Parsing of X.509 certificates in a WAP environment

Examensarbete utfört i informationsteori
av
Fredrik Asplund
LiTH-ISY-EX-3288-2002
2002
Parsing of X.509 certificates in a WAP environment

Examenarbete utfört i informationsteori
vid Linköpings tekniska högskola
av

Fredrik Asplund, 771115-6930
LiTH-ISY-EX-3288-2002

Handledare: Per Ståhl
Examinator: Viiveke Fåk
Linköping 2002-11-13
Parsning av X.509 certifikat i en WAP-miljö

This master thesis consists of three parts. The first part contains a summary of what is needed to understand a X.509 parser that I have created, a discussion concerning the technical problems I encountered during the programming of this parser and a discussion concerning the final version of the parser. The second part concerns a comparison I made between the X.509 parser I created and a X.509 parser created "automatically" by a compiler. I tested static memory, allocation of memory during runtime and utilization of the CPU for both my parser (MP) and the parser that had a basic structure constructed by a compiler (OAP). I discuss changes in the parsers involved to make the comparison fair to OAP, the results from the tests and when circumstances such as time and non-standard content in the project make one way of constructing a X.509 parser better than the other way. The last part concerns a WTLS parser (a simpler kind of X.509 parser), which I created.

Nyckelord

Computer Security, Parser, Program comparison, Programming, Public key encryption, WTLS certificate, X.509 certificate
Abstract

My master thesis can be divided into three parts. First I was to program a X.509 parser, then I was to compare that parser to another parser that had a basic structure created by a compiler and after that I was to program a verifier to accompany the parser.

Chapter 1 contains the technical and social background to my master thesis.

Chapter 2 is the specification of my master thesis.

Chapter 3 contains a summary of what is needed to understand my X.509 parser, a discussion concerning the technical problems I encountered during the programming and a discussion of the result from this part of the master thesis.

Chapter 4 concerns part two of my master thesis, the comparison. I tested static memory, allocation of memory during runtime and utilization of the CPU for both my parser (MP) and the parser that had a basic structure constructed by a compiler (OAP). I discuss changes in the parsers involved to make the comparison fair to OAP, the results from the tests and when circumstances such as time and non-standard content in the project make one way of constructing a X.509 parser better than the other way.

The last part of my master thesis that was about creating a verifier was changed into creating a WTLS parser, which is a simpler kind of X.509 parser. I completed such a parser and have included a correction of Ericsson’s specification for this work in this essay as chapter 5.
Index

Abstract ................................................................. 4  
1. Background .......................................................... 6  
  1.1. Technical background ......................................... 6  
  1.2. Social background ............................................. 6  
  1.2.1. My situation .................................................. 7  
  1.2.2. The situation at the company ............................... 8  
2. The specification of my master thesis ............................ 10  
3. The X.509 parser .................................................... 11  
  3.1. General summary ............................................... 11  
  3.1.1. X.509 certificates ............................................ 11  
  3.1.2. Abstract Syntax Notation One (ASN.1) .................. 14  
  3.1.3. Basic Encoding Rules (BER) ............................... 15  
  3.1.4. Distinguished Encoding Rules (DER) .................... 17  
3.2. Clarifications ...................................................... 17  
  3.2.1. X.509 vs. WAP certificates ................................. 17  
  3.2.2. BER vs. DER encoding ..................................... 18  
  3.2.3. The handle .................................................... 18  
  3.2.4. The ASN.1 definition of DSA ............................... 19  
  3.2.5. Ericsson’s error handling .................................. 19  
3.3. The end result ..................................................... 20  
4. The comparison ..................................................... 21  
  4.1. Something about OAP .......................................... 21  
4.2. Static memory .................................................... 21  
  4.2.1. Changes ....................................................... 21  
  4.2.2. The test ....................................................... 22  
  4.2.3. Observations ................................................. 22  
  4.2.4. Conclusions .................................................. 22  
4.3. Allocation of memory during runtime ......................... 22  
  4.3.1. Changes ....................................................... 22  
  4.3.2. The test ....................................................... 22  
  4.3.3. Observations ................................................. 23  
  4.3.4. Conclusions .................................................. 23  
4.4. Utilization of the CPU .......................................... 23  
  4.4.1. Changes ....................................................... 23  
  4.4.2. The test ....................................................... 23  
  4.4.3. Observations ................................................. 24  
  4.4.4. Conclusions .................................................. 24  
4.5. General conclusions ............................................ 24  
5. The third part ....................................................... 25  
6. Summary and conclusions .......................................... 26  
7. References .......................................................... 28  
Appendix A .............................................................. 29  
Appendix B .............................................................. 35  
Appendix C .............................................................. 41  
  Table 1 ................................................................. 41  
  Table 2 ................................................................. 42  
  Table 3 ................................................................. 42  
Appendix D .............................................................. 44
1. Background
This chapter details the circumstances that led up to and influenced my master thesis. First the technical background is described and thereafter I discuss relevant social circumstances at my workplace.

1.1. Technical background
Public key encryption involves two keys\(^1\) - a public key and a private key - that both are associated with an entity that wishes to authenticate its identity electronically, sign data and/or encrypt data. Knowledge of one key in the pair does not infer any knowledge about the other key. Anyone who has the public key can encrypt data and be confident that only those who have access to the private key can decrypt the encrypted data.

![Figure 1. Simplified view of public key encryption.](image)

The opposite is also true, data encrypted with the private key can only be decrypted correctly with the public key. Because of this the owner of the private key can be confident in being the only entity capable of creating signatures that those who have access to the public key recognize as valid ones.

The properties of public key encryption is clearly useful since an entity who wishes to use it only has to keep its private key secret, while the public key can be distributed freely. Left to solve is the problem that entities that retrieve a public key must be able to determine if it is the correct public key for the entity they wish to reach or a fake.

Because of the properties of public key encryption it is used extensively in Internet architecture and there the solution to the problem above often is the use of public key certificates (i.e. the requested public key together with useful data signed by some entity who in some way is trusted by the entity who requests the public key).

Today WAP technology has been developed to make it possible for users to surf on the Internet with mobile phones. The developers of this technology have studied the X.509 specification\(^2\), which is part of the X.500 specifications that associate a unique certificate (the form of the certificate is also defined) containing a public key with every user in a system, when making the technology secure. Among other things the form of the X.509 certificate has been used as a basis for the recommendation\(^3\) of what a WAP-certificate should contain.

1.2. Social background
The following chapters will all be about what I did at Ericsson and not how it was to work there. This does not have to be an important or even interesting issue, but I believe that the situation at my workplace created some important problems that I had to overcome to do my work and this made this issue worth discussing.

\(^1\) The term "key" is used to rely the circumstances present -i.e. encryption- and is actually a number. The algorithm used defines which numbers that can be used.

\(^2\) [8]

\(^3\) [2]
All this was nothing that I had not anticipated, since I have done work for large companies before. I knew that these companies often have some kind of social structure that if not understood properly can complicate your work. Important to note is that the important thing about these analyses is that I actually made them. They are not overly well supported by theory or -due to a limit on how many pages I could spend on this- argument. They do however, explain some decisions I made when programming my parser (for example when to stop programming some parts of it), which is something I will have reason to note later in this essay. That is their merit and the reason they are here at all.

1.2.1. My situation

When analyzing the problems I encountered I first had to differ between problems created by the fact that I was new at a workplace and those created by the workplace itself. This was because the problems created by the fact that I was new at Ericsson were created by me and therefore much more likely to be solvable (to change yourself is often simpler than to change something external). There were some problems created by the workplace that I could solve by changing things there, but they were not many and therefore I thought it was better to make this separation of problems to see which were the more easily solvable. This had two benefits. First of all I did not build up any stress or anxiety by thinking that I could change things I in fact probably had no control over and secondly I did not let problems that were solvable stay unsolved.

So, what problems were created by the fact that I was new at Ericsson? Well, not many really. I did not work with anyone else on the master thesis and most of the work was programming, so apart for the contact with those working on the same project the only contact I ever had with “Ericsson” was that I sat in a building with that name on the wall outside while I worked. This of course makes this analysis less general and more likely to be erroneous, but as I previously noted the importance of this analysis lies in the way it affected me and nothing else.

One thing that became an issue almost instantly was that I did not have any knowledge about the computer software at Ericsson. The development tools for incorporating a program into the Ericsson program structure required both knowledge about the software and knowledge about the system in general. A coworker at Ericsson was supposed to teach me and some other people doing their master thesis this, or rather -since a complete teaching would take too much time- set up the system for us. Eventually he had not time to set up this for me and later on other things became more important so that the last part of the specification of the master thesis unfortunately never was realized.

The next notable thing was that since I did not have access to any direct way to test my program against the Ericsson system it was even more important that I could ask the right people questions about my work. This was of course hard since the organization was unfamiliar to me, which meant that I was often very surprised about trivial things such as error management or the lack of it. Luckily I could turn to my supervisors for help regarding this so that the right people could give me an answer to the most important questions in a reasonable time.

Apart from this there were not many notable problems created by me being new at Ericsson. There was the usual amount of bad luck, for example I did not have full access to a proper compiler for my parser until three weeks after schedule, but nothing apart from the ordinary.
1.2.2. The situation at the company
During my time at Ericsson this company was in a bit of turmoil. I was therefore not 
surprised to find that the social structure at Ericsson had been influenced by something I 
guessed was regression as a social defense (i.e. the way of countering anxiety created by 
demands from inner or outer sources by trying to avoid the problem). These problems 
were not in any way impossible to overcome so the workplace was not a bad one, but they 
were problems that had to be consciously overcome.

1.2.2.1. Diffusion of responsibility
The first thing I noticed was something I put down to be a general lack of knowledge about 
who was actually responsible for what. This for instance meant that the specifications for 
the programming often contained errors that someone that had given the whole structure 
a thought would not have done; errors that later were hard to correct since "someone else" 
with less important work on their hands could always decide about them. It was as if those 
who created certain specifications did not have enough knowledge about that to be 
 specified to make a correct job, but rather were chosen to do this simply because they 
were considered as capable in general. They seemed to balk at claiming responsibility for 
some parts of the project.

