Emerald Insight

International Journal of Pervasive Computing and Communications

A password-authenticated secure channel for App to Java Card applet communication

Michael Hölzl Endalkachew Asnake Rene Mayrhofer Michael Roland

Article information:

To cite this document:
Michael Hölzl Endalkachew Asnake Rene Mayrhofer Michael Roland , (2015),"A password-authenticated secure channel for App to Java Card applet communication", International Journal of Pervasive Computing and Communications, Vol. 11 Iss 4 pp. 374 - 397
Permanent link to this document:
http://dx.doi.org/10.1108/IJPCC-09-2015-0032

Downloaded on: 07 November 2016, At: 22:28 (PT)
References: this document contains references to 42 other documents.
To copy this document: permissions@emeraldinsight.com
The fulltext of this document has been downloaded 167 times since 2015*

Users who downloaded this article also downloaded:

(2015),"An analysis of tools for online anonymity", International Journal of Pervasive Computing and Communications, Vol. 11 Iss 4 pp. 436-453 http://dx.doi.org/10.1108/IJPCC-08-2015-0030

(2015),"The accessibility and usage of smartphones by Arab-speaking visually impaired people", International Journal of Pervasive Computing and Communications, Vol. 11 Iss 4 pp. 418-435 http://dx.doi.org/10.1108/IJPCC-09-2015-0033

Access to this document was granted through an Emerald subscription provided by emerald-srm:563821 []

For Authors

If you would like to write for this, or any other Emerald publication, then please use our Emerald for Authors service information about how to choose which publication to write for and submission guidelines are available for all. Please visit www.emeraldinsight.com/authors for more information.

About Emerald www.emeraldinsight.com

Emerald is a global publisher linking research and practice to the benefit of society. The company manages a portfolio of more than 290 journals and over 2,350 books and book series volumes, as well as providing an extensive range of online products and additional customer resources and services.

Emerald is both COUNTER 4 and TRANSFER compliant. The organization is a partner of the Committee on Publication Ethics (COPE) and also works with Portico and the LOCKSS initiative for digital archive preservation.

*Related content and download information correct at time of download.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

The current issue and full text archive of this journal is available on Emerald Insight at:
www.emeraldinsight.com/1742-7371.htm

Received 4 September 2015
Revised 4 September 2015
Accepted 7 September 2015

A password-authenticated secure channel for App to Java Card applet communication

Michael Hölzl
JRC u'smile and Institute of Networks and Security, Johannes Kepler University Linz, Linz, Austria
Endalkachew Asnake
JRC u'smile, University of Applied Sciences Upper Austria, Hagenberg, Austria
Rene Mayrhofer
JRC u'smile and Institute of Networks and Security, Johannes Kepler University Linz, Linz, Austria, and
Michael Roland
JRC u'smile, University of Applied Sciences Upper Austria, Hagenberg, Austria

Abstract

Purpose – The purpose of this paper is to design, implement and evaluate the usage of the password-authenticated secure channel protocol SRP to protect the communication of a mobile application to a Java Card applet. The usage of security and privacy sensitive systems on mobile devices, such as mobile banking, mobile credit cards, mobile ticketing or mobile digital identities has continuously risen in recent years. This development makes the protection of personal and security sensitive data on mobile devices more important than ever.

Design/methodology/approach – A common approach for the protection of sensitive data is to use additional hardware such as smart cards or secure elements. The communication between such dedicated hardware and back-end management systems uses strong cryptography. However, the data transfer between applications on the mobile device and so-called applets on the dedicated hardware is often either unencrypted (and interceptable by malicious software) or encrypted with static keys stored in applications.

Findings – To address this issue, this paper presents a solution for fine-grained secure application-to-applet communication based on Secure Remote Password (SRP-6a and SRP-5), an authenticated key agreement protocol, with a user-provided password at run-time.

Originality/value – By exploiting the Java Card cryptographic application programming interfaces (APIs) and minor adaptations to the protocol, which do not affect the security, the authors were able to implement this scheme on Java Cards with reasonable computation time.

Keywords Mobile devices, Java Card, Secure channel, Secure element, Smart card, SRP

Paper type Research paper

This work has been carried out within the scope of u'smile, the Josef Ressel Center for User-Friendly Secure Mobile Environments, funded by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, NXP Semiconductors Austria GmbH and Österreichische Staatsdruckerei GmbH. This article is an extended version of M. Hölzl, E. Asnake, R. Mayrhofer and M. Roland. Mobile Application to Java Card Applet Communication using a Password-authenticated Secure Channel. In *Proceedings of the 12th International Conference on Advances in Mobile Computing & Multimedia (MoMM2014)*. ACM, Kaohsiung, Taiwan, December 2014.

International Journal of Pervasive Computing and Communications
Vol. 11 No. 4, 2015
pp. 374-397
© Emerald Group Publishing Limited
1742-7371
DOI 10.1108/IJPCC-09-2015-0032

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

1. Introduction

Nowadays, mobile devices such as mobile phones, tablets or smart watches have become an indispensable part of our daily life, but these are encountering many security threats (cf. Khan *et al.*, 2012; La Polla *et al.*, 2013; Mayrhofer, 2014). They can not only be easily stolen or lost but also face attacks from malicious third-party applications. We also have to be aware that the flash memory on these mobile devices cannot be trusted, and unauthorized individuals have the possibility to gain personal, business or other security and privacy sensitive data from that memory. To provide a secure environment for storing data, applications often use an additional tamper-resistant hardware like smart cards or secure elements. Special variants of them run with Java Card technology, which allows the execution of small Java-based applications (applets). One advantage of these tamper-resistant units is the protection of data and executed code against various physical and software attacks. Communication between the card and an off-card application is only possible through a standardized interface for exchanging the so-called *Application Protocol Data Units* (APDU). There are many applications, such as the systems by Ruiz-Martinez *et al.* (2007) and Mantoro and Milisic (2010), that take advantage of this tamper resistance by storing master passwords on the hardware. However, when using smart cards for applications, it is important to consider that the communication outside the applet is not automatically secured. This is especially relevant in an environment where malicious third-party applications could eavesdrop on the data transfer.

The main motivating goal in this paper is to provide an infrastructure for third-party applications on mobile devices to securely communicate through a password-authenticated secure channel with applets running on the Java Card. In an infrastructure where applications running on the main processor also have a corresponding applet running on the tamper-resistant environment, they could then use this additional hardware module to securely store security and privacy sensitive data. A mobile Web browser could, for example, use it to store passwords. Or, company services could use it for storing private keys for corporate virtual private network (VPN) access. The advantage of additional hardware is that malware running on the device would not have the possibility to directly read this password storage. Even if these passwords were encrypted on the flash memory using a master password, an attacker would still be able to brute-force this master password using an offline dictionary attack. Although there are ways to make brute-force attacks on encrypted user-chosen passwords harder (e.g. using a key-derivation function), this remains problematic as users tend to use weak and easily guessable passwords on mobile devices as demonstrated by Landman (2010) and Ben-Asher *et al.* (2011).

There are existing protocols for current smart cards that provide secure communication between the card and an off-card application. The respective standards are defined by GlobalPlatform (GP)[1]. However, these standards only provide a secure interface for managing applications (installation, removal, etc.) and card-specific data (e.g. personalization of applications with user-specific data). A secure channel is established between a back-end server and a security domain on the card using a shared secret. Consequently, using this secure management channel from within a mobile device application to communicate with the applets would require that shared secret to be stored within the application data. An attacker who gained access to that shared key could therefore access and manipulate that security domain with all contained applets

and data. To protect the data of each application, a more fine-grained secure channel for application-to-applet transmission is needed (cf. Hölzl *et al.*, 2013). In this paper, we address this issue by proposing an implementation of a fine-grained secure channel that is authenticated with a user-provided PIN or password, compatible to current Java Cards and time efficient for the user.

