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)

Smartcards are used commercially and within industry for authentication, encryption, decryption, signing and verifying data. This paper aims to look into how the smartcard 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

Smartcards 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.

Smartcards 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 smartcards have their own on-board micro-controller, to allow all of these operations to take place on the smartcard 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 smartcard and the PIN/password. In many industries, for applications such as, banking/ payment systems, telecommunications, healthcare and public sector transport, smartcards are used due to the security they are believed to provide.

The most common API (application programming interface) that is used to communicate with smartcards 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 smartcards 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 smartcard 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 smartcard 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 smartcard’s 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	05	L	X	00	(II) Encrypt Data
00	2a	80	05	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	00	00	-	FL	-	(P) Select File [append previous path]
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	-	Reallocate Binary 256 bytes
80	32	00	03	L	X	-	Reallocate 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

3.2.2 Diffie Hellman

3.2.3 ECC

also explain ECCDH

3.3 Symmetric Encryption

explain this in an overview term

3.3.1 Block Ciphers

1. AES
2. DES
3. Triple DES

3.3.2 ECB Mode

3.3.3 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 Smartcard

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

5.1 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 6

PKCS #11 Functions - APDU analysis

As discussed in chapter 2, PKCS #11 API functions are broken down into multiple ISO 7816 command-response pairs via the use of the card vendors middleware *(their implementation of the PKCS #11 API)*. This implementation is at the manufacturer's discretion, who may wish to use proprietary commands. The aim of this chapter is to analyse the APDU traces of 9 PKCS #11 functions, and how the card manufacturer has decided to split them into command-response pairs.

This is because previous literature has shown that some poor implemenations has revealed PIN's and encryption keys at the APDU layer which heavily violate the PKCS #11 standard. Our analysis will include a step by step guide as to how the functions are broken down into command response pairs at the APDU level. In addition, the analysis will provide explanations of how the implementation operates and therefore be able to suggest possible vulnerabilities, some of which will be investigated in chapter 7.

All the traces are reported in appendix B. These traces have been shortened to only include the command-response pairs which are deemed to be the most important to the evaluation of a function. However, the full traces for each function are given within the project directory. We will refer to appendix B including the corresponding command response pair which are given in brackets to allow ease of locating the steps throughout our analysis.

C_sign and *C_verify* are not included in this analysis as earlier work [?] on this card show that both of these functions operate in a similar manner to C_encrypt and C_decrypt. *C_createObject* is also not included in this analysis, because it operates in a similar fashion to *C_generateKey* and *C_generateKeyPair*. The only difference is that the key values are provided by the user rather than being computer generated.

To prevent repetition we list the dependencies (other functions that must be run before hand) at the start of each section when necessary.

It is worth stressing that chapter 6 will only provide explanations of the possible vulnerabilities that might be present. We leave all the investigations of them to chapter 7.

6.1 Initialization

Initialization is process whereby the smart-card sends 2 serial numbers to the API, in order for the API to have the knowledge of which model of smart-card it is to communicate with. 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. Send to the API the card serial number (B.1 - command 4,5)
3. Send to the API the 2nd serial number (B.1 - command 8,9)

The trace doesn't reveal sensitive information. These steps are required in order for the card 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 the serial numbers are required by the library in order for it to operate correctly for the specific type of card.

6.2 C_login

Function dependencies $\rightarrow$ *[Initialization]*

The login function has 5 main components (these are listed below).

1. Open file control parameters for file X_1 (B.2 - command 10)
2. Select file X_2 (B.2 command 11)
3. Request challenge from the card (B.2 - command 12)
4. Use the proprietary VERIFY command for verification of PIN value (B.2 - command 13)
5. Clear security Status (B.2 - command 20)

The API requests to view the file control parameters for file X_1. The file location as stated within the trace is at '3F 00 00 20 00'. This holds the retry counter value, which dictates how many attempts are left before the card is blocked *(A value of 'A0' states the card is currently in a blocked*

status). To make it easier to locate within the trace we have highlighted the current value in bold which is of 'AA', meaning there is 10 attempts left.

Following this check, file X_2 is selected. We assume that this file holds a pointer to EEPROM memory where the PIN is securely stored by the smartcard. This file cannot be accessed with the inter-industry command 'READ BINARY' nor the proprietary command 'READ FILE CONTROL PARAMETERS'. However if this file is not selected PIN verification fails despite having the correct PIN value. This finding was reported by (enter name) last year [?].

After file X_2 has been selected, the API requests the smartcard to send an 8 byte challenge. Upon receiving said challenge, the API then responds with a 16 bytes of data for PIN verification using the proprietary 'VERIFY' command. Assuming the correct PIN has been supplied the user is now authenticated with the card, and the last step clears the security status.

The evaluation of multiple traces of the login function shows that the response calculated appears to have an element of randomness. From studying the traces alone, the method of calculating the value of the response, given the PIN and the smartcards challenge, cannot be intuitively determined.

No immediate sensitive information is revealed in this trace, due to the use of the challenge-response algorithm used for verifying a user's PIN. However, if an attacker can successfully reverse engineer the challenge-response protocol, the PIN can be calculated given 1 successful login trace. Furthermore, if an attacker can find a method for changing the file control parameters, they might be able to reset the retry counter at the APDU level to allow an unlimited number of attempts of logging in. These possible vulnerabilities will be discussed in detail in chapter 7.

6.3 C_findObject

[For this trace to show meaningful information we have generated and stored a triple DES key onto the card, before running the findObject function. The label of the key was 'des3' with id '01']

Function dependencies $\rightarrow$ *[Initialization, Login]*

The findObjects function is split into 3 main components (these are listed below).

1. Select and open 'cmapfile' [List files command]. This stores all file locations for attributes of the keys (B.3 - command 32, 33, 34)

2. Using the data from ’cmapfile’, Open the first attribute key file (B.3 - command 36)
3. Open the next attribute key file. (Which is empty) (B.3 - command 38, 39)

First, a cmapfile is accessed. This file stores the location of key attribute files. Following this, attribute files are accessed. The attribute files store all the information regarding a specific keys attributes (listed in section 2.1.2). Finally the next (and last) file is opened. In the case more than one key was stored on the card at the time of running this function the next file would contain the attributes of the next key. However, since there is only 1 key, the next file is empty (containing all zero’s) thus determining there are no more key objects to be found.

Due to key and attribute files being stored separately, this function call reveals no sensitive information. This finding was reported by (name) 2015 [?] and is stated within our literature review. Only the attributes are opened and listed at the APDU layer. These attributes can be printed at the API layer as well.

A possibility for an attack is present here. The ISO 7816 proprietary ’REAL-LOCATE BINARY’ and the inter-industry ’UPDATE BINARY’ commands can be used to modify a file. This will allow modification of attributes.

As reported in the literature review, the work by (name) 2015 [?] on the same card shows that they were able to modify any attributes. They tested the modification of CKA_sensitive & CKA_extractable from false to true. The change in this direction is not permitted by the PKCS #11 standard. However the ability to change these at the APDU level was achieved. This approach still yielded no significant results, as keys and attribute are stored in separate files, thus the keys could still not be loaded. This is discussed in chapter 7.

6.4 C_generateKey

[For this trace we generate a triple DES key, key length of 24 bytes.]

Function dependencies → [Initialization, Login]

The generateKey function has 9 main components (these are listed below).

1. Open secure messaging (B.4 - commands 26, 27, 28)
2. Generate 24 random bytes and send them to the API via secure messaging (B.4 - command 29)

3. Close secure messaging (B.4 - command 30)
4. commands skipped include finding spare file for attributes and opening the file.
5. Update file with key attributes (B.4 - commands 40, 41)
6. Open secure messaging [again] (B.4 - commands 42, 43, 44)
7. Open key file directory via secure messaging (B.4 - command 45)
8. Create key file for triple DES key via secure messaging (B.4 - command 46)
9. Close secure messaging (B.4 - command 47)

Secure messaging session have an initialization process whereby the API and smartcard generate 2 session keys S_{Enc} and S_{Mac}. Once the initialization is completed the following communications data fields are encrypted with S_{Enc} and a checksum is computed via C-MAC (see section 3.4.2) using S_{Mac}. They provide confidentiality and integrity of the data being sent within the data fields of the commands during a secure messaging session.

Two different secure messaging sessions are used. The first is to request the card to generate 24 bytes (same as the key length) and send it to the API. The second is to place the key value into the key file. The attribute file is created without the use of secure messaging as it does not contain sensitive information.

The analysis of the communication trace suggests that the card is requested to generate the key value from a 'get challenge' request (within secure messaging). We assume that the generated bytes are used to be stored in the key file as the key value. At this stage it is not possible to verify this. The reason being that two different secure messaging sessions are used, and therefore have different session keys. This causes the encryption of the generated bytes and the storing of them in a file, to be different. Despite the possibility they might be identical.

As discussed in the literature review chapter, previous hardware security modules created by 'Athena smartcard solutions' have in the past requested the card to generate random bytes and use them as the key value (when generateKey function is called). This was completed without the use of secure messaging. Thus, it appears that they might be using a similar approach but with the use of secure messaging (to prevent revealing the encryption keys in plain text at the APDU layer) on this smartcard.

If the assumption of the generated bytes being used as the key value holds, it may result in a significant vulnerability that could be exploited. This will only be valid if the secure messaging protocol (that is a proprietary implementation by the card vendor) can be reversed engineered. This should allow an attacker to inject a new key value or decrypt the key value to be

stored. This would be in breach of the PKCS #11 standard, depending on attribute values set.

A second vulnerability is also present. Unlike previous attacks whereby attributes are modified after a key has been created. An attacker could modify the attributes during the key generation process (this would take place at step 5, in the component list above). Changing attribute values would allow for the key to be extracted as soon as it is created (this has been checked). It is worth noting that the user will be aware of it, as it is printed at the API layer. In addition they will also be able to see the modified attributes which they had not set. Thus we determine this to be a vulnerability, but not as serious as the previous one mentioned. As if it were utilised, the user would most likely notice and therefore generate a new key to be used.

Both of these possible vulnerabilities are discussed in chapter 7.

6.5 C_generateKeyPair

[For this trace we generate an RSA-1024 public/private key pair]

Function dependencies → [Initialization, Login]

We find 15 main components in this generation of the RSA public and private key pair (these are listed below).

1. Create public key attribute file (B.5 - command 54)
2. Add public key attributes to file [inc. public exponent] (B.5 - commands 56, 57)
3. Create private key attribute file (B.5 - command 59)
4. Add private key attributes to file (B.5 - commands 61, 62)
5. Create Private CTR RSA Key file (B.5 - command 64) [ref crypto section] [?]
6. Select temporary file (B.5 - command 65)
7. Generate RSA Key Pair (B.5 - command 66)
8. Select temporary file (B.5 - command 67)
9. Get RSA public key (B.5 - command 68)
10. Select parent folder of private key attribute file (B.5 - command 69)
11. Create new file, with public modulus and additional info (B.5 - command 70

12. Select temporary file (B.5 - command 71)
13. Get RSA public key (B.5 - command 72)
14. Select public attribute file (B.5 - command 73)
15. Add public modulus to attribute file (B.5 - command 74)

The rest of the communication we believe to be resetting the temporary file, and adding file location information to file control parameters of parent directories. (These commands are not included in the traces within the appendix)

Components 1, 3, 5, 7, 9, 11, and 13 transfer data that are wrapped within an ASN.1 BER encoding [?]. We have used an online tool [?] to decode the data fields. The following are the ASN.1 BER decodings of the commands that have these wrapping.

