

A Tour through the Realms of Reverse Engineering

Ricardo J. Rodríguez

@RicardoJRdez * rjrodriguez@unizar.es * www.ricardojrodriguez.es

© All wrongs reversed



Universidad
Zaragoza

November 23, 2016

Legacy Systems
Room A.12

Outline

- 1 Introduction to Reverse Engineering
- 2 Reverse Engineering of Protocols
- 3 Reverse Engineering of Software
- 4 Reverse Engineering of Integrated Circuits/Smart Cards
- 5 Conclusions



Agenda

- 1 Introduction to Reverse Engineering
 - What is Reverse Engineering?
 - Motivation
 - Approaches to Reverse Engineering

2 Reverse Engineering of Protocols

3 Reverse Engineering of Software

4 Reverse Engineering of Integrated Circuits/Smart Cards

5 Conclusions



Introduction to Reverse Engineering (I)

Reverse Engineering

- **Figure out how something works** from an exhaustive analysis
- **Improvement of legacy products/systems**
- **Different application domains**
 - Hardware (legacy hardware)
 - Software (e.g. Samba)



Introduction to Reverse Engineering (II): Motivation (1)

Motivation

- Interoperability
- **Non-existent documentation**
- Final product analysis
- **Security audit**
- **Industrial (or military) espionage** (e.g. II WW)
- Removal of anti-copy or limited use protections
- Creation of duplicates without license
- Academic
- **Innate curiosity**
- **Learn about the errors** of other people



Introduction to Reverse Engineering (II): Motivation (2)

Find bugs in software

- **Incorrect checking of buffer limits** (buffer overflow)
- **Use of data without previous validation**
- **Cyclic routines** for input data
- **Byte-level copy operations**
- **Pointer arithmetic** based in user-input data
- **"Trust" in security systems with dynamic inputs**



Introduction to Reverse Engineering (III): Approaches

- **White-box**
 - Full knowledge (e.g., source code)
 - E.g.: *WhiteBox SecureAssistant, IDAPro, SourceScope...*



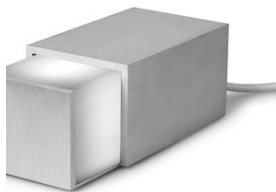
Introduction to Reverse Engineering (III): Approaches

- **White-box**

- Full knowledge (e.g., source code)
- E.g.: *WhiteBox SecureAssistant, IDAPro, SourceScope...*

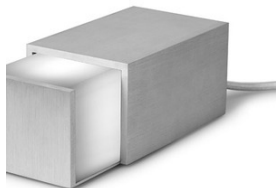
- **Gray-box**

- Partial knowledge
- E.g.: Rational's Purify (use/consumption of memory), Valgrind



Introduction to Reverse Engineering (III): Approaches

- **White-box**
 - *Full* knowledge (e.g., source code)
 - E.g.: *WhiteBox SecureAssistant, IDAPro, SourceScope...*
- **Gray-box**
 - *Partial* knowledge
 - E.g.: Rational's Purify (use/consumption of memory), Valgrind
- **Black-box**
 - *Null* knowledge
 - Analyse outputs of the system depending on different inputs



Reverse Engineering

Choose your side!

On hardware

- Reverse engineering of integrated circuits/smart cards
 - Low-level details
 - Physics and electronic knowledge (plus special hardware)
- Reverse engineering of devices
 - How system works?
 - How are the inputs and outputs?

On software

- Reverse engineering of protocols
 - How network layer works
- Reverse engineering of software
 - Low-level code analysis: assembly, calling conventions
 - Use of debugging/disassembler/decompiler tools
 - Programming

Agenda

- 1 Introduction to Reverse Engineering
- 2 Reverse Engineering of Protocols**
- 3 Reverse Engineering of Software
- 4 Reverse Engineering of Integrated Circuits/Smart Cards
- 5 Conclusions



Reverse Engineering of Protocols

- **Message format**
 - **Normally done by manual analysis**
 - Recent research on automatic analysis
 - Message clustering
 - Emulate protocol implementation tracing message processing
- **Protocol inference**
 - **Get the state-machine of the protocol**
 - Two techniques
 - **Off-line learning**: observes communication and build a state-machine that matches observed message sequences → NP-complete problem
 - **On-line learning**: the predictor is updated at each step with the given data. Polynomial time

Check out papers of J. Caballero et al. (2007, 2009) and Cho et al. (2010)

Agenda

- 1 Introduction to Reverse Engineering
- 2 Reverse Engineering of Protocols
- 3 Reverse Engineering of Software**
 - What is it?
 - Types of Analysis
- 4 Reverse Engineering of Integrated Circuits/Smart Cards
- 5 Conclusions



Reverse Engineering of Software

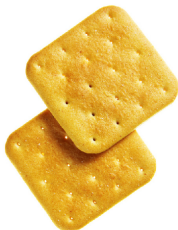
Reverse code engineering

- Also known as *cracking* (also as Software Reverse Engineering)
- **Remove code protections** (copyrights)
- NOT always bad: bugs detection, potential exploits, . . . in your programs

Reverse Engineering of Software

Reverse code engineering

- Also known as *cracking* (also as Software Reverse Engineering)
- **Remove code protections** (copyrights)
- NOT always bad: bugs detection, potential exploits, ... in your programs
- **Crackers**: something else than a (salt) cookies...
 - **NOT MISTAKE** with *CRiminal hACKERS*



Reverse Engineering of Software

- Involves knowledge of assembler
- Involves knowledge of file formats
- Involves knowledge of Operating System
- Involves knowledge of networks

Reverse Engineering of Software

- Involves knowledge of assembler
- Involves knowledge of file formats
- Involves knowledge of Operating System
- Involves knowledge of networks
- Involves knowledge of laws
 - Jail is cold

Methods

- Analyze information exchange (on computer bus and network)
- Disassembler: read PUSH EAX instead of 0x50
- Decompilation: recreate the high-level representation of the bytes

Reverse Engineering of Software

Well-known RCE examples

- Samba
 - They reverse-engineer unpublished information about how Windows file sharing worked
- Wine
 - They reverse-engineer unpublished information about Windows API
- OpenOffice
 - They reverse-engineer unpublished information about Microsoft Office file formats
- Mac OS System 4.1
 - Reversed in 1987 by Bell Laboratories to enable this OS to execute on RISC machines of their own

Static Analysis

- Code **is not executed** (cold analysis)
- Steps:
 - 1 Analyse the **PE header**
 - 2 **Read the code** (disassembler)
 - 3 **Search for strings**
 - 4 **THINK!**

Static Analysis

- Code **is not executed** (cold analysis)
- Steps:
 - 1 Analyse the **PE header**
 - 2 **Read the code** (disassembler)
 - 3 **Search for strings**
 - 4 **THINK!**

Can you figure out what is doing? → **Not enough in all binaries**

Dynamic Analysis

- Code **is executed** (hot analysis)
- Steps:
 - 1 Analyse the **PE header**
 - 2 **Read the code** (debugger)
 - 3 **Search for strings**
 - 4 **Observe the execution**
 - 5 **THINK!**

Dynamic Analysis

- Code **is executed** (hot analysis)
- Steps:
 - 1 Analyse the **PE header**
 - 2 **Read the code** (debugger)
 - 3 **Search for strings**
 - 4 **Observe the execution**
 - 5 **THINK!**

Can you find out ***now*** what is doing?

Reverse Engineering of Software

Understanding Assembler from Code – a MUST!

```
int main(int argc, char* argv[])
{
    printf("Hello, world! My name is %s and
           I've %d years old.", "Restituto", 23);
    return 0;
}
```

```
.section .rdata,"dr"
LC0:
    .ascii "Restituto\0"
    .align 4
LC1:
    .ascii "Hello, world! My name is %s and
           I've %d years old.\0"
.text

_main:
LFB6:
    push ebp
    mov  ebp, esp
    and  esp, -16
    sub  esp, 16
    call ___main
    mov  DWORD PTR [esp+8], 23
    mov  DWORD PTR [esp+4], OFFSET FLAT:LC0
    mov  DWORD PTR [esp], OFFSET FLAT:LC1
    call _printf
    mov  eax, 0
    leave
    ret
```



Reverse Engineering of Software

Basic techniques

- CD-Check
 - Check presence of a specific drive (e.g., AoE II, AvP Gold Editon)
- Event-fishing
 - Windows follows event-driven paradigm
 - Detect where a certain message is handled. That is, reveal the function that handles an input
- Serial-fishing
 - Find the correct serial for registering a software
- Keygenning. Two types:
 - Self-keygenning: patch the binary to show the correct key by itself
 - Keygen: replicate the key code generation

Reverse Engineering of Software

Example of CD Check (1)

```
0041F18A . 83F8 05      JMP     EAX,5
0041F18D > 74 0D      JNE SHORT empres2.0041F19C
0041F18F . 5E        POP     ESI
0041F190 . 33C0      XOR     EAX,EAX
0041F192 . 5B        POP     EBX
0041F193 . 81C4 0C020000 ADD    ESP,20C
0041F199 . C2 0400   RETN   4
0041F19C > 8D8424 140100 LEA    EAX,DWORD PTR SS:[ESP+114]
0041F1A3 . 68 00010000 PUSH  100
0041F1A8 . 8D4C24 10  LEA    ECX,DWORD PTR SS:[ESP+10]
0041F1AC . 50        PUSH   EAX
0041F1AD . 8D5424 18  LEA    EDX,DWORD PTR SS:[ESP+18]
0041F1B1 . 51        PUSH   ECX
0041F1B2 . 8D4424 14  LEA    EAX,DWORD PTR SS:[ESP+14]
0041F1B6 . 52        PUSH   EDX
0041F1B7 . 50        PUSH   EAX
0041F1B8 . 8D4C24 28  LEA    ECX,DWORD PTR SS:[ESP+28]
0041F1BC . 68 00010000 PUSH  100
0041F1C1 . 51        PUSH   ECX
0041F1C2 . 56        PUSH   ESI
0041F1C3 . FF15 94C16100 CALL   DWORD PTR DS:[K&KERNEL32.GetVolume
0041F1C9 . 85C0      TEST   EAX,EAX
0041F1CB > 75 0B      JNZ SHORT empres2.0041F1D8
0041F1CD . 5E        POP     ESI
0041F1CE . 5B        POP     EBX
0041F1CF . 81C4 0C020000 ADD    ESP,20C
0041F1D5 . C2 0400   RETN   4
0041F1D8 > 8B53 24    MOV    EDX,DWORD PTR DS:[EBX+24]
0041F1DB . 8D4424 14  LEA    EAX,DWORD PTR SS:[ESP+14]
0041F1DF . 81C2 FD020000 ADD    EDX,2FD
0041F1E5 . 52        PUSH   EDX
0041F1E6 . 50        PUSH   EAX
0041F1E7 . E8 74021E00 CALL   empres2.005FF460
0041F1EC . 83C4 08    ADD    ESP,8
0041F1EF . F7D8      NEG    EAX
0041F1F1 . 1BC0      SBB   EAX,EAX
0041F1F3 . 5E        POP     ESI
0041F1F4 . 40        INC    EAX
0041F1F5 . 5B        POP     EBX
0041F1F6 . 81C4 0C020000 ADD    ESP,20C
0041F1FC . C2 0400   RETN   4
```

```
pFileSystemNameSize =
pFileSystemNameBuffer
pFileSystemFlags
pMaxFilenameLength
pVolumeSerialNumber
MaxVolumeNameSize = 1
VolumeNameBuffer
RootPathName
GetVolumeInformationA
```

```
00130DB0 00130DCC
00130DB4 0013E6B5 ASCII "AOE2"
00130DB8 00EA0078
```

Reverse Engineering of Software

Example of CD Check (2)

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h
|
:10001898 FF1568110110      Call dword ptr [10001168]
:1000189E 83F805      cmp eax, 00000005
:100018A1 7569      jne 1000190C
:100018A3 33ED      xor ebp, ebp
:100018A5 BF01000000      mov edi, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
[:100018FD(C)]
|
:100018AA 8D4C2414      lea ecx, dword ptr [esp+14]
:100018AE 57      push edi
:100018AF 51      push ecx
:100018B0 8D542454      lea edx, dword ptr [esp+54]

* Possible StringData Ref from Data Obj ->"istrack\02d.cda"
|
:100018B4 68C4600110      push 100160C4
:100018B9 52      push edx
:100018BA 895C2458      mov dword ptr [esp+58], ebx
:100018BE E89A0F0000      call 1000285D
:100018C3 8D44245C      lea eax, dword ptr [esp+5C]
```