For the implementation of this channel, we use the *Secure Remote Password* (SRP) protocol in its latest revisions SRP-6a and SRP-5 by Wu (1998, 2002) and the IEEE Computer Society (2009). SRP is a password-authenticated key agreement protocol which is based on the Diffie and Hellman (1976) key exchange and can be either constructed from or reduced to Diffie-Hellman (Wu, 1998). The main extension is that a Diffie-Hellman key exchange is not authenticated, while SRP can be used for password-based mutual authentication over insecure communication channels. Besides SRP, there are multiple other similar password-authenticated key agreement (PAKE) protocols such as SPEKE by Jablon and Ma (1996), J-PAKE by Hao and Ryan (2011), EKE by Bellovin and Merritt (1992), OKE by Lucks (1997), AuthA by Bellare and Rogaway (2000) and others. There are also PAKE variants which make use of elliptic curve cryptography (ECC). While it has been shown that elliptic curve cryptography is faster than other public key algorithms (Han *et al.*, 2002; Gayoso Martinez *et al.*, 2005), the usage of such protocols is restricted on Java Cards due to the limitations in terms of ECC support in the current Java Card 3.0. However, with the usage of proprietary classes, which were added by smart card manufacturers, it is possible to support such functionality.

Finally, we chose SRP because it can be used without any licensing, has no known flaws in its current implementation, provides an elliptic curve variant and is already included in other cryptographic protocols such as the transport layer security (TLS) protocol by Taylor *et al.*, 2007. One disadvantage of this protocol is the high complexity of computations due to modulo operations on big numbers. Therefore, a pure software implementation of this protocol on a Java Card with severe restrictions such as an 8-bit CPU and little memory is very time-consuming and not practical. In this paper, we present a solution on how to execute SRP on low-end hardware platforms within a reasonable authentication time and an overall protocol runtime below four seconds. In our terms, we define reasonable authentication time as the time a user is willing to wait for a secure channel to be authenticated after password entry. Based on the Nielsen Norman Group, we assume that this value is below 1 second[2]. The key points and main contributions of our approach are:

- Implementation of a password-authenticated secure channel protocol on Java Cards, which is suitable for comparably weak passwords and does not require to store credentials in the mobile device flash memory, by exploiting the RSA public key encryption operation for a significant increase in computation performance.
- Minor adaptations to the SRP protocol scheme to optimize verification time after PIN/password entry.
- We provide an elliptic curve variant which is based on the standardized SRP-5 and incorporates our proposed adaptations for optimized verification time.
- Memory optimizations for the implementation to reduce required transient and persistent memory during protocol execution.
- An open-source implementation for developers.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

2. Related work

There have been multiple previous publications in the area of providing an authenticated channel for Java Card applets. From an industry standardization point of view, the *GP Secure Channel Protocols* (SCP) are one of the most relevant ones. The GP specification defines standards for secure channel protocols, namely, SCP01, SCP02 and SCP03, to establish secure communication between a Java Card and an off-card application. According to GP Card Specification 2.2 by GlobalPlatform (2009), there are two ways in which an applet could handle a secure channel, namely, *Direct Handling* and *Indirect Handling*. In *Direct Handling*, an applet is fully responsible for implementing the protocol and defining its own security domain. The other approach is *Indirect Handling* where the applet uses readymade services provided by security domains to handle the SCP. This enables the applet to be implemented independently from the protocol and leaves secure channel related computation to the security domain it belongs to.

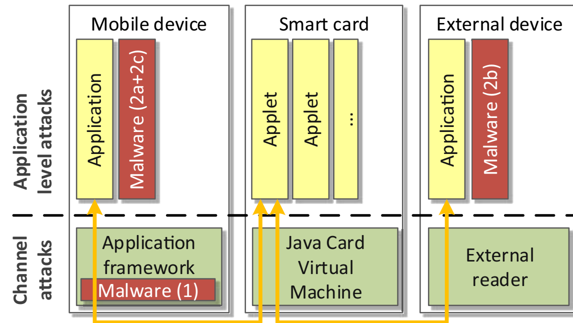
One of the main advantages in using a GP SCP for secure application-to-applet communication is that it is an industry standard and a widely used protocol with application programming interface (API) support on Java Cards. For our use, cases we consider the off-card environment as potentially insecure, while we trust in the security of the Java Card. Because GP SCP authentication relies on static shared keys between communicating parties, the insecurity of the off-card application breaks the security of the protocol. These limitations force us to look into other password-based authentication schemes for a more fine-grained secure channel protocol.

Various previous papers have been published that make use of the Java Card crypto API to execute modulo operations on the card's cryptographic co-processor for an efficient implementation of different protocols. Sterckx *et al.* (2009) discuss simplified methods to use *Direct Anonymous Attestation* (DAA) (Brickell *et al.*, 2004) on Java Cards. As many other cryptographic protocols, DAA also involves computation of modulo operations on big numbers. In their paper, they show that these modulo operations can be computed more efficiently with the help of the cryptographic co-processor compared to a pure software implementation. Tews and Jacobs (2009) show different RSA variants and performance measurements of Brands' protocols for zero-knowledge proof (Brands, 2000). This paper also proposes an efficient implementation of a protocol by exploiting the Java Card crypto API to perform modulo operations on the cryptographic co-processor.

Our implementation of big numbers and modular arithmetic in the Java Card applet is based on this previous scientific work.

3. Threat model

Our approach for implementing SRP on smart cards is based on the Android platform due to availability of open-source projects which enable access to smart cards and secure elements. In addition to this, the openness of the platform makes it easier to analyze different types of attacks. Hence, we use the Android Operating System (OS) to specify our threat model categories (Figure 1 gives an overview of these categories in a mobile device context):



Note: The arrows depict the secure channel between application and applet

Figure 1.
Overview of our threat model

3.1 Channel attacks

The Android OS has built-in security features which protect applications against different types of threats. One of these features is application sandboxing. Every Android application has a unique user ID that is assigned at install time. At run-time, an application runs in a sandbox where communication to other applications is made possible via Inter Process Communication (IPC) facilities. One IPC facility used in our implementation is a service. Android services are application components that are used to handle long-running tasks in the background. They also provide a client-server interface which can be used by different applications to request specific functions of a service. One of the possible channel attacks on Android applications which make use of IPC facilities is *service hijacking*. This happens when an application creates a connection with a malicious service instead of the intended one, as discussed by Chin *et al.* (2011). With such kind of possibilities, an attacker can gain full control over the message exchange between an application and Java Card applets, which creates a suitable condition for active channel attacks.

3.2 Attack on application level

This sort of attack can come from mobile applications as well as over NFC:

- *Malicious applications*: We consider two different threats from malicious applications: the first threat can be caused by an application that has been granted root privileges or that uses privilege escalation exploits, as demonstrated by Höbarth and Mayrhofer (2011). Such kind of malware can access application private storage and conduct different attacks on the content (e.g. brute-force attack on encrypted passwords). In the second threat, the malware does not need root privileges, but makes use of smart card services (cf. Roland *et al.*, 2012). If a malicious application has sufficient privileges to access smart card services, it can impersonate a legitimate application and try to brute-force the password to establish a channel to the applet. With methods discussed in this paper, the applet is protected against the first threat, while the second should, in the worst case, end up with denial of service.
- *Attacks from an external card reader*: Usually, smart cards and secure elements can be accessed from an external card reader through either the contact or

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

contact-less interfaces. An attacker who has physical access to the card and the mobile device or an attacker who is able to communicate with the smart card over *Near Field Communication* (NFC) could launch the same attacks as a malware mobile application. Additionally, we also have to consider attack scenarios over the contact-less interface (e.g. relay attacks over NFC by Hancke (2005)).

- *User interface (UI) masquerading or control by malicious entity*: Mobile malware that gains control over the UI (i.e. finds methods to listen to users' input), or is faking the UI can steal users' passwords, which could result in severe consequences. This is especially problematic when the user is not aware of the compromised password. These threats are out of scope of this paper but could potentially be addressed by using a trusted execution environment (TEE) for secure password input (e.g. TrustZone[3]). Hence, the password for establishing a secure channel cannot be stolen by a malicious application. However, the methods discussed in this paper still help to secure the data path between application and applet after password entry.

4. Secure remote password

The SRP protocol was introduced by Wu (1998) and is a password-authenticated key agreement protocol that can be used over insecure channels for providing password-based secure key agreement and authentication. Similar to the key exchange by Diffie and Hellman (1976), an eavesdropper is not able to guess the computed session key even with the knowledge of the complete data transfer. The big advantage of SRP in comparison to Diffie-Hellman is that SRP provides password-based mutual authentication. Additionally, the properties of SRP make the protocol resistant against most prominent attacks (e.g. off-line dictionary attacks, replay attacks). The IEEE Computer Society (2009) also standardized an elliptic curve variant of this protocol (SRP-5) in the IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques (IEEE 1,363.2-2008). In the upcoming description of the SRP protocol, we will start by explaining the discrete logarithm variant (SRP-6a) and our proposed adaptations to it. We will elaborate on the elliptic curve variant of the protocol (SRP-5) in Section 4.4.