1. Create public key attribute file (B.5 - command 54)
Application 2(4 elem)
[10] (1 byte) 04
[03] (2 byte) **01 40**
[00] (2 byte) 01 A7
[06] (8 byte) 00 20 00 20 00 20 00 20

3. Create private key attribute file (B.5 - command 59)
Application 2(5 elem)
[10] (1 byte) 04
[03] (2 byte) **02 00**
[00] (2 byte) 01 23
[04] kx s0
[06] (8 byte) 00 00 00 20 00 20 00 20

5. Create private CTR RSA key file (B.5 - command 64)
Application 2(6 elem)
[10] (1 byte) 04
[03] (2 byte) **00 41**
[00] (2 byte) 00 80
[05] (5 byte) 05 0C 20 00 A3
[06] (14 byte) 00 00 00 FF 00 FF 00 20 00 20 00 00 00 20
Application 17(0 elem)

7. Generate RSA key pair (B.5 - command 66)
[12] (2 elem)
[00] (1 byte) 06

[01] (3 byte) 01 00 01

9,13. Get RSA public key (B.5 - commands 68, 72)
Application 73(2 elem)
[01] (128 byte) D1 EF 7C A5 06 A1 87 FD 5F 13 5B 25 B7 16..
[02] (3 byte) 01 00 01

11. Create new file with public modulus and additional info (B.5 - command 70)
Application 2(6 elem)
[10] (1 byte) 04
[03] (2 byte) **00 81**
[00] (2 byte) 00 80
[05] (5 byte) 05 08 20 00 A3
[06] (14 byte) 00 00 00 FF 00 FF 00 20 00 20 00 00 00 20
Application 17(2 elem)
[16] (3 byte) 01 00 01
[17] (128 byte) D1 EF 7C A5 06 A1 87 FD 5F 13 5B 25 B7...

RSA key pair generation occurs on the card. A dedicated processor is used for this, hence the need to change to temporary files to access the generated key and then store the public information. We notice no sensitive information being revealed within the traces in plain text. We are of the opinion that the information provided that is wrapped within the ASN.1 BER encoding, especially for **5**, might have exported the private key and the additional parameters required for CRT RSA keys [?]. To test this theory we have deleted the generated key and generated another RSA key. This resulted in the public modulus of the RSA key to change, but the remaining parameters did not. A change in the public modulus would cause a change in both the private exponent and CRT parameters. This leads us to conclude that no part of the private key is exported in plain text within the communication traces.

As we will explain in the destroyObject function analysis, when a key is deleted from the card, the location in memory where the keys are securely stored (i.e. cannot be read) is revealed. We have used the destroyObject function to delete both RSA public and private keys. This enabled us to find the locations of the key and attribute files for both keys. These results are reported below:

	Key File	Attribute File
RSA public key	3F0030003002 **00 81** (11. [03])	3F0030003002 **01 40** (1. [03])
RSA private key	3F0030003002 **02 00** (3. [03])	3F0030003002 **00 41** (5. [03])

Table 6.1: File Location Table

Our work shows that components 1, 3, 5, and 11 (listed above) are used to save the details of the location in the smartcard's memory of the RSA public & private attribute and key files. The corresponding componenets with the file location parameters are reported in brackets in the table 6.1. Components 9 and 13 reveal the RSA public exponent and modulus used to be saved into attribute and key files. Finally, component 7 is used to give the size of the RSA key and its public exponent for key pair generation on the card.

Therefore, the only vulnerabilities we notice are, the ability to modify attributes as the key is being generated (components 2 and 4). This was discussed within generateKey function analysis and did not prove to be a vulnerability that would not be noticed by the user. The second vulnerability is the disclosure of the file location of the private key. The opening and possible reading of the key value is discussed as part of the destroyObject function analysis and also in chapter 7.

6.6 C_destroyObject

[For this trace we delete/destroy a triple DES (key option 2) 16 byte key and its attribute file]

Function dependencies → [Initialization, Login, FindObjects]

Once the key and attribute files have been located and the user is authenticated with the card, there are 6 main components to the destruction of the object. These are listed below:

1. Select counter file for key attributes (B.6 - command 49)
2. Update counter (B.6 - command 50)
3. Select key file (B.6 - command 53)
4. Delete key file (B.6 - command 54)
5. Select key attribute file (B.6 - command 55)
6. Delete key attribute file (B.6 - command 56)

Work conducted by (name) 2015 [?] did show that the counter file maintains track of how many keys have been created and deleted in a certain path within the smartcard's memory. We are unsure about its specific use (except for maintaining track of the number of keys) as we have only seen it being updated upon deletion of an object, and the new counter value being appended to the new keys attributes (the one created after deleting an

object). The original and updated counter value is reported in appendix B.6 - command 37 and 50 respectively.

Once the counter file has been updated, the key attribute file and the key file are selected and deleted using the inter-industry command 'DELETE FILE'. This is the first finding where the location of a key file is accessed without the use of secure messaging. The finding is consistent with the earlier work [?]. Thus we think that this may present a vulnerability which would allow us to open and read key values at the APDU level.

We attempt to use the 'SELECT FILE' command and then 'READ BINARY' command to try and read the files data. This resulted in an error message = '69 81', meaning the command is incompatible with the file structure. In out next step, we try the other command which is 'OPEN FILE CONTROL PARAMETERS'. This did successfully work. We find that the output is wrapped in an ASN.1 BER encoding, thus we decoded it this gave the following result:

Application 2(6 elem)
[07] (1 byte) 08
[03] (2 byte) **00 C1**
[00] (2 byte) 00 18
[10] (1 byte) 04
[06] (14 byte) 00 00 00 FF 00 FF 00 00 00 00 00 00 00 00
[05] (4 byte) 01 0C 10 00

The above decoding is similar to what we have reported for the RSA private key analysis in section 6.5. The 16 byte key that would be used for triple DES is not present. There is however pointers to the file that we just read the parameters of *(00 C1),* and other additional parameters that we are unsure of their use.

All security policies [?] that we have reviewed concerning 'Athena smartcard solutions' suggest that keys are held in a non-readable memory to outside actors, and only when required are encrypted internally (by the smartcard's OS) and stored in the EEPROM (electrically erasable programmable read only memory) for them to be used for the security operation required of them.

Thus we did not find any security vulnerabilities from this function. And also eliminates one of the vulnerabilities we suggested within the generateKeyPair function analysis. We discuss further in chapter 7.

6.7 C_encrypt

[For this function we use a triple DES key to encrypt a string 'TestString123456'] using ECB mode]

Function dependencies → [Initialization, Login, FindObjects]

The commands given for this trace are actually repeated and therefore occur twice for one encrpytion of a given string. This is due to PKCS #11 library being programmed in C by the card's manufacturer. The resulting length of the encryption is assumed to be of an undetermined size (for block ciphers and even RSA this can be pre-calculated given the input length). However the implementation runs the encryption twice, the first run is to calculate the resulting length of the encryption, and the second run saves the result in a buffer that has been pre-allocated the correct number of bytes in a char array.

This function only has only 2 main components (these are listed below).

1. Select key file (B.7 - command 52)
2. Encrypt data (B.7 - command 53)

The key file is selected and then the encryption APDU command is called with the string in the data field. As explained in destroyObject function analysis, despite the key file location being revealed here, it does not present a vulnerability. The key file can neither be opened nor reveal the value of the key. Instead it provides pointers to other locations in the smartcard's memory that hold the key value securely. It is reasonable to assume (though unable to test or verify) that only the smartcard's operating system has access to these locations.

Therefore we find no evidence of other vulnerabilities that can be exploited.

6.8 C_decrypt

[For this trace we use the same triple DES key to decrypt the message we previously encrypted]

Function dependencies → [Initialization, Login, FindObjects]

The decrypt function operates nearly in an identical manner to the encrypt function. The only difference is that it simply changes one byte (P1 in the APDU command header) in command 65. The key is loaded and then the decrypt APDU command is sent, with the encrypted string placed in the

data field. For the same reasons stated in the encrypt function, the APDU command is sent twice.

It also has 2 main components (these are listed below).

1. Select key file (B.8 - command 64)
2. Decrypt data (B.8 - command 65)

As the decrypt function operates in a similar fashion to the encrypt function, we find no vulnerabilities here either. However we did find a similar implementation flaw that has been reported in earlier work [?] regarding decryption using an AES block cipher key in ECB mode. For an unknown reason the first 8 bytes are removed from the encrypted data. This only decrypts and returns the second 8 bytes. As this does not present a security vulnerability, but it is rather an issue with the functionality of the decrypt function. Therefore for this reason we do not analyse it any further.

6.9 C_setAttribute

[For this trace we modify the label of the triple DES key from 'des3' to 'changed']

Function dependencies → [Initialization, Login, FindObjects]

Similar to encrypt and decrypt, this function has only 2 main components. The first one selects the attribute file, and the following 2 commands update the whole file to include the new name of the label to the key. The components are listed below:

1. Select key attribute file (B.9 - command 51)
2. Modify attribute file to change the label name from 'des3' to 'changed' (B.9 - commands 52,53)

Command chaining (see section 2.2) was used as the number of bytes within the attribute file exceeded the limit of 256. Command chaining is indicated by the use of '90' instead of '80' as the CLA byte. This was seen in the 'REALLOCATE BINARY' command that was used to modify the attribute file.

We notice that at the APDU layer attributes can be changed via the 'REALLOCATE BINARY' command. This is a vulnerability that was discovered and tested in previous work [?]. With the main focus on changing the attribute

values of CKA_sensitive and CKA_extractable from true to false and false to true respectively. The change in this direction is not permitted by the PKCS #11 standard. However, this was conducted at the APDU layer. The tests carried out in the literature found that due to the split between attribute files and key value files, the key value could still not be extracted despite the modification to the attributes.

Thus, despite the vulnerability being present it does not appear to cause a major security issue, in the sense that the key value is still stored securely. We discuss this further in chapter 7.

6.10 C_unwrap

[For this trace we used an RSA public key to encrypt a triple DES key value = 12345678. With its encryption saved, we then create a template for the key attributes and used the RSA private key to unwrap the key value and template, to save the key on the card]

Function dependencies → [Initialization, Login, FindObjects, Encrypt]

There are 10 main components involved in the unwrapping of a key. 2 different secure messaging sessions are used (just like in the generateKey function). The first secure messaging session is used to unwrap the encrypted key into a temporary file. The second secure messaging session is used to save the triple DES key into a key file. This follows after the attribute file has been created in between these 2 secure messaging sessions. The order of the components are listed below:

1. Open secure messaging (B.10 - command 92, 93, 94)
2. Select a file [the file location is encrypted and therefore unknown] (B.10 - command 95)
3. Unwrap key and attributes (B.10 - command 96)
4. Close secure messaging (B.10 - command 97)
5. Select key attribute file (B.10 - command 119)
6. Add key attributes to file (B.10 - command 120, 121)
7. Open secure messaging (B.10 - command 122, 123, 124)
8. Select the key file directory (B.10 - command 125)
9. Create key file (B.10 - command 126)
10. Close secure messaging (B.10 - command 127)

In a scenario where we are not already in the knowledge of the key value, the secure messaging sessions are the main vulnerability that we will aim to try and exploit. While the key is not exported at the APDU layer in plain text. Successfully reverse engineering the secure messaging protocol, would allow an attacker to exploit it. This can be achieved by injecting our own session keys and decrypt the key value. This was discussed in analysis of the generateKey function.

Furthermore, an attacker can also modify the attributes as the key is saved within the card for the first time. But as discussed in the generatKey function analysis, this attack would make the user aware of the fact that the key has been exported and the attributes have been changed. This cannot be deemed as serious a vulnerability compared to one that would expose the key in plain text at the APDU layer without the user's knowledge. These possible vulnerabilities are discussed further in chapter 7.

6.11 C_wrap

As discussed in section [?] of the literature, the wrap/decrypt attack is carried out at the API level whereby one key can be given wrapping and decryption functionalities within its attributes. This allows for any key (regardless of the attributes) to be wrapped, exported out of the card, and then decrypted by the same key. This would reveal its key value in plain text. This vulnerability bypassed all controls that the attributes of a key are expected to provide. Due to this vulnerability discovery, it is reasonable to assume that 'Athena smartcard solutions' decided to remove the ability to wrap keys completely in an attempt to prevent this type of attack. This reasoning is based on the fact that requesting any key to wrap another key at the API level causes an error: *CKR_Function_Not_Supported*.

From the analysis of table 2.1 (see section 2.2.4), we find a correlation between the P1 and P2 parameters for the APDU commands for encryption, decryption and unwrapping. These values are tabulated below:

P1	P2	Command
82	05	encrypt
80	05	decrypt
80	0A	unwrap
?	?	wrap

The CLA and INS bytes are identical for all of the above commands.

Given the fact that for encrypt and decrypt, P2 remains the same and only P1 alters from '82' to '80', we infer that if the wrap command still exists at the APDU layer, then the values for P1 and P2 would be **82** and **0A**

respectively. Therefore, an example full command would be: **00 2a 82 0a L Data 00**. Were L is the length of the data, and the data is the key to be wrapped.