```
* Possible StringData Ref from Data Obj ->"rb"
|
:100018C7 68C0600110      push 100160C0
:100018CC 50      push eax
:100018CD E8780F0000      call 1000284A
:100018D2 8BF0      mov esi, eax
:100018D4 83C418      add esp, 00000018
:100018D7 3BF3      cmp esi, ebx
:100018D9 7424      je 100018FF
:100018DB 56      push esi
:100018DC 6A0B      push 0000000B
:100018DE 8D4C2428      lea ecx, dword ptr [esp+28]
:100018E2 6A04      push 00000004
:100018E4 51      push ecx
:100018E5 E8180E0000      call 10002702
:100018EA 8B4C2458      mov ecx, dword ptr [esp+58]
:100018EE 56      push esi
:100018EF 03E9      add ebp, ecx
:100018F1 E88F0D0000      call 10002685
:100018F6 83C414      add esp, 00000014
:100018F9 47      inc edi
:100018FA 83FF10      cmp edi, 00000010
:100018FD 7EAB      jle 100018AA

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
[:100018D9(C)]
|
:100018FF 83FF10      cmp edi, 00000010
:10001902 7508      jne 1000190C
:10001904 81FDC09A4200      cmp ebp, 00429AC0
:1000190A 741F      je 1000192B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
[:10001902(C)]
|
:1000190C 8A442413      mov al, byte ptr [esp+13]
:10001910 FE0C      inc al
:10001912 3C7A      cmp al, 7A
:10001914 8B442413      mov byte ptr [esp+13], al
:10001918 0F8E6DFFFFFF      jle 1000188B
:1000191E 5F      pop edi
:1000191F 5E      pop esi
:10001920 5D      pop ebp
:10001921 33C0      xor eax, eax
:10001923 5B      pop ebx
:10001924 81C4A0000000      add esp, 000000A0
:1000192A C3      ret
```

Reverse Engineering of Software

Example of serial-fishing (1)

Serial hardcoded in the code



Reverse Engineering of Software

Example of serial-fishing (1)

Serial hardcoded in the code

```
00424FEC PUSH registro.00422798 UNICOD "english.dll"
0042505D MOV DWORD PTR SS:[EBP-90],registro.0042 UNICOD "Trial version expired"
00425080 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICOD "This trial version has exp
004250FB MOV DWORD PTR SS:[EBP-90],registro.0042 UNICOD "Este producto ha caducado"
0042511E MOV DWORD PTR SS:[EBP-80],registro.0042 UNICOD "Este producto ha caducado.
00425187 PUSH registro.00422734 UNICOD "alt"
004251D0 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICOD "ERROR!"
0042542B PUSH registro.00422E48 UNICOD "utilities 77 backdoor"
00425454 PUSH registro.00422E14 UNICOD "RC10-FFGH-PPBA-9999"
004255C0 PUSH registro.00422798 UNICOD "english.dll"
00425625 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICOD "Registration"
0042563C PUSH registro.00422E78 UNICOD "There is a problem when tr
00425651 PUSH registro.00422E60 UNICOD "H"
```



Reverse Engineering of Software

Example of serial-fishing (1)

Serial hardcoded in the code

```
00424FEC PUSH registro.00422798 UNICODE "\english.dll"
0042505D MOV DWORD PTR SS:[EBP-90],registro.0042 UNICODE "Trial version expired"
00425080 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICODE "This trial version has exp
004250FB MOV DWORD PTR SS:[EBP-90],registro.0042 UNICODE "Este producto ha caducado"
0042511E MOV DWORD PTR SS:[EBP-80],registro.0042 UNICODE "Este producto ha caducado.
00425187 PUSH registro.00422734 UNICODE "alt"
004251D0 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICODE "ERROR!"
0042542B PUSH registro.00422E48 UNICODE "utilities 77 backdoor"
00425454 PUSH registro.00422E14 UNICODE "RC10-FFGH-PPBA-9999"
004255C0 PUSH registro.00422798 UNICODE "\english.dll"
00425625 MOV DWORD PTR SS:[EBP-80],registro.0042 UNICODE "Registration"
0042563C PUSH registro.00422E78 UNICODE "There is a problem when tr
00425651 PUSH registro.00422E60 UNICODE "H"
```

(Un)fortunately, not so common nowadays... ☺

Reverse Engineering of Software

Example of serial-fishing (2) – file fwdv1.zip

The screenshot shows the URSoft W32Dasm Ver 10 interface. The main window displays the disassembly of a file named 'vl.exe'. A modal dialog box titled 'W32Dasm List of String Data Items' is open in the foreground. The dialog contains the following text:

To Search Disassembly for String Data, Double Click on

"Fishing with DiLA v0.1"
"Sorry, wrong code!"
"Success! Thank you for playing "
"Uñijiii"

Buttons at the bottom of the dialog include 'Close', 'Copy All', 'Copy View', and 'Cancel Search'.

The background window shows the disassembly of 'vl.exe' with the following details:

- Disassembly of File: vl.exe
- Code Offset = 00000400, Code Size = 00000200
- Data Offset = 00000800, Data Size = 00000200
- Number of Objects =
- Object01: .text
- Object02: .rdata
- Object03: .data
- Object04: .rsrc
- Number of Dialogs =
- Name: DialogID_03E9,
- Number of Imported M
- Import Module 001
- Import Module 002
- Import Module 001

Reverse Engineering of Software

Example of serial-fishing (2) – file fwdv1.zip

```
* Reference To: user32.ShowDialogParamA, Ord:008Ah
|
:00401016 E87B000000          Call 00401096
:0040101B 6A00                    push 00000000

* Reference To: kernel32.ExitProcess, Ord:0080h
|
:0040101D E886000000          Call 004010A8
:00401022 55                      push ebp
:00401023 8BEC                    mov ebp, esp
:00401025 817D0C11010000        cmp dword ptr [ebp+0C], 00000111
:0040102C 754D                    jne 0040107B
:0040102E 817D10EB030000        cmp dword ptr [ebp+10], 000003EB
:00401035 7542                    jne 00401079
:00401037 6A00                    push 00000000
:00401039 6A00                    push 00000000
:0040103B 68EA030000            push 000003EA
:00401040 FF7508                    push [ebp+08]

* Reference To: user32.GetDlgItemInt, Ord:00F3h
|
:00401043 E854000000          Call 0040109C
:00401048 3D9A020000            cmp eax, 0000029A
:0040104D 7416                    je 00401065
:0040104F 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"Fishing with DiLA v0.1"
|
-----
:00401051 6800304000            push 00403000

* Possible StringData Ref from Data Obj ->"Sorry, wrong code!"
|
:00401055 6817304000            push 00403017
:0040105B FF7508                    push [ebp+08]

* Reference To: user32.MessageBoxA, Ord:019Dh
```



Reverse Engineering of Software

Example of serial-fishing (2) – file fwdv1.zip



Reverse Engineering of Software

Example of self-keygenning (1) – file X-Converter.zip

```
00401A6D | . E8 46010000 CALL <JMP.&kernel32.lstrcatA>
00401A72 | . 68 62544000 PUSH xconv.00405462
00401A77 | . 68 1D544000 PUSH xconv.0040541D
00401A7C | . E8 3D010000 CALL <JMP.&kernel32.lstrcmpA>
00401A81 | . 83F8 00 CMP EAX, 0
00401A84 | . 74 02 IF_CQNDT_FALSE 00401A88
```

```
0012F1A0 | 00401A81 | CALL to lstrcmpA from xconv.00401A7C
0012F1A4 | 0040541D | String1 = "7BBE9ABB-63D0532B-20205A3A"
0012F1A8 | 00405462 | String2 = "quequeque"
0012F1AC | 7E3B21CC | RETURN to user32.7E3B21CC from user32.7E398600
0012F1B0 | FFFFFFFF
0012F1B4 | 0012F228
0012F1B8 | 00000000
0012F1BC | 00401A87 | RETURN to xconv.00401A87 from xconv.00401A81
```

- Recall last example: lstrcmpA
 - Offset 0040541D: valid key

Let's make the binary speak by itself. . .



Reverse Engineering of Software

Example of self-keygenning (2) – file X-Converter.zip

MessageBox function

Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

Syntax

```
C++  
  
int WINAPI MessageBox (  
    _In_opt_ HWND hWnd,  
    _In_opt_ LPCTSTR lpText,  
    _In_opt_ LPCTSTR lpCaption,  
    _In_     UINT uType  
);
```



Reverse Engineering of Software

Example of self-keygenning (3) – file X-Converter.zip

004016A5	. 68 81000000	PUSH 0x81
004016AA	. FF75 08	PUSH DWORD PTR SS:[EBP+0x8]
004016AD	. E8 94040000	CALL <JMP.&user32.GetDlgItemTextA>
004016B2	. E8 4A020000	CALL xconv.00401901
004016B7	. 83F8 01	CMPL EAX, 0x1
004016BA	√ 74 32	JE SHORT xconv.004016EE
004016BC	. 8005 B8534000	ADD BYTE PTR DS:[0x4053B8], 0x1

0040169E	. 6A 1B	PUSH 0x1B
004016A0	. 68 62544000	PUSH xconv.00405462
004016A5	. 68 81000000	PUSH 0x81
004016AA	. FF75 08	PUSH DWORD PTR SS:[EBP+0x8]
004016AD	. E8 94040000	CALL <JMP.&user32.GetDlgItemTextA>
004016B2	√ E9 F3050000	JMP xconv.00401C9A
004016B7	. 83F8 01	CMPL EAX, 0x1
004016BA	√ 74 32	JE SHORT xconv.004016EE
004016BC	. 8005 B8534000	ADD BYTE PTR DS:[0x4053B8], 0x1
004016C3	. 803D B8534000	CMPL BYTE PTR DS:[0x4053B8], 0x3
004016CA	√ 0F84 9C000000	JE xconv.00401760

00401C9A	E8 52FCFFF	CALL xconv.00401901
00401CB0	90	PUSH EAX
00401CB1	90	NOP
00401CB2	6A 00	PUSH 0x0
00401CB4	6A 00	PUSH 0x0
00401CB6	68 10544000	PUSH xconv.0040541D
00401CB8	6A 00	PUSH 0x0
00401CC2	E9 F0E9FFF	CALL user32.MessageBoxA
00401CC7	90	JMP xconv.004016B7
		NOP

RSC11 *7BBE9ABD-2289532B-20205



Reverse Engineering of Software

Example of self-keygenning (3) – file X-Converter.zip



Reverse Engineering of Software

Example of keygenning (1) – file X-Converter.zip

```
00401668 .v0f85 24010000 00401668 6A 0E PUSH 0E
00401671 .68 0E PUSH xconv.0040542D
00401673 .68 20544000 PUSH 0E
00401676 .68 00000000 PUSH 0E
0040167D .FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401680 .E9 C1040000 CALL <JMP.>user32.GetDlgItemTextA
00401685 .83F8 0C CMP EBX,0C
00401688 .7F 4B JS SHORT xconv.004016DE
0040168D .83F8 04 CMP ERX,4
00401690 .7C 46 JS SHORT xconv.004016D5
00401693 .68 20544000 PUSH xconv.0040542D
00401694 .68 06334000 PUSH xconv.00403306
00401699 .E9 26050000 CALL <JMP.>kernel32.lstrcpyA
0040169E .6A 1B PUSH 1B
004016A0 .68 62544000 PUSH xconv.00405462
004016A5 .68 81000000 PUSH 81
004016A8 .FF75 08 PUSH DWORD PTR SS:[EBP+8]
004016AD .E9 54040000 CALL <JMP.>user32.GetDlgItemTextA
004016B2 .E9 4A020000 CALL xconv.00401901
004016B7 .83F8 01 CMP ERX,1
004016BA .74 32 JS SHORT xconv.004016FE
004016BC .8005 B0534000 ADD BYTE PTR DS:[4053B0],1
004016C3 .803D B0534000 CMP BYTE PTR DS:[4053B0],3
004016C8 .v9f84 30000000 JS xconv.00401752
004016D0 .E9 9 81000000 JMP xconv.00401756
004016D5 .> 6A 10 PUSH 10
004016D7 .68 73004000 PUSH xconv.00403079
004016DC .68 53324000 PUSH xconv.00403253
004016E1 .FF75 08 PUSH DWORD PTR SS:[EBP+8]
004016E4 .E9 69040000 CALL <JMP.>user32.MessageBoxA
004016E9 .E9 07000000 JMP xconv.00401755
004016EE .E9 A8030000 CALL xconv.00401A9B
004016F3 .C605 72434000 MOV BYTE PTR DS:[404372],1
004016FA .6A 40 PUSH 40
004016FC .68 33304000 PUSH xconv.00403023
00401701 .68 3E304000 PUSH xconv.0040303E
00401706 .FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401709 .E9 4A040000 CALL <JMP.>user32.MessageBoxA
0040170E .6A 00 PUSH 0
```

```
Count = E (14.)
Buffer = xconv.0040542D
ControlID = 86 (128.)
hWnd
GetDlgItemTextA

String2 = "DeRtH"
String1 = xconv.00403306
lstrcpyA
Count = 1B (27.)
Buffer = xconv.00405462
ControlID = 81 (129.)
GetDlgItemTextA

Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
Title = "Sorry"
Text = "Sorry username must be at least 4 characters long and not n
hOwner
MessageBoxA

Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
Title = "Thank you!"
Text = "Registration done. Thank you for registering this program!"
hOwner
MessageBoxA
Result = 0
```

- $4 \leq$ length name ≤ 12
- Length serial: 27 (read)
- Checking procedure: 00401901
- Name buffer: buf_42D

Reverse Engineering of Software

Example of keygenning (2) – file X-Converter.zip

00401901	# 53	PUSH EBX	
00401902	. 57	PUSH EDI	
00401903	. 52	PUSH EAX	
00401904	. 51	PUSH ECX	
00401905	. 68 2D544000	PUSH kconv.0040542D	
0040190A	. E8 BB020000	CALL <JMP.&kernel32.lstrlenA>	[String = "DeRtH"
0040190F	. 83F8 0C	CMPL ERX, 0C	lstrlenA
00401912	. 7C 02	JE SHORT kconv.00401916	
00401914	. EB 75	JMP SHORT kconv.0040190B	
00401916	> B9 0C000000	MOV ECX, 0C	
0040191B	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 0	
00401922	. 49	DEC ECX	
00401923	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040192A	. 49	DEC ECX	
0040192B	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
00401932	. 3BC1	CMPL ERX, ECX	
00401934	. 74 55	JE SHORT kconv.0040190B	
00401936	. 49	DEC ECX	
00401937	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
0040193E	. 3BC1	CMPL ERX, ECX	
00401940	. 74 49	JE SHORT kconv.0040190B	
00401942	. 49	DEC ECX	
00401943	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
0040194A	. 3BC1	CMPL ERX, ECX	
0040194C	. 74 3D	JE SHORT kconv.0040190B	
0040194E	. 49	DEC ECX	
0040194F	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
00401952	. 3BC1	CMPL ERX, ECX	
0040195B	. 74 31	JE SHORT kconv.0040190B	
0040195D	. 49	DEC ECX	
0040195E	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
00401962	. 3BC1	CMPL ERX, ECX	
00401964	. 74 25	JE SHORT kconv.0040190B	
00401966	. 49	DEC ECX	
00401967	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
0040196E	. 3BC1	CMPL ERX, ECX	
00401970	. 74 19	JE SHORT kconv.0040190B	
00401972	. 49	DEC ECX	
00401973	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
0040197A	. 3BC1	CMPL ERX, ECX	
0040197C	. 74 0D	JE SHORT kconv.0040190B	
0040197E	. 49	DEC ECX	
0040197F	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 30	
00401986	. 49	DEC ECX	
00401987	. 3BC1	CMPL ERX, ECX	
00401989	. 74 00	JE SHORT kconv.0040190B	
0040198B	> E8 2D544000	PUSH kconv.0040542D	[String2 = "DeRtH"
00401990	. 68 30544000	PUSH kconv.00405430	String1 = kconv.0040543A
00401995	. E8 2A020000	CALL <JMP.&kernel32.lstrcpyA>	lstrcpyA
0040199A	. 6A 21	PUSH 21	Length = 21 (33.)
0040199C	. 68 8A534000	PUSH kconv.0040538A	Destination = kconv.0040538A
004019A1	. E8 06020000	CALL <JMP.&kernel32.RtlZeroMemory>	RtlZeroMemory
004019A6	. 6A 21	PUSH 21	Length = 21 (33.)



Reverse Engineering of Software

Example of keygenning (3) – file X-Converter.zip

00401901	# 53	PUSH EBX	
00401902	. 57	PUSH EDI	
00401903	. 52	PUSH EAX	
00401904	. 52	PUSH EAX	
00401905	. 68 20544000	PUSH XCONV.00405420	
00401906	. E8 BB020000	CALL <JMP.&kernel32.lstrlenA>	[String = "DeRtH"
0040190F	. 83FB 0C	CMPL EAX, 0C	lstrlenA
00401912	. 7C 02	JE SHORT XCONV.00401916	
00401914	. EB 75	JMP SHORT XCONV.0040190B	
00401916	> B9 0C000000	MOV ECX, 0C	
0040191B	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 0	
00401922	. 49	DEC ECX	
00401925	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
0040192A	. 49	DEC ECX	
0040192B	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
00401932	. 3BC1	CMPL EAX, ECX	
00401934	. 74 55	JE SHORT XCONV.0040190B	
00401936	. 49	DEC ECX	
00401937	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
0040193E	. 3BC1	CMPL EAX, ECX	
00401940	. 74 49	JE SHORT XCONV.0040190B	
00401942	. 49	DEC ECX	
00401943	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
0040194A	. 3BC1	CMPL EAX, ECX	
0040194C	. 74 30	JE SHORT XCONV.0040190B	
0040194E	. 49	DEC ECX	
0040194F	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
00401955	. 3BC1	CMPL EAX, ECX	
0040195B	. 74 31	JE SHORT XCONV.0040190B	
0040195D	. 49	DEC ECX	
0040195E	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
00401962	. 3BC1	CMPL EAX, ECX	
00401964	. 74 25	JE SHORT XCONV.0040190B	
00401966	. 49	DEC ECX	
00401967	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
0040196E	. 3BC1	CMPL EAX, ECX	
00401970	. 74 19	JE SHORT XCONV.0040190B	
00401972	. 49	DEC ECX	
00401973	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
0040197A	. 3BC1	CMPL EAX, ECX	
0040197C	. 74 00	JE SHORT XCONV.0040190B	
0040197E	. 49	DEC ECX	
0040197F	. C681 20544000	MOV BYTE PTR DS:[ECX+405420], 20	
00401986	. 49	DEC ECX	
00401987	. 3BC1	CMPL EAX, ECX	
00401989	. 74 00	JE SHORT XCONV.0040190B	
0040198B	. 68 20544000	PUSH XCONV.00405420	
00401990	. 68 30544000	PUSH XCONV.00405430	[String2 = "DeRtH"
00401995	. E8 2A020000	CALL <JMP.&kernel32.lstrcpyA>	String1 = XCONV.0040543A
0040199A	. 6A 21	PUSH 21	lstrcpyA
0040199C	. 68 8A534000	PUSH XCONV.0040538A	Length = 21 (33.)
004019A1	. E8 96200000	CALL <JMP.&kernel32.RtlZeroMemory>	Destination = XCONV.0040538A
004019A6	. 6A 21	PUSH 21	RtlZeroMemory
			Length = 21 (33.)

```
eax = strlen(buf_42D);
ecx = 12;
buf_42D[ecx] = 00;
for(; eax != ecx; ecx--)
    buf_42D[ecx] = 20;
```



Reverse Engineering of Software

Example of keygenning (4) – file X-Converter.zip

00401901	# 53	PUSH EBX	
00401902	. 57	PUSH EDI	
00401903	. 52	PUSH EAX	
00401904	. 51	PUSH ECX	
00401905	. 68 2D544000	PUSH xconv.0040542D	
0040190A	. E8 BB020000	CALL <JMP.&kernel32.lstrlenA>	[String = "DeRtH"
0040190F	. 83F8 0C	CFI EAX, 0C	lstrlenA
00401912	. 7C 92	JE SHORT xconv.00401916	
00401914	. EB 75	JMP SHORT xconv.0040190B	
00401916	> B9 0C000000	MOV ECX, 0C	
0040191B	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 0	
00401922	. 49	DEC ECX	
00401923	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040192A	. 49	DEC ECX	
0040192B	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
00401932	. 3BC1	CFI EAX, ECX	
00401934	. 74 55	JE SHORT xconv.0040190B	
00401936	. 49	DEC ECX	
00401937	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040193E	. 3BC1	CFI EAX, ECX	
00401940	. 74 49	JE SHORT xconv.0040190B	
00401942	. 49	DEC ECX	
00401943	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040194A	. 3BC1	CFI EAX, ECX	
0040194C	. 74 3D	JE SHORT xconv.0040190B	
0040194E	. 49	DEC ECX	
0040194F	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
00401956	. 3BC1	CFI EAX, ECX	
00401958	. 74 31	JE SHORT xconv.0040190B	
0040195A	. 49	DEC ECX	
0040195B	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
00401962	. 3BC1	CFI EAX, ECX	
00401964	. 74 25	JE SHORT xconv.0040190B	
00401966	. 49	DEC ECX	
00401967	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040196E	. 3BC1	CFI EAX, ECX	
00401970	. 74 19	JE SHORT xconv.0040190B	
00401972	. 49	DEC ECX	
00401973	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
0040197A	. 3BC1	CFI EAX, ECX	
0040197C	. 74 0D	JE SHORT xconv.0040190B	
0040197E	. 49	DEC ECX	
0040197F	. C681 2D544000	MOV BYTE PTR DS:[ECX+40542D], 20	
00401986	. 49	DEC ECX	
00401987	. 3BC1	CFI EAX, ECX	
0040198B	> 68 2D544000	PUSH xconv.0040542D	
00401990	. E8 30E54000	PUSH xconv.0040543A	
00401995	. E8 2A020000	CALL <JMP.&kernel32.lstrcpyA>	[String2 = "DeRtH"
0040199A	. 6A 21	PUSH 21	String1 = xconv.0040543A
0040199C	. 68 9A534000	PUSH xconv.004053BA	lstrcpyA
004019A1	. E8 96020000	CALL <JMP.&kernel32.RtlZeroMemory>	[Length = 21 (33.)
004019A8	. 6A 21	PUSH 21	Destination = xconv.004053BA
			RtlZeroMemory
			[Length = 21 (33.)

```
eax = strlen(buf_42D);
ecx = 12;
buf_42D[ecx] = 00;
for(; eax != ecx; ecx--)
    buf_42D[ecx] = 20;

buf_43A = buf_42D;
buf_3BA = zeros(33);
```



Reverse Engineering of Software

Example of keygenning (5) – file X-Converter.zip

004019BE	. 6A 10	PUSH 10	
004019C0	. 68 1D544000	PUSH xconv.0040541D	[Length = 10 (16.) Destination = xconv.0040541D
004019C5	. E9 E2010000	CALL <JMP.&kernel32.RtlZeroMemory>	RtlZeroMemory
004019D0	. 81F2 FFFFFFF0F	XOR EDX,0FFFFFFF	
004019D6	. 52	PUSH EDX	[<><> Format = "%2x" s = xconv.004053B8 vsprintfA
004019D7	. 68 C9324000	PUSH xconv.004032C9	
004019DC	. 68 B8534000	PUSH xconv.004053BA	
004019E1	. E9 36010000	CALL <JMP.&user32.wsprintfA>	vsprintfA
004019E6	. 83C4 0C	ADD ESP,0C	
004019E8	. 8B15 31544000	MOV EDI,DIWORD PTR DS:[405431]	
004019EF	. 81F2 63739802	XOR EDX,2987363	
004019F5	. 52	PUSH EDX	[<><> Format = "%2x" s = xconv.004053DB vsprintfA
004019F6	. 68 C9324000	PUSH xconv.004032C9	
004019FB	. 68 DB534000	PUSH xconv.004053DB	
00401A00	. E9 17010000	CALL <JMP.&user32.wsprintfA>	vsprintfA
00401A05	. 83C4 0C	ADD ESP,0C	
00401A08	. 8B15 35544000	MOV EDI,DIWORD PTR DS:[405435]	
00401A0E	. 81F2 697A0000	XOR EDX,7A69	
00401A14	. 52	PUSH EDX	[<><> Format = "%2x" s = xconv.004053FC vsprintfA
00401A15	. 68 C9324000	PUSH xconv.004032C9	
00401A1A	. 68 FC534000	PUSH xconv.004053FC	
00401A1F	. E9 F9000000	CALL <JMP.&user32.wsprintfA>	vsprintfA
00401A24	. 83C4 0C	ADD ESP,0C	
00401A27	. 68 B8534000	PUSH xconv.004053BA	[ConcatString = "" ConcatString = ""
00401A2C	. 68 1D544000	PUSH xconv.0040541D	StringToAdd = ""
00401A31	. 68 82010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = "%-"
00401A36	. 68 2D334000	PUSH xconv.0040332D	ConcatString = ""
00401A3B	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A40	. E9 73010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = ""
00401A44	. 68 DB534000	PUSH xconv.004053DB	ConcatString = ""
00401A4F	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A54	. E9 64010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = ""
00401A5A	. 68 2D334000	PUSH xconv.0040332D	ConcatString = ""
00401A5F	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A65	. E9 55010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = ""
00401A6B	. 68 FC534000	PUSH xconv.004053FC	ConcatString = ""
00401A68	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A72	. 68 82010000	CALL <JMP.&kernel32.lstrcatA>	String2 = "" String1 = "" lstrcatA
00401A77	. 68 62544000	PUSH xconv.00405462	
00401A7C	. 68 1D544000	PUSH xconv.0040541D	
00401A81	. E9 36010000	CALL <JMP.&kernel32.lstrcatA>	
00401A84	. C9 00	CMPL 00	
00401A86	. 74 02	JE SHORT xconv.00401A88	
00401A88	. EB 07	JMP SHORT xconv.00401A8F	
00401A89	. B8 01000000	MOV EAX,1	
00401A8D	. EB 07	JMP SHORT xconv.00401A96	
00401A8F	. B8 00000000	MOV EAX,0	
00401A94	. EB 00	JMP SHORT xconv.00401A96	
00401A96	. 59	POP ECX	
00401A97	. 5D	POP EDI	
00401A98	. 5F	POP EDI	
00401A99	. 5B	POP EBX	
00401A9B	. C3	RETN	

