Finding Vulnerabilities in Low-Level Protocols

Nordine Saadouni

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2017

Abstract

Basic example of an abstract (this will be changed)

Smart cards are used commercially and within industry for authentication, encryption, decryption, signing and verifying data. This paper aims to look into how the smart card interacts with an application at the lower level. PKCS#11 (public key cryptography system?) is the standard that is implemented at the higher level and then broken down into command/response pairs sent as APDU traffic to and from the smart card. It is the APDU low-level protocol that will be analysed to see if any vulnerabilities are present with regard to the smart cards tested.

Acknowledgements

Acknowledgements go here.

Table of Contents

Chapter 1

Introduction

Smart-cards are formally known as integrated circuit cards (ICC), and are universally thought to be secure, tamper-resistant devices. They store and process, cryptographic keys, authentication and user sensitive data. They are utilised to preform operations where confidentiality, data integrity and authentication are key to the security of a system.

Smart-cards offer what seems to be more secure methods for using cryptographic operations. (And should still provide the same level of security that would be offered to un-compromised systems, compared to those that are compromised by an attacker). This is partly due to the fact that the majority of modern smart-cards have their own on-board micro-controller, to allow all of these operations to take place on the card itself, with keys that are unknown to the outside world and stored securely on the device. Meaning the only actor that should be able to preform such operations would need to be in possession of the smart-card and the PIN/password. In many industries, for applications such as, banking/ payment systems, telecommunications, healthcare and public sector transport, smart-cards are used due to the security they are believed to provide.

The most common API (application programming interface) that is used to communicate with smart-cards is the RSA defined PKCS#11 (Public Key Cryptography Standard). Also known as 'Cryptoki' (cryptographic token interface, pronounced as 'crypto-key'). The standard defines a platform-independent API to smart-cards and hardware security modules (HSM). PKCS#11 originated from RSA security, but has since been placed into the hands of OASIS PKCS#11 Technical Committee to continue its work (since 2013). [reference wikipedia PKCS#11].

In the previous 10-15 years, literature has shown a great deal of research into the examination of the PKCS#11 API, and the security it provides.

Yet little attention has been paid to that of the lower-level communication (command-response pairs), in which the higher level API is broken down into. It is this area that we wish to research within this paper. The reasoning is simple. If we cannot trust the security of the low-level commands that implement the high level API functions, then in turn we cannot trust the security of the high level API. This is analogous to C code being complied down to binary data to be operated on by the CPU. The addition of two integers cannot be considered correct in C, unless the corresponding binary instructions sent to the CPU are correct as well.

The research of the low-level communication will take two forms.

1. An analysis of the raw communication between PC and Smart-card for all API function calls
2.

Before we move into the above analysis, supporting material must be introduced. This the rest of this paper will be organized as follows.

Chapter 2

Background

In this chapter we give background knowledge on two essential standards that are required for the use of the contact smart-card we analyse in this paper. These two standards are PKCS#11 and ISO 7816.

2.1 PKCS#11

This is the API that each card manufacture implements themselves.

2.1.1 Key Object

2.1.2 Attributes

```
template = (\\
(CKA\_CLASS, LowLevel.CKO\_SECRET\_KEY), [private, public, data, cert (X.5
(CKA\_KEY_TYPE, LowLevel.CKK_DES), [AES,DES,RSA,ECC]
(CKA\_LABEL, label), [name]
(CKA\_ID, chr(id)), [id]
(CKA\_PRIVATE, True), [requires authentication]
(CKA\_SENSITIVE, True), [cannot be extracted unencrypted]
(CKA\_ENCRYPT, True), [can be used for encrypting data]
(CKA\_DECRYPT, True), [can be used for decrypting data]
(CKA\_SIGN, True), [[can be used for signing data]
(CKA\_VERIFY, True), [can be used for verifiying data]
(CKA\_TOKEN, True), [can be stored permanently on the device]
(CKA\_WRAP, True), [can encrypt a key to be extracted]
(CKA\_UNWRAP, True), [can decrypt an encrypted key]
(CKA\_EXTRACTABLE, False)) [is allowed to be extracted from the device]
```

2.1.3 Most Common Functions

2.2 ISO 7816

This is the international standards organization that defines the low level communication protocol between smartcards and the API on the computer.

2.2.1 Command Structure

2.2.2 Response Structure

2.2.3 Inter-Industry/ Proprietary

2.2.4 Most Common Commands

CLA	INS	P1	P2	Lc	Data Field	Le	Description
00	-	-	-	-	-	-	Inter-industry (II)
80	-	-	-	-	-	-	Proprietary (P)
Xc	-	-	-	-	-	-	Secure Messaging (SM)
00	84	00	00	-	-	L	(II) Get Challenge [# bytes = L]
00	b0	X	X	-	-	L	(II) Read Binary [# bytes = L]
00	c0	00	00	-	-	L	(II) Get Remaining Bytes
00	d6	X	X	L	X	-	(II) Update Binary
00	e0	X	X	L	X	-	(II) Create Binary
00	47	00	00	L	params	-	(II) Generate RSA KeyPair
00	e4	00	00	00	-	-	(II) Delete File
00	2a	82	0e	L	X	00	(II) Encrypt Data
00	2a	80	0e	L	X	00	(II) Decrypt Data
00	2a	9e	12	L	X	00	(II) Sign Data
00	2a	80	0a	L	X	00	(II) Unwrap Key (RSA_PKCS)
80	a4	08	00	L	FL	-	(P) Select File [FL = file location; L = len(fl)]
80	a4	08	0c	L	FL	-	(P) Read File Control Parameters
80	20	00	00	10	R	-	(P) Verify PIN [R = Response]
80	28	00	00	04	00 00 00 20	-	(P) Clear Security Status
80	30	01	00	00	-	-	(P) List Files
80	48	00	80	00	-	-	(P) Get Cards Public Key (genkey)
80	48	00	00	00	-	-	(P) Get Cards Public Key (genkeypair)
80	86	00	00	L	X	00	(P) Open Secure Messaging
80	86	ff	ff	-	-	-	(P) Close Secure Messaging
90	32	00	03	ff	X	-	Relocate Binary 256 bytes
80	32	00	03	L	X	-	Relocate Binary L bytes (changes file)

Table 2.1: Common APDU Commands

2.2.5 File Structure

Chapter 3

Cryptographic Operations

In this chapter we describe different cryptography standards, and give a brief overview of the terminology and maths used behind some of the types of cryptography. This will be used to aid our explainations in later chapters that regard the cards functionality and cryptographic operations used to attempt the secure transfer of sensitive information.

3.1 Hash Function

explain this in an overview term

3.2 Asymmetric Encryption

public and private keys for communicating over an insecure channel

3.2.1 RSA

Chinese Remainder Theorem RSA Keys

3.3 Symmetric Encryption

explain this in an overview term

3.3.1 DES

3.3.2 Triple DES

3.3.3 AES

3.3.4 ECB Mode

3.3.5 CBC Mode

3.4 Message Authentication Codes

explain what a message authentication code is, what its used for
(wikipedia have good diagrams and explanations of this!!)