4.1 SRP key exchange procedure

Scheme 1 describes the steps of the original SRP protocol. Communication between both parties is generally separated into two phases:

	Client		Server
1.		$\xrightarrow{I}$	(lookup s, v)
2.	$x = H(s, I, P)$	$\xleftarrow{s}$	
3.	$A = g^a$	$\xrightarrow{A}$	$B = kv + g^b$
4.	$u = H(A, B)$	$\xleftarrow{B}$	$u = H(A, B)$
5.	$S = (B - kg^x)^{a+u\cdot x}$		$S = (Av^u)^b$
6.	$M_1 = H(A, B, S)$	$\xrightarrow{M_1}$	(verify M_1)
7.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, S)$
8.	$K = H(S)$		$K = H(S)$

Scheme 1. Original SRP-6a protocol scheme as described by Wu (2002)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

(1) a key agreement phase where both parties calculate the shared secret (Steps 1-5 in Scheme 1); and
(2) a mutual verification phase where they authenticate each other (Steps 6-8 in Scheme 1).

For the key agreement phase, public keys A and B are calculated by modular exponentiation with a private key exponent a and b. On the server-side, the public part g^b is additionally XORed with the password verifier v multiplied by constant k (Steps 1-4).

After exchanging public parameters A and B, the client computes the shared secret S based on the user identification (identifier I and password P), while the server does the same using the pre-computed verifier v (Step 4 in Scheme 1). Based on this shared secret, both parties have to mutually authenticate each other in the verification phase. This is done by exchanging the verifiers M_1 and M_2 with the corresponding opposite party (Steps 6-7). Then both parties ensure that legitimate values of the client password x and the server verifier v are used in the key agreement phase. After this verification, they compute the same session key $K = H(S)$ (Step 8) and use this as basis for securing future data transfers against eavesdroppers or active attackers. A list of all SRP protocol notations can be found in Table I.

4.2 Proposed protocol adaptations for Java Card applets

To improve computation time and memory usage of the protocol on a Java Card, we made minor changes to the SRP protocol (changes are highlighted in bold font in our protocol in Scheme 2). First, we combine the transmission of the server's public key B

g, n	The generator g and a large prime modulo number n for all computations
s	A random user's salt for the password
I, P	Identifier and password of the user
k	Constant multiplier, computed from the hash of the modulo and concatenated with g
x	Private key derived from identifier, password and salt
v	The password verifier calculated from g^x
u	Random scrambling parameter, publicly revealed
a, b	Ephemeral private keys, generated randomly and not publicly revealed
A, B	Corresponding public keys
$H(\cdot)$	One-way hash function
K	Computed session key

Table I. SRP protocol notation as described by Wu (1998, 2002)

	Client		Server
1.	$A = g^a$	$\xrightarrow{A}$	$B = kv + g^b$
2.	$u = H(A, B)$	$\xleftarrow{B,s}$	$u = H(A, B)$
3.	$x = H(s, P)$		
4.	$S = (B - kg^x)^{a+u\cdot x}$		$S = (Av^u)^b$
5.	$\boldsymbol{K = H(S)}$		$\boldsymbol{K = H(S)}$
6.	$M_1 = H(\boldsymbol{u}, S)$	$\xrightarrow{M_1}$	(verify M_1)
7.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(\boldsymbol{u}, M_1, S)$

Scheme 2. Java Card applet. Our protocol implementation of SRP-6a with minor changes to improve memory consumption and performance

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

with the user's salt s. The current revision of SRP-6a by Wu (2002) uses an additional round of transmission to agree on the identifier I and the salt s at the beginning. In our elaborated use case of fine-granular application-to-applet communication, we only need one secure channel instance for each applet. Therefore, we do not need an identifier, as we only have one verifier v – which is computed during applet installation – and can directly start the communication by sending the public key A to the Java Card (see Steps 1 and 2 in Scheme 2). This reduces the amount of required round trips from three to two (a similar approach was also suggested as an optimized version in the original protocol publication, cf. Wu (1998)).

As in the original version, the password verifier v is also pre-computed but uses the elliptic curve generator G in the equation: $v = d_x \cdot G$. The other constants and variables are also the same as listed in Table I with the difference that G is used instead of g.

The second adaptation influences the sequence of calculating the session key K. Usually, this key is calculated after the mutual verification phase (see Step 8 in the original SRP protocol Scheme 1). To reduce the time required for the verification (Step 6 and 7 in Scheme 2), we moved the calculation of K to the key agreement phase (Steps 1-5 in Scheme 2). On the server-side (the smart card), this first key agreement phase does not require any authentication of the user and can therefore be done before or while the user is typing password or PIN (or while running other authentication mechanisms like fingerprint, face unlock, special touch patterns, etc.) On the client-side, Steps 3 to 5 are performed after the password has been entered. However, the computation time of these operations is negligible on a high-performance mobile device processor. The actual time a user has to wait for the server to establish a secure channel is therefore reduced to the key verification phase in Steps 6 and 7. So, in the use case of a password manager on the tamper-resistant hardware, the first data exchange starts when the application is opened. While the user then enters the password, the more computationally intensive operations of the key agreement phase can be executed simultaneously on the server-side. After the user enters the password, the server only requires the verifier M_1 to verify the secure channel and give access to the password manager.

The third adjustment is related to the computation of verifiers M_1 and M_2. In the original protocol, these verifiers were computed as $M_1 = H(A, B, S)$ and $M_2 = H(A, M_1, S)$. However, the authors of the protocol suggest this approach as only one possible way to mutually verify client and server. In our approach, we suggest another efficient way of computing the verifiers. Instead of using the public keys A and B for computing M_1 and M_2, we use the scrambling parameter u. This change optimizes the time of execution for the verification steps on the Java Card side. For example, using SHA-256 as a hash function and a modulus N of 2048 bit, the verification of M_1 will require computing SHA-256 over a 768 bytes input. This is reduced to 288 bytes by replacing (A, B) with (u) and reduces the number of intermediate operations required to compute M_1 and M_2.

4.3 Security analysis

From a security point of view, we argue that these changes do not affect the security of the verification and the key agreement. The first two adaptations change the sequence without actually changing the communication and computations from the original SRP-6a scheme.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

For the third proposed adaptation (the change in the verification), we argue that the security of the protocol is not affected as other proven protocols use a similar approach of double hashing. The most famous one is *Hash-based Message Authentication Code* (HMAC) by Bellare *et al.* (1996), which also makes use of a hash inside a hash function.

4.4 SRP with elliptic curve cryptography

Besides the discrete logarithm based protocol SRP-6 (as described in the previous section) and SRP-3, the IEEE Computer Society (2009) also standardized a variant of the SRP protocol, with version number SRP-5, using Elliptic Curve Cryptography (ECC) (IEEE 1,368.2-2008). As stated by Anoop (2007), mathematical operations in ECC are performed on an Elliptic Curve (EC) $x[3] + ax + b$, where different values for a and b define different curves. Several organizations, such as the NIST (Barker *et al.*, 2012), the European Network of Excellence in Cryptology II (2012) (ECRYPT II), Certicom Research (2010) as part of the Standards for Efficient Cryptography Group (SECG) or the ECC Brainpool working group RFC (written by Lochter and Merkle, 2010), specify such values. These values are always defined together with other elliptic curve domain parameters (cf. Anoop, 2007): (p, a, b, G, n, h) for ECs over $\mathbb{F}_p$ and $(m, f(x), a, b, G, n, h)$ for ECs over $\mathbb{F}_{2m}$. The main goal of each of these specified curves is to ensure that the Elliptic-Curve Discrete-Logarithm Problem (ECDLP) is difficult, which is the problem an attacker has to solve to retain a private key, given a user's public key, as described by Koblitz (1987).

The major advantage of ECC is that smaller key sizes are required for the same security level as in non-ECC public-key cryptography. Table II gives an overview of the security level of several key sizes in ECC compared to RSA by different organizations. While the numbers slightly vary between these organizations, they all show that the advantage of smaller required key sizes gets even better for higher security levels. EC roughly requires key sizes that are double the size of the required security level. In RSA, the required key sizes increases up to a factor of 60 (e.g. RSA requires a key size of 15,360 bits for 256 bit security in the NIST (2012) key size comparison).