```
eax = strlen(buf_42D);  
ecx = 12;  
buf_42D[ecx] = 00;  
for(; eax != ecx; ecx--)  
    buf_42D[ecx] = 20;
```

```
buf_43A = buf_42D;  
buf_3BA = zeros(33);  
buf_3DB = zeros(33);  
buf_3FC = zeros(33);  
buf_41D = zeros(16);
```



Reverse Engineering of Software

Example of keyenging (6) – file X-Converter.zip

0040198E	. 6A 10	PUSH 10	
00401990	. 68 10544000	PUSH xconv.00405410	[Length = 10 (16.) Destination = xconv.00405410 RtlZeroMemory
004019C4	. 8B15 2D544000	MOV EDI, DWORD PTR DS:[40542D]	
004019D0	. 81F2 FFFFFFFF	XOR EDI, 0FFFFFFF	
004019D6	. 52	PUSH EDI	[CXX] Format = ""%u" s = xconv.004053BA vsprintf
004019D7	. 68 C9324000	PUSH xconv.004032C9	
004019DC	. 68 B0534000	PUSH xconv.004053BA	
004019E1	. E9 36010000	CALL <JMP.>user32.vsprintfA	
004019E9	. 8B15 31544000	MOV EDI, DWORD PTR DS:[405431]	
004019EF	. 81F2 63739802	XOR EDI, 2987363	
004019F5	. 52	PUSH EDI	[CXX] Format = ""%u" s = xconv.004053DB vsprintf
004019F6	. 68 C9324000	PUSH xconv.004032C9	
004019FB	. 68 D8534000	PUSH xconv.004053DB	
00401A00	. E8 17010000	CALL <JMP.>user32.vsprintfA	
00401A05	. 83C4 0C	ADD ESP, 0C	
00401A08	. 8B15 35544000	MOV EDI, DWORD PTR DS:[405435]	
00401A0E	. 81F2 697A0000	XOR EDI, 7A69	
00401A14	. 52	PUSH EDI	[CXX] Format = ""%u" s = xconv.004053FC vsprintf
00401A15	. 68 C9324000	PUSH xconv.004032C9	
00401A1A	. 68 F0000000	PUSH xconv.004053FC	
00401A1F	. E8 79010000	CALL <JMP.>user32.vsprintfA	
00401A24	. 83C4 0C	ADD ESP, 0C	
00401A27	. 68 B0534000	PUSH xconv.004053BA	[StringToAdd = "" ConcatString = ""
00401A28	. 68 10544000	PUSH xconv.00405410	[lstrcatA StringToAdd = "" ConcatString = ""
00401A31	. E8 82010000	CALL <JMP.>kernel32.lstrcatA	[lstrcatA StringToAdd = "" ConcatString = ""
00401A36	. 68 2D334000	PUSH xconv.0040332D	[StringToAdd = "" ConcatString = ""
00401A3B	. 68 10544000	PUSH xconv.00405410	[lstrcatA StringToAdd = "" ConcatString = ""
00401A3E	. E8 79010000	CALL <JMP.>kernel32.lstrcatA	[lstrcatA StringToAdd = "" ConcatString = ""
00401A44	. 68 D8534000	PUSH xconv.004053DB	[StringToAdd = "" ConcatString = ""
00401A4A	. 68 10544000	PUSH xconv.00405410	[lstrcatA StringToAdd = "" ConcatString = ""
00401A4F	. E8 64010000	CALL <JMP.>kernel32.lstrcatA	[lstrcatA StringToAdd = "" ConcatString = ""
00401A54	. 68 2D334000	PUSH xconv.0040332D	[StringToAdd = "" ConcatString = ""
00401A59	. 68 10544000	PUSH xconv.00405410	[lstrcatA StringToAdd = "" ConcatString = ""
00401A5E	. E8 55010000	CALL <JMP.>kernel32.lstrcatA	[lstrcatA StringToAdd = "" ConcatString = ""
00401A63	. 68 F0534000	PUSH xconv.004053FC	[StringToAdd = "" ConcatString = ""
00401A68	. 68 10544000	PUSH xconv.00405410	[lstrcatA StringToAdd = "" ConcatString = ""
00401A6D	. E8 46010000	CALL <JMP.>kernel32.lstrcatA	[lstrcatA StringToAdd = "" ConcatString = ""
00401A72	. 68 62544000	PUSH xconv.00405462	[String2 = "" String1 = ""
00401A77	. 68 10544000	PUSH xconv.00405410	[String2 = "" String1 = ""
00401A7C	. E8 30010000	CALL <JMP.>kernel32.lstrcpwA	[lstrcpwA
00401A81	. 83F8 00	CMP EAX, 0	
00401A84	. 74 02	JE SHORT xconv.00401A88	
00401A86	. <EB 07	JMP SHORT xconv.00401A8F	
00401A88	. > B8 01000000	MOV EAX, 1	
00401A8D	. <EB 07	JMP SHORT xconv.00401A96	
00401A8F	. > B8 00000000	MOV EAX, 0	
00401A94	. <EB 00	JMP SHORT xconv.00401A96	
00401A96	. > 59	POP EAX	
00401A97	. 5A	POP EDI	
00401A98	. 5B	POP EDI	
00401A99	. 5C	POP EBX	
00401A9B	. C3	RETN	

```
eax = strlen(buf_42D);  
ecx = 12;  
buf_42D[ecx] = 00;  
for(; eax != ecx; ecx--)  
    buf_42D[ecx] = 20;
```

```
buf_43A = buf_42D;  
buf_3BA = zeros(33);  
buf_3DB = zeros(33);  
buf_3FC = zeros(33);  
buf_41D = zeros(16);
```

```
edx = buf_42D; // 4 bytes  
edx ^= 0xFFFFFFFF;  
buf_3BA = <edx in string>  
// Recall endianness!!
```



Reverse Engineering of Software

Example of keygenning (7) – file X-Converter.zip

0040198E	. 6A 10	PUSH 10	
00401990	. 65 10544000	PUSH xconv.0040541D	[Length = 10 (16.)
004019C5	. E9 E2010000	CALL <JMP.&kernel32.RtlZeroMemory>	RtlZeroMemory
004019C9	. 8B15 20544000	MOV EDX, DWORD PTR DS:[40542D]	
004019D0	. 81F2 FFFFFFFF	XOR EDI, 0FFFFFFF	
004019D5	. 52	PUSH EDI	{CXX}
004019D7	. 68 C9324000	PUSH xconv.004032C9	Format = "%2x"
004019DC	. 68 B0534000	PUSH xconv.004053BA	s = xconv.004053BA
004019E1	. E9 36010000	CALL <JMP.&user32.wsprintfA>	wsprintfA
004019E9	. 8B15 31544000	MOV EDX, DWORD PTR DS:[405431]	{CXX}
004019EF	. 81F2 63739802	XOR EDX, 2987363	ConcatString = ""
004019F5	. 52	PUSH EDI	Format = "%2x"
004019F6	. 68 C9324000	PUSH xconv.004032C9	ConcatString = ""
004019FB	. 68 D0534000	PUSH xconv.004053DB	s = xconv.004053DB
00401A00	. E9 17010000	CALL <JMP.&user32.wsprintfA>	wsprintfA
00401A09	. 8B15 35544000	MOV EDX, DWORD PTR DS:[405435]	{CXX}
00401A0E	. 81F2 697A0000	XOR EDI, 7A69	Format = "%2x"
00401A14	. 52	PUSH EDI	ConcatString = ""
00401A15	. 68 C9324000	PUSH xconv.004032C9	Format = "%2x"
00401A1A	. 68 F5340000	PUSH xconv.004053FC	s = xconv.004053FC
00401A1F	. E9 F8000000	CALL <JMP.&user32.wsprintfA>	wsprintfA
00401A24	. 83C4 0C	ADD ESP, 0C	
00401A27	. 68 B0534000	PUSH xconv.004053BA	{StringToAdd = ""
00401A2C	. 65 10544000	PUSH xconv.0040541D	ConcatString = ""
00401A31	. E9 82010000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA
00401A36	. 68 2D334000	PUSH xconv.0040332D	{StringToAdd = ""
00401A3B	. 68 10544000	PUSH xconv.0040541D	ConcatString = ""
00401A40	. E9 82010000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA
00401A45	. 68 D0534000	PUSH xconv.004053DB	{StringToAdd = ""
00401A4A	. 68 10544000	PUSH xconv.0040541D	ConcatString = ""
00401A4F	. E9 6A010000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA
00401A54	. 68 2D334000	PUSH xconv.0040332D	{StringToAdd = ""
00401A59	. 68 10544000	PUSH xconv.0040541D	ConcatString = ""
00401A5E	. E9 55010000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA
00401A63	. 68 FC534000	PUSH xconv.004053FC	{StringToAdd = ""
00401A68	. 65 10544000	PUSH xconv.0040541D	ConcatString = ""
00401A6D	. E9 46010000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA
00401A72	. 68 62544000	PUSH xconv.00405462	{String2 = ""
00401A77	. E9 10544000	PUSH xconv.0040541D	String2 = ""
00401A7C	. E9 30010000	CALL <JMP.&kernel32.lstrcpwA>	lstrcpwA
00401A81	. 83F8 00	CHP EAX, 0	
00401A84	> 74 02	JZ SHORT xconv.00401A88	
00401A86	> EB 07	JMP SHORT xconv.00401A88	
00401A89	> 85 01000000	MOV EAX, 1	
00401A8D	> EB 07	JMP SHORT xconv.00401A96	
00401A8F	> B9 00000000	MOV EAX, 0	
00401A94	> EB 00	JMP SHORT xconv.00401A96	
00401A96	> 5A	POP ECX	
00401A97	> 5A	POP EDI	
00401A98	> 5F	POP EDI	
00401A99	> 5E	POP EBX	
00401A9E	> C3	RETN	

```
eax = strlen(buf_42D);
ecx = 12;
buf_42D[ecx] = 00;
for(; eax != ecx; ecx--)
```

```
buf_43A = buf_42D;
buf_3BA = zeros(33);
buf_3DB = zeros(33);
buf_3FC = zeros(33);
buf_41D = zeros(16);
```

```
edx = buf_42D; // 4 bytes
edx ^= 0xFFFFFFFF;
buf_3BA = <edx in string>
// Recall endianness!!
```

```
edx = (buf_42D + 4); // 4 bytes
edx ^= 0x2987363;
buf_3DB = <edx in string>
// Recall endianness!!
```



Reverse Engineering of Software

Example of keygenning (8) – file X-Converter.zip

004019BE	. 6A 10	PUSH 10	
004019C0	. 68 1D544000	PUSH xconv.0040541D	[Length = 10 (16.)
004019C5	. F3 E2010000	CALL <JMP.&kernel32.RtlZeroMemory>	Destination = xconv.0040541D
004019CA	. 8B15 2D544000	MOV EDI, DWORD PTR DS:[40542D]	RtlZeroMemory
004019D0	. 81F2 FFFFFFF0F	XOR EDI, 0FFFFFFF	
004019D6	. 52	PUSH EDI	<>>
004019D7	. 68 C9324000	PUSH xconv.004032C9	Format = "%2x"
004019DC	. 68 B8534000	PUSH xconv.004053BA	s = xconv.004053BA
004019E1	. EB 36010000	CALL <JMP.&user32.wsprintfA>	vsprintfA
004019E6	. 83C4 0C	ADD ESP, 0C	
004019E9	. 8B15 31544000	MOV EDI, DWORD PTR DS:[405431]	<>>
004019EF	. 81F2 63739802	XOR EDI, 2987363	Format = "%2x"
004019F5	. 52	PUSH EDI	s = xconv.004053DB
004019F6	. 68 C9324000	PUSH xconv.004032C9	vsprintfA
004019FB	. 68 D8534000	PUSH xconv.004053DB	
00401A00	. EB 17010000	CALL <JMP.&user32.wsprintfA>	
00401A08	. 8B15 35544000	MOV EDI, DWORD PTR DS:[405435]	<>>
00401A0E	. 81F2 69700000	XOR EDI, 7069	Format = "%2x"
00401A14	. 52	PUSH EDI	s = xconv.004053FC
00401A15	. 68 C9324000	PUSH xconv.004032C9	vsprintfA
00401A1A	. 68 FC534000	PUSH xconv.004053FC	
00401A1F	. EB F3000000	CALL <JMP.&user32.wsprintfA>	
00401A27	. 68 B8534000	PUSH xconv.004053BA	[StringToAdd = ""
00401A2C	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A31	. 68 0A010000	CALL <JMP.&kernel32.lstrcatA>	StringToAdd = ""
00401A36	. 68 2D334000	PUSH xconv.0040332D	ConcatString = ""
00401A3B	. 68 1D544000	PUSH xconv.0040541D	StringToAdd = ""
00401A40	. EB 73010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = ""
00401A4D	. 68 D8534000	PUSH xconv.004053DB	StringToAdd = ""
00401A4A	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A4F	. EB 64010000	CALL <JMP.&kernel32.lstrcatA>	StringToAdd = ""
00401A54	. 68 2D334000	PUSH xconv.0040332D	ConcatString = ""
00401A59	. 68 1D544000	PUSH xconv.0040541D	StringToAdd = ""
00401A5E	. EB 55010000	CALL <JMP.&kernel32.lstrcatA>	ConcatString = ""
00401A63	. 68 FC534000	PUSH xconv.004053FC	StringToAdd = ""
00401A68	. 68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A6D	. EB 44010000	CALL <JMP.&kernel32.lstrcatA>	String2 = ""
00401A72	. 68 62544000	PUSH xconv.00405462	String1 = ""
00401A77	. 68 1D544000	PUSH xconv.0040541D	lstrcatA
00401A7D	. EB 30010000	CALL <JMP.&kernel32.lstrcatA>	
00401A82	. 83F8 00	CFE XOR, 0	
00401A84	. > 74 02	JE SHORT xconv.00401A88	
00401A86	. > EB 07	JMP SHORT xconv.00401A8F	
00401A88	. > BB 01000000	MOV EDI, 1	
00401A8D	. > < 07	JMP SHORT xconv.00401A96	
00401A8F	. > BB 00000000	MOV EDI, 0	
00401A94	. > EB 00	JMP SHORT xconv.00401A96	
00401A96	. > 59	POP ECX	
00401A97	. > 5A	POP EDI	
00401A98	. > 5F	POP EDI	
00401A99	. > 5B	POP EBX	
00401A9B	. > C3	RETN	