All this could of course be explained by the fact that many of the specifications were 
created while the work on other parts of the project was still unfinished or even 
unspecified. I doubt this explanation however since the errors did not concern 
contradictions between parts, but rather strange data structures that were incapable of 
supporting the original X.509 specifications or the usage of words like for example 
“session handling” when a simple list of pointers was the result when the specifications 
were implemented.

It was also not simply a case of many mistakes in the specifications. There were of 
course these too, for example I spent a week incorporating a sorting algorithm that was 
eventually proven unnecessary, but these were not simple things that a proper reading of 
fundamental documentation could have helped avoid.

This was a serious problem since I was forced to leave big holes in my programming 
for later to compensate for functions I could not be given a straight answer about how they 
should be done. Faced with this dilemma I had to analyze the pattern of errors to decide 
what to do about them. I put the pattern of some very specific parts of the parser not being 
correctly specified down to a phenomenon called diffusion of responsibility. Other 
possible explanations could for example be faults in my own person, the technical system 
being implemented itself or simply some coworkers at Ericsson being too arrogant to work 
with me. At that moment I however did not -and I still do not- think these explanations or 
any other -especially those based solely on individuals- plausible. I consider this 
phenomenon noticeable at Ericsson if not a serious problem for someone not new to the 
organization.

I could of course not do much about this problem since I was nowhere near being 
responsible for anything like specifications myself. To counter it I tried to take control of as 
much of the development of the inner workings of the parser as I could and always keep 
myself active in the discussions about the parser. This kept the diffusion of responsibility 
from doing anything except augmenting other problems at the workplace.

---

[11], page 115.
1.2.2.2. Silence and denial

This augmentation that the diffusion of responsibility was responsible for was mostly concerning my cooperation with a group of people in Germany who worked on the same project. They did a lot of good work on the project and were nice people who did not seem to put it below them to help someone working on a master thesis, the problem was that they seemed to be very stressed and therefore sensitive to the structural problems Ericsson had. This was noticeable as silence and denial\(^5\). The reason that I did not think these phenomena stemmed from other things than the way group relations had been affected at Ericsson was simply that they also affected my supervisor. Of course other explanations could be valid, such as language barriers or arrogance, but since I could not find a solvable source to the problems I did not change my opinion about these phenomena during my work there.

Silence is when for example information and questions are met with silence. This happened often enough and at last it was noticeable even to the others on the project. At times answers to important emails were unrelated to the question or simply did not come.

Denial is when information or instructions are partly ignored, someone hear only what that person wants to hear. This showed itself in some corrections to problems, which did not solve the problem but rather changed something unrelated. The most noticeable situation however, was when I got an email stating that they planned to let someone in Germany implement the WTLS parser, which I at that time had already completed weeks before.

These were the social problems that interfered with my work the most and I decided rather early what to do about them. I simply waited as long as I could and then did as much as I could to close the holes in the functions in my parser left by unfinished specifications. Thereafter I documented that which was left unsolved and let it become someone else’s problem.

\(^5\) [11], page 113.
2. The specification of my master thesis

The specification of my master thesis was:

1) To program a parser as it was defined in a specification\(^6\) from Ericsson. It should be able to read through a X.509 certificate and recognize at least that which is demanded by the WAP recommendation\(^7\). The parser should thereafter be able to:

   a) Determine if the structure of the certificate is valid.
   b) Determine what recognizable components the certificate contains.
   c) Determine where in the certificate buffer the different main fields are located.

   Together with the parser I was to create a collection of functions that could use the information determined by the parser to retrieve and parse some of the fields in the certificate.
   I was also to verify that the parser would work in the environment that it was intended to work in.

2) Optimize the ASN.1 definition of a X.509 certificate so that an existing ASN.1 compiler automatically could generate code that gave the same functionality as the code I had programmed in part 1, no more, no less.
   Thereafter I was to compare the code I had created in part 1 with the code created by the compiler regarding:

   a) The execution speed in the system that the parser was intended to work in.
   b) Memory used during execution.
   c) How easy or difficult the code was to read and understand.
   d) The time it would take for a normal user to create the code.
   e) Redundancy in the code.

3) The parser is to be used by a verifier to verify that certificates are valid and authentic. If I had the time I was supposed to work on such a verifier.

\(^6\) [7], page 5.
\(^7\) [2], page 22.
3. The X.509 parser

A X.509 parser relies on the definition of a X.509 certificate and I used the definition found in [1] (with the extra restrictions and definitions from [2]). This Request For Comment (RFC) has both become incorporated in the standards and been replaced with [5] and [6], but so far as programming my parser is concerned there is no important difference between any of these versions. This -and because [1] is the most cited source and de facto standard for software programmers regarding X.509 certificates- is why I have referred to this document directly when programming my parser. However, [1] is a bit unclear on some things and I have therefore been forced to seek clarifications from other sources.

This chapter begins with a general summary of a part of what a programmer needs to know to program a X.509 parser. This includes information found in [8], but also definitions found above that document in the hierarchy of standards (i.e. definitions regarding a whole or bigger subpart of a X.500 directory). The summary is not intended to describe everything needed to program a X.509 parser, since this knowledge should be attained by reading and understanding [1] and [8]. The summary is instead there to give a general background to X.509 certificates and emphasize what is important for a reader to understand the rest of this chapter.

Next part of the chapter is made up of the clarifications I have researched to be able to program my X.509 parser and the last part is a discussion of the end result.

3.1. General summary

To be able to understand the clarifications I have researched a reader must at least have some basic knowledge about X.509 certificates, ASN.1 and some of the defined transport encodings.

3.1.1. X.509 certificates

In a public-key cryptography system a user who wishes to exchange data with some remote system must be confident that the public key he eventually obtains is owned by the correct remote system. This is assured with the use of the X.509 specification and the part it plays in the X.500 structure for public key certificates.
A is a user that in a safe way wants to obtain the public key owned by B. The entities marked as CAs are Certificate Authorities which both A and B trust and which all have a certificate signed by the CA above and all CAs directly below them. A has a certificate which is certified by CA$_C$, so A knows CA$_C$’s public key. CA$_B$ has a certificate signed by CA$_C$, so A can verify the public key of CA$_B$. CA$_C$ has a certificate signed by CA$_B$ and B’s certificate is signed by CA$_B$. The certificates A, CA$_A$, CA$_B$, CA$_C$ and B together form a chain of certificates, a so-called certificate path. By walking up to CA$_B$ and then down to B, A can verify B’s certificate (and therefore B’s public key).

Both the mechanism for verifying a X.509 certificate and the basic structure of the X.509 certificate\(^9\) remains the same today as when defined in 1988. Using one of the supported algorithms in its defined way\(^10\) the data intended for the users of the certificate (or sometimes a hash of this data) is used to generate a signature value. This value and information about the algorithm used is then appended to the data to create the actual certificate. The algorithm used defines how the signature value is used when verifying the certificate.

The outer structure of the X.509 version 3 certificate\(^11\) is defined as:

\[
\text{Certificate ::= SEQUENCE \{ }
\text{tbsCertificate TBSCertificate,}
\text{signatureAlgorithm AlgorithmIdentifier,}
\text{signatureValue BIT STRING \}}
\]

\(tbsCertificate\) is the data intended for the users of the certificate, \(signatureAlgorithm\) identifies the cryptographic algorithm used by the issuing CA to sign the certificate and \(signatureValue\) is a digital signature computed upon \(tbsCertificate\) (or a hash of \(tbsCertificate\)).

In the example above A wants to acquire the public key of B. Usually A would then locate the right database and retrieve the certificate path to and public key of B, but let’s assume that A this time wants to be a 100% sure that the certificate eventually retrieved is B’s. How A does this then depends on both the certificates in the certificate path and which protocols are used, but the following example is definitely a real-life possibility.

![Figure 3. X.509 verification process.](image)

A has the public key of CA$_C$ and starts by acquiring the certificate of CA$_B$ that CA$_C$ has signed with it’s private key. A decrypts the \(signatureValue\) of this certificate using CA$_C$’s public key and the defined way for that particular cryptographic algorithm. If the result matches \(tbsCertificate\) A confirms that nothing else in \(tbsCertificate\) invalidates the certificate and -if not- is assured of having the correct public key of CA$_B$. Repeating this procedure A eventually reaches B and can be confident that the public key obtained then is the correct one.

However, a lot of things in the X.509 specification has changed and among those things is the X.509 certificate itself. The picture below (figure 4) details which fields can be found in a X.509 version 3 certificate.

---

\(^{9}\) [1], page 70.
\(^{10}\) [1], page 57.
\(^{11}\) [1], page 15.
Most of these fields have names that directly give away their content\(^ {12}\), but the optional \texttt{issuerUniqueID}, \texttt{subjectUniqueID} and \texttt{extensions} fields are perhaps not so selfexplanatory.

The \texttt{issuerUniqueID} and \texttt{subjectUniqueID} contains information to be used to separate certificates if subject and/or issuer names have been reused over time and if either of these fields are present the version must be 2 or 3. These two fields and the only optional basic field -the version field- are small and therefore do not make the construction of a parser for X.509 certificates much more difficult although they are optional fields\(^ {13}\).

The \texttt{extensions} field contains at least one extension and if this field is present the version must be 3. Specifications often define extensions that applications claiming to adhere to the specification have to recognize; furthermore, this field usually carries much of the data in X.509 version 3 certificates. Clearly, this means that the addition of this field in the X.509 specification makes the construction of a parser for X.509 certificates much more difficult.

\[14\] demand that applications recognize -among others- the key usage extension. The notation\(^ {15}\) for this extension is hard to understand for someone who lacks knowledge about it, but this is not really important in this example so please bear with me. The structure for this extension is defined as:

\[
\text{Extension} ::= \text{SEQUENCE} \{ \\
\text{extnID} \text{OBJECT IDENTIFIER}, \\
\text{critical} \text{BOOLEAN DEFAULT FALSE}, \\
\text{extnValue} \text{OCTET STRING} \}
\]

\[
\text{id-ce-keyUsage} \text{OBJECT IDENTIFIER ::= \{ id-ce 15 \}}
\]

\[
\text{KeyUsage} ::= \text{BIT STRING} \{ \\
\text{digitalSignature} (0), \\
\text{nonRepudiation} (1), \\
\text{keyEncipherment} (2), \\
\text{dataEncipherment} (3), \\
\text{keyAgreement} (4), \\
\text{keyCertSign} (5), \\
\text{cRLSign} (6), \\
\text{encipherOnly} (7), \\
\text{decipherOnly} (8) \}
\]

\(^{12}\) See above for an explanation of the basic structure.