Organization	Security (bits)	RSA (bits)	EC (bits)
NIST (2012)	80	1,024	160
	112	2,048	224
	128	3,072	256
	192	7,680	384
	256	15,360	512
ECRYPT II (2012)	80	1,248	160
	112	2,432	224
	128	3,248	256
	192	7,936	384
	256	15,424	512
	96	1,536	192
SECG (2010)	112	2,048	224
	128	3,072	256
	192	7,680	384
	256	15,360	521

Table II. Key size comparison between RSA and EC for different security levels by several organizations

4.4.1 Differences to SRP-6a. The sequence of the SRP-6a protocol is mostly the same in the EC variant. The difference, in terms of mathematical operation, is that multiplications with EC points are performed instead of modular exponentiation and point additions are performed instead of modular multiplications. For example, rather than $A = g^a$, we have $Q_A = d_a \cdot G$ (where G is the generator EC point, d_a the private key and Q_A the resulting public key).

The other main difference to the SRP-6a protocol is the creation of a pseudo random EC point E. This point is derived from the x-coordinate of the password verifier V using the Random Element Derivation Primitive (REDP) and should prevent collisions as well as obscures exponential relationship in the public key. The main steps of this REDP are:

- creating a hash of the input;
- add padding, hash it again and define the outcome as the x-coordinate of the new EC point;
- computing the y-coordinate using the appropriate EC equation (e.g. x[3] + $ax + b$); and
- verify if the point is on the curve and return to the second step if it is not, after incrementing the hashing result of the first step.

A detailed description of this primitive can be found in the standard IEEE-1363.2-2008 by the (IEEE Computer Society, 2009, the ECREDP-1 function in Section 8.2.17 should be used for SRP-5).

4.4.2 Protocol establishment. Scheme 3 shows the complete EC variant of the SRP protocol based on the SRP-5 standard in IEEE 1,368.2-2008. As in the SRP-6a variant, the protocol is divided in two phases: Steps 1-7 are the key agreement phase and steps 8-9 are the verification phase. In the key agreement phase, the protocol establishment is initiated when the client generates the public key $Q_A = d_A \cdot G$ and sends it to the server. On the server-side, the public key Q_B is computed by multiplying the private key d_B with the EC base point G and adding the password derived EC point E. The password derived EC points E and the password verifier V do not need to be computed during this phase because they can be pre-computed and stored on the server-side. These values only change if the password changes or a different curve is used.

With the knowledge of both public keys, the server computes the random scrambling parameter u by using the x-coordinates of Q_A and Q_B. Furthermore, it can also compute

	Client		Server
1.	$Q_A = d_A \cdot G$	$\xrightarrow{Q_A}$	$Q_B = d_B \cdot G + E$
2.	$u = H(Q_A, Q_B)$	$\xleftarrow{Q_B, s}$	$u = H(Q_A, Q_B)$
3.	$x = H(s, I, P)$		
4.	$V = x \cdot G$		
5.	$E = REDP(X(V))$		
6.	$S = (Q_B - E) \cdot (d_A + u \cdot x)$		$S = d_B \cdot (Q_A + u \cdot V)$
7.	$K = H(S)$		$K = H(S)$
8.	$M_1 = H(u, S)$	$\xrightarrow{M_1}$	(verify M_1)
9.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(u, M_1, S)$

Scheme 3. Elliptic curve version of our protocol implementation based on SRP-5 in IEEE 1363.2-2008 by the IEEE Computer Society (2009)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

the shared secret $S = d_B \cdot (Q_A + u \cdot V)$ and the session key $K = H(S)$. Considering a smart card as the server-side, this should already be performed before returning the public key Q_B and the user's salt s to the client-side (as illustrated in Step 2). As a consequence, the session key K can be computed while the client-side still asks the user to enter the PIN/password. After the user entered this PIN/password and the smart card returned the public parameters (Q_B, s), the client-side can also compute the session key K with all the required intermediate steps (Steps 2-7 in Scheme 3). As the client-side is assumed to be a mobile phone providing good processing speed, these steps can be executed efficiently with reasonable performance.

After a successful key agreement phase, both parties should have the same key K. To mutually verify that this assumption is true, server and client perform the same two steps as in SRP-6a (compare Steps 6-7 in Scheme 2 and Steps 8-9 in the EC variant of Scheme 3).

5. Secure session

In any symmetric key cryptographic system, the first step is an agreement on a secure session key between communicating parties. This session key is used to establish a confidential and authentic channel between the parties. The standard approach in providing confidentiality and authenticity is to use two different algorithms for each purpose with an authenticated encryption scheme known as Encrypt-then-MAC. Recently, different algorithms have been standardized to provide both functionalities at the same time. A well-known example for this would be *Galois/Counter Mode* (GCM) from ISO/IEC 19772:2009. However, such algorithms or operation modes are not yet included in the Java Card standard (Table III).

For a secure session between the card and the off-card application, the smart card standard provides a protocol named *Secure Messaging* (SM) which is defined in ISO/IEC 7816-4. ISO/IEC 7816-4 is a standard for inter-industry message exchange between smart cards and external interfaces that also defines the APDU message structure. APDU messages are exchanged in command-response pairs which have a header and an optional body part. The SM standard defines a separate BER-TLV[4] coded format for encoding command and response APDUs for secure transmission. It also states types of algorithms to be used for confidentiality and authentication. The choice of specific algorithms and parameters are implementation dependent.

Table III. Notation of the variables and primitives that are different in the EC variant compared to our adapted version of SRP-6a (cf. Table I)

G	The EC base point
V	The password verifier calculated from $x \cdot G$ (x is computed as in the SRP-6a variant with $H(s, P)$)
u	Random scrambling parameter, publicly revealed. Only the x-coordinate of the public key points Q_A and Q_B are used $\rightarrow u = H(X(Q_A), X(Q_B))$
E	Pseudo random EC point which is generated using the derivation primitive *REDP* and the x-coordinate of the password verifier V
REDP(·)	The Random Element Derivation Primitive used for generating the pseudo random EC point E
$X(\cdot)$	Primitive to get the x-coordinate of an EC point
d_A d_B	Ephemeral private keys, generated randomly and not publicly revealed
Q_A, Q_B	Elliptic curve public key points

As pointed out in the introduction section, the GP-secure channel is managed by a security domain of an applet. To establish application- to applet-level secure messaging, the protocol should be implemented on applet level. We implement the ISO/IEC 7816-4 secure messaging based on ETSI TS 102 176-2. This specification defines a set of algorithms and protocols for constructing secure channels between off-card applications and signature creating devices. The secure messaging construction provided in this document is ISO/IEC 7816-4 compliant. The changes we make for our implementation are related to using newer versions of algorithms than used in that specification.

5.1 Confidentiality

According to ISO/IEC 7816-4, confidential command and response APDUs are exchanged using specific BER-TLV data objects protected by a suitable encryption algorithm. ETSI TS 102 176-2 specifies two algorithms for confidentiality: 3DES and AES-128 bit in *Cipher Block Chaining* (CBC) mode.

For our implementation, we use AES-256 in CBC mode with a random initialization vector (IV), as this is supported by the Java Card 2.2 crypto API and provides stronger confidentiality. The random IV is generated on the card and is shared with the off-card application during the key agreement phase together with the user's salt in Step 2 of the protocol (Scheme 2). Padding is performed according to ISO/IEC 7816-4 with mandatory byte value *0x80* followed by zeros to fill the block.

5.2 Message authentication (MAC)

As in the previous case, a BER-TLV data object is defined for exchanging message authentication codes for both command and response APDUs. The MAC data object for a command APDU is constructed from header bytes, data objects containing encrypted information and the expected response length. The MAC data object for the response APDU is constructed from data objects containing an encrypted response and the status word. In addition, both request and response APDU MAC computations include a counter variable which is incremented for every transmission. This prevents replay attacks on both off-card application and applet side. ETSI TS 102 176-2 specifies CBC-MAC with 3DES and AES as algorithms for computing MACs. In both cases, it encrypts the last block, i.e. the output of the normal CBC-MAC operation, with a different key to protect against attacks on variable length messages (cf. Bellare *et al.*, 2000).