```
eax = strlen(buf_42D);
ecx = 12;
buf_42D[ecx] = 00;
for(; eax != ecx; ecx--)
    buf_42D[ecx] = 20;
```

```
buf_43A = buf_42D;
buf_3BA = zeros(33);
buf_3DB = zeros(33);
buf_3FC = zeros(33);
buf_41D = zeros(16);
```

```
edx = buf_42D; // 4 bytes
edx ^= 0xFFFFFFFF;
buf_3BA = <edx in string>
// Recall endianness!!
```

```
edx = (buf_42D + 4); // 4 bytes
edx ^= 0x2987363;
buf_3DB = <edx in string>
// Recall endianness!!
```

```
edx = (buf_42D + 8); // 4 bytes
edx ^= 0x7A69;
buf_3FC = <edx in string>
// Recall endianness!!
```



Reverse Engineering of Software

Example of keygenning (9) – file X-Converter.zip

004019EE	6A 10	PUSH 10	[length = 10 (16.)
004019EB	68 1D544000	PUSH xconv.0040541D	Destination: xconv.0040541D
004019C5	E9 E2010000	CALL <JMP.&kernel32.RtlZeroMemory>	RtlZeroMemory
004019C9	8B15 2D544000	MOV EDI, DWORD PTR DS:[40542D]	
004019C0	81F2 FFFFFFFF	XOR EDI, 0FFFFFFF	
004019D6	52	PUSH EDI	
004019D7	68 C9324000	PUSH xconv.004032C9	{CWD
004019D0	68 B4534000	PUSH xconv.004053BA	Format = "%*x"
004019E1	68 36010000	CALL <JMP.&user32.wsprintfA>	% = xconv.004053BA
004019E6	83C4 0C	ADD ESP, 0C	wsprintfA
004019E9	8B15 31544000	MOV EDI, DWORD PTR DS:[405431]	
004019EF	81F2 63739802	XOR EDI, 2987363	{CWD
004019F5	52	PUSH EDI	Format = "%*x"
004019F6	68 C9324000	PUSH xconv.004032C9	% = xconv.004053DB
004019FB	68 D6534000	PUSH xconv.004053DB	wsprintfA
004019A0	E9 17010000	CALL <JMP.&user32.wsprintfA>	
004019E5	83C4 0C	ADD ESP, 0C	
00401A08	8B15 35544000	MOV EDI, DWORD PTR DS:[405435]	
00401A0E	81F2 697A0000	XOR EDI, 7A69	{CWD
00401A14	52	PUSH EDI	Format = "%*x"
00401A15	68 C9324000	PUSH xconv.004032C9	% = xconv.004053FC
00401A1A	68 FC534000	PUSH xconv.004053FC	wsprintfA
00401A1F	E9 F0000000	CALL <JMP.&user32.wsprintfA>	
00401A27	68 B4534000	PUSH xconv.004053BA	{StringToAdd = ""
00401A2C	68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A21	E9 20310000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A36	68 2D334000	PUSH xconv.004032D0	{StringToAdd = ""
00401A40	68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A48	E9 79010000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A45	68 D6534000	PUSH xconv.004053DB	{StringToAdd = ""
00401A4A	68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A4F	68 64010000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A54	68 2D334000	PUSH xconv.004032D0	{StringToAdd = ""
00401A59	68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A5E	E9 55010000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A63	68 F5340000	PUSH xconv.004053FC	{StringToAdd = ""
00401A68	68 1D544000	PUSH xconv.0040541D	ConcatString = ""
00401A6D	E9 46010000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A77	68 1D544000	PUSH xconv.0040541D	{String2 = ""
00401A7C	E9 3D010000	CALL <JMP.&kernel32.lstrcatA>	{lstrcatA
00401A81	83F8 00	CMPL EAX, 0	
00401A84	> 74 02	JNE SHORT xconv.00401A89	
00401A86	> EB 07	JMP SHORT xconv.00401A8E	
00401A88	> BB 01000000	MOV EBX, 1	
00401A8D	> EB 07	JMP SHORT xconv.00401A96	
00401A8F	> BB 00000000	MOV EBX, 0	
00401A94	> EB 00	JMP SHORT xconv.00401A96	
00401A96	> 59	POP ECX	
00401A97	> 5A	POP EDI	
00401A98	> 5F	POP EDI	
00401A99	> 5B	POP EBX	
00401A9B	> C3	RETN	

```
eax = strlen(buf_42D);
ecx = 12;
buf_42D[ecx] = 00;
for(; eax != ecx; ecx--)
    buf_42D[ecx] = 20;
```

```
buf_43A = buf_42D;
buf_3BA = zeros(33);
buf_3DB = zeros(33);
buf_3FC = zeros(33);
buf_41D = zeros(16);
```

```
edx = buf_42D; // 4 bytes
edx ^= 0xFFFFFFFF;
buf_3BA = <edx in string>
// Recall endianness!!
```

```
edx = (buf_42D + 4); // 4 bytes
edx ^= 0x2987363;
buf_3DB = <edx in string>
// Recall endianness!!
```

```
edx = (buf_42D + 8); // 4 bytes
edx ^= 0x7A69;
buf_3FC = <edx in string>
// Recall endianness!!
```

```
buf_41D = buf_3BA & '-'
        & buf_3DB & '-' & buf_3FC;
```

Reverse Engineering of Software

Example of keygenning (10) – file X-Converter.zip

```
void generate_key(char *name, int len)
{
    int bufLen = 32;
    char    buf_3BA[bufLen], buf_3DB[bufLen],
            buf_3FC[bufLen], buf_41D[15],
            nameFilled[12];

    // Init
    memset(nameFilled, ' ', 12);
    for(int i = 0; i < len; i++)
        nameFilled[i] = name[i];

    memset(buf_3BA, 0, bufLen);
    memset(buf_3DB, 0, bufLen);
    memset(buf_3FC, 0, bufLen);
    memset(buf_41D, 0, 15);

    // Compute 1st num
    sprintf(buf_3BA, "%X",
            compute_number(nameFilled, 0, 0xFFFFFFFF));
    // Compute 2nd num
    sprintf(buf_3DB, "%X",
            compute_number(nameFilled, 4, 0x2987363));
    // Compute 3rd num
    sprintf(buf_3FC, "%X",
            compute_number(nameFilled, 8, 0x7A69));

    // Build key
    sprintf(buf_41D, "%s-%s-%s",
            buf_3BA, buf_3DB, buf_3FC);
    printf("Dear %s, your key is %s.\n", name, buf_41D);
}

int compute_number(char *name,
                   int init, int xor_val)
{
    int edx = name[init];

    for(int i = 1; i < 4; i++)
        edx ^= name[init + i] << 8*i;