\(^{13}\) Optional as in "not necessarily there", actually the specification defines the unique identifier fields as \texttt{OPTIONAL} and the version field as \texttt{DEFAULT}.

\(^{14}\) Page 25.

\(^{15}\) See 3.1.2. for an explanation of ASN.1, which is the notation used here.
This defines that the key usage extension is identified by the OID\textsuperscript{16} 2-5-29-15 (id-ce is the OID branch 2-5-29), could either be critical or non-critical and should contain a sequence of bits that have their meaning defined by [1] (even if the names of the “KeyUsage” bits -such as “keyEncipherment”- give rather strong clues to the information they are meant to convey). Important to note is that:

1. An OID signals which extension is present and since the number of OID authorities are not limited the number of different extensions that a parser can come across are not limited either. The extension field has the potential to become very large.
2. The X.509 specification defines that applications handling X.509 certificates must reject a certificate if they encounter a critical extension they do not recognize within it\textsuperscript{17}, so even a small error in the way a parser handles extensions could mean it will reject many valid certificates.
3. This extension is one of the least complicated ones, but despite this the meaning that its components carries cannot be deduced by simple looking at the notation describing its structure. It is clearly impossible to construct any automatic handling of unknown extensions or general handling of the known ones.

This should suffice to make it clear that the extension field makes a X.509 certificate much more difficult to parse.

3.1.2. Abstract Syntax Notation One (ASN.1)

To make software development manageable there is the need for programmers to be able to specify parts of systems without concern for how they are actually to be implemented or represented. This makes it possible to state axioms, which can be “proven” only after the part is actually implemented and used in other higher-level parts simply by assumption. One way of making this simplification of software development possible, which for instance Open Systems Interconnection\textsuperscript{18} (OSI) uses, is to use abstraction.

OSI’s method of defining abstract objects is called Abstract Syntax Notation One (ASN.1) and is simply a notation for describing abstract types and values. This notation is the one used to define the X.509 certificate\textsuperscript{19} and the definition itself the “axiom” I could use in the development of my parser. With it I knew what kind of information to expect in the data representing the certificate.

A description of the notation could (and does) fill books. For us the important thing to note is that ASN.1 defines four groups of types\textsuperscript{20} and that the types in all these groups (except some in the “other types” group) each have a distinct tag\textsuperscript{21} that identify that type.

- Simple types are the ones that have no components; these are the “building blocks” of the structured types.

    Examples of simple types are INTEGER (an arbitrary integer) and IA5String (an arbitrary string of IA5 characters).

\textsuperscript{16} The OBJECT IDENTIFIER (OID) type denotes an object identifier, a sequence of integer components that identifies an object such as an algorithm, an attribute type, or practically anything relevant. These OID’s are all organized together in a single tree, but different authorities supervise the different branches.
\textsuperscript{17} [1], page 24.
\textsuperscript{18} An internationally standardized architecture that governs the interconnection of computers from the physical layer up to the user application layer.
\textsuperscript{19} [1], page 70.
\textsuperscript{20} Simple types, structured types, tagged types and other types.
\textsuperscript{21} [3], page 209.
• Structured types have components.

Examples of structured types are SEQUENCE (an ordered collection of one or more types) and SET (an unordered collection of one or more types).

• Tagged types are derived from other types by either implicit or explicit tagging.

Implicitly tagged types are derived from other types by changing the tag of the underlying type.

Explicitly tagged types are derived from other types by adding an outer tag to the underlying type. In effect, explicitly tagged types are structured types consisting of only the underlying type.

• Other types are the few types that cannot be constructed in one of the above three ways.

One type from the “other type” group is the ANY type, which denotes an arbitrary value of an arbitrary type.

With the types from these four groups an ASN.1 programmer should be able to describe any data structure abstractly.

3.1.3. Basic Encoding Rules (BER)

While the ASN.1 definition of a X.509 certificate tells us what kind of information to expect, to program a X.509 certificate parser you still need to know how this information is going to be represented when received. The OSI standard \(^{22}\) for this is the Basic Encoding Rules for ASN.1 (BER).

BER describes three ways of representing data. The sender has some choice about how to encode the data, but the choice is limited by the ASN.1 value to be represented and whether the length is known or not. The three ways of encoding data are primitive encoding \(^{23}\), constructed definite-length encoding and constructed indefinite-length encoding.

Regardless of which way of encoding is used a value encoded under BER contains three or four parts.

1. Identifier Octet. The first octet (i.e. a data octet of 8 bits) identify the class of the ASN.1 value, any tag number if present and whether the method used is primitive or constructed.
2. Length Octet. After the identifier octet there is one or several octets \(^{24}\) that either identify how many of the following octets sent represent the present ASN.1 value (definite length) or identify that this is unknown (indefinite length).
3. Content Octet. These octets are either the representation of a value meant to be relayed (primitive) or BER encodings of the components of some ASN.1 value (constructed).

\(^{22}\)[12]

\(^{23}\) Primitive encoding can only be definite-length.

\(^{24}\) “Short form” is when the octet directly after the identifier octet defines the length of the value; “long form” is when that octet instead defines how many octets following it define the length of the value.
4. End-of-content Octet. If the constructed indefinite-length way of encoding data is used the content octets are followed by two octets (00 00). If one of the other ways of encoding is used these two octets are absent.

An ASN.1 type is the INTEGER type, which is an arbitrary integer. Under BER an arbitrary value of this type can -by definition- only be encoded using primitive encoding.

One encoding of the integer value 5 is 02 01 05.

Another ASN.1 type is the T61String, which is an arbitrary string of T.61 characters. This type could be encoded in any of the three ways of encoding, this since BER defines a constructed encoding of a T61String as an encoding where the content octets give the concatenation of the BER encodings of consecutive substrings.

The encoding of the T61String “eeeeee” could for example be any of the following:

```
14 06 65 65 65 65 65 65     Primitive encoding
14 81 06 65 65 65 65 65     Long form of length octets
34 0A 14 03 65 65 65 14 03 65 65 65 14 03 65 65 65 00 00  Constructed indefinite length encoding*
```

* note that all the inner values (the substrings) in this example have definite length encoding.

This structure of BER combined with the fact that the sender under BER does not send unnecessary data makes it possible to construct ASN.1 structures where it could be impossible for the receiver to determine exactly what values the sender actually sent. This problem is solved by tagging.

As stated earlier there are two ways to tag an ASN.1 structure, either you use implicit tagging or explicit tagging. Under BER implicit tagging is done by changing the identifier octet of the value to be tagged. Explicit tagging under BER is done by adding an extra outer identifier and length pair to the value.

A SEQUENCE that contains two to four INTEGER values could in ASN.1 be written:

```
integerSequence ::= Sequence {
  integer1 INTEGER,
  [0] integer2 INTEGER OPTIONAL,
  integer3 INTEGER
  [0] integer4 INTEGER OPTIONAL }
```

Integer2 and integer4 are optional values and under BER optional values are not included in the content octets. This means that a receiver that is handed three integer values inside an integerSequence cannot determine which integer values that has been handed over. The components of this SEQUENCE have to be tagged.

```
integerSequence ::= Sequence {
  integer1 INTEGER,
  [0] integer2 INTEGER OPTIONAL,
  integer3 INTEGER
  [0] integer4 INTEGER OPTIONAL }
```

Note that not all components need to be tagged for the SEQUENCE to become unambiguous. The difference between the octets sent in each case could be (using one of the possible encodings and writing out only the part that differs):

```
. . . 02 01 05 . . .    No tagging
. . . 80 01 05 . . .    Implicit tagging
. . . A0 03 02 01 05 . . .    Explicit tagging
```
3.1.4. Distinguished Encoding Rules (DER)
The Distinguished Encoding Rules for ASN.1 (DER) is a subset of BER. Under DER any ASN.1 value has exactly one possible encoding, something which is needed when unique encoding is needed. To define this subset DER imposes four extra, general rules:

1. When the length is between 0 and 127, the short form of length must be used
2. When the length is 128 or greater, the long form of length must be used, and the length must be encoded in the minimum number of octets.
3. For simple string types and implicitly tagged types derived from simple string types, the primitive, definite-length method must be employed.
4. For structured types, implicitly tagged types derived from structured types, and explicitly tagged types derived from anything, the constructed, definite-length method must be employed.

Beside these general rules there are also some specific rules for certain ASN.1 types (two examples of these kind of types are BIT STRING and SEQUENCE).

3.2. Clarifications
The previous parts of this chapter may give the reader the impression that most of the work done in preparation of part 1 was reading through documentation and standards. As a matter of fact I spent more time on discussion boards and sorting through email lists than reading specifications even before I started programming.

This was because even though the standards are the most fundamental knowledge in this field of expertise there are people working in a lot of different companies implementing these standards. This means that how these people in general interpret the standards is how the standards have to be interpreted by me. If those who make standards are unclear on something and the effect is that a majority of the tools to create X.509 certificates create what the standards would define as an erroneous certificate, then a X.509 parser should probably just as well ignore the standards regarding this “something”.

The following part is as previously stated made up of the clarifications I have researched, but it can just as easily be read as a description of how the programming work on part 1 went.

3.2.1. X.509 vs. WAP certificates
Up until now I have referred to my parser as a X.509 parser although the original parser specification speaks about a parser that works in a WAP environment. This is because this was one of the first things I had to clarify, that is: how big is the actual difference between the different certificates defined in [1] and [2]?

Well, I found that for me the difference between the WAP recommendation and the X.509 specification was next to nothing. There can basically be three big differences between a correctly implemented X.509 parser (note that I do not write verifier here) and a correctly implemented WAP parser.

25 [2], page 22.
1. A X.509 parser could recognize more attributes than specified in [2] for the subject or issuer field. My parser clearly does this since it does not only recognize all attributes defined as a must in both [1] and [2], but also the EmailAddress attribute\(^{26}\) and all attributes on the 2.5.4. branch (the default name attribute branch) on the OID tree.

2. A X.509 parser could recognize more extensions than specified in [2]. My parser also does this. Not only does my parser recognize those extensions demanded from [1] and the Domain Information extension specified in [2], but also the Issuer Alternative Name, Policy Mappings, Policy Constraints, Name Constraints and CRL Distribution Points extensions (the authorityAccessInfo extension is actually the Authority Information Access extension).

