 129 bytes length - 1024 bits - for digital signature of Certificates Authority.

		89 3c 7a 7c f2 95 23 1c a2 56 ca 09 04 b6 74 d3 f5 84 15 ff cc 03 1e 2a b4 69 ac 83 42 03 c7 b6 d1 a0 e1 51 8c eb a1 3f 32 53 89 ee 5b ae 72 6e a6 16 ac 88 77 38 23 cb bc 8e 15 7f 36 4f 8c 92 4d e9 53 96 0b 08 c9 cd 92 fd 46 fc 9e 05 11 bd ba cb 6f 30 53 e8 94 48 45 84 68 c8 ba c1 57 89 75 a1 32 f2 35 01 1b e7 17 87 9c b7 6d 8b eb af 7a 9a		This certificate is self signed and the signature is generated with itself private key. In this case all the previous fields from the certificate has been concatenated and signed with the private key and can be verified with itself public key.
			} } }	End BIT STRING End SEQUENCE with 0x81 0x9F length End SEQUENCE with 82 0x01 0xC3 length

So, from the previous table it is obvious the following for the X509 v3 self signed certificate:

- the serial number is 1183548260 (0x46 0x8B 0x83 0x64 hex);
- the certificate is signed with RSA over the SHA-1 hash algorithm;
- the issuer's distinguished name is CN = ORANGE Romania; C = RO;
- the subject's distinguished name is CN = ORANGE Romania; C = RO;
- the issuer and the subject are the same, so the certificate is self signed;
- the certificate was issued on July 04, 2007 11:24:20 and will expire on October 02, 2007 11:24:20;
- the certificate contains a 1024 bit RSA public key;
- the certificate contains the signature of all the fields in last section

The obtained ASN.1 Notation for the certificate from the previous listing of the self signed certificate is [4]:

```
SEQUENCE {
 SEQUENCE {
  [0] {
    INTEGER 2
 }
 INTEGER 1183548260
 SEQUENCE {
  OBJECT IDENTIFIER
  rsaWithSha1(1.2.840.113549.1.1.5)
      NULL
 }
 SEQUENCE {
   SET {
     SEQUENCE {
      OBJECT IDENTIFIER countryName (2.5.4.6)
      PrintableString ‘RO’
     }
   }
   SET {
    SEQUENCE {
     OBJECT IDENTIFIER commonName (2.5.4.3)
     PrintableString ‘ORANGE Romania’
    }
   }
 }
 SEQUENCE {
   UTCTime ‘070704112420Z’
   UTCTime ‘071002112420Z’
 }
 SEQUENCE {
   SET {
     SEQUENCE {
         OBJECT IDENTIFIER countryName (2.5.4.6)
         PrintableString ‘RO’
     }
   }
   SET {
     SEQUENCE {
         OBJECT IDENTIFIER commonName (2.5.4.3)
         PrintableString ‘ORANGE Romania’
     }
   }
 }
SEQUENCE {
  SEQUENCE {
   OBJECT IDENTIFIER rsaEncryption (1.2.840.11354
     9.1.1.1)
   NULL
  }
  BIT STRING 0 unused bits, encapsulates { //public
     key}
  SEQUENCE {
   OBJECT IDENTIFIER rsaWithSha1 (1.2.840.113549
      .1.1.5)
   NULL
  }
  BIT STRING 0 unused bits, encapsulates { //signat
      ure}
 }
}
```

The systems for obtaining public keys based on certificates are simple and cheap to implement, due to an important characteristic of the digital certificates: the certificates may be distributed without requiring protection through the average security systems (certification, integrity and confidentiality). This because the public key should not be kept secret; thus, the digital certificate that includes it is not secret. There are no requirements for certification or integrity, because the certificate self-protects (the digital signature of AC in the certificate ensures its certification, as well as its integrity).
Consequently, the digital certificates may be distributed and moved by unsecured communication liaisons: by unsecured file servers, by systems of unsecured folders and/or communication protocols that do not endure the security.

3. Protocols for Certificates Management

The interaction between the components of a public key infrastructure requires the existence of some protocols for the certificates management. The elements involved in the PKI management are:

- the subject of a certificate, that may be a person or an application and represents the final entity (EE – End Entity);
- Certification Authority – CA, that starts a digital certificate, coupling the identity of an user with his public key and certifies this association;
- AI (Registration Authority – RA) – certification of the persons, name deliverance key generation.

