ON THE PERFORMANCE OF CERTIFICATE REVOCATION PROTOCOLS BASED ON A JAVA CARD CERTIFICATE CLIENT IMPLEMENTATION

K. Papapanagiotou, K. Markantonakis, Q. Zhang, W.G. Sirett and K. Mayes
The Information Security Group Smart Card Centre (Founded by Vodafone, G&D and the ISG), Royal Holloway, University of London

Abstract: The use of certificates for secure transactions in smart cards requires the existence of a secure and efficient revocation protocol. There are a number of existing protocols for online certificate revocation and validation, among which OCSP and SCVP are the most widely used. However there are not any real applications testing the efficiency of these protocols when run in a smart card, even though the advantages of such an implementation are promising. In this paper we examine the details of the implementation of these protocols, emphasising on the issues arisen from the limitations of the smart cards. We also discuss the performance results from the implementation of OCSP and SCVP in a multi-application smart card environment. Results from two different Java Card platforms are presented and analyzed.

Key words: Certificate Revocation, OCSP, SCVP, Java Card

1. INTRODUCTION

In recent years, X.509 certificates have been used more and more in smart cards in order to validate users, establish secure channels or perform secure transactions. One of the most significant problems of Public Key Infrastructures (PKI) is certificate revocation and validation. Any PKI deployment, whether it includes smart cards or not, should provide efficient and secure mechanisms for certificate validation. Until now, protocols, such as the Online Certificate Status Protocol (OCSP) [15] and the Simple

Certificate Validation Protocol (SCVP) [12] have been designed and successfully implemented in many systems. Their weaknesses and general issues concerning their use have been discussed extensively in academic literature [2]. However, none of these protocols have been specifically designed or tested for smart cards.

In this paper we outline the design of a smart card certificate validation model and discuss the issues surrounding its implementation. We also implement OCSP and SCVP in two different Java Card platforms, which from now on will be referred to as Vendor A and Vendor B, using the Java Card API Ver. 2.1 [22]. Our goal is to create a transparent, lightweight, and independent application that will provide certificate validation function to any other application in a multi-application smart card environment. It will be completely trusted by all other applications on the card to perform certificate validation on their behalf. Thus, other applications should not need to be aware of the specifics of the underlying protocols. Our purpose is, firstly, to present some performance measurements concerning the implementation of the most widely used revocation protocols in a smart card. Secondly, we want to highlight the issues regarding the design and development of a validation protocol in such a limited processing and memory environment. Such protocols are an essential part of every PKI. It is important that we identify the existing limitations in their implementation, as input into future designs for optimised solutions.

We will briefly describe the certificate validation protocols and then analyze the issues surrounding certificate validation on a smart card. Subsequently, we will present the smart card certificate validation model and the entities involved. Furthermore, we provide the implementation details of the model along with providing performance results and timings. Finally, we provide some concluding remarks and discuss directions for further research.

2. THE CONCEPT OF ON-LINE CERTIFICATE VALIDATION

Certificate revocation and validation using Certificate Revocation Lists [9] may in some cases be inefficient or indeed inadequate. Protocols for online certificate validation have been proposed to solve the issues surrounding certificate revocation. These protocols can in theory be used in a smart card architecture to validate certificates for card based applications.

2.1 Certificate Validation Protocols

Many methods and protocols have been proposed for online certificate validation [2]. Currently, the most widely used are OCSP [15] and SCVP [12], which are simple client-server protocols involving requests and responses that provide the current status of one or more certificates. A client can send a request to a server (usually called "responder"), asking for the status of one or more certificates. The server responds with a signed message containing the status of the certificate(s), the time when the responder last updated the status information and the time of the creation of the message. Responses should be signed by the CA, a trusted or an authorized responder.

In OCSP version 1 [15] (OCSPv1), the certificate is referenced using its serial number along with the issuer's name and public key. Thus, it is necessary for the client sending the request to construct and validate the full certificate path from the queried certificate to the root CA. Certificate path construction can be difficult to implement in environments with limited processing power, such as smart cards. As a result, other solutions were proposed, like OCSP version 2 (OCSPv2) [16] and SCVP [12]. OCSPv2 practically added two more possibilities to reference the certificate: the client can send the entire certificate or a hash of some specific fields from it. Consequently, OCSP clients do not have to do any certificate path construction. Nevertheless, the size of the request message is significantly increased. Even though the draft for OCSPv2 has expired since 2002 [16], we believe that it may be applicable to smart cards.