3. A X.509 parser could recognize other, not all or differently implemented signature algorithms and/or public key types than a WAP parser. My X.509 parser does not only recognize those defined in the WAP recommendation\(^{27}\), but also all those defined in the X.509 specification\(^{28}\).

Because of this I opted to call my parser a X.509 parser instead of a WAP parser.

### 3.2.2. BER vs. DER encoding

The next step was to study the general structure of a X.509 certificate. An issue that arose quickly was if the standards demanded that the certificate should be encoded in the DER subset of BER or not? In [1] I found sentences such as “For signature calculation, the certificate is encoded using the ASN.1 distinguished encoding rules (DER) [X.208].”\(^{29}\) while at the same time the examples at the end of [1] included certificates using BER\(^{30}\). Other sources only made me more aware of the different interpretations in this area\(^{31}\).

At last I found the “solution” as a discussion on an email list. The discussion on this list is included in this document as Appendix A and the solution I refer to is the one where you demand that tbsCertificate is in DER while the rest can be encoded under other encoding rules (which in this case was specified by Ericsson as only BER even though their written specification\(^{32}\) states otherwise).

### 3.2.3. The handle

Even though the actual data transmitted by Ericsson does not make a parser aware that this is a WAP environment\(^{33}\), there are still certain limits set by the WAP system itself. There is precisely little computing power and memory space, something that less important processes such as my parser has to adjust to. I discussed this with my supervisor and we found that:

\(^{26}\) [1], page 23.  
\(^{27}\) [2], page 16 and 17.  
\(^{28}\) [1], page 57.  
\(^{29}\) [1], page 15.  
\(^{30}\) [1], page 123.  
\(^{31}\) [4], certificate chapter.  
\(^{32}\) [7], page 3.  
\(^{33}\) As noted in 3.2.1..
1. Not every certificate sent to the parser will contain much information important to the verifier. As a matter of fact most of them will probably only contain 2 or 3 important fields.
2. Most fields are very simple to parse once you find them, but the task to find them could be very time consuming if you start at the beginning of the certificate.
3. There would be a need for the verifier to quickly find out if a certificate contains any critical extensions, especially if these extensions are unknown.

To make the parser conform as much as possible to the limits set by the WAP environment we introduced the handle[^34], i.e. a linked list mostly made up of pointers to the different main fields in the certificate. The first part of the handle is made up of those fields that are always present in a X.509 certificate (so that the verifier can jump directly to any of these fields) and the second part contains list objects with an identifier for a optional field, a pointer to that field and an indicator for critical/non-critical status. The second part is ordered by a recursive function so that all unknown extensions are put before the known.

This set-up makes it possible for my parser to avoid parsing unimportant fields, save memory space by only writing the offset of the most important fields to memory and quickly give the verifier information about the extensions present.

### 3.2.4. The ASN.1 definition of DSA

During the work on my parser I came across several small deviations from the standard in the majority of the certificates, but the most significant was the “DSA deviation”. In the X.509 specification it is stated that the DSA parameter fields should be omitted if there are no parameters present[^35] (i.e. there should not be any empty fields to signal this in either the signatureAlgorithm, the subjectPublicKeyInfo or the signature field). However, most DSA fields without parameters are created in the same way as RSA fields without parameters should be implemented, that is with a parameters component of the ASN.1 type NULL. I found the reason for this in a non-standard source[^36]:

> "Another pitfall to be aware of is that algorithms which have no parameters have this specified as a NULL value rather than omitting the parameters field entirely. The reason for this is that when the 1988 syntax for AlgorithmIdentifier was translated into the 1997 syntax, the OPTIONAL associated with the AlgorithmIdentifier parameters got lost. Later it was recovered via a defect report, but by then everyone thought that algorithm parameters were mandatory. Because of this the algorithm parameters should be specified as NULL, regardless of what you read elsewhere."

Since this fact is not mentioned in important sources such as [1], I opted for a solution in my parser where both the deviation and the standard were allowed.

### 3.2.5. Ericsson’s error handling

When I first implemented my parser I used C++ where I could benefit from the error handling built into that language. This was a mistake since Ericsson’s core phone system only accepts basic C (not C++), but a natural one to make if you consider the circumstances[^37].

[^34]: This was later incorporated in the specification, [7], page 5.
[^35]: [1], page 60 and 63.
[^36]: [4], signature chapter.
[^37]: See 1.2.1.
The code changes were a minor problem compared to the fact that I had to start over with the error handling in my parser. This time I decided to study Ericsson’s directives for error handling thoroughly before programming anything. This was a bit problematic however, since I found out that Ericsson actually did not have any directives for this. At first I thought I had missed something, but as I eventually talked to an employee of Ericsson involved in this area of programming I was told once and for all that:

1. In case of an error a process in an Ericsson system should try to recover.
2. If the process cannot recover it should instead terminate and while doing so free all memory it has allocated.

This was no problem to implement; to sprinkle a lot of macros across the code is a usual way to handle this in C and that was also the main suggestion I got. This is also one of the first things I learnt that you should try to avoid when programming in C (disregarding the fact that people do it all the time), since it could very easily make the code instable and hard to check for errors - not mentioning that you do not handle the error in any way. Lacking other choices I finally put it all down to the fact that Ericsson has to cope with too many programmers to create some coherent error handling procedure that someone in my situation could use. The error handling was therefore at last implemented with macros despite the disadvantages.

3.3. The end result
The end result was a good, working X.509 parser, with only four major problems that could be identified at the end of part 1:

1. The parser can only parse almost as detailed in the specification\(^{38}\).
2. The parser could be hard to understand for someone that wants to update or change the code. This is mostly because of the macros discussed above.
3. The last part of part 1 in the specification for the master thesis, to verify that the parser can work in the environment that it was intended to work in, could not be realized\(^{39}\).
4. Although most functions have been tested fully, there remain some that have not been tested at all or tested to some lesser extent than Ericsson would want. This is because I have not been able to find good enough test certificates or certificate creating tools that implement all fields in a X.509 certificate (especially examples of extensions are hard to find). This mean that if I somehow have misinterpreted the documentation concerning these fields, it will go unnoticed until someone actually tries to parse a certificate with such a field. However, the testing status of all fields has been reported.

I consider this part of the master thesis a success, since Ericsson was satisfied -even impressed- by the result and none of the problems above could be considered among the worst problems a program could have or in any way hinder the usage of the parser.

---

\(^{38}\) See 1.2.2.1..
\(^{39}\) See 1.2.1..
4. The comparison
The two parsers that I compare in this chapter are MP, the parser I have programmed in 1, and OAP, a parser whose basic structure was created with the “OSS ASN compiler for C”. Some parts of MP have changed during and after the comparison, but nothing that has any impact on the results in this chapter has been affected.

4.1. Something about OAP
OAP was created from one of my purposes optimized version\textsuperscript{40} of the ASN.1 module\textsuperscript{41} from the WAP recommendation. Before I feed this optimized code into the OSS ASN compiler I added those modules that are imported (and all the modules they refer to - which eventually meant something like 10,000 lines of code - since I did not want to accidentally remove some important piece of code). This do not mean that OAP contains a lot of unnecessary code, the documentation clearly states that only directly referenced code from imported modules are incorporated in the final result\textsuperscript{42}.

The final result is as stated above only a basic structure. Initializing of the process environment and functions that give the parsed fields to the user still has to be implemented, which meant that to look for redundancy or how difficult the code was to understand became meaningless.

Unfortunately I did not have the time to give OAP the same functionality as MP, which means that a comparison could easily have become erroneous or ambiguous. To avoid this, compensate for it and avoid giving my parser any undeserved credit I tried to give OAP the upper hand in all the tests below. How well I succeeded with this is detailed under the “changes” parts.

4.2. Static memory
By static memory is meant memory that the compiled source code for the parser occupies outside runtime.

4.2.1. Changes
To make OAP work I had to add an initializing function, an exit function and a function that could retrieve a certain certificate field to a user. For the retrieving function I chose to implement the function to retrieve the serial number of a certificate. This because this field is so simple that it can only be parsed in one unique way, which should minimize unknown factors in the comparison. In MP I instead removed all field parsing functions and their definitions except those needed to retrieve the serial number. Since code for such things as wrapping and session handling was not added to OAP it should be that parser that had the upper hand in this test.

\textsuperscript{40} Appendix B.
\textsuperscript{41} [2], page 19.
\textsuperscript{42} [3], page 112.
4.2.2. The test
I generated and studied MAP-files\textsuperscript{43} for MP and OAP. From these I removed the parts that concerned code from the surrounding system (in this case, Windows) since that code would eventually be provided by the telephone system itself. From the result I thereafter removed the functions for input during testing and those definitions for output that will not be used in the final system.

4.2.3. Observations\textsuperscript{44}
MP was much smaller than OAP even after I removed a big part with information from DLLs and such. The most interesting observation was that OAP was bigger then MP only because of one thing - the definitions of names and data structures that the OSS ASN.1 compiler had put in OAP. I also noticed that OAP probably would have smaller functions than MP even after session handling and such have been incorporated.

4.2.4. Conclusions
My conclusions was that MP is more effective regarding static memory and that how you define the data structures that are to be parsed (i.e. names and so on) can make great difference in the static memory size of OAP. However, to ascertain how big this difference can become there is a need for tests were the data structures in the ASN.1 modules are redefined.

4.3. Allocation of memory during runtime
Allocation of memory during runtime is exactly what it sounds like, i.e. how much memory a parser allocate during its execution.

4.3.1. Changes
The fairest comparison would be that if both parsers were made to run a test program with all the field functions from the specification included. To make that possible I would had to have finished the programming of OAP. This was with the time I had left something unrealistic. However, due to the fact that OAP’s allocation of memory during runtime depends much less on how many fields are parsed than MP’s allocation do, this problem could be overcome. I simply let MP parse all fields while OAP only had to parse the serial number field. With this test case I could even make a guess on how the verifier should behave to make the most effective use of the parsers.

4.3.2. The test
Since I did not have access to more sophisticated testing tools I used Microsoft Spy++, Process Viewer and PEBrowse Professional Interactive. The reason I used PEBrowse instead of Dependency Walker was that DW does not support surveillance of the linking of DLLs during runtime. PEBrowse can also step through the execution of the parsers while at the same time showing the memory allocation at the lowest level.