In our implementation, we use AES-CMAC by Song *et al.* (2006) because it fixes security weaknesses of CBC-MAC (cf. Dworkin (2005)).

5.3 Key derivation

From the SRP key agreement, we obtain a 256-bit session key K. Because the secure messaging standard provides confidentiality and authentication, we need a different key for each purpose. The derivation of encryption and authentication keys from the shared secret is done in accordance with ANSI X9.63 (American National Standards Institute, 2001) using a cryptographic hash function and a counter. If K is the session key from the key agreement, H the secure hash function and c the counter variable; encryption and authentication keys are derived with:

$$K_{\text{Enc}} = H(K \| c) \quad \text{where} \quad c = 1$$

$$K_{\text{Auth}} = H(K \| c) \quad \text{where} \quad c = 2$$

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

We use SHA-256 as the hash function. The newly derived encryption and authentication key parameters serve only for one secure messaging session.

6. Implementation

6.1 Implementation of SRP-6a for key agreement

In recent years, the SRP protocol has been adopted by different standards (e.g. TLS-SRP by Taylor *et al.* (2007)), and it has also been implemented in many cryptographic libraries[5]. In contrast, there is no implementation of SRP for Java Cards (even in the latest Java Card specifications). However, the Java Card environment supports the execution of SRP within hardware even though the explicit operations in the API are missing. In our implementation, we exploit the Java Card 2.2 crypto API to perform server-side SRP computation in a reasonable amount of time. Our implementation supports SRP prime modulus sizes of 1,024, 1,536 and 2,048 bits. For the evaluation of the protocol, we also included 512- and 768-bit versions. According to recommendations by the European Network of Excellence in Cryptology II (2012), the modulus in RSA or Diffie-Hellman should be at least 1,248-bit long to get 80-bit security. Therefore, we recommend using 1,536- and 2048-bit implementations.

The important building blocks of the Java Card protocol implementation are discussed in more detail in the following:

- *Server-side static parameters (before Step 1 of protocol Scheme 2)*: On the server-side, the password verifier v, the multiplier parameter k and the product of the two (kv) remain constant as long as the user password is not changed. This is a performance advantage as the static values can be computed once during applet installation and stay the same until the user changes the password.
- *Server secret key generation (before Step 1 in protocol Scheme 2)*: For our implementation, we use a 256-bit random output from the smart cards secure random number generator. According to RFC 5054 by Taylor *et al.* (2007), the ephemeral secret parameters a and b should be at least 256-bits long. The advantage of sticking to this minimum recommended length is a better performance in modular exponentiation operations involving the secret parameters.
- *Public parameter computation (Step 1 in Scheme 2)*: The public key parameter is computed as $B = kv + g^b$. Because kv is already computed during applet initialization, the remaining operations are one modular addition and one modular exponentiation.
- *Shared secret computation (Step 4 in Scheme 2)*: The shared secret is computed as $S = (Av^u)^b$. The variable A is the public key of the client, u the random scrambling parameter and b the ephemeral secret value. This computation contains one modular multiplication and two modular exponentiations. The main difficulty in computing this operation on Java Card is that there is currently no support for modulo operations over big numbers. Since Java Card 2.2, a single class has been added to support big numbers which, however, does not support modular arithmetic operations. Moreover, it is included under optional packages and is not implemented by most Java Cards available on the market today. These limitations forced us to look for other options to perform operations on big numbers.

6.1.1 Big numbers and modular arithmetic. Our implementation for modulo operations over big numbers is motivated by previous scientific work of Sterckx *et al.* (2009);

Tews and Jacobs (2009) and Bichsel *et al.* (2009) – especially, by BigNat[6] (based on research by Tews and Jacobs (2009)), a generic open-source library for big numbers which supports modulo operations. Because of memory limitations, we only implement specific big number operations tailored to our purpose:

- *Big numbers are handled with byte arrays*: While it is also possible to use arrays of short data types, using byte arrays is more convenient and can be used by native APIs with no need for conversion.
- Big number modular addition and subtraction can be computed efficiently on a Java Card Virtual Machine using basic subtraction and addition in base-256 encoded numbers. For example, with our implementation, a single modular addition and subtraction of 1,024 bit numbers on JCOP 2.4.1 smart cards takes an average of 27 or 54 ms, depending on the result being less or greater than modulus n.
- *Big number modular exponentiation*: The suitable way for big number modular multiplication as explained by Sterckx *et al.* (2009) and Tews and Jacobs (2009) is to leverage the Java Card RSA crypto API which is accelerated by a cryptographic co-processor. Tews and Jacobs (2009) state that a pure Java Card implementation for multiplication of 2,048-bit long numbers takes about 64 seconds. From such a performance, we can conclude that an implementation of SRP without hardware acceleration is impractical. The usage of hardware for big number modulo operations is made possible by using RSA public key encryption without padding. The Java Card crypto API for RSA supports setting the plain text parameter m and the public exponent e of the RSA encryption $c \equiv m^e \pmod{n}$. By simply using our exponent as exponent of the RSA public key (e) and a base padded with leading zeros as the plain text (m), the cipher text we get from encryption with the public key is the result of a modular exponentiation. With the methods mentioned above the computation for a 1,024-bit public parameter B completes in less than 150 ms on JCOP 2.4.1 smart cards.
- Finally, modular multiplications are converted to modular exponentiations using the binomial theorem, as discussed by Sterckx *et al.* (2009) and Tews and Jacobs (2009). This reduces the equation to modular additions, subtractions and squaring, which can be computed efficiently on the cryptographic co-processor of the smart card:

$$x \cdot y = \frac{((x+y)^2 - x^2 - y^2)}{2}$$

  This operation requires three modular exponentiations (which can be done with RSA public key encryption as mentioned in the previous bullet point), 1-4 additions, 2-3 subtractions and one right shift for the division by 2. The number of additions and subtractions varies if single results are out of the modulus range. This also influences the overall time (e.g. on JCOP 2.4.1 a single 1,024-bit multiplication takes between 130 and 230 ms). This method is used for the computation of the shared secret S (step 4 in Scheme 2) and kv (during installation time).

6.1.2 Implementation notes. As of the latest Java Card specification (Java Card 3.0), there is no support for transient RSA public keys. Transient RSA keys are included in JCOP-specific extension API. However, to support as many devices as possible, we recommend to use standard Java Card facilities. The drawback of using static RSA keys,

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

which reside in persistent memory, is that read and write operations on EEPROM are very slow compared to operations on transient memory. One solution for this is to initialize two RSA public key objects, one with fixed exponent 2 for squaring and another one for 32-byte-long ephemeral exponents u and b. Using separate public key objects for squaring reduces the amount of write operations to the EEPROM for each authentication cycle. In total, we currently need two EEPROM rewrites per cycle.

6.2 Elliptic curve SRP implementation

As it is the case for SRP-6a, the EC variant of SRP is also not implemented on smart cards. This means that to implement this EC variant, we also need to apply the same or similar methods as described in the Section 6.1. Hence, in this section, we will only discuss the different and additional methods that are need to realize the EC variant of SRP.

6.2.1 Elliptic curve primitives. As we can see in Steps 1 and 6 of Scheme 3, the EC version of SRP requires point multiplication and addition (e.g. $Q_B = d_B \cdot G + E$) to compute the shared secret. Fournaris and Koufopavlou (2008) and Anoop (2007) show that these operations have to follow the *group law* and have a considerable computational cost if they are not implemented in hardware.

On Java Card, point multiplications can be done using the API for the Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol by Koblitz (1987). In the usual procedure of ECDH, both parties multiply the public key of the counterpart with the own private key to get the common shared secret S (e.g. $S = d_A \cdot Q_B$, where Q_B is the public key of the counterpart and d_A the own private key). This step in the protocol can be exploited to multiply any EC point on the curve with any natural number.

More difficult is performing an EC point addition, as the latest release of Java Card (version 3.0) still does not provide an API for that. A pure software solution would require to implement an algorithm that finds the modular multiplicative inverse of big numbers, which is considered computationally complex, as stated by Anoop (2007). This means that the only reasonable way for EC point addition is to use proprietary classes by the smart card manufacturer (e.g. JCOP allows addition of 192-bit EC points since version 2.4.1). The downside of this is that the implementation is therefore not platform independent. Still, to the best of our knowledge, there is currently no other feasible way to perform EC point addition on smart cards.