SCVP [12] is a protocol that can provide further information than just revocation status. An SCVP Server can perform certificate path construction and validation as well as revocation checking. The request message includes either the entire certificate or a hash of the certificate and can be signed if needed. The server can be instructed to provide certification path for the certificate, revocation status or both. SCVP Responses are always signed, unless an error message is given back. Practically, they contain the same information with OCSP responses, in terms of time and date values. SCVP messages are encapsulated in Cryptographic Message Syntax (CMS) [6] data structures, which results in larger and more complex messages.

2.2 Issues surrounding the Smart card Certificate validation

Nowadays, there are numerous proposals for smart card applications exploiting public key cryptography, which implies the handling of certificates by the smart card. Therefore, an efficient mechanism for certificate validation is considered essential. However, the implementation

of such a mechanism has to address some significant issues, mainly caused by limitations of the smart card environment.

First of all, it is considered more secure and efficient for a smart card application to be able to determine the validity of a certificate without having to rely on an application that doesn't reside on the card. The tamper-resistant nature of the card makes resident application and data more trustworthy and secure. On the other hand, an off-card certificate validation application should be trusted by the card and would require a secure communication channel with the card.

Most difficulties regarding the implementation of certificate validation protocols on smart cards are the result of the limited processing power and memory of the card. The limited processing capability of a smart card makes the creation of a certification path on the card very time and memory consuming. Thus, the use of OCSPv1 is not recommended, as it requires a fully validated certification path. Apart from that, it is not very efficient to implement a fully functional ASN.1 [10] parser and DER [11] encoder in such a limited environment. X.509 Certificates, OCSP and SCVP protocols are all designed using ASN.1 notation. Thus, the creation and parsing of messages can be very time consuming. In addition OCSPv1 messages require extraction of fields from the queried certificates, which demands the presence of an ASN.1 parser or a package that handles X.509 certificates.

Smart cards have also a very limited memory. An X.509 certificate may occupy more than 1000 bytes [17] of memory. This is not such a problem for the card's capacity as it is for the communication channel between the card and the reader. An APDU data buffer [8] can hold up to 255 bytes of data, so a series of APDUs is needed for an X.509 certificate to be transmitted to the card. An OCSPv2 message also contains numerous other fields and a digital signature, which increase even more the total size of the messages. SCVP messages can contain only a hash of the certificate, so SCVP is expected to be more efficient than OCSP. Hashing the certificate is also possible in OCSPv2, however only a part of the certificate is hashed and thus, a package handling X.509 certificates would be needed. Nevertheless, a hash function requires more processing power and thus, is expected to need more time for execution. SCVP response messages also contain the certificate of the SCVP server, so they are expected to be as big as OCSP responses. Finally, both OCSP and SCVP protocols require some time checks to be done to determine the validity of the responses. Such checks cannot be done on-card as the card doesn't have a clock and, thus, knowledge of current time or date. Thus, in order to prevent replay attacks, nonces [12, 15] should be used.

The Open Mobile Alliance has already published a candidate version of an OCSP profile for mobile environments [18]. Its goal is to enable the use of OCSP in mobile devices with limited resources that use the Wireless

Application Protocol (WAP). This profile sets requirements and constraints on OCSP in order to have smaller, simpler and more easily processed messages. Nonetheless, it is not specifically designed for smart cards.

2.3 Motivation

As the need for the use of certificates in transactions with smart cards increases, the use of an online validation mechanism provided within the card becomes very attractive. An evaluation of the different online validation protocols is required in order to determine which one is more efficient for smart card use. The limitations of smart cards bring about the issues presented previously, which need to be met in a real world application.