\textsuperscript{43} A MAP-file is a generated text file that contains a description of the contents of some other file.
\textsuperscript{44} Appendix C, table 1.
4.3.3. Observations
Regarding MP the most interesting observation is that the handle does not occupy any noticeable memory space. OAP on the other hand, occupy a lot of memory space both when it initializes the global variables and when the certificate is decoded. Unfortunately I could not ascertain the memory allocation inside the functions during runtime, but the working set in PV hints at that this is not much. OAP allocates double the amount of memory during runtime compared to MP even when MP parses the whole certificate in a single run, something that I considered a big difference.

4.3.4. Conclusions
My first conclusion was that how the verifier chooses to utilize MP could really make a difference here. If MP is made to directly allocate all the memory it needs during runtime the memory allocated is quite a lot more than if MP is made to allocate and then release memory one field function a time (this is the difference between start and main\(^{46}\)), since the memory allocated by the handle is not much. The large memory allocation by OAP is probably due to its kind of general parsing of the whole certificate directly. There is not much anyone can do about this, since that is the principle that the OSS ASN.1 compiler bases its code on.

4.4. Utilization of the CPU
I tested how fast the parsers were by testing how much they utilized the CPU.

4.4.1. Changes
Here I used the same set-up as when I tested the static memory. The fairest comparison would again be to let both parsers run through a whole certificate, since MP this time benefits from the fact that OAP retrieves so much information already from the beginning. This was not possible, so I settled for the best possible compromise, that to parse only a single field and only make conclusions on this general one field parsing case.

4.4.2. The test
In this test I used the functions time() and clock() to measure the CPU utilization and since both parsers where burdened by the same functions this extra “weight” should not interfere in the comparison. I was forced to use these functions since the programs I had access to, MS++ and PV, measures the CPU-time in 0.15 ms steps, which is a too big step size for these short programs. The best would be to count CPU-cycles directly through hardware measurements, which clock() does not but professional benchmarking programs do.

\(^{45}\) Appendix C, table 2.
\(^{46}\) Appendix C, table 2.
4.4.3. Observations

`clock()` also has its limits, which is noticeable in the results from MP. It is not the step size that is the problem, but the CPU-cycles are simply too few for `clock()` to register. On the other hand it is possible to see that MP goes from scratch to retrieving a single field much faster than OAP.

4.4.4. Conclusions

Exactly as in the earlier test the more specialized coding of MP gives better results than the general coding of OAP. Here it is much harder to conclude anything, but it is clear that MP parses a single field much faster than OAP. However, after the first pass OAP does not need to parse more of the certificate and can also give a verifier all those fields present in the specifications (at least as long as the ASN.1 modules are correct and the creators of the certificate adhere to the standard). The use of the version of MP used in this test demands that the verifier present can continue parsing certain fields where MP ends its parsing\(^{48}\). If you take all this into account a reasonable conclusion is that certificates inside a certificate path are probably parsed faster by MP, while certificates that the verifier could need many fields from -such as those at both ends of the path- could be parsed faster by OAP.

4.5. General conclusions

That I have programmed MP from scratch is clearly seen in all the tests I have done. Already in the beginning of the development of MP I focused on programming so that it allocated a small memory space and could retrieve the general fields in a certificate fast. So MP was created with a successful focus on those things I tested, while OAP was created from a general compiling tool.

So is MP better then OAP for someone in Ericsson’s situation? It is only possible to answer that question if you take into account the time it has taken to program MP. There have of course for me been some problems in the development of MP that a regular employee of Ericsson would not have encountered. Taking this in regard, however, I still estimate a development time of about 6 weeks for this parser. OAP also took some time to develop and would demand even more time if all functions were to be realized. If someone who was familiar with the OSS ASN.1 compiler did it I would still estimate that that person would need at least 2 weeks for the modification of the ASN.1 modules, the wrapper functions, the initialiser functions, and so on. To ensure a good result that person would probably need even more time. The difference would be about one man month. That cost have been kept down since I worked on this parser for my master thesis, but at the same time there were other problems\(^{49}\).

Taking all this into account, together with the fact that the assumption that the parsing will be mostly of a few fields in each certificate is not supported by any empiric evidence, is to the disadvantage of MP. At the same time the credibility of MP benefits from the results in all the tests above. My final conclusion is that MP is the best parser in this case, but that a problem with simpler ASN.1 code, less time to complete or no non-standard content could very well be solved easier and at a lesser cost by using the OSS ASN.1 compiler.

\(^{47}\) Appendix C, table 3.
\(^{48}\) See 1.2.2.2.
\(^{49}\) See 1.2.
5. The third part

The last part of my master thesis was originally that I was to work on a verifier if I had the time to do so. During the last weeks of my master thesis this was changed so that I instead was supposed to implement as much of a WTLS-parser that I had the time to do. A WTLS certificate is nothing more than a much smaller, less complicated and easier to parse version of a X.509 certificate, which meant that I rather quickly completed such a parser as described in the specification\textsuperscript{50} I was given from Ericsson.

There is not much to say about this, except that the specification contains errors. It is erroneously stated that the Signature field has a fixed size if rsa\_sha or ecdsa\_sha is used\textsuperscript{51}. The WAP, WTLS specification\textsuperscript{52} actually states that the signature field begins with two bytes that indicate the size of the signed message digest if either rsa\_sha or ecdsa\_sha is used, so the part about the signature field in [9] should be replaced with:

**Signature**: Depending on which signature algorithm is used we have for
- anonymous: No bytes in this field.
- rsa\_sha: The next two 2 bytes specifies the length n (1\(\leq n\leq 2^{16} - 1\)) in bytes of the signed message digest and this is followed by the n-byte signed message digest.
- ecdsa\_sha: The next two 2 bytes specifies the length n (1\(\leq n\leq 2^{16} - 1\)) in bytes of the signed message digest and this is followed by the n-byte signed message digest.

After I was done I compared this parser with my X.509 parser and compiled a list of differences between them since they will work together in the finished verifier structure. This list is incorporated in this essay as Appendix D.
6. Summary and conclusions

My master thesis contained three parts. I was to program a X.509 parser, compare it to another parser that had a basic structure created by a compiler and then I was to program a verifier to accompany the parser.

The parser was constructed successfully. Among other things I solved problems regarding:

- Unclear documentation.
- Differences between the majority of the implementations of some standards and the standards themselves.
- Demands from the environment meant to contain the parser.
- Unclear policies on error handling at Ericsson.

I guess the most daunting problem was that to find the differences between the standards and the real-life implementations of these standards. This was more often than not a matter of finding some error the majority of the certificates had and thereafter tracking it down to some obscure error made by some large company in the beginning of their certificate creation era. The origin of changes in the parser implementation could often be traced to some non-standard phenomena and since there is not many texts about this kind of problems this is maybe one of the most important incitements to get in contact with the certificate creating community on for example Internet discussion boards. The chance of overlooking one of these non-standard phenomena should however be small, since they probably have to be important and big enough to be noted easily to not have been changed according to the standard yet (it should be possible for each company to alone change small, unimportant non-standard content in certificates without too much effort).

With this in mind I feel confident in that the end result did only have four major problems, which was that it:

- Differed from Ericsson’s specification regarding some small details.
- Could be hard to update.
- Was not tested in its real environment.
- Was not fully tested.

None of these problems could be considered extreme or in any way hindered the usage of the parser.

The comparison was also completed successfully. I tested static memory, allocation of memory during runtime and utilization of the CPU for both my parser (MP) and the parser that had a basic structure constructed by a compiler (OAP). In all of these tests I had to change the parsers involved to make the comparison fair to OAP, something that in reality meant that OAP had the upper hand in all the tests. Despite this MP had better results in all the tests made. My conclusion became that MP was the better parser, but since the completion time was so much shorter for OAP there could be situations were it would be a better idea to develop a parser like OAP. An example of a situation when this would be wiser is when the ASN.1 code is simple, not much time is at hand for the project and no non-standard content is likely to appear.

The part of the master thesis that was about creating a verifier was changed into creating a WTLS parser, which is a simpler kind of X.509 parser. I completed such a parser and have included a correction of Ericsson’s specification for this in this essay. This
change was the sensible thing to do since the original plan to create a verifier could be work cut out for a master thesis all in itself. Bringing together the different kind of certificates would mean more study of specifications and of course the creation of more specifications by Ericsson. The programming work would perhaps not be more than that put into the parsers for the different certificates, but totally different since it would aim at working with information instead of gaining it from a certificate structure.

I foresee problems for instance:

- If those who constructed the different parsers have not thought about the fact that the different certificates are treated differently by the WAP system, for instance not all certificates have their length noted before them in the buffer of certificates delivered by such a system to a verifier.
- In creating functions for the exact handling of for instance the different public cryptographic algorithms or the comparison of certificate fields. This is not only affected by the specifications regarding different certificates, but also by what kind of importance profiles such as for instance PKIX are given.
- In the creation of an efficient verifier regarding the handling of the vast number of extensions available to certificate creators, especially if some extension have been overlooked and/or have to be added at this late stage of the programming.

I guess the result in the end will be either spectacular or a spectacularly horrible nightmare of error solving.
7. References


[3] Olivier Dubuisson; ASN.1 - Communication between Heterogeneous Systems; 5:th June 2002


[9] Per Ståhl; Parsing of WTLS Certificates; 3:rd June 2002


[11] Lars Svedberg; Gruppsykologi: Om grupper, organisationer och ledarskap; 1997

Appendix A

Answer 11
Ford, Warwick [WFord@verisign.com]
My memory is not what it used to be, but I thought we deliberately did not specify the encoding rules to be used for anything except the innermost ToBeHashed. The rationale is that the more outer ASN.1 is a true part of the surrounding protocol, and the encoding for that is established by presentation protocol mechanisms (or maybe an application protocol rule). For example, the presentation protocol (if it existed) might negotiate PER, in which case all the encoding of SIGNED (except for the innermost) would be PER.
/Warwick

Answer 10
Sharon Boeyen [sharon.boeyen@entrust.com]
OK, so that complicates things. Hoyt, I'd like to try to retrace the history. In the 88 text that ASN.1 comment about the inner sequence needing to be DER is not present. The other text about DER is and my reading is that the intent at that time at least was for the whole structure that was instantiated as a result of the macro to be DER. Somewhere along the line that asn.1 comment was added. I'd like to find out what led to that. Was it a DR and if so perhaps some background on the DR would help. If it was through an amendment we probably won't be able to track it as easily. I know you have all the historical texts. I only have stuff since I took over editorship. I'll check my files, but you may be able to find it more quickly :-)
/Sharon

Answer 9
Housley, Russ [rhousley@rsasecurity.com]
In Peter Gutmann's X.509 Style Guide, at least one implementation uses BER on the outermost SEQUENCE.
/Russ

Answer 8
Sharon Boeyen [sharon.boeyen@entrust.com]
Russ do you have a sense of how implementors do it today - is the PKIX text something they all fully understand and agree with or is this a detail that may not have been paid much attention? FYI, I did look into what Entrust CAs do - they DER encode both. I also looked into some of our client side systems (didn't do an exhaustive check) and expect, from what I did check, that they could handle either situation.