    return edx^xor_val;
}

/* Example of first num computation
edx = nameFilled[0];
edx ^= nameFilled[1] << 8;
edx ^= nameFilled[2] << 16;
edx ^= nameFilled[3] << 24;
// XOR it
edx ^= 0xFFFFFFFF; */
```



Agenda

- 1 Introduction to Reverse Engineering
- 2 Reverse Engineering of Protocols
- 3 Reverse Engineering of Software
- 4 Reverse Engineering of Integrated Circuits/Smart Cards**
 - Near Field Communication (NFC): What is it?
 - MIFARE classic: What is it?
 - Related Work
 - Android and NFC: A Tale of L♥ve
 - Problem Analysis
- 5 Conclusions



Reverse Engineering of Integrated Circuits/Smart Cards



Near Field Communication ¿?



Near Field Communication: What is it? (I)

- Bidirectional short-range contactless communication technology
 - Up to 10 cm
- Based on RFID standards, works in the 13.56 MHz spectrum
- Data transfer rates vary: 106, 216, and 424 kbps



Near Field Communication: What is it? (I)

- Bidirectional short-range contactless communication technology
 - Up to 10 cm
- Based on RFID standards, works in the 13.56 MHz spectrum
- Data transfer rates vary: 106, 216, and 424 kbps

Security based on proximity concern: physical constraints

Near Field Communication: What is it? (I)

- **Bidirectional short-range contactless communication technology**
 - Up to 10 cm
- Based on RFID standards, **works in the 13.56 MHz spectrum**
- Data **transfer rates vary**: 106, 216, and 424 kbps

Security based on **proximity concern: physical constraints**

Main elements & operation modes

- Two main elements:
 - **Proximity Coupling Device** (PCD, also NFC-capable device)
 - **Proximity Integrated Circuit Cards** (PICC, also NFC tags)
- Three operation modes:
 - **Peer to peer**: direct communication between parties
 - **Read/write**: communication with a NFC tag
 - **Card-emulation**: an NFC device behaves as a tag

Near Field Communication: What is it? (II)

“Big” actors

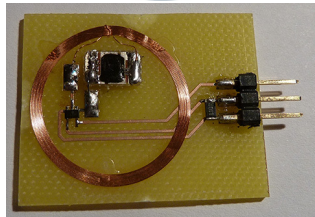


NFC Forum

- **Non-profit industry association**
- Formed on March 18, 2004
- Founders: NXP Semiconductors (formerly Philips Semiconductors), Sony and Nokia
- **Promotes implementation and standardisation** of NFC
- 190 member companies (June 2013). Some located at Spain:
 - Applus
 - AT4 Wireless

Near Field Communication: What is it? (III)

Real actors (1)



PICC

- Proximity Integrated Circuit Card
- Commonly named as *tag*
- Passive or active (depends on power supply)
 - Widely used (cheaper): passive ones
- It contains:
 - Internal capacitor
 - Stores the energy coming from the reader
 - Resistor

Near Field Communication: What is it? (III)

Real actors (2)

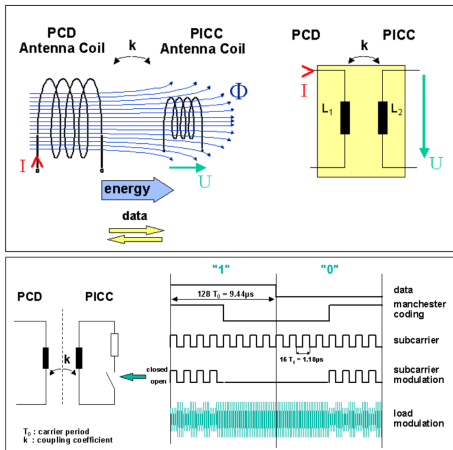


PCD

- Proximity Coupling Device
- Commonly named as *reader/writer*
- Active (forced)
- Contains the **antenna**
 - Communication at the 13.56MHz ($\pm 7\text{kHz}$) frequency
 - Electronic field

Near Field Communication: What is it? (IV)

An interesting reading on this topic...



[Taken from 13.56 MHz RFID Proximity Antennas

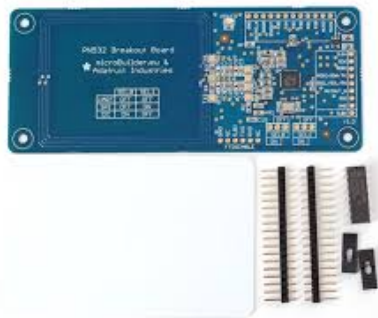
(http://www.nxp.com/documents/application_note/AN78010.pdf)]

Near Field Communication: Where is it used? (V)



Near Field Communication

Lab Environment



Hardware

- AdaFruit PN532
- A computer

Software

- C compiler
- NFC Library (`libnfc`)
- NFC tools (`nfc-tools`)

MIFARE classic ¿?



MIFARE Classic (I): What is it?

MIFARE product family

- Introduced in 1995 by NXP
- “Advanced technology for RFID identification”
- Based on **ISO/IEC 14443 Type A/B 13.56 MHz** standard
- Several products:
 - Ultralight
 - **Classic**
 - DESFire
 - SmartMX



MIFARE Classic (I): What is it?

MIFARE product family

- Introduced in 1995 by NXP
- “Advanced technology for RFID identification”
- Based on **ISO/IEC 14443 Type A/B 13.56 MHz** standard
- Several products:
 - Ultralight
 - **Classic**
 - DESFire
 - SmartMX
- **50M reader and 5B card components** sold
- **~ 80% contactless ticketing credentials** (according to ABI Research)



MIFARE Classic (II): Some of its common uses

Some systems using MIFARE Classic

- **Access Controls**
 - University of Zaragoza
 - Personal entrance Schiphol Airport (AMS)
 - Dutch military bases
 - Hotel room keys
 - Many office and official buildings
- **Ticketing events**
- **Public transport systems**
 - OV-Chipkaart (NL)
 - Oyster card (London, UK)
 - Smartrider (AU)
 - EMT (Málaga, Spain)
 - Wikipedia: <http://en.wikipedia.org/wiki/MIFARE>



MIFARE Classic (III): Internal Structure (1)

Logical Structure

- **EEPROM memory**
- Basic unit: **16B block**
- A **sector** is a set of blocks
- **Two size variants:**
 - 1KB (16 sectors, 4 blocks each)
 - 4KB (40 sectors, first 32 sectors are 4-block, the rest 16-block)



MIFARE Classic (III): Internal Structure (1)

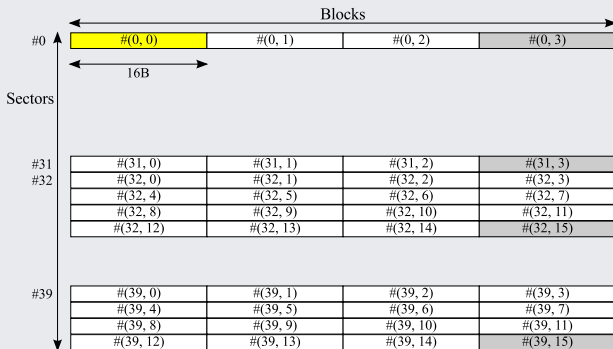
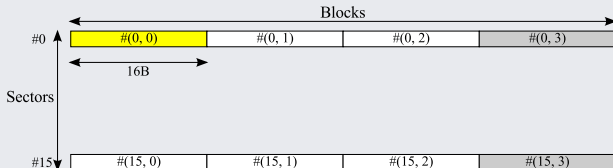
Logical Structure

- **EEPROM memory**
- Basic unit: **16B block**
- A **sector** is a set of blocks
- **Two size variants:**
 - 1KB (16 sectors, 4 blocks each)
 - 4KB (40 sectors, first 32 sectors are 4-block, the rest 16-block)

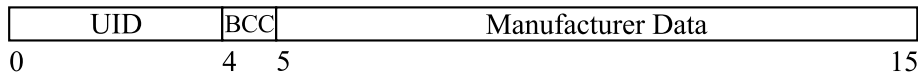
Let me show you this graphically. . .



MIFARE Classic (III): Internal Structure(2)



MIFARE Classic (III): Internal Structure (3)

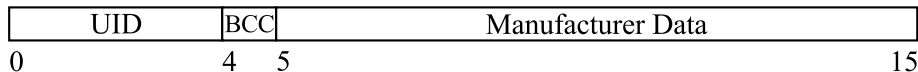


Manufacturer block

- **Sector 0, block 0** (yellow one in previous slide)
- Contains:
 - UID (4B)
 - BCC (bit count check, 1B): XOR-ing of UID bytes
 - Manufacturer data (11B)
- **Set and locked by manufacturer → read only!**



MIFARE Classic (III): Internal Structure (3)



Manufacturer block

- **Sector 0, block 0** (yellow one in previous slide)
- Contains:
 - UID (4B)
 - BCC (bit count check, 1B): XOR-ing of UID bytes
 - Manufacturer data (11B)
- **Set and locked by manufacturer → read only!**
 - Not the case for some Chinese cards 😊

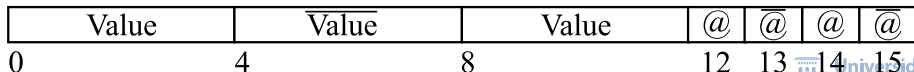


MIFARE Classic (III): Internal Structure (4)

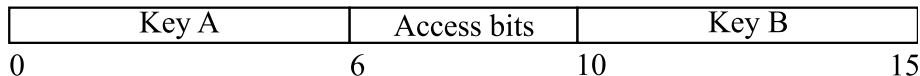
Storing data...

Storing data into blocks

- **Read/write block**
 - You can store data as you want, no matter how
- **Data block**
 - Predefined format (look below!)
 - Don't worry: APIs will help you!
 - Only need a *value*, it puts all the values properly on its own...
 - Contains:
 - Value (twice)
 - Value negated (once)
 - 1-byte address (twice)
 - 1-byte address negated (twice)



MIFARE Classic (III): Internal Structure (5)



Sector trailer

- **Last one in each sector** (grey ones in previous slide)
- Contains:
 - Key A
 - Access Bits
 - Key B
- **Authentication per sector** before any operation is allowed
- Access bits define how is the auth. required and what operations are allowed
- **Having fun with access bits may provoke a useless tag!**
- **Keys are set to FFFFFFFFh at delivery**

MIFARE Classic (III): Internal Structure (6)

Operations

Operation	Description	Valid for...		
		R/W block	Value block	Sector trailer
Read	Reads a memory block	√	√	√
Write	Writes a memory block	√	√	√
Increment	Reads the value, increments it and stores		√	
Decrement	Reads the value, decrements it and stores		√	
Transfer	Transfers contents of internal register to a block		√	
Restore	Loads contents of a block to internal register		√	

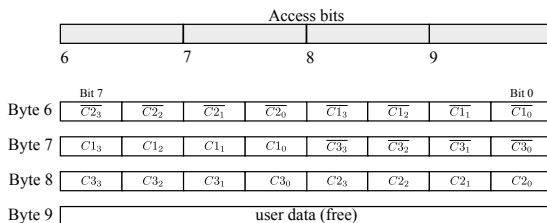


MIFARE Classic (III): Internal Structure (7)

Access Conditions

Access Bits	Valid Commands	Block
$C1_0 C2_0 C3_0$	(all operations)	0
$C1_1 C2_1 C3_1$	(all operations)	1
$C1_2 C2_2 C3_2$	(all operations)	2
$C1_3 C2_3 C3_3$	Read, Write	3

- 3 bits define access conditions for every data block and sector trailer
- Stored non-negated and negated
- Commands are executed only after a successful authentication



MIFARE Classic (III): Internal Structure (8)

Access Conditions for sector trailer

Access Bits			Access condition for ...					
C1	C2	C3	Key A		Access bits		Key B	
			read	write	read	write	read	write
0	0	0	-	key A	key A	-	key A	key A
0	0	1	-	key A	key A	key A	key A	key A
0	1	0	-	-	key A	-	key A	-
0	1	1	-	key B	key A (or B)	key B	-	key B
1	0	0	-	key B	key A (or B)	-	-	key B
1	0	1	-	-	key A (or B)	key B	-	-
1	1	0	-	-	key A (or B)	-	-	-
1	1	1	-	-	key A (or B)	-	-	-

(- means never)

Recall: show mfcab tool (<https://bitbucket.org/rjrodriguez/mfcab>)

MIFARE Classic (III): Internal Structure (9)

Access Conditions for data blocks

Access Bits			Access condition for...				Application
C1	C2	C3	Read	Write	Increment	Decrement, Transfer, Restore	
0	0	0	key A (or B) [†]	key A (or B)	key A (or B)	key A (or B)	Transport configuration
0	0	1	key A (or B) [†]	-	-	key A (or B)	Value block
0	1	0	key A (or B) [†]	-	-	-	R/W block
0	1	1	key B	key B	-	-	R/W block
1	0	0	key A (or B)	Key B	-	-	R/W block
1	0	1	key B	-	-	-	R/W block
1	1	0	key A (or B)	key B	key B	key A (or B)	Value block
1	1	1	-	-	-	-	R/W block

(- means never)

[†] if key B can be read in the sector trailer, then it cannot be used for authentication

MIFARE Classic: Communication Protocol (I)

Protocol steps

- 1 Get the tags in the reader's range
- 2 Select only one tag (anticollision loop)
- 3 Access a block, with key A or key B (starts authentication step)

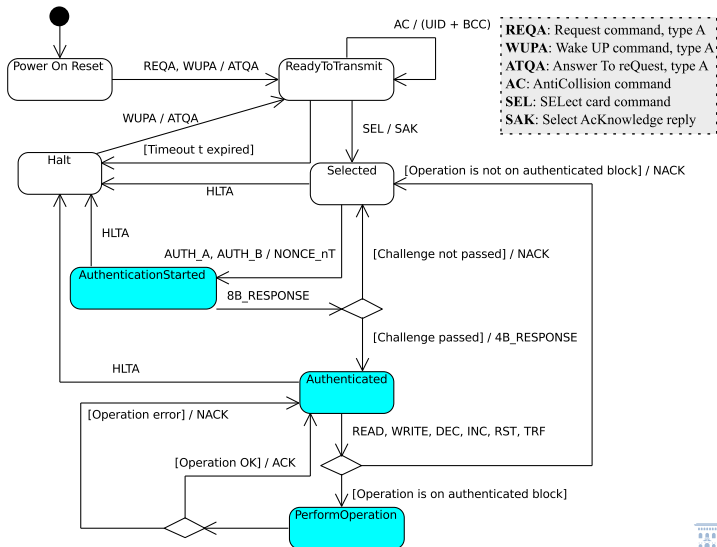
Authentication step

- Challenge-response mutual authentication using nonces
 - Nonce: randomly generated information
 - Nonces generated from a LFSR (next slides)

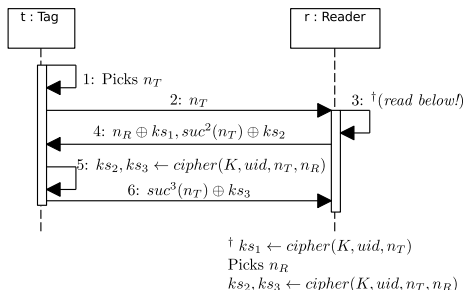


MIFARE Classic: Communication Protocol (II)

UML-SM of a NFC tag



MIFARE Classic: Communication Protocol (III)



• Three-pass authentication

① Send nonce (n_T) as challenge

- Generated by a 16-bit LFSR ($g(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$)

② Send response and other nonce n_R as challenge

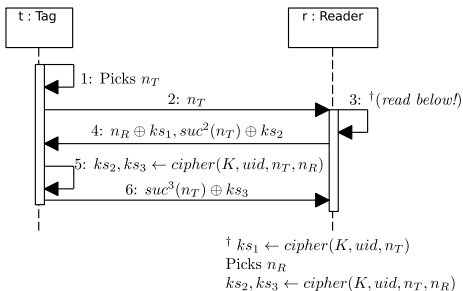
③ Send response

- **Note:** from n_T , communication is ciphered

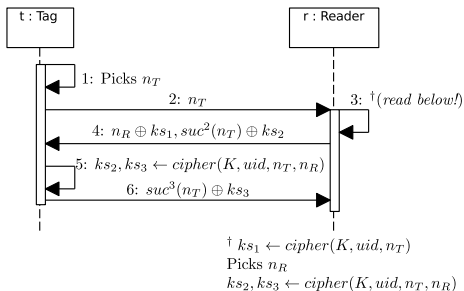
MIFARE Classic: Communication Protocol (IV)

Known plaintext [GKMRVSJ-ESORICS-08]

- **Recall:** n_T is in plaintext



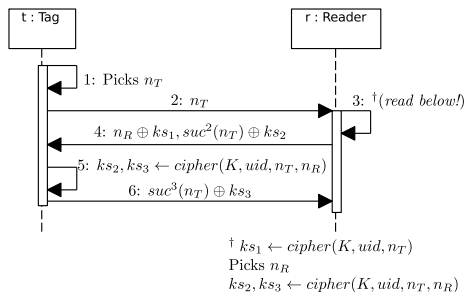
MIFARE Classic: Communication Protocol (IV)



Known plaintext [GKMRVSJ-ESORICS-08]

- **Recall:** n_T is in plaintext
- Given n_T , compute $suc^2(n_T) \rightarrow ks_2 = n_T \oplus suc^2(n_T)$

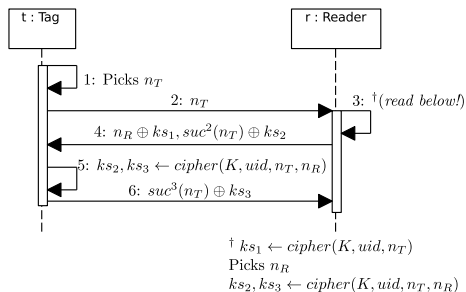
MIFARE Classic: Communication Protocol (IV)



Known plaintext [GKMRVSJ-ESORICS-08]

- **Recall:** n_T is in plaintext
- Given n_T , compute $suc^2(n_T) \rightarrow ks_2 = n_T \oplus suc^2(n_T)$
- When tag does not send last response, some readers time out and send HLT command XORed ks_3
 - HLT command is known, then we recover ks_3

MIFARE Classic: Communication Protocol (IV)



Known plaintext [GKMRVSJ-ESORICS-08]

- **Recall:** n_T is in plaintext
- Given n_T , compute $suc^2(n_T) \rightarrow ks_2 = n_T \oplus suc^2(n_T)$
- When tag does not send last response, some readers time out and send HLT command XORed ks_3
 - HLT command is known, then we recover ks_3
- **Eavesdropping a successful authentication session**
 - ks_2, ks_3 recovered from $suc^2(n_T) \oplus n_T, suc^3(n_T) \oplus n_T$

MIFARE Classic: CRYPTO1 (I)

- Proprietary **stream cipher**. Key length of **48 bits**
- **“Security by obscurity”** principle
- Hardware on-chip: **faster cryptographic operations!**
- Reversed some years ago. . . :
 - K. Nohl and H. Plötz: “Mifare: Little Security, Despite Obscurity”, in *Chaos Communication Congress*, 2007. Reverse engineering **on silicon implementation**
 - García et al.: “Dismantling MIFARE Classic”, in *ESORICS* 2008. **Fully disclosed the entire encryption algorithm**



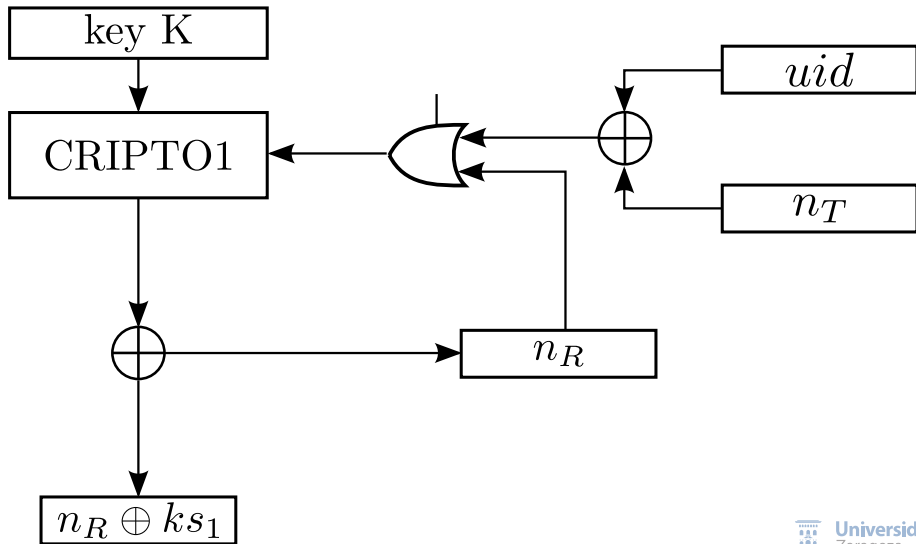
MIFARE Classic: CRYPTO1 (I)

- Proprietary **stream cipher**. Key length of **48 bits**
- **“Security by obscurity” principle**
- Hardware on-chip: **faster cryptographic operations!**
- Reversed some years ago. . . :
 - K. Nohl and H. Plötz: “Mifare: Little Security, Despite Obscurity”, in *Chaos Communication Congress*, 2007. Reverse engineering **on silicon implementation**
 - García et al.: “Dismantling MIFARE Classic”, in *ESORICS* 2008. **Fully disclosed the entire encryption algorithm**
- **Linear Feedback Shift Register (LFSR) + two-layer non-linear filter generator**
 - At every clock tick, register is shifted one bit to the left
 - Leftmost bit: discarded
 - Feedback bit: computed with $g(x)$

$$g(x) = x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1$$

MIFARE Classic: CRYPTO1 (II)

Initialisation diagram



MIFARE Classic: Known Weaknesses (I)

On Pseudo-Random Number Generator

MOST CRITICAL weakness

Low entropy

- **LFSR generating nonces: 16-bit length**
- 0.6 seconds to generate ALL possible nonces [NESP-USENIX-08]
- **Generator resets to a known state every time the tag starts operating**
 - Just a wait a fixed number of clock cycles. . .
 - Experimentally possible to get the same nonce every 30ms using Proxmark 3 reader

MIFARE Classic: Known Weaknesses (II)

On Cryptographic Cipher

$X_9, X_{11}, X_{13}, \dots, X_{47}$

Keystream generation

- Odd bits as inputs to the filter functions
- Divide-and-Conquer technique
 - Split even, odd bits in groups
 - Firstly focus on odd group:
 - After 2 shifts, new input is $x_{11}, x_{13}, \dots, x_{47}$ and x_{49}
 - **Used for generating two keystreams**
 - Explore what bits generate the right keystreams
- **Attack: Recover all sector keys without the needed of a genuine reader**



MIFARE Classic: Known Weaknesses (III)

On Cryptographic Cipher

$X_9, X_{11}, X_{13}, \dots, X_{47}$

Leftmost bit not used in filter generator

- First 9 bits unused
- **Attack: Rollback LFSR state bit a bit**
 - Recover the initial state of LFSR

Statistical Bias [C-SECURITY-09]

- With a $\pi = 0.75$, ks_1 is independent of the last three bits of n_R
- **Attack: card-only attack**
 - Recover one key, then apply nested authentication attack [GKMRVSJ-ESORICS-08]
 - Does not require any pre-computation
 - Extremely fast, and requires a few hundred queries
 - Further information: <http://eprint.iacr.org/2009/137.pdf>

MIFARE Classic: Known Weaknesses (IV)

On Communication Protocol

One-Time Padding (OTP)

- ISO-14443-A: every byte sent is followed by a parity bit
- MIFARE Classic **computes parity bit over plaintext instead of ciphertext**
- **LFSR is not shifted after parity bit encryption**



MIFARE Classic: Known Weaknesses (IV)

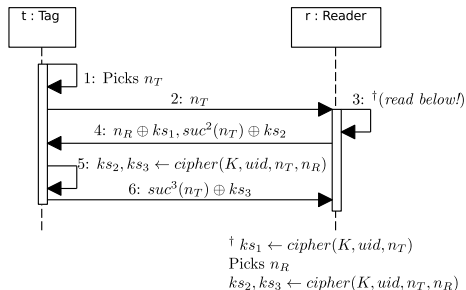
On Communication Protocol

One-Time Padding (OTP)

- ISO-14443-A: every byte sent is followed by a parity bit
- MIFARE Classic **computes parity bit over plaintext instead of ciphertext**
- **LFSR is not shifted after parity bit encryption**
- **Next plaintext and parity bit use the same keystream** → OTP seems not to be OTP...
- More examples of violating OTP property:
 - Venona Project (U.S. counter-intelligence program during Cold War)
 - Point-to-Point Tunneling Protocol (PPTP)
 - IEEE 802.11 WEP

MIFARE Classic: Known Weaknesses (V)

On Communication Protocol



Information Leak from Parity

- Second step in authentication, reader sends $n_R, suc^2(n_T)$
- **PICC checks parity bits in n_R before checking $suc^2(n_T)$**
 - When parity is incorrect, PICC does not answer
 - When $suc^2(n_T)$ is incorrect, it answers **NACK** (transmission error)
- **NACK sent encrypted $\rightarrow ks_3$ can be recovered**

MIFARE Classic: Known Weaknesses (VI)

On Deployment

Default Keys

- Some chip manufacturers leave default keys on chips
- This is obvious, as companies must make the effort to do system integration for clients... (sic!)
- RTFM: Chip manufacturer warns about CHANGING default keys
- Default keys are well-known and documented

FFFFFFFFFFFFh	000000000000h	1A982C7E459Ah
A0A1A2A3A4A5h	B0B1B2B3B4B5h	AABBCCDDEEFFh
D3F7D3F7D3F7h	4D3A99C351DDh	

Related Work (I)

On MIFARE Classic weaknesses analysis (1)

- NP-CCC-07 K. Nohl and H. Plötz, “**Mifare: Little Security, Despite Obscurity**”, in *Chaos Communication Congress*, 2007.
- GKMRVSJ-ESORICS-08 García et al., “**Dismantling MIFARE Classic**”, in *Procs. of the European Symposium on Research in Computer Security (ESORICS)*, 2008.
- KHG-CARDIS-08 G.d Koning Gans et al., “**A Practical Attack on the MIFARE Classic**”, in *Procs. of the Smart Card Research and Advanced Applications Conference (CARDIS)*, 2008.
- NESP-USENIX-08 K. Nohl et al., “**Reverse-Engineering a Cryptographic RFID Tag**”. In *USENIX Security Symposium*, 2008.
- GRBS-SP-09 F.D. García et al., “**Wirelessly Pickpocketing a Mifare Classic Card**”, in *Procs. of the 30th IEEE Symposium on Security and Privacy (S&P)*, 2009.

Related Work (II)

On MIFARE Classic weaknesses analysis (2)

C-SECURITY-09 N.T. Courtois, “**The Dark Side of Security by Obscurity and Cloning MiFare Classic Rail and Building Passes Anywhere, Anytime**”. In *Procs. of the Int. Conf. on Security and Cryptography (SECURITY)*, 2009

GRBS-SP-09 F.D. García et al., “**Wirelessly Pickpocketing a Mifare Classic Card**”, in *Procs. of the 30th IEEE Symposium on Security and Privacy (S&P)*, 2009

Tan-MScThesis-09 W.H. Tan, “**Practical Attacks on the MIFARE Classic**”, Imperial College London, 2009

On NFC Attacks

VK-NFC-11 R. Verdult and F. Kooman, “**Practical Attacks on NFC Enabled Cell Phones**”. In *Procs. of the 3rd Int. Workshop on Near Field Communication*, 2011

Related Work (III)

On MIFARE Attacks

- Sogeti ESEC Pentest: “Playing with NFC for fun and coffee”
- BackTrack Linux: “RFID Cooking with Mifare Classic” (2012)
- C. Miller, “Exploring the NFC Attack Surface”, in *BlackHat US*, 2012.
- ComputerWorld article: “Android NFC hack enables travelers to ride subways for free, researchers say” (2012)
- HackPlayers: “Cómo colarse en el metro de forma elegante” (2012)
- Security ArtWork: “Hacking RFID, rompiendo (...) Mifare” (2010)

On NFC-related issues

- R. Lifchitz, **Hacking the NFC credit cards for fun and debit** (Hackito Ergo Sum 2012)
- J.M. Esparza, **Give me your credit card, the NFC way** (NcN'12)
- J. Vila, R.J. Rodríguez, **Practical Experiences on NFC Relay Attacks with Android: Virtual Pickpocketing Revisited**. In *RFIDSec 2015*

Reverse Engineering of Integrated Circuits/Smart Cards

Extending the lab environment



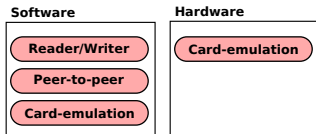
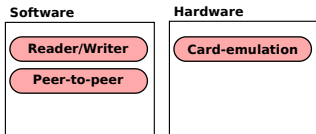
- NFC brings “cards” to mobile devices
- Payment sector is quite interested in this new way for making payments
 - 500M NFC payment users expected by 2019
- Almost 300 smart phones available at the moment with NFC capabilities
 - Check <http://www.nfcworld.com/nfc-phones-list/>
 - Most of them runs **Android** OS

Time to buy a NFC-capable device!

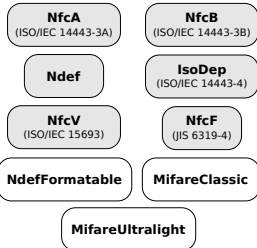
Android and NFC: A Tale of Love (I)

Recap on evolution of Android NFC support

NFC operation modes supported



Android 2.3.3 Gingerbread (API level 10)



Android 4.2 Jelly Bean (API level 17)

NfcBarcode

Android CyanogenMod OS 9.1

IsoPcdA
(ISO/IEC 14443-4A)

IsoPcdB
(ISO/IEC 14443-4B)

thanks to Doug Year

Android 4.4 KitKat (API level 19)

NfcAdapter.ReaderCallback added



Android and NFC: A Tale of L♥ve (II)

Digging into Android NFC stack

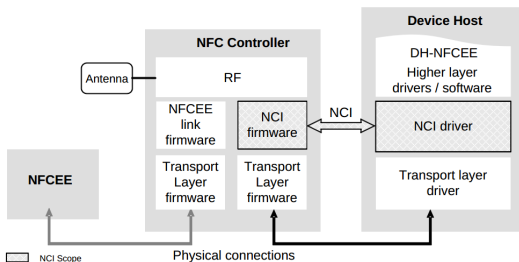
- **Event-driven framework, nice API support**
- **Two native implementations** (depending on built-in NFC chip)
 - `libnfc-nxp`
 - `libnfc-nci`



Android and NFC: A Tale of L♥ve (II)

Digging into Android NFC stack

- **Event-driven framework, nice API support**
- **Two native implementations** (depending on built-in NFC chip)
 - libnfc-nxp
 - libnfc-nci
- **NXP dropped in favour of NCI:**
 - **Open architecture**, not focused on a single family chip
 - **Open interface** between the NFC Controller and the DH
 - **Standard** proposed by NFC Forum



Reverse Engineering of Integrated Circuits/Smart Cards

Problem Analysis

Specific goals

- Figure out the pair of keys (A, B)
- Make a dump of a real card
- Study the card content
- Check any integrity about unauthorised content alteration
- Make a clone card
- Do a mobile app for card-hacking



Reverse Engineering of Integrated Circuits/Smart Cards

Using mfoc

- Two different Classic version
 - MIFARE Classic 1K (T1)
 - MIFARE Classic 4K (T2)