Until now, there is little public information relating to the implementation of a validation protocol on the card. X.509 certificates can be relatively large in size, a fact that makes their management and manipulation in a smart card environment difficult. Nevertheless, X.509 is currently the most widely used certificate format and most validation protocols are to be used with such certificates [12, 15]. A smart card application that will implement these protocols can provide significant feedback concerning the practical and theoretical issues of certificate validation in smart cards. The implementation of a certificate validation protocol on a smart card can also facilitate the management of certificates within the card. The protocol can be implemented in a separate, stand-alone application which provides a shareable interface to all other applications in a multi-application smart card. As a result, any application can use the protocol, without being aware of its details. Many different validation servers can be registered to the card, which can decide where to send the validation request.

3. A SMART CARD CERTIFICATE VALIDATION MODEL

The design of a smart-card software solution can be easily split into three parts: the card side, the pc-client side and the server side. In our case we have two applications on the card side: a generic application and a validation client. The pc-client side acts as a gateway between the validation client on the card and the validation server. The entities that are involved and the technology that we used to implement them are described in this section.

3.1 The entities involved

The entities that take part in a certificate validation protocol on a smart card are the following:

- Smart card Application (SA): A third party application (applet) which has an X.509 certificate and wishes to use the card's validation functionality.
- Smart card Validation Client (SVC): It implements the validation protocol and provides its functionality to other applications through a shareable interface.
- PC-Client Terminal Application (PCAP): It communicates with the card and forwards Certificate Validation Requests from the card to the server, and Responses from the Server to the card. It also provides the APDU commands needed by the SVC to perform the validation.
- Server: This can be any server that supports OCSP, SCVP or other online certificate validation protocols.

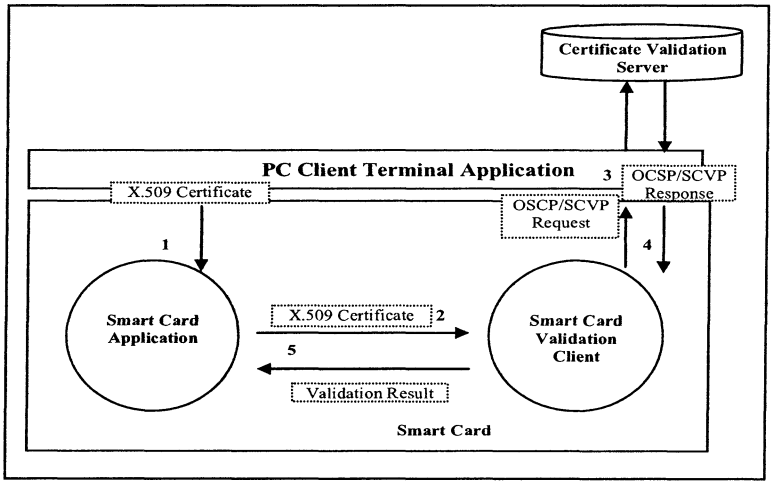

Figure 1. Message flow during protocol execution

Figure 1 illustrates the message flow during protocol execution. More specifically, in 1 the SA receives the certificate from the PCAP, and then in 2 invokes the method of SVC to send the certificate to the SVC using the Shareable Interface Object (SIO). Further on, the SVC formats an OCSP or SCVP Request and forwards it to the Server (3), which then issues an OCSP or SCVP Response (4). Finally, the SVC gets the response by the PCAP (4), verifies it and sends the result back to SA (5).

3.1.1 The Smart card Application

The SA can be any application using X.509 certificates. In our implementation it receives an X.509 certificate from the PCAP. The typical size of such a certificate is greater than 255 bytes, which is the size of the command APDU data buffer. Consequently, data will have to be sent in blocks and a series of APDUs can be used to send an X.509 Certificate. It should be noted that the Java Card environment restricts the maximum size of arrays. As a result, the maximum size of a certificate is not only limited by the small command APDU data buffer but also by the idiosyncrasies of the underlying Java Card platform.

Once receiving a command APDU containing a part of a certificate (*Send Certificate*), the SA calls a shareable interface method of the SVC which stores the certificate. Java Card SIOs [14] only allow passing of primitive types as parameters. Thus, for passing the certificate to the SVC, a global array, in our case the APDU buffer, had to be used [14]. The overall security of this approach is discussed and evaluated in [14]. A different command APDU (*Get Result*) returns the result of the validation protocol to the SA. The SA must be aware of the SVC's Shareable Interface methods.