3.4.1 Hash Based - Message Authentication Codes (HMAC)

3.4.2 Cryptographic Based - Message Authentication Codes (CMAC)

3.5 One Time Passwords

What is a one time password → what forms are there?

3.5.1 Hash Based - One Time Passwords (HOTP)

3.5.2 Time Based - One Time Passwords (TOTP)

Chapter 4

Tools

4.1 PCSC-lite

4.2 Virtual Smart-Card

frank something Explain how this creates a virtual smartcard reader This is then used to run API functions, and the commands are relayed to the actual card reader and therefore card.

4.3 Man in The Middle (MiTM)

Chapter 5

Related Work / Literature Review

This will be a brief chapter and will discuss all of the research I have conducted.
Mainly regarding PKCS#11 API attacks due to the small amount of literature that is available for APDU level attacks I shall also explain why some of the attacks are not able to be conducted on the particular card I am reviewing

Chapter 6

PKCS#11 Functions - APDU analysis

In this chapter we analyse the APDU traces of 9 functions from the API. C_sign and C_verify were not included in this analysis as from analysis of the traces from the previous work on this card it was clear that both of these functions operated in a similar manner to C_encrypt and C_decrypt. C_createObject was also omitted form this analysis, as the function is not used for security operations, and rather for the creation and storing of certificates and data. The analysis will include a step by step guide of how the functions are broken down into command response pairs at the APDU level, and also suggest methods for attacking the card using this knowledge.

All of the traces are provided in appendix B, and have been shortened to only include the parts which we deem to be the most important to the evaluation of how each function is broken down into APDU command response paris. However, the full traces for each function are given within the project directory. In the analysis to come we will refer to appendix B, with the corresponding command response pair in brackets to allow ease of locating the steps we analyse.

6.1 Initialization

The first process before anything can occur on the card is the initialization. This occurs as soon as a session is opened with the card. The process has the following steps:

1. Open the laser PKI file and select the applet (B.1 - command 1)
2. Open the file and send to the API the cards serial number (B.1 - command 4,5)

3. Open a second file and send to the API another serial number (B.1 - command 8,9)

None of these traces reveal sensitive information. But is required in order for the card to be able to operate and communicate with the API. The API is a generic library that works with many different hardware security modules created by the 'Athena smartcard solutions'. Hence this information is required by the library in order for it to operate correctly for the specific type of card.

6.2 C_login

The login function has 5 main parts. The API locates the file control parameters to check if the retry counter is greater than 0. In the APDU trace this is highlighted in bold and has a value of AA. From there the PIN file is selected, a challenge is asked for by the card and a response is calculated from the knowledge of the PIN and the challenge. Once this is completed the security status of the card is cleared. This is done because every security operation requires a new login by the API. While the user only logs in once, at the APDU layer this occurs many times. These steps are listed below:

1. Open file control parameters for the PIN file (B.2 - command 10)
2. Open the PIN file (B.2 command 11)
3. Request challenge from card (B.2 - command 12)
4. Verify PIN (B.2 - command 13)
5. Clear security Status (B.2 - command 20)

From evaluating multiple traces of the login, it is clear that the response calculated seems to have an element of randomness. From studying the traces alone, the method of calculating the response cannot be intuitively determined.

While no immediate sensitive information is revealed in this trace as the challenge-response algorithm used for verifying a users PIN hides its value, successfully reverse engineering the protocol will all an attacker to calculate the users PIN given successful login trace.

Furthermore, if an attacker can find a method for changing the file control parameter, it might be possible to reset the retry counter at the APDU level to allow an unlimited number of attempts of logging in.

6.3 C_findObject

1. Selects and opens ’cmapfile’. This stores all file locations for attributes of the keys (B.3 - command 32,33,34)
2. Based on those locations, finds and opens the attribute file for the des3 key (B.3 - command 36)
3. And also opens the last attribute file to determine there are no more keys (B.3 - command 38,39)

6.4 C_generateKey

1.

6.5 C_generateKeyPair

This saves the CRT instead of the private key. explain the structure

1.

6.6 C_destroyObject

1.

6.7 C_encrypt

1.

6.8 C_decrypt

1.

6.9 C_setAttribute

1.

6.10 C_unwrap

1.

Chapter 7

Attempts To Attack At the APDU Level

In this chapter we explain the design, implementation and results of the attacks we attempted at the APDU level. The two main attacks we focus on are reverse engineering the PIN authentication protocol, and reverse engineering the secure messaging protocol. The motivations behind this come from literature stated in chapter X.

The smartcard we analyse uses a challenge-response algorithm to prevent the PIN being sent over in plain text. This is an improvement upon other card vendors implementations whereby they do send the PIN over in plain text for it to be verified. Successfully reverse engineering this challenge-response protocol will allow an attacker to calculate the victims PIN given one successful login trace. Furthermore, it will also remove the first dependency on the use of the API. Being able to communicate to the card directly gives an attacker a higher capability. But as of right now, since we are unaware of how the login occurs the API must be used.

The second dependancy on the use of the API, is secure messaging. When the generateKey function from the API is called to create block cipher keys, it is the computer that requests the card to generate the key, send it over using secure messaging, and then use secure messaging again to place the key into a file to be stored on the card. This again is an improvement upon previous hardware security modules from the manufacture 'Athena- Smartcard solutions' (specifically ASE KeyPro), whereby this was completed without the use of secure messaging, and therefore the block cipher keys where needlessly exported over in plain text which would allow an attacker to sniff them. If this protocol is successfully reversed engineered, it will be possible to sniff the keys by decrypting the messages which contain the keys that are in transit from computer to the smart-card.