We ran a simple test to see if this command operates on the card:

1. Wrap key (string ’12345678’) (B.11 - command 1)

The result of the test shows an error at the APDU level. The error value was ’6A 80’, meaning the parameters within the data field are incorrect. This suggests that the command does exist and most likely corresponds to the wrap function.

The data field of unwrap is encrypted, as it is used within secure messaging. However if this was reversed engineered and we are able to understand the data field of the unwrap command, it will be highly likely that we would be able to create the correct data field for the wrap command. This would allow us to bypass all attribute controls of keys at the API level, and implement the wrap/decrypt attack at the APDU level. This in turn would allow all keys on the card to be exported out of the card in plain text. This would be a serious violation of the PKCS #11 standard.

Chapter 7

Attempts To Attack At the APDU Level (change)

In this chapter we explain the motivations, implementation and results of the attacks we attempt as the APDU level. For clarity purposes we first give a brief overview of the API's function analysis at the APDU layer that we conducted within chapter 6. This analysis in combination with previous work completed on the same smartcard and for that matter other smartcards as well, drove the motivations behind the attacks we selected to conduct.

7.1 Motivations

List a table of all attacks, tick and cross them, then go on to explain them in the following two sub sections.

7.1.1 Vulnerabilities not investigated

The *initialization* function did not present any vulnerabilities that would be able to be exploited and was just needed in order for the API to correctly operate.

We did mention in the *Login* function analysis it might be possible to either use 'REALLOCATE BINARY' or 'DELETE FILE' and 'CREATE FILE' in order to modify the parent directories file control parameters for the PIN file that is stored on the card. The aim of this would be to change the retry counter before it reached 0, which would cause the card to be in a blocked state. If this is possible then an unlimited number of attempts at guessing the users PIN would be possible. We decided to not focus on this attack as it would not be particularly useful without the knowledge of the challenge-response algorithm that is used to verify the users PIN.

FindObject function did present the possibility to locate the object and at the APDU level modify any attribute. The two that have the most significance would be CKA_extractable and CKA_sensitive. This had been completed within previous work [?] and revealed no significant vulnerability. This was due to the fact that once the attributes had been set as part of the key generation process, the attribute file and key key file are separated and thus prevent the key from being exposed in plain text. Since this attack had already been investigated we did not see it fit to investigate it any further.

GenerateKey and *GenerateKeyPair* functions both provided a vulnerability whereby attributes could be modified on the fly as the key was being generated. This did leave the possibility of the key value being exposed as the key had not been generated yet. And effectively the card would be told to generate the key with the attributes the attacker sets. This vulnerability would always have to be exploited upon key generation and would be visible at the API level and therefore the user of the smartcard. Therefore while it is a viable attack to investigate, we decided to look into methods that would allow all keys to be exported in plain text regardless of whether or not the key was already generated and saved on the card.

Encrypt, Decrypt, GenerateKeyPair (private key) and *DestroyObject* functions all exported in plain text the directory of the key file. As we discussed in chapter 6 (destroyObject analysis) we thought that this might have provided the ability to open the file and read the key value in plain text. This hypothesises turned out to be incorrect, and instead the files stored data wrapped in an ASN.1 BER encoding, that had pointers to locations within the cards memory where the key values are stored securely (non-readable to outside actors).

Unwrap function did not provide any major vulnerabilities either. Instead we would like to know exactly how the data field is built, but this cannot be inferred from traces as the command is only ever used within secure messaging. Therefore the data field is encrypted. The reason for wanting to understand how the command works is explained in the next section.

7.1.2 Vulnerabilities to investigate

From studying the traces for the ***login*** function it was apparent that a challenge-response algorithm has been implemented in an attempt to provide a secure method for transporting the PIN for verifaction by the smartcard without ever revealing its value in plain text (which would be against the PKCS #11 standard). This is an improvement implementations from other card manufactures whereby they did just send the PIN in plain text over the APDU layer for the smartcard to verify it, leaving an attacker the ability to simply sniff the communication and extract the users PIN number. However the challenge-response algorithm that 'Athena smartcard solutions' have implemented on this card still leaves the potential for an

attacker to reverse engineer the protocol, and then brute force the PIN until the same response is calculated to be sent to the card. Doing this would give the attacker the knowledge of the users PIN and also remove the first dependency on the use of the API by an attacker. This is therefore the first attack that we investigate.

Furthermore the use of secure messaging to encrypt and also provide an integrity check (that the commands have not been altered in transit) in the form a a message authentication code being appended to the command was found within two functions. These are ***GenerateKey*** and ***unwrap***. Successfully reverse engineering the secure messaging protocol would allow for two objectives to be complete. Firstly within the generateKey function an attacker would be able to implement a man in the middle attack to either alter the key being generated with a value known to the attacker, or simply decrypt the key to find out its value. This would be possible regardless of the attributes set for the key, and therefore would violate the standard to a large extent if the attributes for CKA_sensitive and CKA_extractable were set to true and false respectively. As an attacker would be able to be in the knowledge of the block cipher keys generated when the card should be the only system with said knowledge.

Secondly is the understanding behind how the ***Unwrap*** APDU command operates. As stated in section 6.11, despite the API removing the ability to use the ***Wrap*** functionality (most likely due the wrap/decrypt attack discovery [?]), with a high probability the APDU command still exists as we managed to send what we believe to be the correct command and received back an error stating the data field parameters were incorrect. Therefore reverse engineering the secure messaging protocol would allow us to fully understand the ***Unwrap*** command data field, and possibly then be able to generate the correct command for wrapping a key. If this is possible, then the wrap/decrypt attack could be implemented at the APDU level, bypassing all attribute controls and exporting any/ all keys that are stored within the card. This would be considered a serious vulnerability.

All the vulnerabilities stated within this section are also all to be implemented at the APDU layer and would not provide the user the knowledge the attack is in process. This is the main reasoning behind selecting these vulnerabilities to investigate, and the fact they are far more serious compared to the others mentioned, in terms of violations of the PKCS #11 API.

7.2 Reverse Engineering PIN Authentication Protocol

The first attack we investigate is the reverse engineering of the PIN authentication protocol. If this is successful an attacker will be able to brute

force and/ or use a dictionary attack to calculate the users PIN. It shall also remove one dependency on the use of the API. As of right now the API is required to login into the smartcard as we are unaware of how the challenge-response algorithm is implemented. Figure 7.1 shows the procedure for logging into the smartcard at the APDU level.

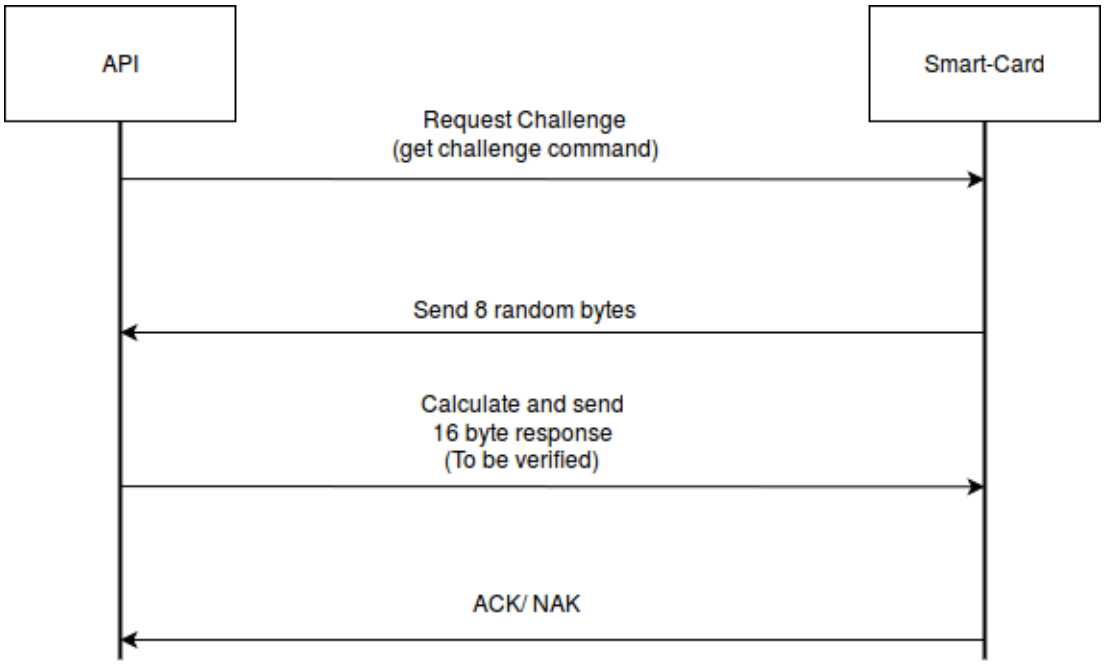


Figure 7.1: Challenge Response Procedures

The API requests (via a 'GET CHALLENGE' command) the smartcard to generate and send back 8 random bytes. The API then calculates a 16 byte response based upon the knowledge of the challenge (8 random bytes) and the users PIN. The 16 byte response is sent to the smartcard (via a 'VERIFY' command), which will return one of two possible outcomes:

1. '90 00' $\rightarrow$ Verification of users PIN succeeded
2. '63 CX' $\rightarrow$ Verification of users PIN failed. (X = number of attempts left before the card is blocked)

The following sections give 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 smartcard. 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 ***truncate(hash('example'))***, means the string, 'example', 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)
MD5	128	16
SHA1	160	20
SHA256	256	32
SHA384	384	48
SHA512	512	64

Table 7.1: Hash Functions *(supported by the card)*

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 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. Some of 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). We found that 16 characters was the upper limit to what a user can set the PIN to be by changing the PIN and incrementing its length until the API returned an error.

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 a given challenge and PIN there is only one corresponding response that API will calculate

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 authenticating a user on a website 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 experiments to generate 16 bytes, given X and the PIN.

[We denote || as the concatenation 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 did not result in finding a match to the response generated by the API. 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 bytes calculated by the API we decided to look at the password-based key derivation function.

PBKDF2 was created as part of PKCS #5 by RSA laboratories [?] and is starting 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 smartcard 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 i number of bytes, therefore the only truncation method used was 'first_16' (table 7.3).

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

- PBKDF2(hash_function, PIN, X, number_of_rounds)

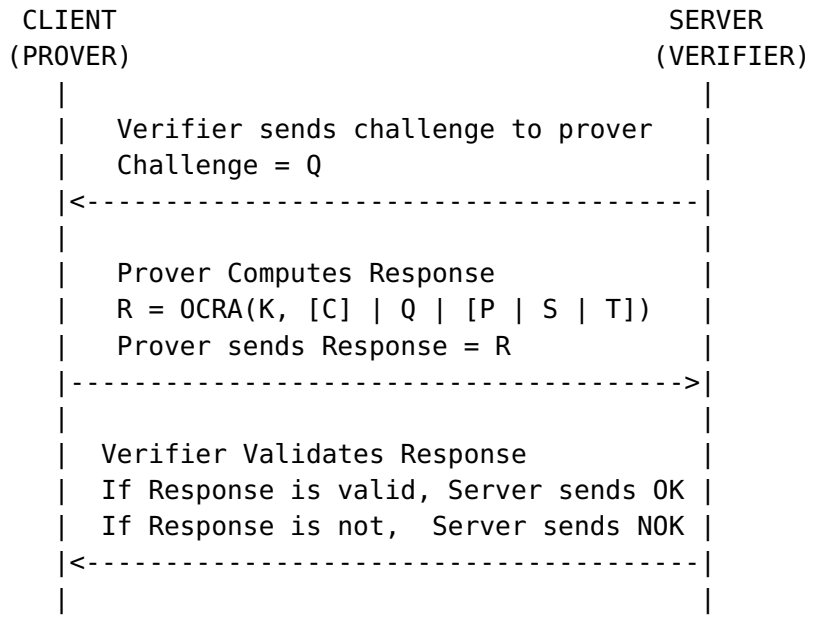
This generated 100,000 experiments per hash function. With 5 hash functions, this resulted in half a million tests being run. Due to the sheer computational power required for this search I decided to parallelize the search based on the hash function, and run them on separate cores of a server. Even 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 challenge-response algorithm, 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 smartcard. We have extracted out the protocol and provided it below:




```
C - Counter, optional.

Q - Challenge question, mandatory, supplied by the verifier.

P - Hashed version of PIN/password, optional.

S - Session information, optional.

T - Timestamp, optional.
```

In the previous section of the paper they explain that OCRA in the instance of this protcol stands for a generic variable to state which form of HOTP is to be used. HOTP (see section 3.5.1 of this paper) can take on different formula's due to using different hash functions. The generic formula is:

$$HOTP(K,C) = Truncate(HMAC(K,C)) \quad \&\& \quad 0x7FFFFFFF$$

where K and C stands for the password (secret key) and is a counter (the challenge) respectively.

