# Verifying X.509 Certificates on Smart Cards

Olaf Henniger, Karim Lafou, Dirk Scheuermann, and Bruno Struif

***Abstract*—**This paper presents a smart-card applet that is able to verify X.509 certificates and to use the public key contained in the certificate for verifying digital signatures that have been created using the corresponding private key, e.g. for the purpose of authenticating the certificate owner against the card. The approach has been implemented as an operating prototype on Java cards.

***Keywords*—**Public key cryptographic applications, smart cards.

## I. Introduction

SMART cards are used as tamper-resistant devices for creating digital signatures, but usually not for verifying digital signatures of unknown origin [1]. A reason for this is the difficulty of verifying X.509 certificates, i.e. electronic documents in the format defined in the ITU-T recommendation X.509 [2] for binding a public key to its owner. The verification of digital signatures is usually carried out in the host PC. The public key of the root certification authority is stored as trust anchor on the hard drive and must be protected there.

The smart-card applet presented in this paper is able to verify X.509 certificates and digital signatures. This allows storing and retaining the trust-anchor public key in the smart card, where it remains securely protected against tampering. When the verification result and the signed data are finally presented to the user via the host PC, still the same attacks are possible as if when the entire signature verification process is carried out in the host PC. Nevertheless, keeping back the trust anchor in the smart card is useful to better ground the trust in it.

The smart card applet can use the public key contained in the certificate for verifying digital signatures that have been created using the corresponding private key. The digital signatures to be verified could, for instance, be created for the purpose of authenticating the certificate owner against the card in a challenge-response protocol using public-key cryptography. For the purpose of device authentication, [1] defines so-called card-verifiable certificates (CVC's). The attribute "card-verifiable" may suggest that other certificate formats, like X.509 certificates or PGP (Pretty Good Privacy) certificates, could not be verified on smart cards, but this is not entirely true. It is convenient if the authentication against the card can be carried out using the widely used X.509 certificates, without a card-specific certificate format.

Manuscript received August 31, 2006.

O. Henniger (phone: +49-6151-869-264; fax: +49-6151-869-224; e-mail: henniger@sit.fraunhofer.de), K. Lafou, D. Scheuermann, and B. Struif are with the Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany.

Related work on verifying X.509 certificates on smart cards has been done in the context of Java cards for authentication in wireless LAN networks [4].

The remainder of this paper is organised as follows. Section II introduces the structure and usage of X.509 certificates. Section III describes the implementation platform. Section IV describes the design of an X.509 parser suitable for smart cards. Section V suggests applications and extensions.

## II. Public-Key Certificates

### A. *Structure of X.509 Certificates*

A public-key certificate is a mechanism for binding a public key to its owner, which can be a person, organisation, or device. A certification authority (CA), usually a trusted third-party commercial service provider, binds a public key to its

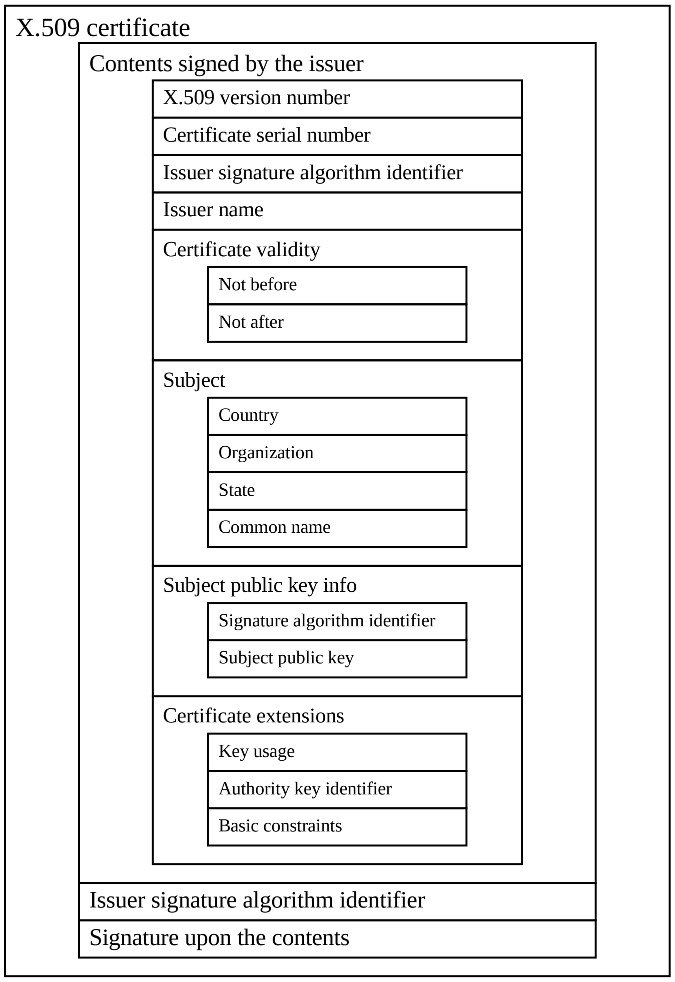

Fig. 1 Structure of an X.509 certificate

owner by digitally signing the public key together with data identifying the owner. These signed data together form a public-key certificate.