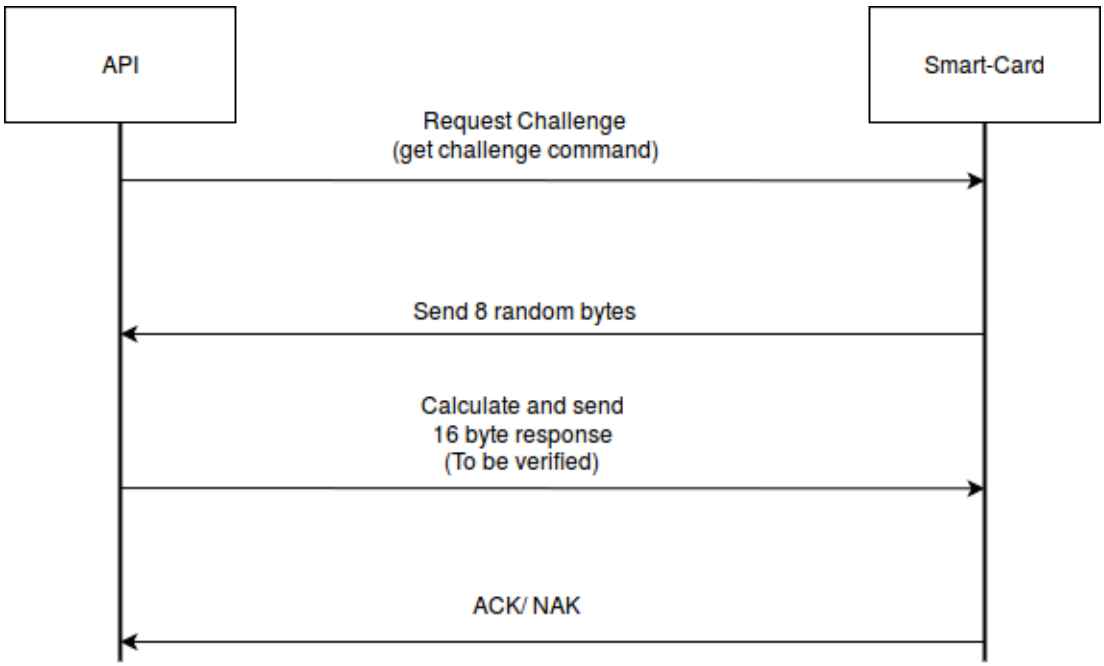
7.1 Reverse Engineering PIN Authentication Protocol

7.2 Reverse Engineering PIN/password Authentication 1st draft

The first attack that I decided to attempt is to reverse-engineer the PIN authentication method. The reasoning behind this is because if this can be successfully done, the PIN can then be inferred from one communication trace sniffed between smart-card and the API (on the computer). The inference comes from the fact that once the method is deduced, an attacker can simply brute force the possible combinations of a PIN, to test if a match is found. (This becomes clearer in the latter sections)

From previous work completed on this card by an MSC student last year [?], and from the analysis conducted in section 6.2, it was quite clear that the card has the following characteristics in terms of PIN authentication. The PKCS#11 API requests a challenge, the smart-card responds with an 8 byte challenge, the API then calculates a 16 byte response (using the 8 byte challenge, and the PIN), the smart-card verifies whether or not the response is correct. There are two response formats to that APDU verification command:

- '90 00' $\rightarrow$ verification succeeded, correct PIN was entered
- '63 CX' $\rightarrow$ verification failed (where X is the number of attempts left before the card is blocked)



The following sections are explanations of the searches that we conducted to try and reverse-engineer the protocol showed above. To give a full understanding of how challenging this part of the project was, we will explain the combinations of possibilities we checked, and the reasoning behind each of them. These will be split up into different 'searches', and increment in terms of new findings and understanding of how the protocol may be implemented.

To aid these explanations, we introduce here 3 key sub-functionalities that most of the conducted searches use. Table 7.1 lists all the hash functions (see section 3.1) that were used, and provides the output length in bits & bytes. Those hash functions were all supported by openSSL and the python package 'hashlib'. Table 7.2 provides the names of the bitwise logical operations that were used to 'join' two bytes together. And table 7.3 provides the description of truncation methods used to reduce the output size of a search down to 16 bytes to match the response provided by the API.

From here on, in the explanation of the searches I will just refer to HASH, JOIN & TRUNCATE which will suggest that all of the elements in the tables have been iterated over and preformed on. For example TUNRCATE(HASH('this is a string')), means the string, 'this is a string', is to be hashed with all the functions in table 7.1, and then truncated to 16 bytes using all the methods listed in table 7.3.

Hash Name	Output Length (bits)	Output Length (Bytes)
MD4	128	16
MD5	128	16
MDC2	128	16
RIPEMD160	160	20
SHA	160	20
SHA1	160	20
SHA224	224	28
SHA256	256	32
SHA384	384	48
SHA512	512	64
WHIRLPOOL	512	64

Table 7.1: Hash Functions

Logical Operations
AND
OR
XOR
NOT AND
NOT OR
NOT XOR

Table 7.2: Bitwise Logical Operations (Joins)

Truncation method	Description?
first_16	Truncates the output by taking the first 16 bytes
last_16	Truncates the output by taking the last 16 bytes
mod_16	Truncates the output by taking modulus 2^{128} [We use 128 because that's the number of bits in 16 bytes]

Table 7.3: Truncation Methods

Before any searches could be conducted, the first task was to extract the values of the 8 byte challenge (denoted X), and the 16 byte response (denoted Y), from a communication trace of C_login. Table 7.4 provides the values for the PIN, X and Y in hexadecimal format.

Data	ASCII	HEX
PIN	’0000000000000000’	30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
Y	N/A	53 17 55 20 F4 30 18 56 80 E6 75 55 E1 91 A7 EC
X	N/A	68 F1 E4 92 85 36 39 A3

Table 7.4

In the very first original experiments, we made assumptions as did previous work on this card [ref Msc] that the PIN number was only numerical characters, only had 4-8 digits and if the PIN was less than 8 characters, it was padded using a special character to form 8 bytes. These assumptions turned out to be false, and thus the elementary experiments were all flawed from the beginning. I have not included a description of those experiments in the following sections, as many of them were in fact very similar to those explained below, just with different assumptions of the PIN. They also used a PIN for the card that was ’12345’, and hence there was an exponential explosion in the number of experiments for a particular search, due to needing to test for different padding schemes and characters. Hence why in these following experiments the PIN had been set to ’0000000000000000’ (16 zeros, no need for padding).

7.2.1 Authentication Protocol Search 1.0 (Password Storage)

Assumptions:

- PIN consists of alpha - numeric characters
- PIN is a maximum of 16 bytes
- PIN is encoded in ASCII characters
- For any PIN that is less than 16 bytes long, there is padding character used to pad the PIN to 16 bytes

Search 1 - Hash functions
In this initial stages we thought that there is a large possibility that the 16 byte response was generated by hashing a combination of the PIN and the 8 byte challenge. This was partly due to common practices used in industry

whereby users passwords are often hashed, and in most cases salted (see section 3.1), before storing them in databases. This practice is more secure than storing plain text passwords, as if an attacker were to gain access to the back end databases storing said passwords, the password itself would not be available to see. For authentication the password is just hashed (and salted, if a salt is used), and then compared against to the stored value. The fact that from multiple traces the 16 byte response seemed to be uniformly random, it supported this hypothesis.

Thus the first search that we completed focused on the fact that a hash function was used to produce the 16 byte response. Below we have listed the methods tested in expirments to generate a 16 bytes, given X and the PIN.

We donote ‖ as the concatination function. Thus for two strings ’string1 ’ ‖ ’string2’ = ’string1 string2’.