We ran experiments using the 5 hash functions listed in table 7.1, and even inter-changed K and C to have each others values instead. This did not result in finding a matching response to the one the API had calculated.

Search 4 - TOTP

With the above in mind, I also wanted to rule out the possible use of *time based one time passwords* (see section 3.5.2). TOTP algorithms are rarely

used within smartcard's due to the complexity of syncing the clocks between the computer and the smartcards processors, and then also accounting for any lag. Despite the low probability of it being used it was worth checking.

This was completed by halting communication between the smartcard 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 verification, 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 that possibility.

7.2.3 Authentication Protocol Search 3.0 (Triple DES Encryption)

With none of the above searches concluding any useful results, and instead just ruling out them as possibilities we moved onto checking an assumption that we were not 100% sure was accurate. This would have been done earlier, however it had taken a while to get the code for the man in the middle attack to be operational, and was required for this. The assumption in question was, that *for a given challenge and PIN, the API will calculate only 1 corresponding response*.

To verify this assumption we used the man in the middle tool to inject our own challenge (that the card normally sends) and set it to '00 00 00 00 00 00 00 00'. Completing this multiple times, we noticed that this assumption was in-fact wrong. Over multiple attempts we saw different responses being calculated by the API, despite the PIN and the challenge being the same. However we also noticed that logins with quick successions of one an other had the same value for the response when the same PIN and challenge were present, and in some cases only the last 8 bytes changed with the first 8 bytes remaining the same.

Due to these observations we decided to run multiple experiments to find characteristics of the 16 bytes response *(which we now believe to be split into two 8 byte parts)*. We did this by logging into the card twice each time, and altering the three parameters *(one at a time)* that we determined to cause the response to alter. The parameters we alter are ***time, PIN*** and the ***challenge***. The results of the experiments are tabulated below:

*[To make the explanation of this clearer we have placed the actual traces of these experiments within **appendix A**. And use variable names for the two 8 byte sections of the 16 byte response. These are denoted below as A and B]*

16 byte response	1st 8 bytes	2nd 8 bytes
Y =	A	B

Time	PIN	Challenge	A	B	Appendix
Different	-	-	χ	χ	A.1.1
Same	Same	Same	✓	✓	A.1.2
Same	**Same**	**Different**	✓	χ	**A.1.3**
Same	Different	Same	χ	χ	A.1.4
Same	Different	Different	χ	χ	A.1.5

Table 7.5: Multiple logins with different parameters

Table 7.5 shows that changing the second the separate logins occur within, or changing the PIN causes the whole 16 bytes to alter. If both login attempts are within the same second, and the same PIN and challenge are given to the API then the exact same 16 byte response is calculated. However the most interesting characteristic discovered within these experiments is when the logins occur within the same second, with the same PIN but different challenge. This causes the first 8 bytes to remain the same, but the second 8 bytes to differ, within the 16 byte response.

This is the characteristic that provides the evidence to support our theory that the 16 byte response is in-fact split into to 8 byte sections. This also shows that that the second 8 bytes is a function that utilises the challenge. As a different challenge causes it to change. Based on these facts we assume the following:

1. The first 8 bytes is a random number N_A = Nonce API, which is seeded with time (in seconds)
2. The second 8 bytes is a another number that verifies the knowledge of both the challenge and N_A. We denote this as V_{AC}
3. A block cipher (see section 3.3) is used to encrypt both of these numbers, to send over to the card for PIN verification
4. A function of the PIN, is the password to the block cipher
5. The PIN is still of maximum length 16 bytes
6. The PIN is still in ASCII format

To sum up we assume at this stage the challenge-response algorithm uses a function of the PIN to be the password to a block cipher which encrypts a message in this format:

$$block\_cipher\_encrypt(N_A||V_{AC})$$

The block ciphers that are supported by the card are listed below:

Block Cipher	Key Size	Block Size
AES	16	16
DES	8	8
Triple DES *(key option 2)*	**16**	**8**
Triple DES *(key option 3)*	24	8

The use of the block cipher AES is ruled out as a possibility as if it was used the characteristic that we found would not be present. A change to the challenge, within the same second and given the same PIN would result in a different encryption of both 8 bytes because AES would operate over the entire 16 byte block. This leaves only DES and triple DES as a viable option.

We eliminated the possibility of standard DES being used due to its key size. It would not make logical sense to use a function of the password and then truncate it down to 8 bytes from 16 bytes. This would drop the security of the login from 256 bits to 128 bits for no reason, as other options are available.

This therefore only left us with triple DES key options 2 and 3. With the fact in mind that the PIN for the user is of maximum length 16 bytes we decided to focus on key option 2 which has 16 byte password. There is a possibility that the password is hashed for example and 24 bytes are used with key option 3, but we decided to first test key option 2 before moving onto 3 in the case that 2 fails.

Due to the assumption that the second 8 bytes is a verification (possible join, or hashing) of both the cards challenge and N_A, then we could not simply just decrypt the 16 byte response and analyse it. As both number could be completely unknown to us, therefore we would have no method of verifying we have the correct protocol. To mitigate this we decided to get 2 different responses, within the same second and with the same challenge, but using different PIN's. If the password to the block cipher is in-fact a function of the PIN, then the correct decryption of both responses with different passwords should evaluate to the same 16 bytes. This is how we will verify if we have found the correct protocol.

Search 5 - Triple DES *(Key option 2)*-ECB

Table 7.6 shows the data for two logins that occurred within the same second, with the same challenge (that we injected), but using two different PINs. We then generate 6 different possible passwords for the encryption using the PIN. The first is simply the PIN in ascii format at 16 bytes (all zeros). The following 5 are the PIN hashed with the hash functions in table 7.1, and truncated only using the first_16 truncation method within table 7.3. Using the generated passwords we decrypt the 16 byte responses using triple DES key option 2, and list the results in table 7.7.

$Response_1$	93 57 F7 53 42 A3 27 3D E6 B8 7E A6 81 98 5B 1C
PIN_1	0000000000000000
$Response_2$	90 6A 74 D1 BD 2C 75 2E 52 bA 17 87 E3 70 51 EF
PIN_2	1111111111111111
$Challenge_{1,2}$	00 00 00 00 00 00 00 01

Table 7.6: Experimental setup

Pin Method	Response	Decryption using Triple DES-ECB
PIN_1	1	A4E8604FFFEABBC7 A85FDCA6E1D8187F
PIN_2	2	23479D5F8CA2EAB7 AE0EB9D09C987484
$MD5(PIN_1)$	1	C4B103E761F3688A 45681FD6B9F7137D
$MD5(PIN_2)$	2	404F5882AFE4600D E8E8C4F918E0C55B
$SHA1(PIN_1)$	1	**FE5DBEA8ECAB2F86** 9357F75342A3273C
$SHA1(PIN_2)$	2	**FE5DBEA8ECAB2F86** 906A74D1BD2C752F
$SHA256(PIN_1)$	1	5F6D92B899EF4557 A1B797A4F65D5B30
$SHA256(PIN_2)$	2	74CED585EABBD62E B847856F6A3AEBC7
$SHA384(PIN_1)$	1	DFF3A6A88508D7CF 802CF22A15E4051C
$SHA384(PIN_2)$	2	A75AB9FEE2A2B445 54618384E33CA723
$SHA512(PIN_1)$	1	473EF6CC4BA45EF7 EA740F801A595B4A
$SHA512(PIN_2)$	2	1C202A831098546D 8EF674B4A0B0D7AD

Table 7.7: *Decryption Results*

The results are interesting. While we did not find an exact match of the entire responses when decrypting both of them, we did find a match for the first 8 bytes. This suggests we have found the correct formatting of the PIN, but the wrong decryption methodology. A change in the second 8 bytes suggest cipher block chaining is used. Hence we move onto search 6.

Search 6 - Triple DES *(Key option 2)*-CBC

Triple DES block cipher using cipher block chaining (CBC) requires an initialization vector (IV) to be used (see section 3.3). The IV is 8 bytes in length. Common practice is to send this in plain text along with the encryption of a message. However it is quite clear that this does not occur. The only other possibility is that the card and the API already have agreed upon a value to use, and always use this same value. The most common practice is too use an IV = 00 00 00 00 00 00 00 00. Despite it being a bad an insecure practice, this still does occur. Hence, in the following experiment we only test the pin method for $SHA1(PIN_{1,2})$ and decrypt the corresponding responses using triple DES-CBC with an IV of zeros.

Pin Method	Response	Decryption using Triple DES-ECB
SHA1(PIN)	1	FE5DBEA8ECAB2F86 0000000000000001
SHA1(PIN)	2	FE5DBEA8ECAB2F86 0000000000000001

Table 7.8: Decryption Results

Table 7.8 lists shows the results of both decryption's. The result is better than what we had originally assumed. The second 8 bytes, which we thought would have been a verification number (join or hash) (showing knowledge of both nonces), is in fact just the cards challenge. This almost completes our attack. As now we can produce a 16 byte response, given the cards challenge, the PIN and a nonce to place at the start of the message to encrypt.

The issue is we are not sure if we can just place any 8 byte nonce at the start of message to be encrypted. As the API might be expecting some sort of counter, and might refuse invalid values. Hence we conduct the final stage of this attack to test if we can place any value there, and hence will have fully reversed engineered the PIN authentication protocol.

To sum up so far this is the protocol we believe to be in place:

- Request and get an 8 byte challenge from the card, denoted $\mathbf{N}_c$
- Password for encryption = Truncate_First_16(SHA1 (PIN)))
- Block cipher used = Triple-DES-CBC
- Initialization Vector = 00 00 00 00 00 00 00 00
- Calculate the response as the following:

$$Encrypt(N_a||N_c)$$

- were $\mathbf{N}_a$ is a nonce determined by the API.

We set N_a to equal 00 00 00 00 00 00 00 00 00 00, calculate what we believe to be a valid response and send it the the API. This experiment is shown within appendix A.2. The result of it is an APDU response of '90 00', successful login!

Thus concluding this attack.

7.3 Reverse Engineering Secure Messaging

The second attack we investigate is the reverse engineering of the secure messaging protocol used by the card. As stated within section 2.2, and within public 'Athena smartcard solutions' security policies [?] for several different cards the vendor produces, the secure messaging initialization is

used to derive 2 session keys. S_{Enc}, S_{Mac} are used to encrypt and compute a message authentication code (MAC) of the messages within a secure messaging session respectively. This provides confidentiality and data integrity of the messages. The decision for how to derive these keys is left entirely upto the card vendor. Which makes this particularly difficult as there are several methods that could be used.

The secure messaging sessions have seen to be used within the ***GenerateKey*** and ***Unwrap*** in the analysis conducted within chapter 6. And also seen to be used within ***CreateObject*** seen in the analysis conducted by an MSC student (place name here) last year [?]. In every case it is used to provide confidentiality of the CKA_value of block cipher keys. Successfully reverse engineering the secure messaging protocol would allow an attacker to use a man in the middle attack to derive two sets of session keys (one with the API and one with the card) which would allow the correct keys to be sent to the card, but also allow the attacker to decrypt them and find out the value is said keys. This would violate the PKCS #11 standard if the keys were set to be sensitive and un-extractable. In addition, it would also be an attack whereby the user would be completely unaware it had taken place. This is considerably worse than previously analysed attacks were there is a large possibility of the user finding out, and therefore changing the keys before an attacker is able to utilise them for malicious purposes.

We also hope to be able to learn how the ***unwrap*** command data field is built and then build the correct command for ***wrap*** at the APDU level. This should be able to be completed by decrypting the data field of the unwarp command when used via the API. With this knowledge it should be possible to implement the wrap/decrypt attack [?] at the APDU level, and will therefore allow the exporting of all keys including RSA private keys which are generated on the card and thought to be securely stored.