6.2.2 Big numbers and modular arithmetic. Operations on big numbers in the EC variant are performed in the same manner as described in Section 6.1.1. Hence, modular exponentiations are performed by exploiting the RSA encryption and modular multiplications are simplified to an equation consisting of modular additions, subtractions, squaring and division by two.

Although the protocol for the EC variant in Scheme 3 does not show any appearances of modular exponentiations and multiplications, we still need these methods for the REDP function. The REDP function generates a pseudo random EC point and needs to, among other things, solve the equation $y = x[3] + ax + b$ (for curves over the prime finite field $\mathbb{F}_p$). This equation requires modular exponentiation, addition as well as multiplication. To make the equation easier to solve on the smart card, it can also be written as $y = (x[2] + a) \cdot x + b$. This leaves us an equation consisting of modular squaring, multiplication and addition (note that multiplications can be converted again as described in Section 6.1.1).

The difficulty in modular squaring and modular multiplication for the EC variant of the protocol results from the smaller modulo sizes in EC cryptography. The current Java Card API only allows moduli higher than 512 bits for RSA encryption/decryption (i.e. keys of length 512, 736, 768, etc. bits). Nevertheless, with EC cryptography, we usually need to operate with values lower than 512 bits. Currently suggested key lengths (and therefore also the moduli) are 192, 224, 256, 384 or 521 bits. Consequently, there are two possible approaches:

(1) The easiest solution would be to use the smallest possible key length for the RSA cipher object (i.e. 512 bits) and initialize it with the EC modulus with leading zeros (e.g. [[320-bits zeros] [192-bits EC modulus]]). However, due to the possible security risks, the usage of leading zeros in the modulus of the RSA cipher object is not allowed by the Java Card framework. Hence, this simple approach is not applicable.

(2) An alternative method to use the hardware-supported RSA encryption for modular squaring is the usage of trailing zeros instead of leading zeros for the modulus. To get a correct result, we then have to apply trailing zeros to the value that should be squared. When squaring a value with trailing zeros, the number of zeros at the end are doubled. For example: squaring the decimal number 100 (2 trailing zeros) results into 10,000 (four tailing zeros). To use this for our purpose, we therefore have to add the half amount of zeros to the value that should be squared compared to the zeros added to the modulus.

An example with decimal numbers: Let us assume we have the equation (4)[2] mod 7 = 2. If we add two trailing zeros to the modulus now, we would also have to add two zeros to the result and one zero to the value which should be squared → 40[2] mod 700 = 200. Instead of 2, the result of the second equation is 200. Nevertheless, removing the same amount of trailing zeros that have been added to the modulus (two) to get the result of the initial equation is trivial now.

In our case, this means that we have to compute the amount of required trailing zeros first (e.g. for a 192-bit EC key inside a 512-bit RSA cipher object, we have $512 - 192 = 320$ zero bits). The modulus of the RSA object, which is used for modular squaring, is then specified with the EC modulus and this computed amount of trailing zeros. For example, the modulus of a 512 RSA object for a 192-bit EC keys would be specified as:

[[192 bits EC modulus] [320 bits zeros]]

Dividing the amount of used zeros (320) by two gives us the amount of zeros that needs to be added to the value that has to be squared:

[[value to be squared] [160 bits zeros]]

By using this value for our RSA encryption object, the squaring will be performed with hardware support and we get the correct result with 320 bit trailing zeros.

6.3 Secure messaging implementation

In our secure messaging implementation, the operations needed to wrap and unwrap APDUs have a small memory overhead because of paddings and intermediate

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

operations. However, no extra memory is needed as the transient memory allocated for the key agreement phase is large enough to be reused for operations in secure messaging.

The encryption operation is straightforward as AES-CBC-256 is supported in Java Card 2.2. For the MAC operation, the algorithm AES-CMAC-128 is not included in the Java Card standard 2.2. However, cipher-based MACs could be implemented efficiently in the Java Card environment if their underlying cipher algorithm can be executed with hardware support. For AES-CMAC-128, the underlying block cipher AES-CBC-128 is supported by the Java Card standard.

Additionally, the Java Card standard 2.2 supports AES-CBC-MAC signatures. The internal operations of AES-CMAC and AES-CBC-MAC are similar, except for the sub-key derivation and XORing of the last input block employed by CMAC. After this XOR operation of the last block with the corresponding sub-key, the remaining operation of AES-CMAC is the same as for AES-CBC-MAC (Song *et al.*, 2006).

7. Performance evaluation

7.1 Secure channel protocol establishment

To evaluate the secure channel protocol establishment for the discrete logarithm variant SRP-6a as well as the EC variant, we established two test scenarios. For the first test scenario, we used a DeviceFidelity microSD SE (credenSE 2.8J), a Samsung Galaxy S3 with the *SuperSmile ROM*[7] and Open Mobile API for accessing the secure element on the card. We made the measurements using *System.nanoTime*() before sending and after receiving the APDU. For the second case, we used a JCOP 2.4.1 external smart card with contact-less interface to give insight on performance differences between different tamper-resistant hardware variants. Measurements were taken using NetBeans application profiler[8] on the client-side. We took 100 measurements for both protocol variants (SRP-6a and the elliptic curve variant) and test cases.

7.1.1 SRP-6a with minor adaptations. In our SRP-6a-based password-authenticated key agreement protocol implementation, we use two rounds of message exchange as stated by Wu (2002). The first round is the key agreement phase where the two parties generate the shared secret S and the session key K. This includes operation Steps 1-5 shown in Scheme 2. The second message exchange round is the verification phase which is shown in Steps 6 and 7. In this section, we show the performance of the server-side computations of these two phases using two Java Card variants and different modulus sizes. Although our main focus is the performance evaluation of the protocol in Java Card applets, we also include the data transfer time between the client application and the Java Cards. This should give a better insight on the actual use cases of the protocol implementation. The results and the evaluation of the SRP-6a protocol establishment are split in four parts:

(1) *Key agreement performance*: This evaluation show the request-response time required from sending the public parameter A in the client application until receiving the public parameter B and the user's salt s (Steps 1 to 5 in Scheme 2). The length of the user salt is 16 bytes, while the public key parameter has a length equal to the prime modulus used in the test.

    From the performance results in Table IV, we observe that there is a considerable difference between the minimum and maximum time required to complete the operations. This time difference is introduced by extra modular additions and subtractions needed to get the right modulus result when there is

Test hardware	Agt.	Verif.	Comp.
MicroSD SE			
min [ms]	2,744	341	3,170
max [ms]	3,318	377	3,787
median [ms]	3,036	356	3,496
External smart card			
min [ms]	1,204	89	1,498
max [ms]	1,477	98	1,960
median [ms]	1,293	90	1,600

Note: Computation times for card-side key agreement phase (agt.), key verification phase (verif.) and complete (comp.) protocol run-time (including data transfer time)

Table IV. SRP-6a with 2048 bit modulo performance

an overflow in other modular additions and subtractions implemented in software. From a cryptanalysis point of view this could give an attacker some information about the ephemeral secret key and verifier parameters used in the SRP protocol. Such issues can be solved by introducing dummy addition and subtraction operations which compensate the time difference. The time difference in the two test scenarios (external smart card and microSD SE) is caused by the differences in hardware implementation and communication channel. Due to the usage of standard file system IO for exchanging data, the microSD SE is much slower in transmitting data (cf. time measurements by Hölzl *et al.* (2013)).

(2) *Performance of verification*: As it is shown in Steps 6 and 7 of Scheme 2, the verification of the shared secret consists of one message exchange of the verifiers M_1 and M_2. For our implementation, we use a SHA-256 hash function, so that both M_1 and M_2 are 32 bytes. However, because the shared secret S is also included in the computation of M_1 and M_2, the performance depends on the length of S which is equal to the size of the prime modulus n.

In many use cases, the performance of the verification stage is close to the actual time that a user has to wait before the secure communication starts. This is because the time intensive key agreement phase can be started in the background while the user enters the PIN/password. As shown in the results of the evaluation in Table IV, the verification operation takes a maximum time of 377 ms (microSD SE) and 98 ms (external smart card).

(3) *Complete secure channel establishment*: This case includes the complete protocol running time required by the client- and server-side. For the client-side computations, we use the Bouncy Castle crypto API[9].