Methods tested that produced a 16 byte output using X and the PIN

- join(truncate(hash(X)), pin))
- truncate(hash(join(pin, X ‖ X)))
- truncate(hash(pin‖X))
- truncate(hash(X‖pin))
- truncate(hash(pin+X))
- truncate(hash(join(pin, square(X))))

[The methods should be read from the most inner brackets, outward. Therefore this means that the first method dictates that X is first hashed using one of the hash functions listed in the table 7.1. The output of that is then truncated to 16 bytes using one of the functions from table 7.3. All iterations of the functions in the tables were tested.]

The following experiment resulted in 2592 individual tests, but did not find a match to the response generated by the API [Y in table 7.4]. Thus we moved onto search 2.

Search 2 - PBKDF2

Following the failure of search 1, but still assuming there is a large possibility of a hash function being used due to the common practices mentioned in search 1, and the characteristics known so far of the 16 byte response Y. Then we decided to look at the password-based key derivation function (PBKDF2), which was created as part of PKCS #5 by RSA laboratories [?]. It has been mentioned in literature [?], and started to be used for more secure password storage as well as for key derivation. Essentially PBKDF2 takes

as input, a password (the PIN), a salt (the 8 byte challenge X), a hash function, and the number of iterative rounds. If the number of iterative round is set to 10, then the salted password would be hashed once, and the output of that would be the input for the next round of hashing. This would be completed 10 times. Literature [?] has shown that the standard for the number of iterative rounds used to be 10,000, back in 2008 (check this date). Now it is suggested to use as many rounds as is computationally feasible by the device. Due to the processing power of a smart-card I assumed that this would not exceed 100,000 rounds, in the case that PBKDF2 was used.

Hence this search generated experiments that ran through $1 \rightarrow 100{,}000$ rounds of PBKDF2. As the default of PBKDF2 is to truncate the output by taking the first X (X here is a variable) bytes only 'first_16' truncation method was used.

Methods tested that produced a 16 byte output using X and the PIN

- PBKDF2(hash_function, PIN, X, number_of_rounds)

This generated 100,000 experiments per hash function. With 12 hash functions, this resulted in 1.2 million tests being run. Due to the sheer computational power required for this search I decided to parallelize the search based on the hash function, and run them on separate cores of a server. Even by improving the efficiency of this search, it still took 2 weeks to conduct.

Again this unfortunately did not result in a match between the 16 byte responses calculated and Y (the API's response). Hence we move onto search 3.

7.2.2 Authentication Protocol Search 2.0 (One Time Passwords)

Search 3 - OCRA: OATH Challenge-Response Algorithm

With no luck of deducing the method the authentication protocol uses, we decided to look into more complex standards that exist and used in different parts of the computing industry for challenge response protocols, rather than password storage techniques. The international engineering task force (IETF) released a paper in 2011 [?]. Section 7.1 of the paper gives a one-way challenge response algorithm which fitted the characteristics of the authentication that takes between the API and the smart-card.

Section 3.5.1 explains hash based one time passwords. But in essence HTOP is:
$HOTP(K,C) = Truncate(HMAC(K,C))\&0x7FFFFFFF$ [?-wiki]

Still to complete.

Search 4 - TOTP
With the above in mind, I also wanted to rule out TOTP. This was completed by halting communication between the smart-card and API using the man-in-the-middle tool (see section 4.2). I did this for upto 2 hours. We were looking for a failed verifaction, despite having the correct PIN. The failure should have been caused by the delay in the response if TOTP was used. This was not found and therefore ruled out the possibility of TOTP.

This section will be explained better, and will also include reasoning behind why TOTP is not often used (time sync problems).

7.2.3 Authentication Protocol Search 3.0 (Triple DES Encryption)

Need to explain multiple logins and the characteristics found!
That lead me to believe DES3 encryption was used. Tabulate and cross out other possibilites!

Search 5
Need to decrypt two different encryptions with different PINs in order to find out if this is the method used!
Raw ASCII password and MD5 hash (due to output size = 16 bytes)
DES3-ECB $\rightarrow$ encrypt (Na $\|$ Vac)

Search 6

7.3 Block Cipher Injection (MiTM)

7.4 Manually Overriding Attributes

This attack was found by previous literature. We simply tested to see if the same attack was feasiable on the smart card we analyse.

To test this we created a DES3 key on the card that had the encrypt attribute set to False. We asked the card to encrypt the string '12345678'. The API returned an error stating that this function was not supported due to the attribute settings. We then override the API by manually sending the command overselves. This results in the encryption taking place on the card, when in-fact it should not.

Chapter 8

Conclusion / Results

This shall summarise the whole report and my findings in regard to low-level vulnerabilities on the card.

Chapter 9

Future work

1. complete the reverse engineering secure messaging
2. If SM is known get wrap functionality to work from APDU layer
3. Try using ’reallocate binary’ to change retry counter

Bibliography

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891âĂŞ921, 1905.

[3] Knuth: Computers and Typesetting,
http://www-cs-faculty.stanford.edu/~uno/abcde.html

[4] Frank Morgner : Creating a Virtual Smart Card
https://frankmorgner.github.io/vsmartcard/virtualsmartcard/README.html

[5] Internet Engineering Task Force (IETF).
OCRA: OATH Challenge-Response Algorithm, June 2011
https://tools.ietf.org/html/rfc6287

Appendices

Appendix A

Attack Traces

A.1 Multiple C_login Traces

A.1.1 Different Pin, Different Nonce

```
----- APDU command/response pair 0 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                        .....

00000000: 00 00 00 00 00 00 00 00  90 00                        ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 C2 29 5F  78 D1 68 29 13 78 BE 7B  . ....)_x.h).x.{
00000010: 8E 61 9B 32 2E                                        .a.2.

00000000: 63 C9                                                 c.

----- APDU command/response pair 1 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                        .....

00000000: 00 00 00 00 00 00 00 01  90 00                        ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 C4 6D 44  03 92 F3 6B EF 13 18 07  . ....mD...k....
00000010: CE 5A A4 B9 27                                        .Z..'

00000000: 63 C8                                                 c.
```

A.1.2 Different Pin, Same Nonce

```
----- APDU command/response pair 0 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                  .....

00000000: 00 00 00 00 00 00 00 00  90 00                  ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 8F 58 1B  91 BC 78 D3 37 A9 D9 FB  . ....X...x.7...
00000010: 9C 20 58 F6 0A                                  . X..

00000000: 63 C9                                           c.

----- APDU command/response pair 1 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                  .....

00000000: 00 00 00 00 00 00 00 00  90 00                  ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 E0 30 33  BB 03 0F 6E 11 08 C0 8D  . ....03...n....
00000010: 1D 9D 85 C4 A6                                  .....

00000000: 63 C8                                           c.
```