In our implementation for the two smart card components, we used Java Card 2.1.1 [22], which is supported by our smart card application development tools. Java card is one of the most widely recognised and used smart card multi-application environments.

3.1.2 The Smart card Validation Client

This entity performs all the functions required by the validation protocol. First, it receives the certificate from the SA. The certificate is stored in a byte array throughout the execution of the protocol, as it is needed for the verification of the Server's response. A command APDU (*Create Request*) triggers the function that constructs a validation request, which is later sent to the PCAP (*Send Request*), broken into data blocks. The PCAP also forwards the validation response to the SVC (*Get Response*) where it is processed and verified (*Process Response*) so that the result can be returned to the SA. The SVC holds the Server's public key so that it can verify the digital signature in the response message. Finally, the SVC provides shareable interface methods to the SA, which are needed for passing the certificate and the result of its validation.

3.1.3 The PC-Client Terminal Application

The terminal application was implemented using PC/SC [19]. PC/SC is currently one of the most widely used and supported card terminal programming environments. PC/SC architecture is widely accepted and implemented by large and established companies such as Microsoft, Apple and Philips. Some of the supported programming languages include Visual Basic and C++. Most smart card manufacturers provide drivers for PC/SC.

The terminal application receives validation requests from the smart card in the form of multiple APDUs. Its role is to combine APDUs and send the validation request to the given validation server, as instructed by the card application. Then, it receives validation responses, breaks them into APDUs and sends them back to the card. The PC/SC terminal application is a completely passive component which simply facilitates the communication between the server and the card application, by selecting the applets of SA and SVC, and transmitting the appropriate command APDUs.

The programming language we used to implement the terminal application was Visual Basic 6. Microsoft's Visual Basic is currently very widely used and also directly supports PC/SC. For the purposes of this paper the terminal application was also configured to send an X.509 certificate to the card application. As Visual Basic does not support X.509 certificate handling and digital signatures, CAPICOM [13], a cryptographic library developed by Microsoft, was used to manipulate certificates.

3.1.4 The Server

A dummy OCSP and SCVP responder was implemented to handle OCSP and SCVP requests. The validation server checks the syntax of request messages and then issues digitally signed OCSP and SCVP responses as required. For the purposes of this paper the server was directly integrated with the PCAP. The language used for the implementation was Visual Basic v6 for compatibility and integration with the PCAP. For testing purposes the sever was configured to always send responses with either valid, invalid or unknown certificate status, regardless of the actual status of the certificate. CAPICOM was used to create digital signatures.

3.2 Implementation Details

The SVC was implemented to handle OCSP requests and responses. The implementation was based on OCSPv2. In order for no certificate manipulation to be required the entire certificate was included in the OCSP request. For maximum efficiency, a specific ASN.1 parser and DER encoder

and decoder were implemented, which can only handle such messages. Moreover, only a few of the OCSP Response acceptance requirements specified in [16] were implemented: the certificate was compared with the original queried certificate and then its status was retrieved. The SVC was implemented to only accept messages from a specific trusted responder.

The verification of the digital signature in the validation response messages may be essential for the protocols, but is also a time-consuming function when performed on the card [20]. The purpose of this paper is to evaluate the performance of revocation protocols and not of the signature algorithms. As a result, and for testing and evaluation purposes, digital signatures were not verified on the card so as not to influence our results. However, smart cards running the SVC should support the most known algorithms for digital signatures and hash functions, as digital signature verification is required for correct execution of both OCSP and SCVP.

4. RESULTS AND PERFORMANCE EVALUATION

In this section we present and evaluate the results and timings we took for each of the two protocols, using two different smart cards.

We have generated a set of results for OCSP and SCVP, using a specified, 573 byte, X.509 certificate. Two different high end Java cards were used provided by Vendor A and Vendor B. The smart card application development tools were also different respectively. Due to Java Card's interoperability, there are only minor changes to the implementation for each card, that don't affect the overall performance. Each protocol was executed 5 times for each card. Timings, expressed in milliseconds, are presented in table 1. They were taken using an APDU monitoring tool attached to a P4 Windows machine. Commands marked with * only involve data I/O. The functionality of each command is explained in section 3. We also include for reference timings required for sending the certificate to the card.