The protocols between the mentioned above are used for the following scopes:

- Establishing CA: when a new CA is established, there must be taken certain steps, such as generating the initial list of revoked certificates or the export of the CA's public key.
- Initializing the final entity: involves obtaining the public key of the root CA and the information request about the PKI management at the level of the final entity
- Certification:
  - Initial registration/certification – when an end entity becomes known to a Certification Authority (CA). After this process CA generates one or more certificates for that end entity.
  - Establishing a pair of keys – it is necessary that each pair of keys to be regularly changed and a new certificate to be issued.
  - Updating a certificate – when a certificate expires, it must be update.
  - Changing the CA's pair of keys.
  - Request for crossed certificates – when a certification authority certifies another certification authority.
  - Updating some crossed certificates
- Publishing of a certificate or list of revoked certificates: involves storing a certificate or a list of revoked certificates where everybody may have access (for instance such a protocol is LDAP).
- Restoration of a pair of keys: when an end entity loses its private key and wishes restoration, if previously RA or CA saved this key.
- Revoking a certificate: when an end entity wishes to revoke (cancel) a certificate, operation that involves a revocation demand and implicitly the update of the list of revoked certificates (CRL – Certificate Revocation Lists).

It is not necessary that these operations should be executed on-line, also existing off-line methods for their fulfillment. It is impossible to find a Certification Authority which to issue certificates for all the owners of pairs of public/ private keys in the world, because it is not practical that all the users in the world trust a single organization or company in what concerns their secret communications. That is why is accepted the idea of the multiple Certification Authorities. Moreover, a single user can have certificates issued by different CAs, for different types of secure communications he wants to establish. As well, practically it is not possible to presume that a users of the public key infrastructure (PKI) already holds the public key of a certain certification authority, CA_1, which issued a certificate for an entity with whom that user wants to secure communicate. Although, in order to obtain the public key of that CA_1, the user may use another certificate, i.e. a certificate issued for that CA_1 by another certification authority, CA_2, whose public key is hold in a secure way by the user. So, the procedure is recursively applied and a user may obtain the public keys for a constantly larger number of Certification Authorities and, correspondingly, the public keys for a constantly larger number of other users. This leads to a general pattern, called certification chain of certification path, on which are based all the main present systems for public keys distribution.

The model of building the certification chain is reproduced in figure 8.