A.1.3 Same Pin, Different Nonce

```
----- APDU command/response pair 0 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                  .....

00000000: 00 00 00 00 00 00 00 00  90 00                  ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 6E 78 D4  D5 61 AD 3C 26 D3 89 E8  . ...nx..a.<&...
00000010: 96 B9 92 0D 40                                  ....@

00000000: 63 C9                                           c.

----- APDU command/response pair 1 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                  .....
```

```
00000000: 00 00 00 00 00 00 00 01  90 00                    ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 6E 78 D4  D5 61 AD 3C 26 BC C6 AA  . ...nx..a.<&...
00000010: 72 D3 95 2B 94                                    r..+.

00000000: 63 C8                                             c.
```

A.1.4 Same Pin, Same Nonce

```
----- APDU command/response pair 0 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                    .....

00000000: 00 00 00 00 00 00 00 00  90 00                    ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 EE D2 F1  54 07 18 8A 8F AB A3 F7  . ......T.......
00000010: 3E 64 17 2D 6E                                    >d.-n

00000000: 63 C9                                             c.

----- APDU command/response pair 1 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                    .....

00000000: 00 00 00 00 00 00 00 00  90 00                    ..........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 EE D2 F1  54 07 18 8A 8F AB A3 F7  . ......T.......
00000010: 3E 64 17 2D 6E                                    >d.-n

00000000: 63 C8                                             c.
```

A.1.5 Different Second

```
----- APDU command/response pair 0 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                    .....

00000000: 00 00 00 00 00 00 00 00  90 00                    ..........
```

```
----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 61 50 65  E1 AF 05 7B C3 35 98 0D  . ...aPe...{.5..
00000010: DC 9D C5 42 96                                    ...B.

00000000: 63 C9                                             c.

----- APDU command/response pair 1 -----
(Inter-Industry) Get Challenge
00000000: 00 84 00 00 08                                    .....

00000000: 00 00 00 00 00 00 00 00  90 00                    .........

----- APDU command/response pair 1 -----
(Proprietary) Verify
00000000: 80 20 00 00 10 BF 73 83  F9 30 B1 74 D2 E4 98 83  . ....s..0.t....
00000010: 3A 9F 1F 37 BA                                    :..7.

00000000: 63 C8                                             c.
```

A.2 Successful Login Injection

```
----- APDU command/response pair 12 -----

(Inter-Industry) Get Challenge
COMMAND from API
00000000: 00 84 00 00 08                                    .....

Do you want to automate the injection your own login response? (Y/n)

RESPONSE
00000000: E7 69 60 B5 C8 FC D2 02  90 00                    .i‘.......

----- APDU command/response pair 13 -----

(Proprietary) Verify
COMMAND from API
00000000: 80 20 00 00 10 4A D1 3D  AB 98 7F C5 18 A9 B3 1F  . ...J.=........
00000010: 2F 96 B4 3C AF                                    /..<.

(Proprietary) Verify
COMMAND injected
00000000: 80 20 00 00 10 32 5A 9F  38 CA 4F BE 44 3A CD E1  . ...2Z.8.O.D:..
00000010: C5 03 84 35 DF                                    ...5.

RESPONSE
00000000: 90 00                                             ..
```

A.3 Open Secure Messaging Traces

A.3.1 Generator = 5, [Not modified]

```
----- APDU command/response pair 24 -----

(Proprietary) Get Card Public Key
COMMAND from API
00000000: 80 48 00 80 00                                          .H...

Do you want to to alter command? (y/N)

RESPONSE
00000000: 80 01 05 81 81 80 F7 B5  15 72 07 22 94 6F C4 08  .........r.".o..
00000010: 64 CB BD AF EA 55 7D BD  8F 55 36 B0 01 C2 8B 2E  d....U}..U6.....
00000020: 32 B6 5D 45 F1 74 5D 38  12 0B AD 9D 2C 03 9C 22  2.]E.t]8....,.."
00000030: 46 68 EB 2E A2 8C 20 95  A8 2E 6C A8 E0 6D 47 F2  Fh.... ...l..mG.
00000040: D3 1E D7 01 F8 15 5C AD  DC 05 70 C0 93 B2 6D 74  ......\.p...mt
00000050: B0 9B 95 E6 4D 8C D2 FC  73 3E CD 0F 30 68 79 A5  ....M...s>..0hy.
00000060: B9 35 F2 41 3F 52 AD AD  32 A0 99 1A 18 3D CC 57  .5.A?R..2....=.W
00000070: 7E 39 DA 47 53 1E 67 15  AB 01 70 7F F2 47 96 71  ~9.GS.g...p..G.q
00000080: 44 23 CE 7B 60 67 82 81  80 3C 52 D2 06 89 28 92  D#.{`g...<R...(.
00000090: 2C AB E6 3C 4E E6 DF 0E  D2 29 F1 01 BE 36 C4 F8  ,..<N....)...6..
000000A0: 54 40 56 F3 4A FA 8D 2E  9B 60 F5 07 BC ED B4 44  T@V.J....`.....D
000000B0: 56 68 5D 82 4C C4 EA D7  96 20 F8 C5 46 A6 E0 16  Vh].L.... ..F...
000000C0: B8 AB A5 D8 43 29 58 53  77 17 09 97 AA 70 68 33  ....C)XSw....ph3
000000D0: 9E F1 41 0A 5F 39 D9 75  24 7F 3A 53 63 61 47 87  ..A._9.u$.:ScaG.
000000E0: 87 7F 88 96 BC BB 83 A1  CB D1 42 E0 EB 99 CF 34  ..........B....4
000000F0: 0E CA 56 4F 2C 57 50 6E  7B 1A FC 1F 90 7A E0 C2  ..VO,WPn{....z..
00000100: 61 09                                             a.

Do you want to alter the response? (y/N)

----- APDU command/response pair 25 -----

(Inter-Industry) Get Remaining Bytes
COMMAND from API
00000000: 00 C0 00 00 09                                          .....

Do you want to to alter command? (y/N)

RESPONSE
00000000: A8 5D D3 30 E3 5C A9 00  39 90 00                       .].0.\..9..

Do you want to alter the response? (y/N)

----- APDU command/response pair 26 -----
```

```
(Proprietary) Open Secure Messaging
COMMAND from API
00000000: 80 86 00 00 80 08 9F EA  A1 DC 8F C3 43 FD FD 4A  ............C..J
00000010: E6 95 7E C0 D3 C6 FE 81  61 59 4B CE 45 21 96 63  ..~.....aYK.E!.c
00000020: 0F AB 19 D8 61 1A B2 6B  00 E2 44 0F 06 A3 5B 60  ....a..k..D...[‘
00000030: 87 76 C0 B7 E9 15 D5 50  DB 17 D6 C1 3C 26 54 47  .v.....P....<&TG
00000040: AA A3 4B DC 2C 14 81 08  84 0D F0 CA FB 49 8B C1  ..K.,........I..
00000050: B1 0B A1 2B 86 20 02 F2  0F 69 F0 56 2C 83 0C 6E  ...+. ...i.V,..n
00000060: A6 6A E9 86 56 47 71 24  0C B7 91 7F 37 85 0A D4  .j..VGq$....7...
00000070: 12 35 1F CE 17 6C D2 52  FB 04 24 CF DD E9 53 BE  .5...l.R..$...S.
00000080: DA 26 EA 54 FB 00                                 .&.T..
Do you want to to alter command? (y/N)

RESPONSE
00000000: 66 56 36 31 16 42 8D 8A  BC 06 BA AC 5D 35 26 F5  fV61.B......]5&.
00000010: BF 58 15 7F 00 4F EF 2F  54 FB C4 F2 10 8F CB D6  .X...O./T.......
00000020: 90 00                                             ..

Do you want to alter the response? (y/N)

----- APDU command/response pair 27 -----

(Proprietary) Get Challenge [SM]
COMMAND from API
00000000: 0C 84 00 00 0D 97 01 20  8E 08 05 E4 4A 19 32 DE  ....... ....J.2.
00000010: 51 CB 00                                          Q..

Do you want to to alter command? (y/N)

RESPONSE
00000000: 87 29 01 BD 69 F3 85 A7  98 2E 08 07 21 88 30 2F  .)..i.......!.0/
00000010: 06 FF 93 E4 2F 31 C5 4A  40 FB 45 3A 45 C1 4A 84  ..../1.J@.E:E.J.
00000020: 7F BA 59 BC 44 8A 70 A0  BC DA FB 99 02 90 00 8E  ..Y.D.p.........
00000030: 08 44 26 95 74 6A 51 A3  72 90 00                 .D&.tjQ.r..

Do you want to alter the response? (y/N)

----- APDU command/response pair 28 -----

(Proprietary) Close Secure Messaging
COMMAND from API
00000000: 80 86 FF FF                                       ....
```