The timing results presented in Table 1 are coherent. Even though execution times for most commands differ between the two cards, we can reach into the same conclusions for the protocols we implemented. It should be noted, that the differences in each card's timings is attributed to the different nature of each card. The statistical analysis of the results that we presented leads us to the remarks that we analyze in this section.

First of all, we observe that for OCSP the *Create Request* command is the least time consuming of all. The OCSP protocol doesn't require any special functions for creating request messages, and thus, an OCSP request is created really fast. On the other hand, for SCVP it appears to be the most time consuming command, excluding commands that handle I/O, as a hash

function has to be computed. The processing of an OCSP response (*Process Response*) is the most time-consuming command for the OCSP protocol, excluding the ones regarding I/O. The comparison of the certificate in the response with the original queried certificate is what makes this command more time-consuming than any other. SCVP responses in contrast, require the comparison of a much smaller byte array and thus demand less time, as a hash of a certificate is, of course, smaller than the certificate itself.

Table 1. Performance results

		Vendor A		Vendor B	
Command	Metrics (clk)	OCSP	SCVP	OCSP	SCVP
*Send Certificate**	Average	13978,48		37614,04	
	Median	13994,71		37363,65	
	Std. Dev.	22,81		563,74	
Create Request	Average	8,56	1351,56	24,83	1932,78
	Median	8,53	1351,84	24,83	1934,21
	Std. Dev.	0,03	8,11	0,02	42,80
*Send Request**	Average	28918,72	20,02	14064,73	25,56
	Median	28951,10	22,56	14035,67	25,56
	Std. Dev.	48,34	3,54	62,85	0,03
*Get Response**	Average	20710,90	24776,68	57034,72	67655,90
	Median	20941,47	24829,20	56719,33	67417,51
	Std. Dev.	2423,76	113,20	718,87	702,24
Process Response	Average	68,17	48,32	421,00	322,18
	Median	66,62	55,94	458,39	312,54
	Std. Dev.	3,21	21,13	52,56	23,50
Get Result	Average	14,47		121,03	
	Median	14,50		114,88	
	Std. Dev.	0,05		12,72	
Total	Average	63699,30	40189,53	109280,35	107671,49
	Median	63976,93	40268,75	108716,75	107168,35

Furthermore, the most time-consuming functions of all are the ones that have to do with the input of the certificate (*Send certificate*), OCSP and SCVP responses (*Get Response*) to the card and the output of OCSP requests (*Send Request*). Concerning the transmission of the certificate to the card, it is obvious that even for a cut-off version of an X.509 certificate a significant amount of time is required. As a result, we have to consider the use of other certificate formats. The transmission of the response messages for both protocols requires the same amount of time, as their size is almost the same. On the other hand, a SCVP request is significantly smaller than an OCSP request. Thus, only a very small fraction of the time needed to send the OCSP request, is required to send a SCVP request message. Consequently,

any decrease in the size of the messages will have a significant impact on the total time in which the protocol is executed. The *Get Result* command only involves changing the value of some variables, and thus, it doesn't interfere with the protocol run. Dring protocol runs it was observed that the cards quickly ran out of memory and the applets needed to be reinstalled. This was attributed to the fact that many large arrays are used. Thus, special care has to be taken for memory allocation and garbage collection.

Overall, SCVP runs faster than OCSP in both cards. This is mostly because a SCVP request is much smaller and thus can be sent much faster than an OCSP request. Even though the creation of a SCVP request is more time consuming, the fact that it can be transmitted in a single APDU makes SCVP more efficient. As we already mentioned, the commands that involve data I/O are the most time consuming and any alteration of the size of the messages has more impact on the time required for a complete protocol run than an improvement of any other command might have.

5. CONCLUSIONS

OCSP and SCVP protocols were implemented on two different Java Card platforms. Despite the issues that came up and the compromises regarding the certificate size, we have shown that it is feasible to implement and run known and widely used certificate validation protocols on a smart card. Even though the memory size of modern smart cards has significantly increased, an X.509 certificate is still quite large to be used in such a limited environment. The time that is required for a certificate to be loaded onto the card is clearly a major factor. However in future work there may be scope to reduce delays by exploiting faster card I/O options.