To begin with we use the API to call the function ***GenerateKey***, as we know that secure messaging is utilised within this function. This gives us an overview of how the secure messaging protocol works. Below is a diagram that visually shows what exactly occurs at the APDU level. (The actual APDU commands are stored within appendix A.3.1)

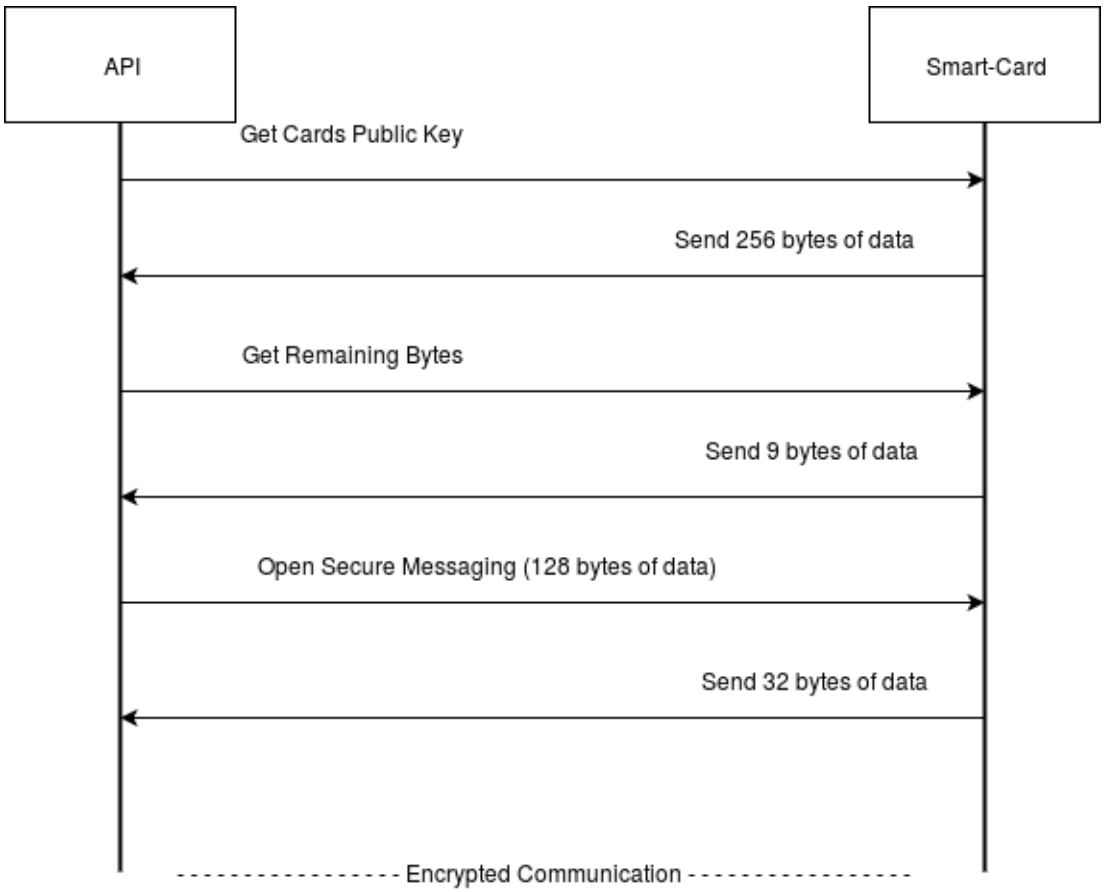


Figure 7.2: Secure Messaging Initialization

As shown in figure 7.2 the API first requests the public key of the card. In total (split between two messages due to the size exceeding 256 bytes) the card sends back *265* bytes worth of data, which is its public key. The API then proceeds to send back to the card 131 bytes worth of data *(from multiple traces we have found that these 131 bytes of data change every single time, a new secure messaging session is opened)*. Finally the card sends 32 bytes of data, which concludes the initialization of secure messaging. Now the card and the API are both aware of two sets of block cipher keys S_{Enc} and S_{Mac} which are used to encrypt all of the following communication.

The first step we took was to analyse the raw bytes within the data field that are being sent over.

7.3.1 Raw byte analysis of the data fields

From multiple traces whereby we opened a secure messaging session we found 9 bytes within the data sent from the card → API that remained constant. This made us think the data might be wrapped in an ASN.1 BER

encoding. Using the online tool [?] we had utilised before (within chapter 6) we found 3 distinctive sets of data within the 265 bytes.

Send 265 bytes of data (cards public key) (A.3.1 - commands 24, 25)
[00] (1 byte) 05
[01] (128 byte) F7 B5 15 72 07 22 94 6F C4 08 ...
[02] (128 byte) 3C 52 D2 06 89 28 92 2C AB E6 ...

These command bytes in the BER encoding of the data have been highlighted in bold within the commands in the appendix to make it clear how the data is split.

We did not find any of the other data fields wrapped in an ASN.1 BER encoding. Thus now we know the following:

Data	Explanation
Cards public key (static key)	(1 byte) = 05 (128 bytes) = F7 B5 15 72 07 22 94 6F C4 08 ... (128 bytes) = 3C 52 D2 06 89 28 92 2C AB E6 ...
API data *public key or encrypted data* (Not static)	(128 bytes) = 08 9F EA A1 DC 8F C3 43 ...
Card random challenge (Not static)	(32 bytes) = 66 56 36 31 16 42 8D 8A ...

Table 7.9: Data organization within secure messaging initialization process

Using the information within table 7.9 we now begin finding protocols that would allow session keys to be derived, and also match the amount of data being sent across from the API to the card and vice versa.

7.3.2 Protocol search

There are three main protocols that would support session keys being derived that would remain a secret between the API and the card. These are: RSA encryption, elliptic curve cofactor diffie hellman (ECCDH) and diffie hellman (DH) [see section 3].

Within security policies for 'Athena smartcard solutions' [?] it states that ECCDH is used for some of their smartcard's. And that the 32 bytes of random data sent by the card is used to XOR with the shared secret derived from ECCDH to create the 2 session keys.

[The same documentation stated that the shared secret dervied from ECCDH was 32 bytes in length, and the session keys were 2 x ***Triple DES (key option 2)*** *16 byte keys]*

The card we analyse supports the key generation of elliptic curve public and private keys for this exact purpose, and for signing and verifying data. Thus the first protocol we look into is ECCDH.

Elliptic Curve Cofactor Diffie Hellman

The smartcard we analyse only supports curves P-192, and P-521 for ECCDH [see section 3]. The security polices state that curve P-256 is used. This is our first indication that the smartcard we analyse is not using ECCDH. However we continue with the investigation in case one of the above curves is used.

The first step we take is to calculate the number of bytes that are required in order to setup the global parameters that must be sent from the card to the API for ECCDH, and also the number of bytes of each public key. A match between these calculations and the number of bytes that are in fact sent across could suggest this protocol is being utilised for secure messaging initialization. *We used OpenSSL to generate the following different public and private keys, along with the global parameters and outputted them in DER format to find the length of the corresponding data.*

Elliptic Curve Field	P-192	P-256	P521
Global Parameter Length	233	233	233
Public Key Length	48	64	131
Total	**281**	**297**	**364**

Table 7.10: Number of bytes required for ECCDH [Card to API]

As shown in table 7.10 non of the curves for ECCDH have a value of 257 bytes in total that would be sent from the card to the API to begin the protocol. None of the public key sizes match 128 bytes which is what the API returns to the card as well. Thus we conclude here that ECCDH is not used and move onto the possibility that RSA is used.

RSA Public Key Encryption of Block Cipher Keys

If RSA is used the only method of deriving keys between the API and the smartcard is for the API to use the public key of the smartcard to encrypt session keys created by the API. The card will then use its private RSA key to decrypt the session keys. Considering that there is 32 bytes sent by the card at the end of the secure messaging initialization process, it is likely that they are XOR'd with the bytes encrypted via RSA, to generate the final session keys to be used.

We will complete the same analysis that we did for ECCDH and calculate the number of bytes required to see if they match the corresponding bytes

sent via the API and the smartcard. These are tabulated below:

RSA	1024-bit	2048-bit
Public Modulus length	128 bytes	256 bytes
Public Exponent length	1 byte = 5	1 byte = 5
Encryption length of 32 bytes	128 bytes	256 bytes

Table 7.11: Number of bytes required for RSA

As we can see from table 7.11 the key option of 2048-bit fits the size of the public key perfectly for the smartcard's public key *(This is on the basis that the public exponent is set to 5)*. However the encryption of the 32 bytes by the API would result in 256 bytes not 128 bytes. This therefore rules out the possibility that RSA-2048 is used, leaving the other possibility to two RSA-1024 keys.

Literature has shown that some manufactures have implemented cards with multiple asymmetric keys, and certain versions of the PKCS #11 API that the card vendor implements will only use one of them. Thus the first test we ran was editing the first bytes of each 128 bytes to see if one or both of the suspected keys were used. For each test we ran, the secure messaging initialization completed, but then upon the first instance of encrypted communication an APDU error was raided 'error with secure messaging'. Thus we found that both 128 byte keys must be used.

The only case that this could be (assuming RSA-1024 is the protocol) is if the session keys were encrypted twice with the two different keys, and both keys have a public exponent of 5. This seemed fairly unreasonable, as in any case two RSA-1024 bit keys are no more secure than just using one. And it would of made more logical sense to implement RSA-2048.

However we continue to check if RSA is in fact used. The next test we run is changing the public exponent. Due to how messages are encrypted with RSA, setting the public exponent to 1 should reveal the message itself, and setting the public exponent to 0 should return a value of 1. This is verified by the following formula:

$$ciphertext = message^{e} \mod n$$

Where e is the public exponent, and n is the public modulus (128 bytes).

	public exponent = 0	public exponent = 1
cipher text *(128 bytes sent from API to card)*	**0**	**1**
Appendix	A.3.3	A.3.2

Table 7.12: Results from altering public exponent

As shown within table 7.12 what we would expect from RSA encryption did not occur. The value of the cipher text when the public exponent was 0 was 0, and 1 when it was 1. This should have been 1 and the value of the un-encrypted message respectively. This does however lead onto our final search for the protocol that is used, and fits the mathematics of the diffie hellman protocol exactly.

Diffie Hellman - Key Agreement Protocol

As stated within section 3 the diffie hellman protocol relies on the fact that the global parameters are known by both parties, then they each share their own respective public keys which allows them to derive a shared secret without ever revealing it in plain text. As a reminder this is the diffie hellman protocol:

Public Parameters
[G must be a primitive root modulo p]
Generator = G
Public modulus = p

From here private keys (a number between 1 and p-1) are selected by each individual entity and generate the public keys as follows:

$$public\_key_1 = G^{private\_key_1} \mod p$$

$$public\_key_2 = G^{private\_key_2} \mod p$$

From here a shared secret **S** can be calculated by each entity with the following formula:

$$S_1 = public\_key_1^{public\_key_2} \mod p$$

$$S_2 = public\_key_2^{public\_key_1} \mod p$$

$$S = S_1 = S_2$$

Thus if the generator were set to be **0**, then the public key calculated by the API would in fact be 0 as well. And in the case the generator is set to **1**, the API would calculate its public key as 1. This matches the properties described in table 7.12. Therefore we conclude due to the mathematical

properties presented that, the secure messaging initialization process is the diffie hellman protcol. Whereby the data sent during the initialization of secure messaging is the following:

Data	Explanation	DH Parameters
Cards public key (static key)	(1 byte) = 05 (128 bytes) = (128 bytes) =	Generator public key or modulo public key or modulo
API data *public key or encrypted data* (Not static)	(128 bytes) =	API's public key
Card random challenge (Not static)	(32 bytes) =	Used for session key derivation (not part of DH)

Table 7.13: Data organization within secure messaging initialization process

The 32 bytes sent at the end of the secure messaging initialization we assume to be used like those stated as part for ECCDH in the security polices we analysed. They will some how be utilised to generate the final session keys S_{Enc} and S_{Mac}. Later we show documentation that suggests methods for doing this. For now the next process is to implement a man in the middle attack for the diffie hellman key agreement so that an attacker is also in the knowledge of the shared secret.

7.3.3 Man in the middle attack - Diffie Hellman Key Agreement Protocol

The standard man in the middle attack for diffie hellman would require two diffie hellman key agreements to occur. One between the API and the attacker, and one between the attack and the smartcard. From the tests run (when we thought the public keys were RSA based) we noticed we can force the API's public key to be 0, by setting the generator to be 0. However we can also force the shared secret to be zero, if we alter the cards public key to be 0 as well. Then the card, the API and the attacker would all be in the knowledge of the shared secret which is simply just zero.