(4) *Evaluation of prime modulus sizes*: In Figure 2, we visualize the influence of different prime modulus sizes to the required computation time. Our protocol supports prime modulus sizes of 1,024, 1,536 and 2048 bit. For the purpose of performance evaluation, we also included 512 and 768 bit though we recommend to use 1,536- and 2,048-bit versions. Higher prime modulus sizes of above 2048 bit are currently not supported due to the limitation of RSA operations in the current Java Card standard.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

The results in Figure 2 show that the key agreement time of the protocol is significantly influenced by the prime modulus size. At the same time, the verification phase of both Java Card variants is not varying much. As already described, the big advantage of this is the minimized time the user has to wait after entering PIN/password.

7.1.2 Elliptic curve variant of SRP. For the measurements of the protocol establishment of the EC variant, we used the EC parameters of the Koblitz curve secp192k1. This curve is recommended by Certicom Research (2010), in conjunction with the Standards for Efficient Cryptography Group (SECG), and defines domain parameters over the prime finite field and supports 192-bit keys. As listed in Table II (cf. security level listing of the SECG, 2010), this provides a security level of 96 bits or an equivalent strength as RSA/DSA with 1,536-bit keys. Note that most organizations providing EC domain parameters currently suggest key sizes of at least 224 bits[10] (equivalent security level as 2024 RSA keys). However, the Java Card used in these tests runs with version 2.2, which does not support curves with key sizes higher than 192 bit.

Table V lists the results of the elliptic curve SRP measurements in both test scenarios (microSD SE and external smart card). Compared to the discrete logarithm variant with

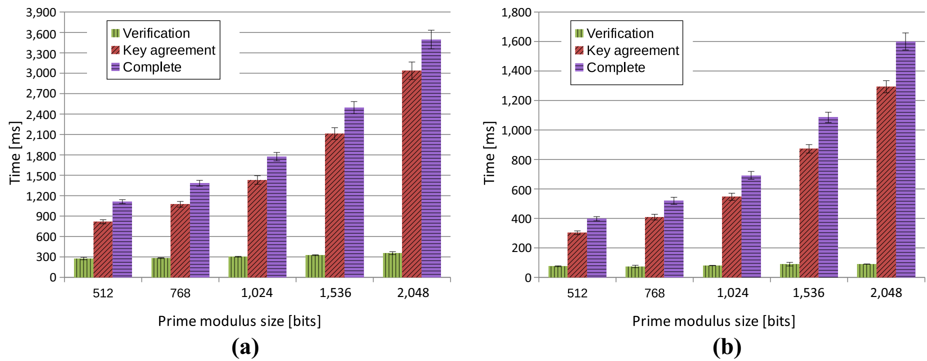


Figure 2. Performance evaluation of the discrete logarithm variant of SRP (SRP-6a) with different modulus sizes and standard deviation on both Java Card variants

Notes: (a) Performance on JCOP 2.4.1 microSD SE; (b) secure channel performance on JCOP 2.4.1 external smart card

Table V. Performance of the elliptic curve SRP with the Koblitz curve secp192k1

Test hardware	Agt.	Verif.	Comp.
MicroSD SE			
min [ms]	988	262	1,401
max [ms]	1,051	315	1,645
median [ms]	1,021	286	1,572
External smart card			
min [ms]	451	60	521
max [ms]	462	68	598
median [ms]	457	60	528

Note: Card-side computation time for key agreement phase (agt.), key verification phase (verif.) and complete (comp.) protocol run-time

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

the equivalent security level (cf. 1,536-bit prime modulus size in Figure 2), the median complete protocol runtime is significantly lower (~1 second less for the microSD and ~550 milliseconds less for the external smart card). This difference is especially high in the key agreement phase (~1 second for microSD and ~400 ms for the external card) but minor in the verification phase (~35 ms for both scenarios). This is because the verification phase did not change in the elliptic curve variant. Still, there are minor differences due to the smaller key sizes that needs to be hashed during verification.

It is also recognizable that the difference between minimum and maximum values in the elliptic curve variant is smaller than in the SRP-6a variant. The reason for this are the missing modular additions and subtractions in the protocol schema of the EC variant.

7.2 Secure messaging performance

In this section, we analyse the performance of our secure messaging implementation with both Java Card variants. To perform these tests, the client-side wraps random data in a secure request APDU object and sends this object. On the Java Card side, a dummy applet receives the incoming secure request APDU and unwraps it (performs MAC verification and decryption) to get the random data from the client. Then these random data are encoded inside a secure response APDU object and is sent back to the client. Table VI shows the median, minimum and maximum results for different packet sizes. The values for the external smart card vary between 90 and 107 ms, with a data rate reaching 1.19 kB/s. The microSD only achieves a median data rate of 186 B/s.

7.3 Memory optimization

In addition to slight adaptations, discussed in Section 6.3, we optimize our implementation by considering performance, memory and security trade-offs:

- Using the APDU buffer for performing and storing intermediate operations and public values.
- Using static memory offsets for different sections instead of allocating several smaller areas to save memory.
- Sharing (reuse) of transient memory between key agreement and secure messaging implementation.

The current implementation with 2,048-bit prime uses 1,040 bytes for byte arrays in persistent (EEPROM) and 834 bytes for byte arrays in transient (RAM) memory.

Test hardware	Data size in bytes 16	32	64	128
MicroSD SE				
min [ms]	322	375	463	655
max [ms]	360	415	534	738
median [ms]	337	384	494	687
External smart card				
min [ms]	88	90	95	105
max [ms]	97	103	107	143
median [ms]	90	92	97	107

Table VI. Card-side secure messaging performance (including data transfer time)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

8. Conclusion

In this paper, we propose an efficient way of implementing a secure communication between Java Card applets and off-card applications in a mutually authenticated secure channel based on the SRP protocol and a standard authenticated encryption scheme. Although the Java Card environment is equipped with the necessary hardware for computation of modulo operations in SRP, limitations in Java Card APIs on accessing the cryptographic co-processors make it challenging to implement SRP with acceptable performance.

By exploiting the RSA encryption API as well as the elliptic curve Diffie-Hellman API and minor adaptations to the protocol, we show that it is possible to implement the SRP-6a and SRP-5 server-side in a Java Card applet with reasonable computation time. For our implementation of the discrete logarithm variant (SRP-6a) with a 2,048-bit-long prime modulus, the complete protocol runs in less than 2 seconds for the smart card and less than 4 seconds for the secure element tests. Moreover, our use case requires that the user only has to wait for the much faster verification phase (i.e. less than 100 ms for the smart card and 400 ms for the secure element), as the time intensive key agreement phase runs simultaneously with the password/PIN entry. With the elliptic curve variant of the protocol, we even achieved a complete protocol run within 600 ms for the smart card variant and 1,700 ms for the secure element (user waiting time is below 70 and 350 ms, respectively). Finally, we also provide an applet-level implementation for the ISO/IEC 7816-4 secure messaging standard. The source code of the whole implementation is available under an open-source license[11].

Notes

1. www.globalplatform.org/
2. www.nngroup.com/articles/response-times-3-important-limits/
3. TrustZone white paper at: www.arm.com/products/processors/technologies/trustzone/index.php
4. Standard for *Tag-Length-Value* encoded data structures.
5. http://en.wikipedia.org/wiki/Secure_Remote_Password_protocol\#Real_world_implementations
6. www.sos.cs.ru.nl/ovchip/
7. http://usmile.at/downloads
8. https://profiler.netbeans.org/
9. www.bouncycastle.org/
10. Overview of currently suggested key sizes by different organizations at: www.keylength.com
11. https://github.com/mobilesec/secure-channel-srp6a-android-lib, https://github.com/mobilesec/secure-channel-srp6a-applet and https://github.com/mobilesec/secure-channel-ec-srp-applet

References

American National Standards Institute (2001), *American National Standard for Financial Service X9.63-2001: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American Bankers Association, available at: http://books.google.at/books?id=vvzkPAAACAAJ

Anoop, M.S. (2007), "Elliptic curve cryptography", *An Implementation Guide*, available at: www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Barker, E., Barker, W., Burr, W., Polk, W., Smid, M., Gallagher, P.D. and For, U.S. (2012), *NIST Special Publication 800-57 Recommendation for Key Management – Part 1: General*, NIST.