Furthermore OCSP is not very efficient for use in a smart card environment. OCSPv1 cannot be used at all, as certificate manipulation and path construction adds a significant overhead. We have shown that an implementation of OCSPv2 is feasible, even though the messages involved are quite large in size. Additionally, checks regarding time cannot be performed on a smart card, even though corresponding fields add up to the total size of the messages. SCVP, which is recommended for use in limited environments, uses a hash of the certificate, significantly reducing the overall size of all messages. This reduces the time required for I/O, but also increases the time required for a construction of a SCVP request. Nevertheless, in total, SCVP runs faster than OCSP.

Currently, we are experimenting with alterations to existing validation protocols as well as system architectures in order to have a more efficient protocol, specifically designed for smart cards. The suggestions of OMA

[18] are also considered, as so far there has not been any known implementation of their protocol. Moreover, recently a new IETF draft on OCSP was submitted [3], describing a lightweight implementation of OSCP, but it is not specifically designed for smart cards. The design of a smart card-specific certificate validation protocol is also examined, as well as the support of other certificate formats [1, 21, 23] that can facilitate certificate revocation and validation. In particular, we need to focus on formats that provide a more compact and efficient way of storing and managing certificates [17] and key pairs [4]. These will enable us to provide more accurate figures and comments on the performance of these protocols in a multi-application smart card environment.

REFERENCES

1. ANSI. X9.68 - 2001: Digital Certificates for Mobile/Wireless and High Transaction Volume Financial Systems: Part 2: Domain Certificate Syntax. 2001
2. A. Arnes. Public Key Certificate Revocation Schemes. PhD thesis. Norwegian University of Science and Technology, 2000
3. A. Deacon and R. Hurst. Lightweight OCSP Profile for High Volume Environments, IETF, 2004
4. N. Feyt and M. Joye. A Better Use of Smart Cards in PKIs. Gemplus Developer Conference, Singapore. Springer Verlag, 2002
5. P. Hoffman. RFC 2634 - Enhanced Security Services for S/MIME. IETF, 1999
6. R. Housley. RFC 2630 - Cryptographic Message Syntax. IETF, 1999.
7. R. Housley, W. Polk, W. Ford and D. Solo. RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF, 2002
8. ISO. ISO/IEC 7816-4, Information technology – Identification cards - Integrated Circuit(s) cards with contacts – Interindustry Commands for Interchange. ISO, 1995
9. ITU-T Recommendation X.509. Information Technology - Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks. 1997
10. ITU-T Recommendation X.681. Information technology - Abstract Syntax Notation One (ASN.1): Information object specification. 1997
11. ITU-T Recommendation X.690. Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). 2002
12. A. Malpani, R. Housley and T. Freeman. Simple Certificate Validation Protocol (SCVP). IETF, 2003
13. Microsoft. CAPICOM Reference. http://msdn.microsoft.com/library/en-us/security/Security/capicom_reference.asp
14. M. Montgomery and K. Krishn. Secure Object Sharing in Java Card. USENIX, 1999
15. M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams. RFC 2560 - X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. IETF, 1999
16. M.Myers, A. Malpani, D.Pinkas. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol version 2. IETF, 2002
17. M. Nyström and J. Brainard. An X.509-Compatible Syntax for Compact Certificates. In Proc. Int. Exhibition and Congress on Secure Networking '99, Springer-Verlag, 1999

18. Open Mobile Alliance. OCSP Protocol Mobile Profile Candidate V1.0, 2004
19. PC/SC Workgroup. Interoperability Specification for ICCs and Personal Computer Systems. http://www.pcscworkgroup.com/, 1997
20. J-J. Quisquater and M. De Soete. Speeding up smart card RSA computations with insecure coprocessors. in Smart Card 2000. Amsterdam, 1991
21. RSA Labs. PKCS #15 v1.1: Cryptographic Token Information Syntax Standard, 2000
22. Sun Microsystems. Java Card 2.1.1 Application Programming Interface, Rev. 1.0. 2000
23. WAP Forum. WAP Certificate and CRL Profiles Specification, 2001