Given all the buffer overflow stuff with ASN.1 flying around lately, I'd feel more comfortable if the outer ASN.1 also had to be DER and indefinite length encodings were not allowed. When I went back to the 1988 text where macros were used, it looks like the intent at that time was for everything to be DER encoded, although I agree that technically things will work if only the tbs is DER encoded.
I suspect the "muddiness" was introduced when we moved away from macros with the "new" asn.1. There are other efficiencies that can be gained with DER encoding as well - for instance if the cert is going to be included in another signed structure, you already have the DER ready.

I would be concerned about any client side systems that may be expecting DER and don't receive it. That's the main reason I'd like to play it safe and have any clarifications to X.509 and PKIX go that way so that nobody breaks (assuming CA vendors encode in DER today).

/ Sharon

Answer 7
Housley, Russ [rhousley@rsasecurity.com]
Sharon:
There are four places in Son-of-2459 that deal with the issue at hand. Frankly, we did a better job in the CRL-related text than the certificate-related text.

Here are the text fragments:
4.1 Basic Certificate Fields
The X.509 v3 certificate basic syntax is as follows. For signature calculation, the certificate is encoded using the ASN.1 distinguished encoding rules (DER) [X.690]. ASN.1 DER encoding is a tag, length, value encoding system for each element.

4.1.1.3 signatureValue
The signatureValue field contains a digital signature computed upon the ASN.1 DER encoded tbsCertificate. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function. This signature value is then ASN.1 encoded as a BIT STRING and included in the signature field. The details of this process are specified for each of the supported algorithms listed in [PKIXALGS].

By generating this signature, a CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

5.1 CRL Fields
The X.509 v2 CRL syntax is as follows. For signature calculation, the data that is to be signed is ASN.1 DER encoded. ASN.1 DER encoding is a tag, length, value encoding system for each element.

5.1.1.3 signatureValue
The signatureValue field contains a digital signature computed upon the ASN.1 DER encoded tbsCertList. The ASN.1 DER encoded tbsCertList is used as the input to the signature function. This signature value is then ASN.1 encoded as a BIT STRING and included in the CRL’s signatureValue field. The details of this process are specified for each of the supported algorithms in [PKIXALGS].

I conclude that only the portions of the structure that are covered by the signature need to be DER encoded. I do not know if there are implementations out there that would fail to decode the certificate id the outermost SEQUENCE was indefinite length encoded.
I will try to change 4.1 to match 5.1. On the other hand, if the conclusion of this discussion is that the whole structure ought to be DER encoded, then I will try to change 5.1 to match 4.1. We are very late in the process, but one sentence that does not contain a MUST might be possible.

/Russ

Answer 6
Sharon Boeyen [sharon.boeyen@entrust.com]
Going back to the 1988 text where this was first defined, the SIGNED MACRO where the asn.1 comments for ENCRYPTED OCTET STRING say "where the octet string is the result of the hashing of the value of ToBeSigned". The comments don't mention DER at this point. But the text preceding the DER rules is the same as it is in the current 4th edition. That ASN.1, to me at least, seems to lean even further that the intent was DER for the whole thing. If anyone needs to see the 88 text I don't have softcopy any more but I suspect Hoyt does.

Hoyt, I agree that we should see if there is implementor consensus. At least DER is the 'safest' to use because anyone expecting DER or BER can handle what you supply, but if an implementor is expecting DER and you send BER that could cause problems. Tim/Russ, I've been kind of holding off waiting to see if you already know of any consensus based on PKIX discussions that would be valuable to this discussion?

/ Sharon

Answer 5
Hoyt L. Kesterson II [hoytkester@earthlink.net]
sharon, you have spotted the conflict. the ASN.1 says that ToBeSigned is encoded in DER for the hash. i see no clear statement that the elements of the COMPONENTS OF is encoded in DER. nor that the encompassing SEQUENCE is in DER (even the text you referenced doesn't discuss the outer sequence).

if the certificate itself is not being transferred as part of a larger signed structure, i don't see how differences in encoding the elements would be detected since the hash does not encompass them. if one implementation encoded a certificate with the surrounding SEQUENCE (or the SEQUENCE forming AlgorithmIdentifier) using an indefinite length encoding and another re-encoded (albeit not the practice i recommend) in the definite form, i suspect the difference would not be detected.

however, if there is some implementor consensus out there, we should consider capturing in the text.

/hoyt

Answer 4
Phil Griffin [phil.griffin@asn-1.com]
I see this differently. The data value is the parameter to the SIGNED {} parameterized type. This is all that need be DER encoded.

See http://www.itu.int/ITU-T/asn1/database/itu-t/x/x509/1997/index.html where this definition is listed in the index.
The input parameter, *ToBeSigned* is the data value that must be a distinguished encoding.

\[
\text{SIGNED}(\text{ToBeSigned}) ::= \text{SEQUENCE} \{ \\
\text{toBeSigned} \ \text{ToBeSigned}, \\
\text{COMPONENTS OF SIGNATURE}(\text{ToBeSigned}) \\
\} \\
\text{SIGNATURE}(\text{ToBeSigned}) ::= \text{SEQUENCE} \{ \\
\text{algorithmIdentifier} \ \text{AlgorithmIdentifier}, \\
\text{encrypted ENCRYPTED-HASH}(\text{ToBeSigned}) \\
\}
\]

and the result of signing this DER encoded value becomes the "encrypted" component in \text{SIGNATURE} {}.

/ Phil

Answer 3
Sharon Boeyen [sharon.boeyen@entrust.com]

Hoyt, this time its me being delinquent with email responses - sorry about that.
Actually, my interpretation of 509 is slightly different than yours, so perhaps there is a need to clarify one way or the other.

Taking the two sentences that precede the DER rules together:

In order to enable the validation of SIGNED and SIGNATURE types in a distributed environment, a distinguished encoding is required. A distinguished encoding of a SIGNED or SIGNATURE data value shall be obtained by applying the Basic Encoding Rules defined in ITU-T Rec. X.690 (1997) | ISO/IEC 8825:1998, with the following restrictions:

Wouldn't a SIGNED data value be a value that includes all three elements, ToBeSigned, AlgorithmIdentifier and ENCRYPTED-HASH? That's my interpretation, especially with the "shall" in the 2nd sentence.

Whether or not it should have been strictly required from a theoretical or functional standpoint is a different question, but my reading of these sentences is that it is required. My interpretation of PKIX clause 4.1 is that they also talk about the full certificate being encoded in DER (because they use the word "certificate" in the text but in the ASN.1 they use Certificate for the outer sequence and TBSCertificate for the one that contains the components that get signed). Tim/Russ is this a correct interpretation of PKIX?

I would be concerned about loosening this up in case some implementations verifying the signature assume both sequences are in DER.

/ Sharon

Answer 2
Hoyt L. Kesterson II [hoytkesterson@earthlink.net]

the x.509 standard still requires the use of the Distinguished Encoding Rules (DER). this is specified in clause 6.1 of the 4th edition (2000) where this is stated
ENCRYPTED-HASH \(\text{ToBeSigned}\) ::= BIT STRING (CONSTRAINED BY \{
-- shall be the result of applying a hashing procedure to the DER-encoded octets --
-- of a value of -- ToBeSigned -- and then applying an encipherment procedure to those octets -- \})

SIGNATURE \(\text{ToBeSigned}\) ::= SEQUENCE {
algorithmIdentifier AlgorithmIdentifier,
encrypted ENCRYPTED-HASH \(\text{ToBeSigned}\)\})

NOTE 5 - The encryption procedure requires the agreements listed in Note 2, and also agreement as to whether the hashed octets are encrypted directly, or only after further encoding them as a BIT STRING using the ASN.1 Basic Encoding Rules.

In the case where a signature is appended to a data type, the following ASN.1 may be used to define the data type resulting from applying a signature to the given data type.

\[\text{SIGNED} \{ \text{ToBeSigned} \} ::= \text{SEQUENCE} \{
toBeSigned \text{ToBeSigned},
COMPONENTS OF SIGNATURE \{ \text{ToBeSigned} \}\}\]  

In order to enable the validation of SIGNED and SIGNATURE types in a distributed environment, a distinguished encoding is required.

there has been some confusion as to what distinguished encoding rules should be followed. in our recent meeting in geneva we agreed that we would refer to the rules in the ASN.1 standard instead of the rules near the end of clause 6.1 in the 509 standard. we will create a technical corrigenda to make clear that the DER defined in ASN.1 should be used.

your mentioning of indefinite length encoding makes me think that someone may have been confused by the text specifying the extension field in the previous editions of x.509. for example, in clause 8 of the 3rd edition the following is stated

The extensions field allows addition of new fields to the structure without modification to the ASN.1 definition. An extension field consists of an extension identifier, a criticality flag, and a canonical encoding of a data value of an ASN.1 type associated with the identified extension.

the term "canonical encoding" was intended to be taken in the broad sense, i.e. DER is the canonical encoding to be used. however, some erroneously read it as a requirement to use the Canonical Encoding Rules (CER) of ASN.1. those rules mandate the use of indefinite length encoding. we removed the use of this term in the 4th edition in response to a defect report, DR214.

i also read your question as asking about the encoding of the data surrounding the signed data, i.e. the encoding of the encapsulating sequence, the type of the resultant hash (see note 5 in clause 6.1), and the algorithmIdentifier component. the rule here is less clear and i have never understood why we don't make this clearer. however, unless the certificate is itself in a signed message, there is no requirement in x.509 to have the that part of the encoding that is not to be computed into the hash field, e.g. the outer SEQUENCE of SIGNATURE, be in DER (sharon, correct me if i am wrong here).

my long held personal opinion is that a canonical encoding like DER should never have been mandated. i have never understood the rational of the process that encodes a structure, e.g. a certificate, in DER, computes a hash of the result of the DER encoding, and encrypts that hash to produce a signature, and then encodes the certificate differently, e.g. BER, for the transfer syntax. thus requiring that a recipient of a signed certificate must decode the transfer syntax and re-encode the resulting data into DER so as to recompute the hash.