Bellare, M., Canetti, R. and Krawczyk, H. (1996), "Keying hash functions for message authentication", *Advances in Cryptology-CRYPTO*, pp. 1-15, available at: http://link.springer.com/chapter/10.1007/3-540-68697-5_1

Bellare, M., Kilian, J. and Rogaway, P. (2000), "The security of the cipher block chaining message authentication code", *Journal of Computer and System Sciences*, Vol. 61 No. 3, pp. 362-399, available at: http://dx.doi.org/10.1006/jcss.1999.1694.

Bellare, M. and Rogaway, P. (2000), "The AuthA protocol for password-based authenticated key exchange", *IEEE P1363*, pp. 136-143.

Bellovin, S. and Merritt, M. (1992), "Encrypted key exchange: password-based protocols secure against dictionary attacks", *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, pp. 72-84.

Ben-Asher, N., Kirschnick, N., Sieger, H., Meyer, J., Ben-Oved, A. and Möller, S. (2011), "On the need for different security methods on mobile phones", *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM, New York, NY, pp. 465-473, available at: http://doi.acm.org/10.1145/2037373.2037442.

Bichsel, P., Camenisch, J., Groß, T. and Shoup, V. (2009), "Anonymous credentials on a standard Java Card", *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ACM, pp. 600-610, available at: http://doi.acm.org/10.1145/1653662.1653734.

Brands, S.A. (2000), *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*, MIT Press.

Brickell, E., Camenisch, J. and Chen, L. (2004), "Direct anonymous attestation", *Proceedings of the 11th ACM conference on Computer and Communications Security*, ACM, New York, NY, pp. 132-145, available at: http://doi.acm.org/10.1145/1030083.1030103.

Certicom Research (2010), "Sec 2: recommended elliptic curve domain parameters", Technical Report, available at: www.secg.org/sec2-v2.pdf

Chin, E., Felt, A.P., Greenwood, K. and Wagner, D. (2011), "Analyzing inter-application communication in android", *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ACM, New York, NY, pp. 239-252, available at: http://doi.acm.org/10.1145/1999995.2000018.

Diffie, W. and Hellman, M. (1976), "New directions in cryptography", *IEEE Transactions on Information Theory*, Vol. 22 No. 6, pp. 644-654.

Dworkin, M.J. (2005), "SP 800-38B. Recommendation for block cipher modes of operation: the CMAC mode for authentication", Technical Report, National Institute of Standards & Technology, Gaithersburg, MD.

European Network of Excellence in Cryptology II (2012), "ECRYPT II yearly report on algorithms and keysizes", pp. 29-34.

Fournaris, A. and Koufopavlou, O. (2008), "Creating an elliptic curve arithmetic unit for use in elliptic curve cryptography", *IEEE International Conference on Emerging Technologies and Factory Automation*, IEEE, Hamburg, pp. 1457-1464.

Gayoso Martinez, V., Sanchez Avila, C., Espinosa Garcia, J. and Hernandez Encinas, L. (2005), "Elliptic curve cryptography: Java implementation issues", *39th Annual 2005 International Carnahan Conference on Security Technology*, IEEE, pp. 238-241.

GlobalPlatform (2009), "Secure channel protocol – GlobalPlatform card specification v2.2 – Amendment D".

Han, J.-H., Kim, Y.-J., Jun, S.-I., Chung, K.-I. and Seo, C.-H. (2002), "Implementation of ECC/ECDSA cryptography algorithms based on Java card", *Proceedings of 22nd International Conference on Distributed Computing Systems Workshops*, pp. 272-276.

Hancke, G. (2005), "A practical relay attack on ISO 14443 proximity cards", Technical Report.

Hao, F. and Ryan, P.Y.A. (2011), "Password authenticated key exchange by juggling", *Proceedings of the 16th International Conference on Security Protocols, Springer-Verlag, Berlin, Heidelberg*, pp. 159-171, available at: http://dl.acm.org/citation.cfm?id=2022815.2022838

Höbarth, S. and Mayrhofer, R. (2011), "A framework for on-device privilege escalation exploit execution on android", *Proceedings of IWSSI/SPMU*.

Hölzl, M., Mayrhofer, R. and Roland, M. (2013), "Requirements analysis for an open ecosystem for embedded tamper resistant hardware on mobile devices", *Proceedings of International Conference on Advances in Mobile Computing and Multimedia, ACM, Vienna*.

IEEE Computer Society (2009), "IEEE standard specifications for password-based public-key cryptographic techniques", IEEE Std 1363.2-2008, pp. 1-127.

Jablon, D.P. and Ma, W. (1996), "Strong password-only authenticated key exchange", *ACM SIGCOMM Computer Communication Review*, Vol. 26 No. 5, pp. 5-26, available at: http://doi.acm.org/10.1145/242896.242897

Khan, S., Nauman, M., Othman, A. and Musa, S. (2012), "How secure is your smartphone: an analysis of smartphone security mechanisms", *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, IEEE, Kuala Lumpur, pp. 76-81.

Koblitz, N. (1987), "Elliptic curve cryptosystems", *Mathematics of Computation*, Vol. 48 No. 177, pp. 203-209, available at: www.ams.org/mcom/1987-48-177/S0025-5718-1987-0866109-5/

La Polla, M., Martinelli, F. and Sgandurra, D. (2013), "A survey on security for mobile devices", *IEEE Communications Surveys Tutorials*, Vol. 15 No. 1, pp. 446-471.

Landman, M. (2010), "Managing smart phone security risks", *Information Security Curriculum Development Conference*, ACM, pp. 145-155, available at: http://doi.acm.org/10.1145/1940941.1940971

Lochter, M. and Merkle, J. (2010), "Elliptic curve cryptography (ECC) brainpool standard curves and curve generation", *RFC* 5639, available at: www.ietf.org/rfc/rfc5639.txt

Lucks, S. (1997), "Open key exchange: how to defeat dictionary attacks without encrypting public keys", *Proceedings of the Security Protocols Workshop, LNCS 1361*, *Springer Berlin Heidelberg*, pp. 79-90, available at: http://link.springer.com/chapter/10.1007/BFb0028161

Mantoro, T. and Milisic, A. (2010), "Smart card authentication for internet applications using NFC enabled phone", *International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, IEEE, Jakarta, pp. D13-D18.

Mayrhofer, R. (2014), "An architecture for secure mobile devices", *Security and Communication Networks*, Vol. 8 No. 10, available at: http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1939-0122

Roland, M., Langer, J. and Scharinger, J. (2012), "Practical attack scenarios on secure element-enabled mobile devices", *4th International Workshop on Near Field Communication (NFC)*, IEEE, Helsinki, pp. 19-24.

Ruiz-Martinez, A., Canovas, O. and Gomez-Skarmeta, A. (2007), "Smartcard-based e-coin for electronic payments on the (mobile) internet", *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, IEEE, Shanghai, pp. 361-368.

Song, J., Poovendran, R., Lee, J. and Iwata, T. (2006), "The AES-CMAC algorithm", RFC 4493 (Informational), available at: http://tools.ietf.org/html/rfc4493

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)

Sterckx, M., Gierlichs, B., Preneel, B. and Verbauwhede, I. (2009), "Efficient implementation of anonymous credentials on java card smart cards", *Information Forensics and Security*, pp. 106-110, available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5386474

Taylor, D., Wu, T., Mavrogiannopoulos, N. and Perrin, T. (2007), "Using the secure remote password (SRP) protocol for TLS authentication", *RFC* 5054, available at: www.ietf.org/rfc/rfc5054.txt

Tews, H. and Jacobs, B. (2009), "Performance issues of selective disclosure and blinded issuing protocols on java card", *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, Springer, pp. 95-111, available at: http://link.springer.com/chapter/10.1007/978-3-642-03944-7_8

Wu, T. (1998), "The secure remote password protocol", *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, Detroit, MI*, pp. 97-111.

Wu, T. (2002), "SRP-6: improvements and refinements to the secure remote password protocol", available at: http://srp.stanford.edu/

Corresponding author
Michael Hölzl can be contacted at: michael.hoelzl_1@jku.at

For instructions on how to order reprints of this article, please visit our website:
www.emeraldgrouppublishing.com/licensing/reprints.htm
Or contact us for further details: **permissions@emeraldinsight.com**

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 22:28 07 November 2016 (PT)