A widely used format for certificates is defined in the ITU-T recommendation X.509 [2]. X.509 is adapted to the Internet in [3]. These specifications define what information, and in what form, a certificate must or may contain. Fig. 1 illustrates the basic structure of X.509 certificates. An X.509 certificate contains the following attributes:

– Version number: Indication of the X.509 version (must be version 3 if certificate extensions are present). Most currently valid X.509 certificates follow version 3.
– Serial number: Unique serial number of the certificate;
– Issuer signature algorithm identifier: Identifier for the signature algorithm used by the issuing CA for signing the certificate contents;
– Issuer: Name of the issuing CA;
– Validity: Validity period for this certificate;
– Subject: Name or alias of the certificate owner;
– Subject public key info: Public key of the certificate owner and identifier for the signature algorithm with which the key is to be used;
– Certificate extensions: Entry for extensions that are attached to the certificate and that cover information about keys and procedures, attributes of owners and issuers, and constraints of the certification path. The standard fields of X.509 certificates turned out not to be sufficient for many applications. Therefore, the syntax of version 3 was extended to allow including additional data. An extension is a triple (type, criticality, and value of the extension). Extensions marked as critical must not be ignored.

Fig. 2 shows the contents of an example X.509 certificate in textual representation (cryptographic data are shortened).

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 3 (0x3)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=DE, L=Darmstadt, O=Trust Me Ltd,
      OU=CA, CN=Premium CA
    Validity
      Not Before: Jul 7 14:42:16 2005 GMT
      Not After:  Aug 4 14:42:16 2007 GMT
    Subject: C=DE, CN=Karim /
      Email=nospam@nowhere.my
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:cb:41:02:03:45:ad:d1:a2:84:f8:c5:dc:6c:
          [..]
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Key Usage:
        Digital Signature, Non Repudiation
  Signature Algorithm: md5WithRSAEncryption
  Signature:
    7c:5e:9b:e6:6d:52:c0:aa:b4:f9:3a:68:18:05:b8:84:
    [..]
```

Fig. 2 Example of X.509 certificate contents