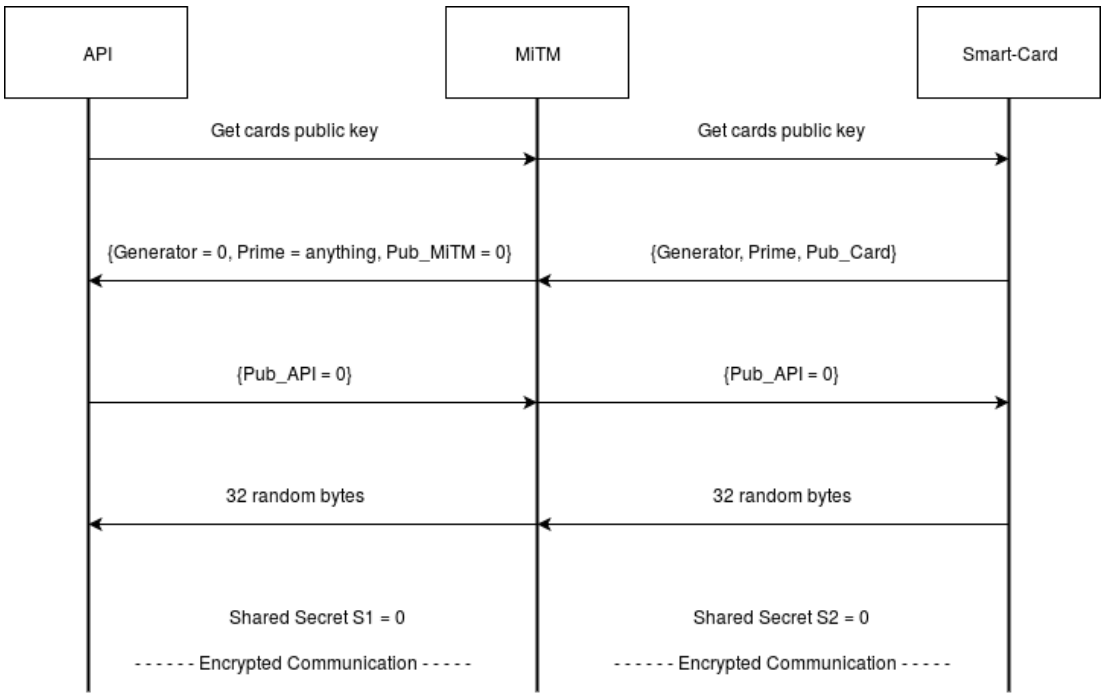


Figure 7.3: Diffie Hellman - Man in The Middle Attack

This is verified by the mathematics behind the diffie hellman protocol.

	API	Smartcard
Generator	0	5
Modulo	p	p
Public Key	$0 = 0^{private\_api} \mod p$	$x_{actual} = 5^{private\_card} \mod p$ $x_{injected} = 0$
Shared Secret $pub_1^{pub_2} \mod p$	$x_{injected}^{private\_api} \mod p$ $\mathbf{0} = 0^{private\_api} \mod p$	$public\_key_{API}^{private\_card} \mod p$ $\mathbf{0} = 0^{private\_card} \mod p$

As we did not know which 128 bytes was the modulus, and which was the public key of the smartcard we had to complete two experiments setting each of the 128 bytes to be zero to test our hypothesis and also setting the generator to be 0. These two tests can be seen within appendix **A.3.4** and **A.3.5**.

As can be seen the first test caused the API to halt. This suggested that this was the modulus and that by setting it 0 caused OpenSSL to try and create a public key with a modulo value of zero which is not should cause OpenSSL to crash. The second test succeeded!

- - continue here - -

Next task if KDF(S) and then re-computing the MAC for 'get challenge' command. Once we have done that we know how to calculate the session

keys which will conclude the reverse engineering!

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. Decrypt and understand the unwrap command
3. Test and get the wrap command to work (hopefully)
4. Implement the wrap/decrypt attack at the APDU level.
5. Try using ’reallocate binary’ and/or ’delete file’ then ’create file’ (must be done with caution! could brick card) to alter the retry counter for logging in. (Then a successful login trace is not required!)

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 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.1.2 Same Pin, Same Challenge

```
----- 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.1.3 Same Pin, Different Challenge

```
----- 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 Different Pin, Same Challenge

```
----- 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.1.5 Different Pin, Different Challenge

```
----- 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.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.3.4 First 128 bytes set to zero and 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 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 82 81  80 3C 52 D2 06 89 28 92  .........<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)

(Halts here)
```

A.3.5 Second 128 bytes set to zero and 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 00 00 00 00 00 00 00  D#.{‘g..........
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 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)
y
```

```
00000000: 00 00 00 00 00 00 00 00  00 90 00                 ...........
response changed!

----- 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: 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.4 Overriding Attribute Controls

A.4.1 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

```
----- APDU command/response pair 26 -----
00000000: 80 48 00 80 00                                        .H...

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.


----- APDU command/response pair 27 -----
00000000: 00 C0 00 00 09                                        .....

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


----- APDU command/response pair 28 -----
00000000: 80 86 00 00 80 84 7F A0  E7 6C 8F AA 50 9C C3 6E  .........l..P..n
00000010: 82 5E 84 B6 E4 F6 77 1C  45 FA AB 06 1B 24 C4 A8  .^....w.E....$..
00000020: 92 03 A9 9C A8 2B BE 1B  28 C4 57 83 A5 5E BB 8D  .....+..(.W..^..
00000030: D2 BF 3F D5 02 8A 7C 13  10 9C 75 06 91 1A 0F 05  ..?...|...u.....
00000040: 55 B4 C9 12 8A 69 59 B6  07 1D 67 F2 8A C9 FA BC  U....iY...g.....
00000050: F3 BE 16 73 51 C0 76 0C  11 E5 0C D3 8C FE 09 E5  ...sQ.v.........
00000060: 1E 52 DE 38 D9 AC 2D EB  C6 A1 C4 8E ED 03 7D 07  .R.8..-.......}.
00000070: 85 B7 FE 66 82 2F 03 65  94 DC 27 77 2B 3A 28 71  ...f./.e..’w+:(q
00000080: 97 08 5D 03 80 00                                 ..]...

00000000: F9 D0 66 F7 48 CB BB E8  CE 93 60 05 99 1B 81 2E  ..f.H.....‘.....
00000010: 73 0B B7 B8 DC 10 A7 84  B3 99 D8 C8 60 D6 48 5A  s...........‘.HZ
00000020: 90 00                                             ..


----- APDU command/response pair 29 -----
00000000: 0C 84 00 00 0D 97 01 18  8E 08 2B 88 7C 0C 8C 24  .........+.|..$
00000010: 00 1F 00                                          ...
```

```
00000000: 87 21 01 69 AB B7 01 F5  F5 8E EA B8 F3 09 D7 5E  .!.i...........^
00000010: F5 26 3C 7F 1D 15 90 B8  40 D4 A1 85 9C 57 3F 27  .&<.....@....W?'
00000020: 87 84 C6 99 02 90 00 8E  08 42 84 88 19 99 3B C2  .........B....;.
00000030: 10 90 00                                          ...


----- APDU command/response pair 30 -----
00000000: 80 86 FF FF                                      ....

00000000: 90 00                                            ..


----- APDU command/response pair 39 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 40          .....?.0.0..@

00000000: 90 00                                            ..


----- APDU command/response pair 40 -----
00000000: 00 D6 00 00 FA 01 03 03  40 01 23 18 00 00 00 00  ........@.#.....
00000010: 04 04 00 00 00 00 01 00  00 01 01 00 02 00 00 01  ................
00000020: 00 00 03 10 00 04 64 65  73 33 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 FF FF FF FF FF 00 11  01 00 18 FF FF FF FF FF  ................
00000070: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000080: FF FF FF 01 00 00 00 04  15 00 00 00 01 02 10 00  ................
00000090: 01 01 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 FF FF FF FF  ................
000000B0: FF 01 03 30 00 01 01 01  04 00 00 01 01 01 05 50  ...0...........P
000000C0: 00 01 01 01 06 00 00 01  00 01 07 50 00 01 01 01  ...........P....
000000D0: 08 50 00 01 01 01 0A 00  00 01 01 01 0C 10 00 01  .P..............
000000E0: 00 01 10 10 00 00 FF FF  FF FF FF FF FF FF 01 11  ................
000000F0: 10 00 00 FF FF FF FF FF  FF FF FF 01 62 50 00     ............bP.

00000000: 90 00                                            ..


----- APDU command/response pair 41 -----
00000000: 00 D6 00 FA 2D 01 00 01  63 00 00 01 01 01 64 00  ....-...c.....d.
00000010: 00 01 01 01 65 00 00 01  01 01 66 00 00 04 31 01  ....e.....f...1.
00000020: 00 00 01 70 00 00 01 01  80 10 00 00 01 00 88 03  ...p............
00000030: 88 03                                            ..

00000000: 90 00                                            ..
```

```
----- APDU command/response pair 42 -----
00000000: 80 48 00 80 00                                        .H...

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.


----- APDU command/response pair 43 -----
00000000: 00 C0 00 00 09                                        .....

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


----- APDU command/response pair 44 -----
00000000: 80 86 00 00 80 C3 88 FD  AF 64 0D 35 77 85 D4 20  .........d.5w..
00000010: 57 10 02 F4 1E 38 51 37  40 31 7F 7F 11 E8 4B 8D  W....8Q7@1....K.
00000020: A5 CE C0 50 EB 6B CE E6  E0 DE E8 34 7C FE 0B 6C  ...P.k.....4|..l
00000030: F0 70 9F E3 5D F7 AA 50  BB 1C F6 8C 00 1B 18 EA  .p..]..P........
00000040: BF 73 E4 BE 75 B6 AE 29  B1 A2 A3 B8 1D 52 FD 19  .s..u..).....R..
00000050: C9 CA 20 FB 80 C2 20 A9  E3 A6 15 6C 11 B3 E9 18  .. ... ....l....
00000060: 13 3F 65 02 28 21 74 72  29 EA E2 27 8B DA 3E 45  .?e.(!tr)..’..>E
00000070: 82 A1 B0 D9 A7 1A 3D F3  5D 4D 27 F4 D2 73 ED 0F  ......=.]M’..s..
00000080: A8 88 41 F2 4F 00                                 ..A.O.

00000000: 14 8C 30 9E D5 10 25 B1  F7 AF 07 E7 25 8B 22 3C  ..0...%.....%."<
00000010: 62 61 8F 24 FB 59 E1 63  D7 B1 08 6D 07 7A DD 93  ba.$.Y.c...m.z..
00000020: 90 00                                             ..


----- APDU command/response pair 45 -----
00000000: 8C A4 08 00 15 87 09 01  E5 61 A8 BF 89 AD D7 FF  .........a......
00000010: 8E 08 C2 B3 32 7B D7 83  C9 D1                    ....2{....

00000000: 99 02 90 00 8E 08 E6 37  E6 BE 12 F8 73 6F 90 00  .......7....so..
```

```
----- APDU command/response pair 46 -----
00000000: 0C E0 08 00 4D 87 41 01  41 03 69 5A A4 EE 5F 44  ....M.A.A.iZ.._D
00000010: 2C 4C A9 FE 46 8D 1F 5B  79 D6 89 68 EB 94 CF FB  ,L..F..[y..h....
00000020: 6B A2 55 F6 65 B7 19 66  B3 67 E0 DF 46 F2 27 22  k.U.e..f.g..F.'"
00000030: AC D8 C1 57 C5 54 5B DF  B9 10 87 58 81 2E 9E 65  ...W.T[....X...e
00000040: 07 B1 6E 14 F8 DE 09 AF  8E 08 8C 79 AD C4 3B E2  ..n........y..;.
00000050: D2 84                                            ..


00000000: 99 02 90 00 8E 08 A5 D0  49 2A C0 91 47 68 90 00  ........I*..Gh..

----- APDU command/response pair 47 -----
00000000: 80 86 FF FF                                      ....