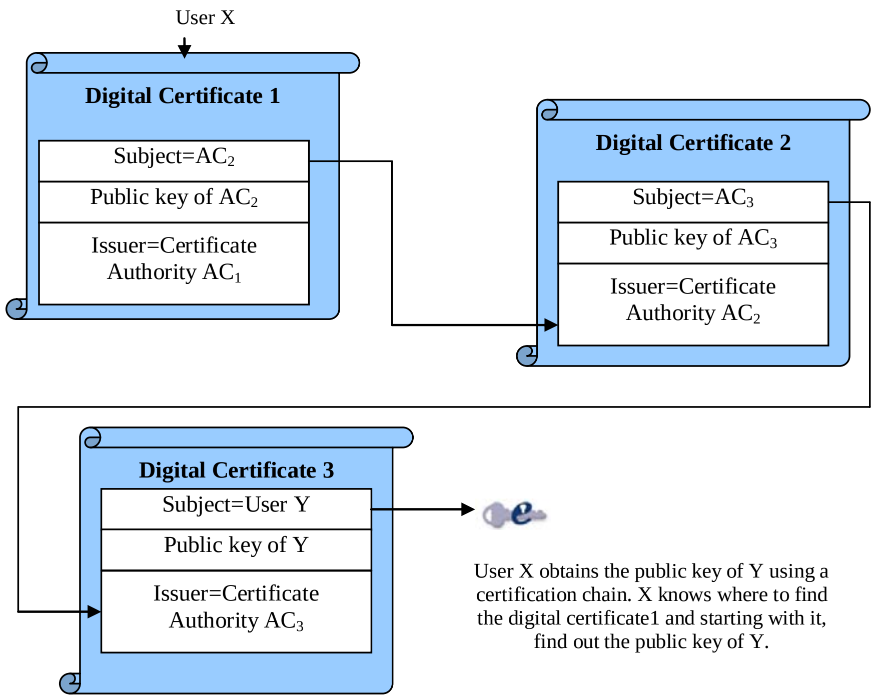

Fig. 8. Certification Chain

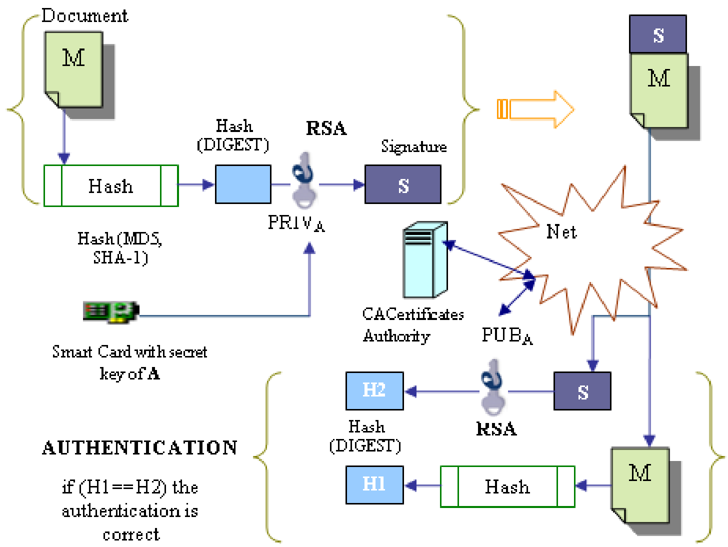

Fig. 9. Digital Signature involving Digital Certificates

The certification is so absolutely necessary in an infrastructure based on cryptography with public keys. But together with this come problems that must be solved, such as: certificate acquisition, its recognition, revocation, distribution and validation. The description of a standard digital signature that involves public certificates is presented in figure 9.

In figure 9 the sender A hashes the original message M. The digest is sent into the smart card that process RSA with the private key stored in secure manner on the smart card. After the smart card processing the signature is attached to the original message using PKCS. The message with signature arrives to the receiver B. The receiver B gets the public key of A using a certification chain from the certificates stored on CA's servers. The public key of A and the signature are the inputs for RSA algorithm and it is obtained the digest H2. The original message is hashed with a dispersion function (MD5 or SHA-1) and it is obtained H1. If H1 is the same as H2 then the signature is valid.

4. Conclusions

The study is important because it seems possible to have different certificates with the same signature value – last field in ASN.1 sequence of a certificate. More information about collisions is in [2]. They "announce a method for the construction of pairs of valid X.509 certificates in which the 'to be signed' parts form a collision for the MD5 hash function. As a result the issuer signatures in the certificates will be the same when the issuer uses MD5 as its hash function". Also, they show "that MD5 collisions can be crafted easily in such a way that the principles derlying the trust in Public Key Infrastructure are violated". There is in research how a tificate trust may be violated by the exploited of MD5 collision over digital certificates. That's why it is important to analysis at byte level of ASN.1 DER/BER encoding of the certificates and their implications in the lic key infrastructures.

References

[1] B. Kaliski Jr. *A Layman's Guide to a Subset of ASN.1, BER, and DER*, RSA Publishing House, USA 1993.

[2] A. Lenstra, X. Wang, and B. de Weger *Colliding X.509 Certificates*, http://www.win.tue.nl/~bdeweger/CollidingCertificates/

[3] http://www.oid-info.com/ cgi-bin/display

[4] C. Toma. *Security in Software Distributed Platforms*, AES Publishing House, Bucharest, 2008.

[5] W. Rankl and Effing. *Smart Card Handbook 3rd Edition*, John Wiley & Sons Publishing House, USA 2004, reprinted 2007.

Cristian TOMA has graduated from the Faculty of Economic Cybernetics, Statistics and Informatics, Economic Informatics specialization, within Academy of Economic Studies Bucharest in 2003. He has graduated from the BRIE master program in 2005 and PhD stage in 2008. In present, he is assistant lecturer at Economic Informatics Department and he is involved in IT&C Security master program. His research areas are in: distributed and parallel computing, mobile applications, smart card programming, e-business and e-payment systems, network security, computer anti-viruses and viruses, secure web technologies and computational cryptography. He is teaching assembly language, object oriented programming, data structures, distributed applications development and advanced programming languages in Economic Informatics Department and he has published 2 books and over 30 papers in indexed reviews and conferences proceedings.