33
one of the concerns voiced in meeting in the mid-90s was that when an intermediate directory system decodes and re-encodes a signed message, and if the re-encoder did not have access to the asn.1 specification of the data, then it did not know how to discern between SET and SET OF since both had the same primitive type value but required different encoding of the data value under DER. my response (other than "then don't do that") was that since the intermediate system knew it was a signed structure, and since the intermediated system was not able to resign the structure (not have the necessary key), the system should treat the structure as a block of octets that should not be changed. instead merely extract the information needed and forward the block of octets onward unchanged.

we tried to give a sense of this method in this text at the end of clause 6.1

Generating a distinguished encoding requires the abstract syntax of the data to be encoded to be fully understood. The Directory may be required to sign data or check the signature of data that contains unknown protocol extensions or unknown attribute syntaxes. The Directory shall follow these rules:

- It shall preserve the encoding of received information whose abstract syntax it does not fully know and which it expects to subsequently sign;
- When signing data for sending, it shall send data whose syntax it fully knows with a distinguished encoding and any other data with its preserved encoding, and shall sign the actual encoding it sends;
- When checking signatures in received data, it shall check the signature against the actual data received rather than its conversion of the received data to a distinguished encoding.

in essence one treats a received signed structure as a string of octets to validate the signature, e.g. recompute hash and compare with the hash recovered by decrypting the received encrypted hash.

/ hoyt

Answer 1
Phil Griffin [phil.griffin@asn-1.com]
Fredrik,
The absolute best place to pose this question is on Hoyt Kesterson's OSI list for a definitive answer from the Directory Rapporteur himself.

'500 list' <osidirectory@az05.bull.com>

/ Phil

-----Original Message-----
[Sent to: First to <asn1@oss.com> and later also to <osidirectory@az05.bull.com>]
I was wondering if someone knows if the DER-encoding of certificates in the X.509-standard is optional or not (old versions of the ASN.1 standards seem to demand this, but newer does not mention it).

Is there valid certificates with parts of them encoded in BER, or rather, is there valid certificates with inner parts (parts of tbsCertificate, signatureAlgorithm or SignatureValue) using the "indefinite-length method" of BER (the only demand being that they have to be re-encoded in DER when an application wants to check validity)? Or does that always make such certificates invalid?
/Fredrik
Appendix B

-- Modification of
WAPCertificateProfiles {joint-iso-itu-t(1) identified-organizations(23) wap(43) modules(0)
certificate-profiles(1)}

DEFINITIONS IMPLICIT TAGS ::==

BEGIN

-- EXPORTS All –

-- All types and values defined in this module are exported for use in other ASN.1
modules.

IMPORTS

informationFramework, selectedAttributeTypes, authenticationFramework,
certificateExtensions
FROM UsefulDefinitions {joint-iso-itu-t(2) ds(5) module(1) usefulDefinitions(0) 3}

ATTRIBUTE, OBJECT-CLASS, top
FROM InformationFramework informationFramework
countryName, organizationName, organizationalUnitName, stateOrProvinceName,
commonName, serialNumber
FROM SelectedAttributeTypes selectedAttributeTypes

-- Added Attributes

surname, givenName, initials, generationQualifier, localityName, title
FROM SelectedAttributeTypes selectedAttributeTypes

-- END Added Attributes

SIGNED, ALGORITHM, EXTENSION, Version, Validity
FROM AuthenticationFramework authenticationFramework

-- Has added code because of other added code
keyUsage, extKeyUsage, certificatePolicies, subjectAltName, basicConstraints,
nameConstraints, policyConstraints, authorityKeyIdentifier, subjectKeyIdentifier,
GeneralName,
FROM CertificateExtensions certificateExtensions

rsaEncryption, sha1WithRSAEncryption
FROM PKCS-1 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
modules(0) pkcs-1(1)}

id-ecPublicKey, ecdsa-with-SHA1, Parameters
FROM ANSI-X9-62 {iso(1) member-body(2) us(840) 10045 module(4) 1}
id-sha1

FROM OIWSECSIGAlgorithmObjectIdentifiers {iso(1) identified-organization(3)
oiw(14) secsig(3) oiwsecsigalgorithmobjectidentifiers(1)};

-- Upper bounds –

wap-ub-depth INTEGER ::= 5

wap-ub-width INTEGER ::= 1

wap-ub-extensions INTEGER ::= 255

wap-ub-attributes INTEGER ::= 255

wap-ub-publicKey INTEGER ::= 5000

wap-ub-digest INTEGER ::= 255

-- Object Identifiers –

wap OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) identified-organizations(23) 43}

wap-ce OBJECT IDENTIFIER ::= {wap 3} -- Certificate extensions branch

wap-ce-domainInformation OBJECT IDENTIFIER ::= {wap-ce 1}

-- Added OID's

id-pe OBJECT IDENTIFIER ::= {1 3 6 1 5 5 7 1}
id-pe-authorityInfoAccess OBJECT IDENTIFIER ::= {id-pe 1}

-- END Added OID’s

-- WAP certificate syntax –

WAPCertificateInfo ::= SEQUENCE {
  version [0] EXPLICIT Version DEFAULT v1,
  serialNumber CertificateSerialNumber,
  signature AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
  issuer Name {{SupportedNamingAttributes}},
  validity Validity,
  subject Name {{SupportedNamingAttributes}},
  subjectPublicKeyInfo SubjectPublicKeyInfo {{SupportedPublicKeyAlgorithms}},
  extensions [3] EXPLICIT Extensions {{SupportedExtensions}} OPTIONAL-- if present, -- version must be v3
}

WAPCertificate ::= SIGNED {WAPCertificateInfo}
CertificateSerialNumber ::= INTEGER -- Recommended to be less than 8 bytes

AlgorithmIdentifier {ALGORITHM:AlgorithmSet} ::= SEQUENCE {
  algorithm ALGORITHM.&id ({AlgorithmSet}),
  parameters ALGORITHM.&Type ({AlgorithmSet}@algorithm) OPTIONAL
}

SupportedPublicKeyAlgorithms ALGORITHM ::= {
  {NULL IDENTIFIED BY rsaEncryption} |
  {Parameters IDENTIFIED BY id-ecPublicKey},... -- For future extensions
}

SupportedSignatureAlgorithms ALGORITHM ::= {
  {NULL IDENTIFIED BY sha1WithRSAEncryption} |
  {NULL IDENTIFIED BY ecdsa-with-SHA1},... -- For future extensions
}

Name {ATTRIBUTE: IOSet} ::= CHOICE {
  rdnSequence SEQUENCE SIZE (1..wap-ub-depth) OF RelativeDistinguishedName
  {{IOSet}}
}

RelativeDistinguishedName {ATTRIBUTE : IOSet} ::= SET SIZE(1..wap-ub-width) OF
AttributeNameTypeAndValue {{IOSet}}

AttributeNameTypeAndValue {ATTRIBUTE : IOSet} ::= SEQUENCE {
  type ATTRIBUTE.&id ({IOSet}),
  value ATTRIBUTE.&Type ({IOSet}@type)
}

-- Has added code because of other added code
SupportedNamingAttributes ATTRIBUTE ::= {
  countryName |
  organizationName |
  organizationalUnitName |
  stateOrProvinceName |
  commonName |
  domainComponent |
  serialNumber |
  surname |
  givenName |
  initials |
  generationQualifier |
  localityName |
  title,
  ... -- For future extensions
}
domainComponent ATTRIBUTE ::= {
    WITH SYNTAX IA5String
    ID { 0 9 2342 19200300 100 1 25}
}

SubjectPublicKeyInfo {ALGORITHM: IOSet} ::= SEQUENCE {
    algorithm AlgorithmIdentifier {{IOSet}},
    subjectPublicKey BIT STRING (SIZE(1..wap-ub-publicKey))
}

Extensions {EXTENSION : IOSet} ::= SEQUENCE SIZE (1..wap-ub-extensions) OF Extension {{IOSet}}

Extension {EXTENSION : IOSet} ::= SEQUENCE {
    extnId EXTENSION.&id ({IOSet}),
    critical BOOLEAN DEFAULT FALSE,
    extnValue OCTET STRING
} (CONSTRAINED BY {-- extnValue must contain a DER-encoded value of type EXTENSION.&Extntype
    -- for the object identified by extnId --})

domainInformation EXTENSION ::= {
    SYNTAX DomainInformation
    IDENTIFIED BY wap-ce-domainInformation
}

DomainInformation ::= SEQUENCE {
    domainInfoFlags DomainInfoFlags DEFAULT {onLineStatusRequest},
    domainAuthorityIdentifier Name {{SupportedNamingAttributes}} OPTIONAL,
    otherExtensions [0] ExtensionReference OPTIONAL,
    ... -- For future extensions
} (CONSTRAINED BY {-- This extension should not be critical--})

DomainInfoFlags ::= BIT STRING {
    onLineStatusRequest (0)
}

authorityInfoAccess EXTENSION ::= {
    SYNTAX AuthorityInfoAccess
    IDENTIFIED BY id-pe-authorityInfoAccess
}

AuthorityInfoAccess ::= SEQUENCE {
    accessDescription AccessDescription
} (CONSTRAINED BY {-- This extension must not be critical--})

AccessDescription ::= SEQUENCE {
}
accessMethod OBJECT IDENTIFIER,
accessLocation GeneralName
}

-- END Added Extension

ExtensionReference ::= SEQUENCE {
    url IA5String, -- URL in accordance with pointing to a DER-encoded value of type Extensions
digest Digest
}

Digest ::= SEQUENCE {
    digestAlgorithm AlgorithmIdentifier {{DigestAlgorithms}} DEFAULT sha1,
digest OCTET STRING (SIZE(8..wap-ub-digest))
}