00000000: 90 00                                            ..
```

B.5 C_generateKeyPair

```
----- APDU command/response pair 54 -----
00000000: 00 E0 01 00 18 62 81 15  8A 01 04 83 02 01 40 80  .....b........@.
00000010: 02 01 A7 86 08 00 20 00  20 00 20 00 20           ...... . . .

00000000: 90 00                                            ..


----- APDU command/response pair 55 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 02 01 40          .....?.0.0..@

00000000: 90 00                                            ..


----- APDU command/response pair 56 -----
00000000: 00 D6 00 00 FA 01 01 01  40 01 A3 16 00 00 00 00  ........@.......
00000010: 04 02 00 00 00 00 01 00  00 01 01 00 02 00 00 01  ................
00000020: 01 00 03 10 00 03 70 75  62 FF FF FF FF FF FF FF  ......pub.......
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 FF FF FF FF FF 00 86  32 00 01 00 01 00 00 00  ........2.......
00000070: 04 00 00 00 00 01 01 10  00 00 FF FF FF FF FF FF  ................
00000080: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000090: FF FF FF FF FF FF FF FF  FF FF 01 02 10 00 01 03  ................
000000A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000000B0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF 01  ................
000000C0: 04 00 00 01 01 01 06 00  00 01 01 01 0A 00 00 01  ................
```

```
000000D0: 01 01 0B 00 00 01 00 01  0C 10 00 01 00 01 10 10  ................
000000E0: 00 00 FF FF FF FF FF FF  FF FF 01 11 10 00 00 FF  ................
000000F0: FF FF FF FF FF FF FF 01  20 00 00 80 A8 FD 0C     ........ ......

00000000: 90 00                                           ..


----- APDU command/response pair 57 -----
00000000: 00 D6 00 FA AD 53 6B 7F  00 00 A8 FD 0C 53 6B 7F  .....Sk......Sk.
00000010: 00 00 B0 AA 47 51 6B 7F  00 00 00 30 00 00 00 00  ....GQk....0....
00000020: 00 00 B0 AA 47 51 6B 7F  00 00 02 30 00 00 00 00  ....GQk....0....
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 11 01 00 00 00 00  ................
00000050: 00 00 58 FD 0C 53 6B 7F  00 00 58 FD 0C 53 6B 7F  ..X..Sk...X..Sk.
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 01 21 00 00 04 00  04 00 00 01 22 00 00 03  ...!........"...
00000090: 01 00 01 01 63 00 00 01  01 01 66 00 00 04 00 00  ....c.....f.....
000000A0: 00 00 01 70 00 00 01 01  80 10 00 00 01 00 93 03  ...p............
000000B0: 93 03                                           ..

00000000: 90 00                                           ..


----- APDU command/response pair 58 -----
00000000: 80 A4 08 00 06 3F 00 30  00 30 02               .....?.0.0.

00000000: 90 00                                           ..


----- APDU command/response pair 59 -----
00000000: 00 E0 01 00 1E 62 81 1B  8A 01 04 83 02 02 00 80  .....b..........
00000010: 02 01 23 84 04 6B 78 73  30 86 08 00 00 00 20 00  ..#..kxs0..... .
00000020: 20 00 20                                         .

00000000: 90 00                                           ..


----- APDU command/response pair 60 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 02 02 00         .....?.0.0...

00000000: 90 00                                           ..


----- APDU command/response pair 61 -----
00000000: 00 D6 00 00 FA 01 02 02  00 01 1F 16 00 00 00 00  ................
00000010: 04 03 00 00 00 00 01 00  00 01 01 00 02 00 00 01  ................
00000020: 01 00 03 10 00 04 70 72  69 76 FF FF FF FF FF FF  ......priv......
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 FF FF FF FF FF 01 00  00 00 04 00 00 00 00 01  ................
00000070: 01 10 00 00 FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000080: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000090: FF FF FF FF 01 02 10 00  01 03 FF FF FF FF FF FF  ................
000000A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000000B0: FF FF FF FF FF FF FF FF  FF 01 03 30 00 01 01 01  ...........0....
000000C0: 05 50 00 01 01 01 07 50  00 01 01 01 08 50 00 01  .P.....P.....P..
000000D0: 01 01 09 50 00 01 00 01  0C 10 00 01 00 01 10 10  ...P............
000000E0: 00 00 FF FF FF FF FF FF  FF FF 01 11 10 00 00 FF  ................
000000F0: FF FF FF FF FF FF FF 01  62 50 00 01 01 01 63     ........bP....c

00000000: 90 00                                           ..


----- APDU command/response pair 62 -----
00000000: 00 D6 00 FA 29 00 00 01  01 01 64 00 00 01 00 01  ....).....d.....
00000010: 65 00 00 01 01 01 66 00  00 04 00 00 00 00 01 70  e.....f........p
00000020: 00 00 01 01 80 10 00 00  01 00 93 03 93 03        ..............

00000000: 90 00                                           ..


----- APDU command/response pair 63 -----
00000000: 80 A4 08 00 06 3F 00 30  00 30 02               .....?.0.0.

00000000: 90 00                                           ..


----- APDU command/response pair 64 -----
00000000: 00 E0 08 00 27 62 81 24  8A 01 04 83 02 00 41 80  ....'b.$......A.
00000010: 02 00 80 85 05 05 0C 20  00 A3 86 0E 00 00 00 FF  ....... ........
00000020: 00 FF 00 20 00 20 00 00  00 20 71 00              ... . ... q.

00000000: 90 00                                           ..


----- APDU command/response pair 65 -----
00000000: 80 A4 00 00 02 00 41                            ......A

00000000: 90 00                                           ..


----- APDU command/response pair 66 -----
00000000: 00 47 00 00 0C AC 81 09  80 01 06 81 81 03 01 00  .G..............
00000010: 01                                              .

00000000: 90 00                                           ..
```

```
----- APDU command/response pair 67 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 02 00 41           .....?.0.0..A

00000000: 90 00                                             ..


----- APDU command/response pair 68 -----
00000000: 80 48 00 00 00                                    .H...

00000000: 7F 49 81 88 81 81 80 D1  EF 7C A5 06 A1 87 FD 5F  .I.......|....._
00000010: 13 5B 25 B7 16 B9 BA A7  21 43 3D DB 51 9D C1 D1  .[%.....!C=.Q...
00000020: 5A 3C 95 7C B6 F0 37 57  83 CF 2D 0B 53 66 C7 11  Z<.|..7W..-.Sf..
00000030: D5 6B FD 28 FA A0 EA 50  1E 2B FD B5 09 49 E2 E7  .k.(...P.+...I..
00000040: 51 67 1B 00 B0 9D 52 CD  22 D8 69 8C 36 74 54 41  Qg....R.".i.6tTA
00000050: 6E 40 58 4F 79 52 E4 D9  00 43 9C 2C 79 FE A6 48  n@XOyR...C.,y..H
00000060: B7 31 8A B2 05 04 C4 DD  B3 86 E6 4F 38 A6 5D 2A  .1.........O8.]*
00000070: CD A8 3F 95 E4 FF 7B 05  1E ED 4A B5 99 69 36 F0  ..?...{...J..i6.
00000080: B9 5B 29 C6 EC B3 25 82  03 01 00 01 90 00        .[)...%.......


----- APDU command/response pair 69 -----
00000000: 80 A4 08 00 06 3F 00 30  00 30 02                 .....?.0.0.

00000000: 90 00                                             ..


----- APDU command/response pair 70 -----
00000000: 00 E0 08 00 B0 62 81 AD  8A 01 04 83 02 00 81 80  .....b..........
00000010: 02 00 80 85 05 05 08 20  00 A3 86 0E 00 00 00 FF  ....... ........
00000020: 00 FF 00 20 00 20 00 00  00 20 71 81 88 90 03 01  ... . ... q.....
00000030: 00 01 91 81 80 D1 EF 7C  A5 06 A1 87 FD 5F 13 5B  .......|....._.[
00000040: 25 B7 16 B9 BA A7 21 43  3D DB 51 9D C1 D1 5A 3C  %.....!C=.Q...Z<
00000050: 95 7C B6 F0 37 57 83 CF  2D 0B 53 66 C7 11 D5 6B  .|..7W..-.Sf...k
00000060: FD 28 FA A0 EA 50 1E 2B  FD B5 09 49 E2 E7 51 67  .(...P.+...I..Qg
00000070: 1B 00 B0 9D 52 CD 22 D8  69 8C 36 74 54 41 6E 40  ....R.".i.6tTAn@
00000080: 58 4F 79 52 E4 D9 00 43  9C 2C 79 FE A6 48 B7 31  XOyR...C.,y..H.1
00000090: 8A B2 05 04 C4 DD B3 86  E6 4F 38 A6 5D 2A CD A8  .........O8.]*..
000000A0: 3F 95 E4 FF 7B 05 1E ED  4A B5 99 69 36 F0 B9 5B  ?...{...J..i6..[
000000B0: 29 C6 EC B3 25                                    )...%

00000000: 90 00                                             ..


----- APDU command/response pair 71 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 02 00 41           .....?.0.0..A

00000000: 90 00                                             ..
```

```
----- APDU command/response pair 72 -----
00000000: 80 48 00 00 00                                        .H...

00000000: 7F 49 81 88 81 81 80 D1  EF 7C A5 06 A1 87 FD 5F  .I.......|....._
00000010: 13 5B 25 B7 16 B9 BA A7  21 43 3D DB 51 9D C1 D1  .[%.....!C=.Q...
00000020: 5A 3C 95 7C B6 F0 37 57  83 CF 2D 0B 53 66 C7 11  Z<.|..7W..-.Sf..
00000030: D5 6B FD 28 FA A0 EA 50  1E 2B FD B5 09 49 E2 E7  .k.(...P.+...I..
00000040: 51 67 1B 00 B0 9D 52 CD  22 D8 69 8C 36 74 54 41  Qg....R.".i.6tTA
00000050: 6E 40 58 4F 79 52 E4 D9  00 43 9C 2C 79 FE A6 48  n@XOyR...C.,y..H
00000060: B7 31 8A B2 05 04 C4 DD  B3 86 E6 4F 38 A6 5D 2A  .1.........O8.]*
00000070: CD A8 3F 95 E4 FF 7B 05  1E ED 4A B5 99 69 36 F0  ..?...{...J..i6.
00000080: B9 5B 29 C6 EC B3 25 82  03 01 00 01 90 00        .[)...%.......


----- APDU command/response pair 73 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 02 01 40           .....?.0.0..@

00000000: 90 00                                             ..


----- APDU command/response pair 74 -----
00000000: 00 D6 00 F5 82 00 80 D1  EF 7C A5 06 A1 87 FD 5F  .........|....._
00000010: 13 5B 25 B7 16 B9 BA A7  21 43 3D DB 51 9D C1 D1  .[%.....!C=.Q...
00000020: 5A 3C 95 7C B6 F0 37 57  83 CF 2D 0B 53 66 C7 11  Z<.|..7W..-.Sf..
00000030: D5 6B FD 28 FA A0 EA 50  1E 2B FD B5 09 49 E2 E7  .k.(...P.+...I..
00000040: 51 67 1B 00 B0 9D 52 CD  22 D8 69 8C 36 74 54 41  Qg....R.".i.6tTA
00000050: 6E 40 58 4F 79 52 E4 D9  00 43 9C 2C 79 FE A6 48  n@XOyR...C.,y..H
00000060: B7 31 8A B2 05 04 C4 DD  B3 86 E6 4F 38 A6 5D 2A  .1.........O8.]*
00000070: CD A8 3F 95 E4 FF 7B 05  1E ED 4A B5 99 69 36 F0  ..?...{...J..i6.
00000080: B9 5B 29 C6 EC B3 25                              .[)...%

00000000: 90 00                                             ..
```

B.6 C_destroyObject

```
----- 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 97 03 97 03 90  00                      .........

----- APDU command/response pair 49 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 03 40 01          .....?.0.0.@.

00000000: 90 00

----- APDU command/response pair 50 -----
00000000: 00 D6 00 04 04 98 03 98  03                      .........

00000000: 90 00                                            ..

----- APDU command/response pair 53 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 00 C1          .....?.0.0...

00000000: 90 00                                            ..


----- APDU command/response pair 54 -----
00000000: 00 E4 00 00                                      ....

00000000: 90 00                                            ..


----- APDU command/response pair 55 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 40          .....?.0.0..@
```

```
00000000: 90 00                                           ..