A.3.2 Generator = 1

```
----- APDU command/response pair 24 -----

(Proprietary) Get Card Public Key
```

```
COMMAND from API
00000000: 80 48 00 80 00                                        .H...

Do you want to to alter command? (y/N)

RESPONSE
00000000: 80 01 05 81 81 80 F7 B5  15 72 07 22 94 6F C4 08  .........r.".o..
00000010: 64 CB BD AF EA 55 7D BD  8F 55 36 B0 01 C2 8B 2E  d....U}..U6.....
00000020: 32 B6 5D 45 F1 74 5D 38  12 0B AD 9D 2C 03 9C 22  2.]E.t]8....,.."
00000030: 46 68 EB 2E A2 8C 20 95  A8 2E 6C A8 E0 6D 47 F2  Fh.... ...l..mG.
00000040: D3 1E D7 01 F8 15 5C AD  DC 05 70 C0 93 B2 6D 74  ......\.p...mt
00000050: B0 9B 95 E6 4D 8C D2 FC  73 3E CD 0F 30 68 79 A5  ....M...s>..0hy.
00000060: B9 35 F2 41 3F 52 AD AD  32 A0 99 1A 18 3D CC 57  .5.A?R..2....=.W
00000070: 7E 39 DA 47 53 1E 67 15  AB 01 70 7F F2 47 96 71  ~9.GS.g...p..G.q
00000080: 44 23 CE 7B 60 67 82 81  80 3C 52 D2 06 89 28 92  D#.{`g...<R...(.
00000090: 2C AB E6 3C 4E E6 DF 0E  D2 29 F1 01 BE 36 C4 F8  ,..<N....)...6..
000000A0: 54 40 56 F3 4A FA 8D 2E  9B 60 F5 07 BC ED B4 44  T@V.J....`.....D
000000B0: 56 68 5D 82 4C C4 EA D7  96 20 F8 C5 46 A6 E0 16  Vh].L.... ..F...
000000C0: B8 AB A5 D8 43 29 58 53  77 17 09 97 AA 70 68 33  ....C)XSw....ph3
000000D0: 9E F1 41 0A 5F 39 D9 75  24 7F 3A 53 63 61 47 87  ..A._9.u$.:ScaG.
000000E0: 87 7F 88 96 BC BB 83 A1  CB D1 42 E0 EB 99 CF 34  ..........B....4
000000F0: 0E CA 56 4F 2C 57 50 6E  7B 1A FC 1F 90 7A E0 C2  ..VO,WPn{....z..
00000100: 61 09                                             a.

Do you want to alter the response? (y/N)
y

00000000: 80 01 01 81 81 80 F7 B5  15 72 07 22 94 6F C4 08  .........r.".o..
00000010: 64 CB BD AF EA 55 7D BD  8F 55 36 B0 01 C2 8B 2E  d....U}..U6.....
00000020: 32 B6 5D 45 F1 74 5D 38  12 0B AD 9D 2C 03 9C 22  2.]E.t]8....,.."
00000030: 46 68 EB 2E A2 8C 20 95  A8 2E 6C A8 E0 6D 47 F2  Fh.... ...l..mG.
00000040: D3 1E D7 01 F8 15 5C AD  DC 05 70 C0 93 B2 6D 74  ......\.p...mt
00000050: B0 9B 95 E6 4D 8C D2 FC  73 3E CD 0F 30 68 79 A5  ....M...s>..0hy.
00000060: B9 35 F2 41 3F 52 AD AD  32 A0 99 1A 18 3D CC 57  .5.A?R..2....=.W
00000070: 7E 39 DA 47 53 1E 67 15  AB 01 70 7F F2 47 96 71  ~9.GS.g...p..G.q
00000080: 44 23 CE 7B 60 67 82 81  80 3C 52 D2 06 89 28 92  D#.{`g...<R...(.
00000090: 2C AB E6 3C 4E E6 DF 0E  D2 29 F1 01 BE 36 C4 F8  ,..<N....)...6..
000000A0: 54 40 56 F3 4A FA 8D 2E  9B 60 F5 07 BC ED B4 44  T@V.J....`.....D
000000B0: 56 68 5D 82 4C C4 EA D7  96 20 F8 C5 46 A6 E0 16  Vh].L.... ..F...
000000C0: B8 AB A5 D8 43 29 58 53  77 17 09 97 AA 70 68 33  ....C)XSw....ph3
000000D0: 9E F1 41 0A 5F 39 D9 75  24 7F 3A 53 63 61 47 87  ..A._9.u$.:ScaG.
000000E0: 87 7F 88 96 BC BB 83 A1  CB D1 42 E0 EB 99 CF 34  ..........B....4
000000F0: 0E CA 56 4F 2C 57 50 6E  7B 1A FC 1F 90 7A E0 C2  ..VO,WPn{....z..
00000100: 61 09                                             a.
response changed!

----- APDU command/response pair 25 -----

(Inter-Industry) Get Remaining Bytes
```

```
COMMAND from API
00000000: 00 C0 00 00 09                                        .....

Do you want to to alter command? (y/N)

RESPONSE
00000000: A8 5D D3 30 E3 5C A9 00  39 90 00                     .].0....9..

Do you want to alter the response? (y/N)

----- APDU command/response pair 26 -----

(Proprietary) Open Secure Messaging
COMMAND from API
00000000: 80 86 00 00 80 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 01 00                                     ......

Do you want to to alter command? (y/N)

RESPONSE
00000000: 7D 2C 25 47 1C 16 34 51  E9 C3 49 38 C8 79 1E ED  },%G..4Q..I8.y..
00000010: A2 6B 20 D4 54 BD 67 0A  D3 85 3E B9 E0 6E D5 5E  .k .T.g...>..n.^
00000020: 90 00                                             ..

Do you want to alter the response? (y/N)

----- APDU command/response pair 27 -----

(Proprietary) Get Challenge [SM]
COMMAND from API
00000000: 0C 84 00 00 0D 97 01 20  8E 08 08 C6 59 9B 57 E6  ....... ....Y.W.
00000010: B4 4E 00                                          .N.

Do you want to to alter command? (y/N)

RESPONSE
00000000: 69 88                                             i.

Do you want to alter the response? (y/N)

----- APDU command/response pair 28 -----
```

```
(Proprietary) Close Secure Messaging
COMMAND from API
00000000: 80 86 FF FF                                    ....
```