DigestAlgorithms ALGORITHM ::= {
    {NULL IDENTIFIED BY id-sha1},
    ... -- For future extensions
}

sha1 AlgorithmIdentifier {{DigestAlgorithms}} ::= {algorithm id-sha1, parameters
SHA1Parameters : NULL}

SHA1Parameters ::= NULL

-- Has added code because of other added code
SupportedExtensions EXTENSION ::= {
    domainInformation |
    keyUsage |
    extKeyUsage |
    certificatePolicies |
    subjectAltName |
    basicConstraints |
    nameConstraints |
    policyConstraints |
    authorityKeyIdentifier |
    subjectKeyIdentifier |
    authorityInfoAccess,
    ... -- For future extensions
}

END
-- Borttagen Kod

wap-oc OBJECT IDENTIFIER ::= {wap 4} -- Object class branch

wap-oc-wapEntity OBJECT IDENTIFIER ::= {wap-oc 1}

wapEntity OBJECT-CLASS ::= {
    SUBCLASS OF { top }
    KIND auxiliary
    MAY CONTAIN { WAPEntityAttributeSet }
    ID wap-oc-wapEntity
}

WAPEntityAttributeSet ATTRIBUTE ::= {
    serialNumber,
    ... -- For future extensions
}

wap-at OBJECT IDENTIFIER ::= {wap 2} -- Attributes branch

-- END Borttagen Kod
### Appendix C

#### Table 1

**MP**

Preferred load address is 00400000

<table>
<thead>
<tr>
<th>Address</th>
<th>Publics by Value</th>
<th>Rva+Base</th>
<th>Lib:Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001:00000000</td>
<td>_Cert_Basic_GetSession</td>
<td>00401000 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000110</td>
<td>_Cert_Basic_EndSession</td>
<td>00401110 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:000001b0</td>
<td>_Cert_Basic_GetLength</td>
<td>004011b0 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000240</td>
<td>_Cert_Basic_AddToMLOList</td>
<td>00401240 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000270</td>
<td>_Cert_Basic_AddToMLOOList</td>
<td>00401270 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:000002c0</td>
<td>_Cert_Basic_GetOIDValueJLM</td>
<td>004012c0 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000360</td>
<td>_Cert_Basic_GetOIDValue</td>
<td>00401360 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000580</td>
<td>_Cert_Basic_Skipper</td>
<td>00401580 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:000006b0</td>
<td>__Cert_OptionalParsingPart</td>
<td>004016b0 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000910</td>
<td>_Do_X509Parser_Init</td>
<td>00401910 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000a70</td>
<td>_Do_X509Parser_Final</td>
<td>00402070 f</td>
<td>MP.obj</td>
</tr>
<tr>
<td>0001:00000b80</td>
<td>_main</td>
<td>004024f0 f</td>
<td>MP.obj</td>
</tr>
</tbody>
</table>

Totally: 4736 bytes

**OAP**

Preferred load address is 00400000

<table>
<thead>
<tr>
<th>Address</th>
<th>Publics by Value</th>
<th>Rva+Base</th>
<th>Lib:Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001:00000000</td>
<td>__ossinit_Opt2@4</td>
<td>00401000 f</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0001:0000050</td>
<td>_Do_X509Parser_Init</td>
<td>00401050 f</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0001:00000a0</td>
<td>_Do_X509Parser_Final</td>
<td>004010a0 f</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0001:00000d0</td>
<td>_Do_X509Parser_GetSerialNumber</td>
<td>004010d0 f</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0001:00000f0</td>
<td>.... [Removed]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001:00004d0</td>
<td>_main</td>
<td>004014d0 f</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0003:0000670</td>
<td>.... [Removed]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003:000030</td>
<td>[Names, Def. of constructed d.types] ....</td>
<td>00410030</td>
<td>OAP.obj</td>
</tr>
<tr>
<td>0003:000030a0</td>
<td>.... [Removed]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Totally: 13056 bytes
V: Virtual Bytes (The virtual memory space used).
W: Working Set (The average memory space used, but actually the memory space reserved for this process as long as the memory usage in the rest of the computer allows it)
H.S: The Heap (The memory space used for dynamically allocated memory).
P.M: The private memory space. Here this coincides with H.S.

Table 2

MP

DLLs used
NTDLL.dll and KERNEL32.dll.

<table>
<thead>
<tr>
<th>Point in time</th>
<th>V</th>
<th>W</th>
<th>H.S/P.M</th>
</tr>
</thead>
<tbody>
<tr>
<td>After start</td>
<td>5636</td>
<td>432</td>
<td>44</td>
</tr>
<tr>
<td>After main</td>
<td>6732</td>
<td>628</td>
<td>108</td>
</tr>
<tr>
<td>After init</td>
<td>6732</td>
<td>628</td>
<td>108</td>
</tr>
</tbody>
</table>

OAP

DLLs used
NTDLL.dll, ossapi.dll, MSVCRT.dll, KERNEL32.dll, USER32.dll, GD132.dll, ADVAPI32.dll, RPCRT4.dll, ossmem.dll, soedapi.dll, soedber.dll, cstrain.dll and soedoid.dll.

<table>
<thead>
<tr>
<th>Point in time</th>
<th>V</th>
<th>W</th>
<th>H.S/P.M</th>
</tr>
</thead>
<tbody>
<tr>
<td>After start</td>
<td>12436</td>
<td>948</td>
<td>136</td>
</tr>
<tr>
<td>After main</td>
<td>13524</td>
<td>1084</td>
<td>200</td>
</tr>
<tr>
<td>After init</td>
<td>14808</td>
<td>1416</td>
<td>240</td>
</tr>
</tbody>
</table>

Table 3

MP

Measured times to finish a run
0.016000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.016000, 0.000000.

Average time to finish a run
0.0032

53 All values are in kilobytes.
54 The points of time are directly after the process starts (start), after the global variables/process environment have been initialised (main) and after the first run through the certificate has been completed.
55 All values are in seconds.
OAP

Measured times to finish a run
0.047000, 0.047000, 0.062000, 0.062000, 0.047000, 0.046000, 0.047000, 0.047000, 0.063000, 0.063000.

Average time to finish a run
0.0531
Appendix D

Differences between the parsers
This is a short summary of the differences between my X509 Parser, my WTLS Parser and the X509 parser specification\textsuperscript{56}. For each difference there is also a suggestion on how to change the implementations when the aim is to combine the X509 and WTLS Parser into some greater structure (i.e. a “general parser”).

All line specifications are according to the files parser.c (06/10/02, 10:23) and parser.h (06/10/02, 10:21).

Define
The implementations have their own defined constants.

The length fields
In both implementations I have used \texttt{uint16} variables to define lengths (the variables presentLength, maxLength and jumpLength are examples of this), which is a difference from the original wish to use \texttt{uint32} variables. The reason for this is the many length fields used/referenced and the fact that a \texttt{uint32} length is -to say the least- overkill here.

How to change: First change all length fields into \texttt{uint32} variables. Secondly change all typecasts (for example those on line 140 and 1053). Finally change the \texttt{Cert_Basic_GetLength} function so that long, definite length fields can be read properly also when they are 3 or 4 bytes long.

Datatypes with similar names
In both implementations we use datatypes that have the same name if we do not take the prefix (which is WTLS or X509) into account. These are the same and can be exchanged for a general datatype with the same structure in both implementations with the following exception:

\texttt{Parser_StringType_t} is a uint8 in the X509 Parser and a uint16 in the WTLS Parser.

How to change: Replace the \texttt{X509Parser_StringType_t} with the \texttt{WTLSParser_StringType_t}.

Different usage
We use the parameters/variables in the same way in both implementations. Therefore a “general parser” could use the functions/parameters in much the same way in relation to both implementations with the following exception:

\textsuperscript{56}[7]
VC_CertificateBuffer_t contains a field that is interpreted as the length of the certificate to be parsed by the X509 Parser. The WTLS Parser interprets the same field as the length of the data left to parse in a buffer containing several certificates handed to a "general parser". The difference here is due to definitions in the WAP, WTLS specification\[^{57}\]. They establish that the length of a WTLS Certificate does not precede it inside a buffer like the one mentioned above, while there must be a length explicitly stated before any X509 Certificates present. Because of this we have also set the WTLS Parser to return the length of the WTLS certificate parsed -by writing in the field mentioned above- before exiting the Init function.

**How to change:** This does not prompt any changes (it is instead needed to implement a "general parser", which can parse the structures defined on the pages mentioned above).

**Cert_Basic_GetOIDValue**
This function is not the same in both implementations.

**How to change:** The Cert_Basic_GetOIDValue in the X509 Parser is the original function and works with the WTLS Parser too. All functions beginning with Cert_ in the X509 Parser can replace the functions with the same name in the WTLS Parser.

**WTLSParser_GetSignatureField and X509Parser_GetSignatureField**
WTLSParser_GetSignatureField does not take the same number of parameters as the X509Parser_GetSignatureField. This is because the signature field in a WTLS certificate cannot contain any parameters.

**How to change:** This does not prompt any changes, but if consistency is an issue then there is no problem in adding -and then ignoring- the ParameterList_p and SizeOfBuffer parameters in WTLSParser_GetSignatureField.

**X509Parser_GetCRLDistributionPoints and X509Parser_GetPublicKeyInfo**
Both these functions differ from the wish of the original specification in that they in some cases do not parse all subfields present in the field they parse. This is because the original specification was incorrect for these fields. The X509Parser_CRLDistributionPoint_t could only contain information about one Generalname -although several could be present- and the EcpkParmeters were not defined.

**How to change:** Union and a lavish use of memory could of course fix this problem, but otherwise I have no suggestions at this time.

\[^{57}\][10], page 63.
WTLSParser_GetPublicKeyInfo and X509Parser_GetPublicKeyInfo

WTLSParser_GetPublicKeyInfo differs from X509Parser_GetPublicKeyInfo in that it does parse all subfields in the Subject Public-key algorithm field. This is because in some cases it needs to be able to do this to manage to eventually get past the field.

**How to change:** This does not prompt any changes, but a “general parser” should be aware of the difference in what it gets in return from WTLSParser_GetPublicKeyInfo and X509Parser_GetPublicKeyInfo.
På svenska
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmanens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmanens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmanens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Fredrik Asplund