```
death@mulita:~/Downloads/mfoc-0.10.2/src$ time sudo mfoc -0 out -P 100
ATQA (SENS RES): 00 04
* UID size: single
* bit frame anticollision supported
  UID (NFCID1): ██████████
  SAK (SEL RES): 06
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092
Fingerprinting based on ATQA & SAK values:
* Mifare Classic 1K
* Mifare Plus (4-byte UID) 2K SL1
* SmartMX with Mifare 1K emulation
[Key: 000000000000] -> [.....]
[Key: f f] -> [.....]
[Key: f f] -> [.....]
[Key: a1 9] -> [xxxxxxxx.....]
[Key: a1 6] -> [xxxxxxxx.....]
[Key: a1 0] -> [xxxxxxxx.....]
[Key: a1 8] -> [xxxxxxxx.....]
[Key: a1 4] -> [xxxxxxxx.....]
[Key: 7 b] -> [xxxxxxxx.....xx]
[Key: 2 3] -> [xxxxxxxx.....xx]
[Key: c 2] -> [xxxxxxxx.....xx]
[Key: 3 0] -> [xxxxxxxx.....xx]
[Key: 5 b] -> [xxxxxxxxxxxxxxxx]
[Key: 3 6] -> [xxxxxxxxxxxxxxxx]
[Key: 6 1] -> [xxxxxxxxxxxxxxxx]
[Key: 0 3] -> [xxxxxxxxxxxxxxxx]
[Key: 0 3] -> [xxxxxxxxxxxxxxxx]
[Key: 0 3] -> [xxxxxxxxxxxxxxxx]
[Key: 01 9] -> [xxxxxxxxxxxxxxxx]
[Key: a1 5] -> [xxxxxxxxxxxxxxxx]
[Key: b1 5] -> [xxxxxxxxxxxxxxxx]
[Key: 7 9] -> [xxxxxxxxxxxxxxxx]