A.3.3 Generator = 0

```
----- APDU command/response pair 24 -----

(Proprietary) Get Card Public Key
COMMAND from API
00000000: 80 48 00 80 00                                 .H...

Do you want to to alter command? (y/N)

RESPONSE
00000000: 80 01 05 81 81 80 F7 B5  15 72 07 22 94 6F C4 08  .........r.".o..
00000010: 64 CB BD AF EA 55 7D BD  8F 55 36 B0 01 C2 8B 2E  d....U}..U6.....
00000020: 32 B6 5D 45 F1 74 5D 38  12 0B AD 9D 2C 03 9C 22  2.]E.t]8....,.."
00000030: 46 68 EB 2E A2 8C 20 95  A8 2E 6C A8 E0 6D 47 F2  Fh.... ...l..mG.
00000040: D3 1E D7 01 F8 15 5C AD  DC 05 70 C0 93 B2 6D 74  ......\.p...mt
00000050: B0 9B 95 E6 4D 8C D2 FC  73 3E CD 0F 30 68 79 A5  ....M...s>..0hy.
00000060: B9 35 F2 41 3F 52 AD AD  32 A0 99 1A 18 3D CC 57  .5.A?R..2....=.W
00000070: 7E 39 DA 47 53 1E 67 15  AB 01 70 7F F2 47 96 71  ~9.GS.g...p..G.q
00000080: 44 23 CE 7B 60 67 82 81  80 3C 52 D2 06 89 28 92  D#.{`g...<R...(.
00000090: 2C AB E6 3C 4E E6 DF 0E  D2 29 F1 01 BE 36 C4 F8  ,..<N....)...6..
000000A0: 54 40 56 F3 4A FA 8D 2E  9B 60 F5 07 BC ED B4 44  T@V.J....`.....D
000000B0: 56 68 5D 82 4C C4 EA D7  96 20 F8 C5 46 A6 E0 16  Vh].L.... ..F...
000000C0: B8 AB A5 D8 43 29 58 53  77 17 09 97 AA 70 68 33  ....C)XSw....ph3
000000D0: 9E F1 41 0A 5F 39 D9 75  24 7F 3A 53 63 61 47 87  ..A._9.u$.:ScaG.
000000E0: 87 7F 88 96 BC BB 83 A1  CB D1 42 E0 EB 99 CF 34  ..........B....4
000000F0: 0E CA 56 4F 2C 57 50 6E  7B 1A FC 1F 90 7A E0 C2  ..VO,WPn{....z..
00000100: 61 09                                          a.

Do you want to alter the response? (y/N)
y

00000000: 80 01 00 81 81 80 F7 B5  15 72 07 22 94 6F C4 08  .........r.".o..
00000010: 64 CB BD AF EA 55 7D BD  8F 55 36 B0 01 C2 8B 2E  d....U}..U6.....
00000020: 32 B6 5D 45 F1 74 5D 38  12 0B AD 9D 2C 03 9C 22  2.]E.t]8....,.."
00000030: 46 68 EB 2E A2 8C 20 95  A8 2E 6C A8 E0 6D 47 F2  Fh.... ...l..mG.
00000040: D3 1E D7 01 F8 15 5C AD  DC 05 70 C0 93 B2 6D 74  ......\.p...mt
00000050: B0 9B 95 E6 4D 8C D2 FC  73 3E CD 0F 30 68 79 A5  ....M...s>..0hy.
00000060: B9 35 F2 41 3F 52 AD AD  32 A0 99 1A 18 3D CC 57  .5.A?R..2....=.W
00000070: 7E 39 DA 47 53 1E 67 15  AB 01 70 7F F2 47 96 71  ~9.GS.g...p..G.q
00000080: 44 23 CE 7B 60 67 82 81  80 3C 52 D2 06 89 28 92  D#.{`g...<R...(.
00000090: 2C AB E6 3C 4E E6 DF 0E  D2 29 F1 01 BE 36 C4 F8  ,..<N....)...6..
000000A0: 54 40 56 F3 4A FA 8D 2E  9B 60 F5 07 BC ED B4 44  T@V.J....`.....D
000000B0: 56 68 5D 82 4C C4 EA D7  96 20 F8 C5 46 A6 E0 16  Vh].L.... ..F...
```

```
000000C0: B8 AB A5 D8 43 29 58 53  77 17 09 97 AA 70 68 33  ....C)XSw....ph3
000000D0: 9E F1 41 0A 5F 39 D9 75  24 7F 3A 53 63 61 47 87  ..A._9.u$.:ScaG.
000000E0: 87 7F 88 96 BC BB 83 A1  CB D1 42 E0 EB 99 CF 34  ..........B....4
000000F0: 0E CA 56 4F 2C 57 50 6E  7B 1A FC 1F 90 7A E0 C2  ..VO,WPn{....z..
00000100: 61 09                                           a.
response changed!

----- APDU command/response pair 25 -----

(Inter-Industry) Get Remaining Bytes
COMMAND from API
00000000: 00 C0 00 00 09                                  .....

Do you want to to alter command? (y/N)

RESPONSE
00000000: A8 5D D3 30 E3 5C A9 00  39 90 00               .].0....9..

Do you want to alter the response? (y/N)

----- APDU command/response pair 26 -----

(Proprietary) Open Secure Messaging
COMMAND from API
00000000: 80 86 00 00 80 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00                               ......

Do you want to to alter command? (y/N)

RESPONSE
00000000: 7E 4B 40 E6 E3 B1 5D 25  2B 02 48 50 B3 63 CC 9E  ~K@...]%+.HP.c..
00000010: 79 41 34 FC 04 B3 57 1C  06 E3 D1 36 3C 24 45 8D  yA4...W....6<$E.
00000020: 90 00                                           ..

Do you want to alter the response? (y/N)

----- APDU command/response pair 27 -----

(Proprietary) Get Challenge [SM]
COMMAND from API
00000000: 0C 84 00 00 0D 97 01 20  8E 08 99 BD 52 69 31 DD  ....... ....Ri1.
00000010: DB FD 00                                        ...
```

```
Do you want to to alter command? (y/N)

RESPONSE
00000000: 69 88                                                  i.

Do you want to alter the response? (y/N)

----- APDU command/response pair 28 -----

(Proprietary) Close Secure Messaging
COMMAND from API
00000000: 80 86 FF FF                                            ....
```