----- APDU command/response pair 56 -----
00000000: 00 E4 00 00                                     ....

00000000: 90 00                                           ..
```

B.7 C_encrypt

```
----- APDU command/response pair 52 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 00 C1         .....?.0.0...

00000000: 90 00                                           ..


----- APDU command/response pair 53 -----
00000000: 00 2A 82 05 13 80 81 10  54 65 73 74 53 74 72 69  .*......TestStri
00000010: 6E 67 31 32 33 34 35 36  00                       ng123456.

00000000: 82 10 B4 F0 97 B6 63 E4  68 7A 8B 00 4F DF 3A C1  ......c.hz..O.:.
00000010: 49 9F 90 00                                       I...
```

B.8 C_decrypt

```
----- APDU command/response pair 64 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 00 C1         .....?.0.0...

00000000: 90 00                                           ..


----- APDU command/response pair 65 -----
00000000: 00 2A 80 05 0B 82 81 08  8B 00 4F DF 3A C1 49 9F  .*........O.:.I.
00000010: 00                                                .

00000000: 80 08 6E 67 31 32 33 34  35 36 90 00              ..ng123456..
```

B.9 C_setAttribute

```
----- APDU command/response pair 51 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 40         .....?.0.0..@
```

```
00000000: 90 00                                           ..


----- APDU command/response pair 52 -----
00000000: 90 32 00 03 FF 01 27 00  00 62 82 01 27 00 03 03  .2....'..b..'...
00000010: 40 01 23 18 00 00 00 00  04 04 00 00 00 00 01 00  @.#.............
00000020: 00 01 01 00 02 00 00 01  00 00 03 10 00 07 63 68  ..............ch
00000030: 61 6E 67 65 64 FF FF FF  FF FF FF FF FF FF FF FF  anged............
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 FF FF FF FF FF FF FF  FF FF FF FF FF FF 00 11  ................
00000070: 01 00 18 FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000080: FF FF FF FF FF FF FF FF  FF FF FF 01 00 00 00 04  ................
00000090: 15 00 00 00 01 02 10 00  01 01 FF FF FF FF FF FF  ................
000000A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
000000B0: FF FF FF FF FF FF FF FF  FF 01 03 30 00 01 01 01  ...........0....
000000C0: 04 00 00 01 01 01 05 50  00 01 01 01 06 00 00 01  .......P........
000000D0: 00 01 07 50 00 01 01 01  08 50 00 01 01 01 0A 00  ...P.....P......
000000E0: 00 01 01 01 0C 10 00 01  00 01 10 10 00 00 FF FF  ................
000000F0: FF FF FF FF FF FF 01 11  10 00 00 FF FF FF FF FF  ................
00000100: FF FF FF 01                                     ....

00000000: 90 00                                           ..


----- APDU command/response pair 53 -----
00000000: 80 32 00 03 30 62 50 00  01 00 01 63 00 00 01 01  .2..0bP....c....
00000010: 01 64 00 00 01 01 01 65  00 00 01 01 01 66 00 00  .d.....e.....f..
00000020: 04 31 01 00 00 01 70 00  00 01 01 80 10 00 00 01  .1....p.........
00000030: 00 99 03 99 03                                  .....

00000000: 90 00                                           ..
```

B.10 C_unwrap

```
----- APDU command/response pair 92 -----
00000000: 80 48 00 80 00                                  .H...

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.


----- APDU command/response pair 93 -----
00000000: 00 C0 00 00 09                                  .....

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


----- APDU command/response pair 94 -----
00000000: 80 86 00 00 80 D0 7E EE  17 C7 31 DD 53 FB 1F D4  ......~...1.S...
00000010: 36 65 EB 7F 2C B0 A2 34  44 80 D7 F4 31 96 12 DF  6e..,..4D...1...
00000020: C8 AD 3C 41 EE 8F 13 C2  8A 3B 8D 6B 73 18 A6 1B  ..<A.....;.ks...
00000030: 46 3E 10 93 5C 2F 35 1C  A3 FC 48 09 DB E4 BB EA  F>..5...H.....
00000040: 3F 1A 11 7D 85 57 2F 85  75 1D 8B F4 E6 39 2B FA  ?..}.W/.u....9+.
00000050: 19 3D 7A BB E3 75 B2 A2  A9 E4 EE 79 4F A6 3F EE  .=z..u.....yO.?.
00000060: FD BF 4A F8 43 8F DA A9  D1 8D 58 63 12 5D C8 E8  ..J.C.....Xc.]..
00000070: 2C 77 8F 5F 96 C0 51 CA  19 B1 80 D5 80 4E 50 8B  ,w._..Q......NP.
00000080: 88 6B 64 43 0D 00                               .kdC..

00000000: FD 74 3B 38 48 7E 0E D9  4D B0 BF E7 66 3D E4 63  .t;8H~..M...f=.c
00000010: 15 24 EC 7B F3 93 C7 90  85 43 E8 DF D9 E0 60 88  .$.{.....C....‘.
00000020: 90 00                                           ..


----- APDU command/response pair 95 -----
00000000: 8C A4 08 00 1D 87 11 01  65 FD 9A B5 09 70 96 93  ........e....p..
00000010: FB 5D 39 FF B3 24 6B 8E  8E 08 04 D7 B0 58 E0 96  .]9..$k......X..
00000020: E6 01                                           ..

00000000: 99 02 90 00 8E 08 67 C9  1F 50 18 5F 6D 6A 90 00  ......g..P._mj..


----- APDU command/response pair 96 -----
00000000: 0C 2A 80 0A 99 87 81 89  01 1D 7A 97 D8 25 8F 60  .*........z..%.‘
00000010: 52 07 AE DC A9 AC 33 7C  6E 12 A9 79 71 B8 36 1B  R.....3|n..yq.6.
00000020: 29 C3 54 C1 A8 29 A4 4F  75 72 4E C6 C5 71 22 88  ).T..).OurN..q".
00000030: 50 0C 29 9F 75 C7 99 39  E9 B6 5B AF A1 65 51 DE  P.).u..9..[..eQ.
00000040: 56 84 6D 30 B6 2F F3 19  6B 83 82 C4 6B AB 59 E3  V.m0./..k...k.Y.
00000050: 2B FD B1 4B FC 3D BE CD  16 C8 C0 69 80 5C 0E 72  +..K.=.....i...r
00000060: C0 0F 24 0A 3E 8A 88 4A  CA 68 02 5C FA B5 36 33  ..$.>..J.h.:63
```

```
00000070: CB 5A F7 BE 86 21 2F 68  DB 5F 46 1D 67 FA C2 8B  .Z...!/h._F.g...
00000080: A9 58 37 5C F0 34 7E FE  FC 1A 78 46 C7 51 0B 13  .X7\.4~...xF.Q..
00000090: B2 97 01 00 8E 08 F9 2D  53 7C AD 46 EB 79 00     .......-S|.F.y.

00000000: 87 11 01 FC E7 96 A5 B1  96 E9 E3 1D 2D 3A 49 46  ............-:IF
00000010: 8C 97 A7 99 02 90 00 8E  08 41 16 94 D4 58 27 D0  .........A...X'.
00000020: 3F 90 00                                          ?..


----- APDU command/response pair 97 -----
00000000: 80 86 FF FF                                       ....

00000000: 90 00                                             ..


----- APDU command/response pair 119 -----
00000000: 80 A4 08 00 08 3F 00 30  00 30 01 03 41           .....?.0.0..A

00000000: 90 00                                             ..


----- APDU command/response pair 120 -----
00000000: 00 D6 00 00 FA 01 03 03  41 01 23 18 00 00 00 00  ........A.#.....
00000010: 04 04 00 00 00 00 01 00  00 01 01 00 02 00 00 01  ................
00000020: 00 00 03 10 00 04 74 65  73 74 FF FF FF FF FF FF  ......test......
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 FF FF FF FF FF 00 11  01 00 08 FF FF FF FF FF  ................
00000070: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000080: FF FF FF 01 00 00 00 04  13 00 00 00 01 02 10 00  ................
00000090: 01 10 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 FF FF FF FF  ................
000000B0: FF 01 03 30 00 01 01 01  04 00 00 01 01 01 05 50  ...0...........P
000000C0: 00 01 01 01 06 00 00 01  00 01 07 50 00 01 01 01  ...........P....
000000D0: 08 50 00 01 01 01 0A 00  00 01 01 01 0C 10 00 01  .P..............
000000E0: 00 01 10 10 00 00 FF FF  FF FF FF FF FF FF 01 11  ................
000000F0: 10 00 00 FF FF FF FF FF  FF FF FF 01 62 50 00     ............bP.

00000000: 90 00                                             ..


----- APDU command/response pair 121 -----
00000000: 00 D6 00 FA 2D 01 00 01  63 00 00 01 00 01 64 00  ....-...c.....d.
00000010: 00 01 00 01 65 00 00 01  00 01 66 00 00 04 FF FF  ....e.....f.....
00000020: FF FF 01 70 00 00 01 01  80 10 00 00 01 00 B0 03  ...p............
00000030: B0 03                                             ..

00000000: 90 00                                             ..
```

```
----- APDU command/response pair 122 -----
00000000: 80 48 00 80 00                                        .H...

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.


----- APDU command/response pair 123 -----
00000000: 00 C0 00 00 09                                        .....

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


----- APDU command/response pair 124 -----
00000000: 80 86 00 00 80 95 3B CF  46 B8 4E 67 E4 6B 97 4B  ......;.F.Ng.k.K
00000010: 70 AD B3 44 22 6A 1B 42  18 4B A9 44 FF 28 FA C0  p..D"j.B.K.D.(..
00000020: 0A EF 44 CD DA C1 28 2B  CF FD 5D 20 48 50 33 59  ..D...(+..] HP3Y
00000030: 7D B7 CB 73 4A EF 28 0A  C7 E4 02 2A 91 A9 F6 55  }..sJ.(....*...U
00000040: 97 D3 A8 DE 21 90 0E 23  0B 9C ED 4B 52 39 46 ED  ....!..#...KR9F.
00000050: 13 1F 7F 9D CB EF 7A DD  7C D7 39 EC 1F BD 2A 3A  ......z.|.9...*:
00000060: 45 48 8F 6C 7E 82 71 E5  14 8F C1 9D F8 E8 53 2B  EH.l~.q.......S+
00000070: D3 AF 3D 7C 11 59 E3 81  F4 0B 08 17 A9 0F 37 69  ..=|.Y........7i
00000080: 90 C1 11 E2 1B 00                                 ......

00000000: B3 0F 6C 66 E6 56 8F 44  55 B2 A6 02 0E 0B 80 01  ..lf.V.DU.......
00000010: FF 89 7A 65 FC 68 25 82  22 C9 97 74 D1 6B 00 AB  ..ze.h%."..t.k..
00000020: 90 00                                             ..


----- APDU command/response pair 125 -----
00000000: 8C A4 08 00 15 87 09 01  82 46 BD FD 60 2D E4 C6  .........F..`-..
00000010: 8E 08 25 35 C0 28 0E E1  20 93                    ..%5.(.. .
```

```
00000000: 99 02 90 00 8E 08 8C D6  A9 A8 99 7F 14 12 90 00  ................


----- APDU command/response pair 126 -----
00000000: 0C E0 08 00 3D 87 31 01  4D 4F 3D AB 31 72 FC F7  ....=.1.MO=.1r..
00000010: B4 84 D1 41 19 1C 22 DF  3F 60 BE 6B 0A 1E 49 5F  ...A.."."?‘.k..I_
00000020: AD 3D 6D 61 5E DA E3 F7  A8 0A 82 EA 65 16 8A 01  .=ma^.......e...
00000030: C5 4F BF 3F 44 73 9C 61  8E 08 A8 A9 A5 4D 55 BB  .O.?Ds.a.....MU.
00000040: E7 B3                                           ..

00000000: 99 02 90 00 8E 08 23 E5  DF 34 11 21 87 1C 90 00  ......#..4.!....


----- APDU command/response pair 127 -----
00000000: 80 86 FF FF                                     ....

00000000: 90 00                                           ..
```

B.11 C_wrap

```
Enter command (spaced integers)
00 42 130 10 08 49 50 51 52 53 54 55 56 00 (integers)
00 2A 82  0A 08 31 32 33 34 35 36 37 38 00 (hexadecimals)
command changed!

RESPONSE
00000000: 6A 80                                           j.
```