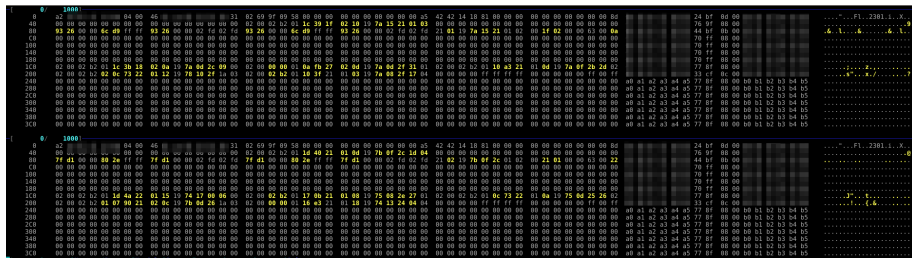
Sector 00 - FOUND_KEY [A] Sector 00 - FOUND_KEY [B]
Sector 01 - FOUND_KEY [A] Sector 01 - FOUND_KEY [B]
Sector 02 - FOUND_KEY [A] Sector 02 - FOUND_KEY [B]
Sector 03 - FOUND_KEY [A] Sector 03 - FOUND_KEY [B]
```

```
Block 15, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3b a2
Block 14, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 13, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 12, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 11, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 10, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 09, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 08, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 07, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 06, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 05, type A, key a 9:03 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 04, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 03, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 02, type A, key a 9:54 5a 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2c
Block 01, type A, key a 9:03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 00, type A, key a 9:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

real 0m35.551s
user 0m0.244s
sys 0m0.004s
```

Reverse Engineering of Integrated Circuits/Smart Cards

Understanding the card content. . .

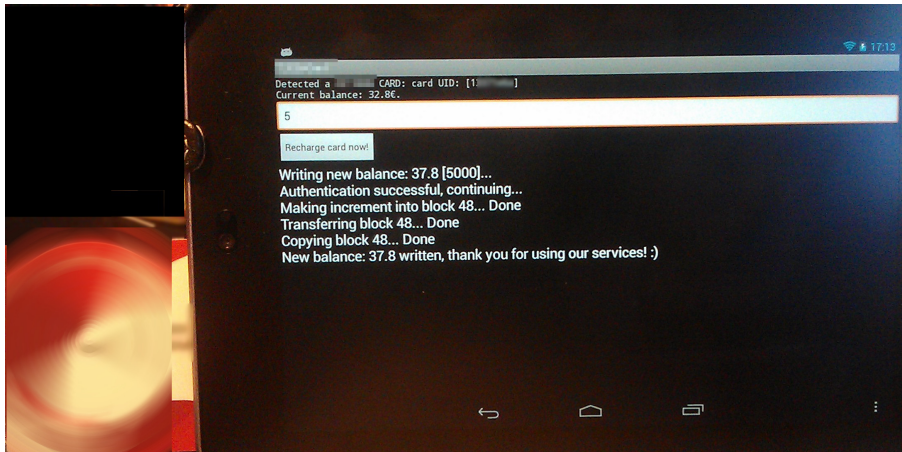


Summary of data

	T1	T2
Card ID	(0, 3)	(10, 3)
Last bus used	(1, 2)	(1, 2)
Current balance	(2, [1, 2])	(12, [1, 2])
Historic	(7, [1, 2, 3]), (8, [1, 2])	(7, [1, 2, 3]), (8, [1, 2])

Reverse Engineering of Integrated Circuits/Smart Cards

Building a PoC in Android O.S. (1)



Reverse Engineering of Integrated Circuits/Smart Cards

Building a PoC in Android O.S. (2) – DEMO



Universidad
Zaragoza

Reverse Engineering of Integrated Circuits/Smart Cards

Recalling the initial goals

Goal	Achieved?	Some remarks
Figure out the pair of keys (A, B)	✓	Some keys are the default ones
Make a dump of a real card	✓	Fast, and simple
Study the card content	✓	Not a single bit encrypted
Check any integrity about unauthorized content alteration	✓	no integrity
Make a cloned card	✓*	A perfect clone (Chine cards rulez!)
Do a mobile app for card-hacking	✓	Android fu\$ing rocks!

Reverse Engineering of Integrated Circuits/Smart Cards

Thinking (and acting?) badly... (1)

What else could be done...

- Identity spoofing
 - Possible penalties for spoofed people
 - Consume the real balance of someone else
- Use of all public services for free
- Black market?
 - Fake recharge point
 - Whether I sold a card illegitimately charged...
- Just put the app in Google Play, and have fun 😊

Reverse Engineering of Integrated Circuits/Smart Cards

Thinking (and acting?) badly... (2): Relay attacks



Event timeline

Nov 2012 Nice chat with J.M. Esparza 😊

Event timeline

Nov 2012 Nice chat with J.M. Esparza 😊

Nov 2012 (ending) Lab environment set and tested (it works!)

Event timeline

Nov 2012 Nice chat with J.M. Esparza ☺

Nov 2012 (ending) Lab environment set and tested (it works!)

Dec 2012 Nice chat with C. Lorenzana ☺ (at STIC CCN-CERT conference)



Event timeline

Nov 2012 Nice chat with J.M. Esparza ☺

Nov 2012 (ending) Lab environment set and tested (it works!)

Dec 2012 Nice chat with C. Lorenzana ☺ (at STIC CCN-CERT conference)

Mar 2013 Confidential report is sent to GDT

Event timeline

Nov 2012 Nice chat with J.M. Esparza ☺

Nov 2012 (ending) Lab environment set and tested (it works!)

Dec 2012 Nice chat with C. Lorenzana ☺ (at STIC CCN-CERT conference)

Mar 2013 Confidential report is sent to GDT

Apr 2013 Report is being handled by CNPIC

Event timeline

Nov 2012 Nice chat with J.M. Esparza ☺

Nov 2012 (ending) Lab environment set and tested (it works!)

Dec 2012 Nice chat with C. Lorenzana ☺ (at STIC CCN-CERT conference)

Mar 2013 Confidential report is sent to GDT

Apr 2013 Report is being handled by CNPIC

May 2013 Company says the problem is known, but does not really care about it. . .

Event timeline

Nov 2012 Nice chat with J.M. Esparza ☺

Nov 2012 (ending) Lab environment set and tested (it works!)

Dec 2012 Nice chat with C. Lorenzana ☺ (at STIC CCN-CERT conference)

Mar 2013 Confidential report is sent to GDT

Apr 2013 Report is being handled by CNPIC

May 2013 Company says the problem is known, but does not really care about it. . .

(today) As they don't care, me neither

Lessons Learned

- It's good to collaborate with police. . . **but you need to be patient**
 - You'll have a good sleep at night and not in jail. . .



Lessons Learned

- It's good to collaborate with police. . . **but you need to be patient**
 - You'll have a good sleep at night and not in jail. . .
 - You also get some **free beer** from Guardia Civil 😊

Lessons Learned

- It's good to collaborate with police. . . **but you need to be patient**
 - You'll have a good sleep at night and not in jail. . .
 - You also get some **free beer** from Guardia Civil 😊
- **Security is not considered (as normally) in a Spanish company**
 - Not at the beginning of a product design
 - Not even when someone spots out the problem
 - They quantify the risk of people exploiting the problem. . .
- This is not U.S., unfortunately (in this case)



Lessons Learned

- It's good to collaborate with police. . . **but you need to be patient**
 - You'll have a good sleep at night and not in jail. . .
 - You also get some **free beer** from Guardia Civil 😊
- **Security is not considered (as normally) in a Spanish company**
 - Not at the beginning of a product design
 - Not even when someone spots out the problem
 - They quantify the risk of people exploiting the problem. . .
- This is not U.S., unfortunately (in this case)

Remember, not economic gain but free beer instead!

Some conclusions...

- MIFARE Classic is like a memory card
- Vulnerable from 2009
- Weaknesses and attacks very well-known and widely documented

Some conclusions...

- MIFARE Classic is like a memory card
- Vulnerable from 2009
- Weaknesses and attacks very well-known and widely documented
- Need to defend against
 - Unauthorized content alteration
 - Relay attacks
 - Clone attacks

Some conclusions...

- MIFARE Classic is like a memory card
- Vulnerable from 2009
- Weaknesses and attacks very well-known and widely documented
- Need to defend against
 - Unauthorized content alteration
 - Relay attacks
 - Clone attacks

Thinking to deploy MIFARE Classic as an access control system?

Some conclusions...

- MIFARE Classic is like a memory card
- Vulnerable from 2009
- Weaknesses and attacks very well-known and widely documented
- Need to defend against
 - Unauthorized content alteration
 - Relay attacks
 - Clone attacks

Thinking to deploy MIFARE Classic as an access control system?
Don't.

Agenda

- 1 Introduction to Reverse Engineering
- 2 Reverse Engineering of Protocols
- 3 Reverse Engineering of Software
- 4 Reverse Engineering of Integrated Circuits/Smart Cards
- 5 Conclusions**



Take-home messages

- **Reverse engineering is an ART** that involves a lot of domains
 - Network protocols
 - Software
 - Integrated circuits/smart cards
 - File formats (forensics)
- **Black-box analysis**: once we found something, keep digging!

Take-home messages

- **Reverse engineering is an ART** that involves a lot of domains
 - Network protocols
 - Software
 - Integrated circuits/smart cards
 - File formats (forensics)
- **Black-box analysis**: once we found something, keep digging!

Never ending learning game!



A Tour through the Realms of Reverse Engineering

Ricardo J. Rodríguez

@RicardoJRdez * rjrodriguez@unizar.es * www.ricardojrodriguez.es

© All wrongs reversed



Universidad
Zaragoza

November 23, 2016

Legacy Systems
Room A.12