```
Certificate  ::=  SEQUENCE {
  tbsCertificate     TBSCertificate,
  signatureAlgorithm AlgorithmIdentifier,
  signatureValue     BIT STRING
}
TBSCertificate  ::= SEQUENCE {
  version         [0]  EXPLICIT Version DEFAULT v1,
  -- If extensions are present, version MUST be v3
  serialNumber         CertificateSerialNumber,
  signature            AlgorithmIdentifier,
  issuer               Name,
  validity             Validity,
  subject              Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo,
  extensions      [3]  EXPLICIT Extensions OPTIONAL
}
Name ::= CHOICE {RDNSequence}
RDNSequence ::= SEQUENCE OF
  RelativeDistinguishedName
RelativeDistinguishedName ::= SET OF
  AttributeTypeAndValue
AttributeTypeAndValue ::= SEQUENCE {
  type  AttributeType,
  value AttributeValue
}
AttributeType ::= OBJECT IDENTIFIER
AttributeValue ::= ANY DEFINED BY AttributeType
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm        AlgorithmIdentifier,
  subjectPublicKey BIT STRING
}
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
  extnId    OBJECT IDENTIFIER,
  critical  BOOLEAN DEFAULT FALSE,
  extnValue OCTET STRING
}
```

Fig. 3 Specification of an X.509 certificate in ASN.1

For specifying the structure of X.509 certificates, [2] and [3] use the Abstract Syntax Notation One (ASN.1) [5]. Fig. 3 shows the specification of an X.509 certificate in ASN.1 [3]. Section IV.A refers back to some of these definitions. For brevity, some optional data elements that may be ignored and some definitions that are not used in this paper have been omitted from Fig. 3.

For a uniform representation of data in different computer systems, a platform-independent encoding is needed. X.509 certificates are encoded using the Distinguished Encoding Rules (DER) for ASN.1 [6], which yield a unique binary representation. Each data element is encoded as a sequence of tag, length, and value. The tag octets identify the type of the data element, the length octets indicate its size, and the value octets contain the actual contents of the data element.

### B. *Structure of Card-Verifiable Certificates*

Unlike X.509 certificates, CVC’s are signed using a signature scheme with message recovery [1]. This saves space and time because essentially only the signature value has to be stored and to be transmitted rather than the signature value together with the plaintext contents. Furthermore, CVC’s do not contain data elements that are hard to be verified on a card, like the validity period.

A CVC contains a certificate holder authorisation (CHA) data element that identifies the role that the certificate holder is allowed to take on in a smart-card application. Checking the

CHA data element contained in a verified certificate may be part of a file-access rule that is enforced on the card [7]. The CHA data element allows giving the right to access a file on the smart card to all individual certificate holders that may take on a certain role, e.g., all certified pharmacists. Role-based authentication using X.509 data elements is not yet specified in [7].

### C. *Verification of Certificates*

A digital signature, created using some private key, can be verified using the corresponding public key. To ensure that the public key that is used really belongs to the remote signer, the corresponding public-key certificate has to be verified. To verify an X.509 certificate, the verifying system has

- to verify the digital signature of the issuing CA upon the certificate contents,
- to check whether the current time is within the validity period of the certificate, and
- to check whether the certificate serial number is not on a suitably recent certificate revocation list (CRL).

In case the public key of the issuing CA is not the trusted anchor, verifying the CA's digital signature would require to verify the CA's public-key certificate; and so on, till finally a certificate is reached that can be verified using the trust anchor. A certification path (or chain) starts with the signer certificate and may proceed through a number of intermediate certificates up to a certificate issued by a trusted CA.

## III. Java Cards as Implementation Platform

Java cards are smart cards with an interpreter (Java Card Virtual Machine) [8] for the execution of processor-independent byte code. Code development for Java cards is based on a subset of Java and Java development tools. Java cards are very well suited for the rapid development of prototype smart-card applications. Their computing speed is rather limited because of the limitations of the smart-card hardware and because the Java byte code is interpreted at run-time. However, thanks to the use of crypto-coprocessors, cryptographic operations on Java cards are in general not slower than on other smart cards.

On Java cards, only a subset of the Java language is available. Java cards support the data types *Boolean*, *byte*, *short*, and optionally also *int*; the data types *char*, *double*, *float*, and *long* are not available. Only one-dimensional arrays are supported. By default, only basic arithmetic operations and no mathematical libraries are available. There is no garbage collection. Objects and arrays once created cannot be deleted; their storage location remains occupied. Therefore, all necessary objects and arrays have to be created when the Java card applet is installed and to be reused later. Dynamic loading of classes is not supported; all necessary classes must be brought onto the Java card at production time or during the installation of a Java card applet.

The Java cards deployed as implementation platform have a CPU word length of 8 bit, 2300 bytes of RAM, 32 Kbytes of EEPROM, and a default clock rate of 3.5712 MHz [9].

## IV. X.509 Parser

### A. *Outline of the Algorithm*

The task of the X.509 parser is to analyze a given X.509 certificate and to extract from it the data elements that are needed for verifying and for using the certificate. The resource constraints on Java cards need to be taken into consideration in the design and implementation of the X.509 parser. Not all possible approaches to parsing X.509 certificates are applicable on smart cards. This section describes a concept of an X.509 parser that is suitable for smart cards.

The X.509 certificate is given as a one-dimensional array on the card. The main ideas of the X.509 parser algorithm are:

- The X.509 certificate is scanned only once. All required information is collected during this run.
- For each required data element the offset, i.e. the distance from the beginning of the array, and the length are stored. Using its offset and length, each data element can be accessed directly from the array. This approach saves memory space as it avoids the duplication of data elements, and also time.

What information from an X.509 certificate is needed for verifying and for using the certificate? In the following, the most important fields of the X.509 certificate, whose offsets and lengths are collected, are listed:

- Name of the certificate owner: The name could consist of several relative distinguished name (RDN) elements (cf. Fig. 3), e.g. common name, organisational unit, organisation country, distinguished name qualifier, state and province name. Since their order is not fixed, one cannot access a certain RDN element by index. Instead, each RDN element's attribute type is compared with the globally unique object identifiers (OID) of the desired RDN elements until they match. If they match, then offset and length of the desired RDN element are noted.
- Public key of the certificate owner;
- Intended usage of the key: The key-usage certificate extension determines the purpose of a certificate. More precisely, it specifies the cryptographic operations that may be accomplished with the key pair. As the order of the certificate extensions is not fixed, one cannot access a certain extension by index. Instead, each extension's OID (cf. Fig. 3) is compared with the OID of the desired extension until they match. If they match, then offset and length of the key-usage extension are noted.

The temporal validity of an X.509 certificate is usually checked based on the current system time and the validity entries in the certificate. However, checking the temporal validity is not easily possible on today's smart cards since they do not possess a system clock. The current implementation ignores the validity entries in the certificate.

### B. Time Complexity

The time complexity of a problem is a measure for the number of steps that an optimal algorithm would need for solving the problem, as a function of the size of the input. Applied to a concrete algorithm, the term time complexity refers, in general, to a measure for the number of steps that this algorithm takes in the worst case, as a function of the size of the input. The exact duration of the program execution on a certain machine is only of secondary interest. Of primary interest is how the duration grows when the size of the input grows. Due to the resource constraints on smart cards, it is important to analyze the time complexity of the X.509 parser algorithm as a function of the certificate size $n$ in bytes.

The X.509 parser algorithm does not process the certificates byte by byte, but attribute by attribute. Because the number of possible attributes is fixed in the specification, their maximum number is constant. Hence, the number of steps required for parsing an X.509 certificate is independent of the certificate size $n$ in bytes; it depends only on the (upper bounded) number of attributes in the certificate. Thus, the X.509 parser algorithm has a constant time complexity $\mathrm{O}(1)$.

This is to the users' advantage. The lower the time complexity of an algorithm, the faster the algorithm will perform its work in practice. Anyway, though parsing does not take longer when the certificate contains a longer public key, verifying the CA's digital signature will take longer, the longer the signature value and the public key of the CA are.

### C. Space Complexity

Due to the limited memory space on smart cards, not only the time complexity but also the space complexity of the X.509 parser algorithm is important. The space complexity of an algorithm is a measure for the number of memory cells that the algorithm needs. Again, of primary interest is how the memory requirement grows when the size of the input grows, but not the exact memory space needed.

In order to process a certificate of the size $n$ bytes in a smart card, an array of size $n$ bytes for storing the certificate and a certain constant number of auxiliary variables for storing the offsets and lengths of the relevant certificate attributes are needed. Thus, the algorithm has linear space complexity $\mathrm{O}(n)$. The longer the certificates and keys are, the more storage space is needed on the smart card.

## V. Conclusions

The presented solution supports an essential part of the verification of X.509 certificates on smart cards. It supports the verification of the signature of a CA upon the certificate contents, yet it does not support checking the temporal validity of the certificate and checking whether the certificate has been revoked.

The temporal validity of a certificate is normally checked based on the current system time and the validity entries in the certificate. However, this examination is not possible on today's smart cards since they do not possess a clock. The current time needs to be communicated to the card in a trustworthy way. A solution of this problem could be that the smart card approximates the current time using the most recent not-before date found in a certificate verified on the card, starting from the smart card's personalisation date, as suggested in [10].

Checking whether the certificate has been revoked or not is not possible since the smart card cannot establish a direct connection to a CRL server. A solution of this problem could be that the host PC communicates the CRL to the card in a trustworthy way. Then, the CRL can be examined on the card.

Due to the resource constraints on smart cards, the time complexity and space complexity of the algorithm play important roles. For algorithms implemented on smart cards, both time and space complexity are desirable to be as low as possible. The parser algorithm has a constant time complexity, i.e. the number of steps required is independent of the certificate size. In addition it has linear space complexity, i.e. the longer the certificates and keys are, the more memory space is required on the smart card that contains the X.509 parser.

The design and implementation of the smart-card applet take the existing standards into consideration to ensure its interoperability with existing signature applications. To put the X.509-based authentication on smart cards into practice, the relevant specifications should be extended to allow formulating role-based access rules using X.509 data elements.

## Acknowledgment

The authors are grateful to Ulrich Waldmann for fruitful discussions.

## References

[1] *Application interface for smart cards used as Secure Signature Creation Devices – Part 1: Basic requirements*, CEN Workshop Agreement CWA 14890-1, 2004

[2] *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks,* ITU-T Recommendation X.509, 2000

[3] R. Housley, W. Polk, W. Ford, and D. Solo, *Internet X.509 Public Key Infrastructure certificate and certificate revocation list (CRL) profile,* Request for Comments RFC 3280, 2002

[4] P. Urien, M. Badra, and M. Dandjinou, "EAP-TLS smartcards, from dream to reality", in *Proc. 4th IEEE Workshop on Applications and Services in Wireless Networks*, Boston, Massachusetts, USA, 2004

[5] *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*, ITU-T Recommendation X.680, 2002

[6] *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),* ITU-T Recommendation X.690, 2002

[7] *Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange,* International Standard ISO/IEC 7816-4, 2005

[8] *Java Card 2.1.1 Virtual Machine Specification*. Sun Microsystems, Revision 1.0, May 2000

[9] *JCOP20 Technical Brief.* Revision 2.3. IBM

[10] *Technical guideline: Advanced security mechanisms for Machine-Readable Travel Documents*, German Federal Office for Information Security (BSI), TR-03110, version 1.0, 2006