A.4 Overriding Attribute Controls

A.5 Encrypt_False

Appendix B

API Function Traces

B.1 Initialization

```
----- APDU command/response pair 1 -----
00000000: 00 A4 04 00 0C A0 00 00  01 64 4C 41 53 45 52 00  .........dLASER.
00000010: 01 00                                           ..

00000000: 90 00                                           ..

----- APDU command/response pair 4 -----
00000000: 80 A4 08 00 06 3F 00 30  00 C0 00               .....?.0...

00000000: 90 00                                           ..


----- APDU command/response pair 5 -----
00000000: 00 B0 00 00 00                                  .....

00000000: 49 44 50 72 6F 74 65 63  74 20 20 20 20 20 20 20  IDProtect
00000010: 20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
00000020: 41 74 68 65 6E 61 20 53  6D 61 72 74 63 61 72 64  Athena Smartcard
00000030: 20 53 6F 6C 75 74 69 6F  6E 73 20 20 20 20 20 20   Solutions
00000040: 49 44 50 72 6F 74 65 63  74 20 20 20 20 20 20 20  IDProtect
00000050: 30 44 35 30 30 30 30 39  32 31 32 32 38 37 39 36  0D50000921228796
00000060: 0D 04 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 10 00 00 00  04 00 00 00 FF FF FF FF  ................
00000080: 00 00 00 00 FF FF FF FF  00 00 00 00 01 00 01 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 90 00                                        ...


----- APDU command/response pair 8 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 03 40 00         .....?.0.0.@.
```

```
00000000: 90 00                                           ..


----- APDU command/response pair 9 -----
00000000: 00 B0 00 02 64                                  ....d

00000000: 41 54 48 45 4E 41 53 4E  C0 AD AA 78 FC 88 42 0D  ATHENASN...x..B.
00000010: 90 00                                           ..
```

B.2 C_login

```
----- APDU command/response pair 10 -----
00000000: 80 A4 08 0C 04 3F 00 00  20 00                  .....?.. .

00000000: 62 2F 87 01 08 83 02 00  20 80 02 00 10 8A 01 04  b/...... .......
00000010: 86 0E 00 FF C0 30 00 FF  00 10 00 FF 00 10 00 00  .....0..........
00000020: 85 0F 00 01 00 00 AA 00  04 10 00 00 00 00 00 FF  ................
00000030: FF 90 00                                        ...

----- APDU command/response pair 11 -----
00000000: 80 A4 08 00 04 3F 00 00  20                     .....?..

00000000: 90 00                                           ..

----- APDU command/response pair 12 -----
00000000: 00 84 00 00 08                                  .....

00000000: 11 B7 B2 80 4B 17 0D A4  90 00                  ....K.....


----- APDU command/response pair 13 -----
00000000: 80 20 00 00 10 1D ED 9E  47 A8 C9 EA CE 37 82 2C  . ......G....7.,
00000010: 92 CF 07 20 2D                                  ... -

00000000: 90 00                                           ..


----- APDU command/response pair 20 -----
00000000: 80 28 00 00 04 00 00 00  20                     .(......

00000000: 90 00                                           ..
```

B.3 C_findObject

```
----- APDU command/response pair 32 -----
00000000: 80 30 01 00 00                                        .0...

00000000: D1 02 00 03 D2 02 03 40  D2 02 03 46 D2 0A 86 7F  .......@...F....
00000010: 63 6D 61 70 66 69 6C 65  90 00                    cmapfile..


----- APDU command/response pair 33 -----
00000000: 80 A4 08 00 06 3F 00 30  00 30 02                 .....?.0.0.

00000000: 90 00                                             ..


----- APDU command/response pair 34 -----
00000000: 80 30 01 00 00                                        .0...

00000000: D1 02 00 00 90 00                                     ......


----- APDU command/response pair 35 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 40           .....?.0.0..@

00000000: 90 00                                             ..


----- APDU command/response pair 36 -----
00000000: 00 B0 00 00 00                                        .....

00000000: 00 03 03 40 01 23 18 00  00 00 00 04 04 00 00 00  ...@.#..........
00000010: 00 01 00 00 01 01 00 02  00 00 01 00 00 03 10 00  ................
00000020: 04 64 65 73 33 FF FF FF  FF FF FF FF FF FF FF FF  .des3............
00000030: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000040: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000050: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000060: FF 00 11 01 00 18 FF FF  FF FF FF FF FF FF FF FF  ................
00000070: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF 01 00  ................
00000080: 00 00 04 15 00 00 00 01  02 10 00 01 01 FF FF FF  ................
00000090: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000000A0: FF FF FF FF FF FF FF FF  FF FF FF FF 01 03 30 00  ..............0.
000000B0: 01 01 01 04 00 00 01 01  01 05 50 00 01 01 01 06  ..........P.....
000000C0: 00 00 01 00 01 07 50 00  01 01 01 08 50 00 01 01  ......P.....P...
000000D0: 01 0A 00 00 01 01 01 0C  10 00 01 00 01 10 10 00  ................
000000E0: 00 FF FF FF FF FF FF FF  FF 01 11 10 00 00 FF FF  ................
000000F0: FF FF FF FF FF FF 01 62  50 00 01 00 01 63 00 00  .......bP....c..
00000100: 61 27                                             a'
```

```
----- APDU command/response pair 37 -----
00000000: 00 B0 01 00 00                                        .....

00000000: 01 01 01 64 00 00 01 01  01 65 00 00 01 01 01 66  ...d.....e.....f
00000010: 00 00 04 31 01 00 00 01  70 00 00 01 01 80 10 00  ...1....p.......
00000020: 00 01 00 99 03 99 03 90  00                       .........


----- APDU command/response pair 38 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 46           .....?.0.0..F

00000000: 90 00

----- APDU command/response pair 39 -----
00000000: 00 B0 00 00 00                                        .....

00000000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000100: 61 2F                                             a/


----- APDU command/response pair 40 -----
00000000: 00 B0 01 00 00                                        .....

00000000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 90  ................
00000030: 00                                                .
```

B.4 C_generateKey

t

B.5 C_generateKeyPair

t

B.6 C_destroyObject

t

B.7 C_encrypt

t

B.8 C_decrypt

t

B.9 C_setAttribute

t

B.10 C_unwrap

t