A Monster Emerges from the Chrysalis

(Experiences reverse-engineering the Luna CA3)

Mike Bond

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Laboratory

10th February 2004

Contents

- Security API attacks
- Introducing the Luna CA3
- Reverse engineering with IDA
- The cloning protocol
  - Stage 1: Finding it
  - Stage 2: Understanding it
  - Stage 3: Breaking it
- Implementing host side interface
- Lessons learned

Acknowledgements

This was a team effort!

Many many thanks to:

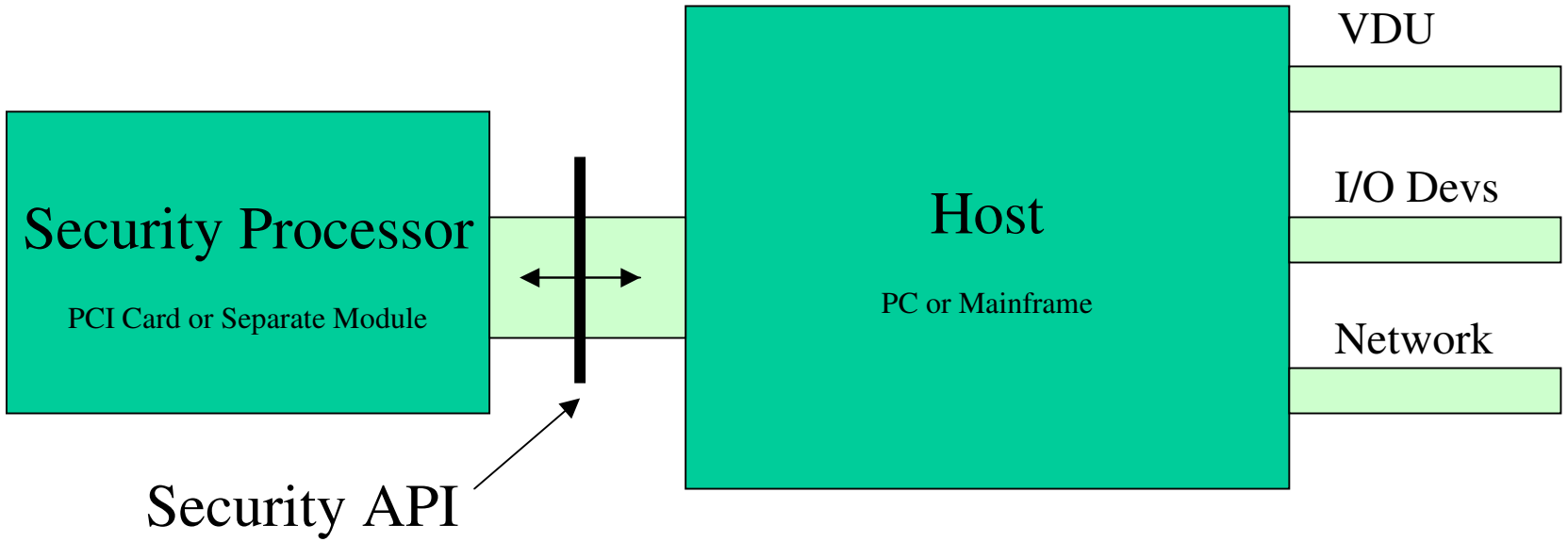
- Steven Murdoch
- Dan Cvrcek

Also thanks to:

- Richard Clayton, IH, Stephen Lewis, Jolyon Clulow
- and many more…

What is a Security API ?

- A command set that uses cryptography to control processing of and access to sensitive data, according to a certain policy



Security API Attacks

- APIs for HSMs have evolved to support more and more transactions and sophisticated features – but they are getting too complex now
- Use the permitted commands of the interface in an unusual sequence to trick a device into revealing secret key material

- Are simpler, quicker and more effective than going in by the ‘front door’?
- *Or are they?*

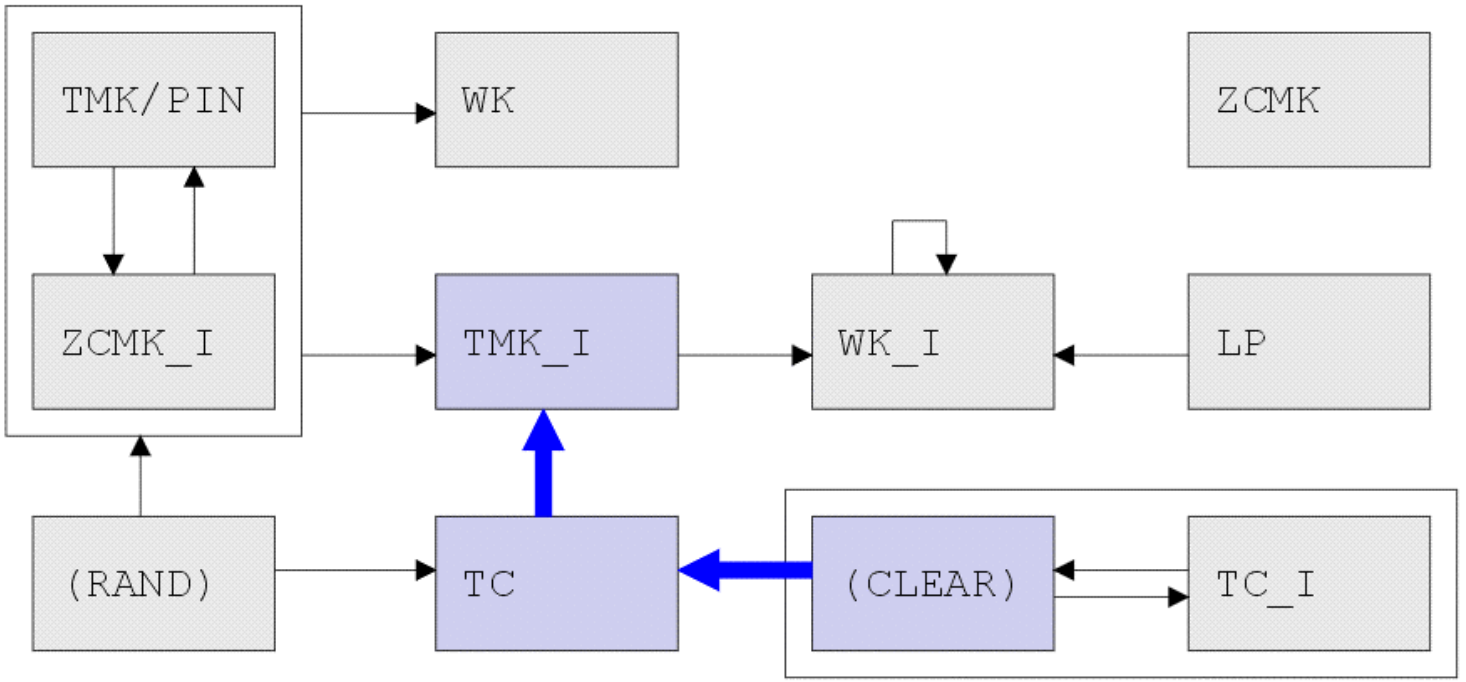
Simple

$U \rightarrow C : PAN$

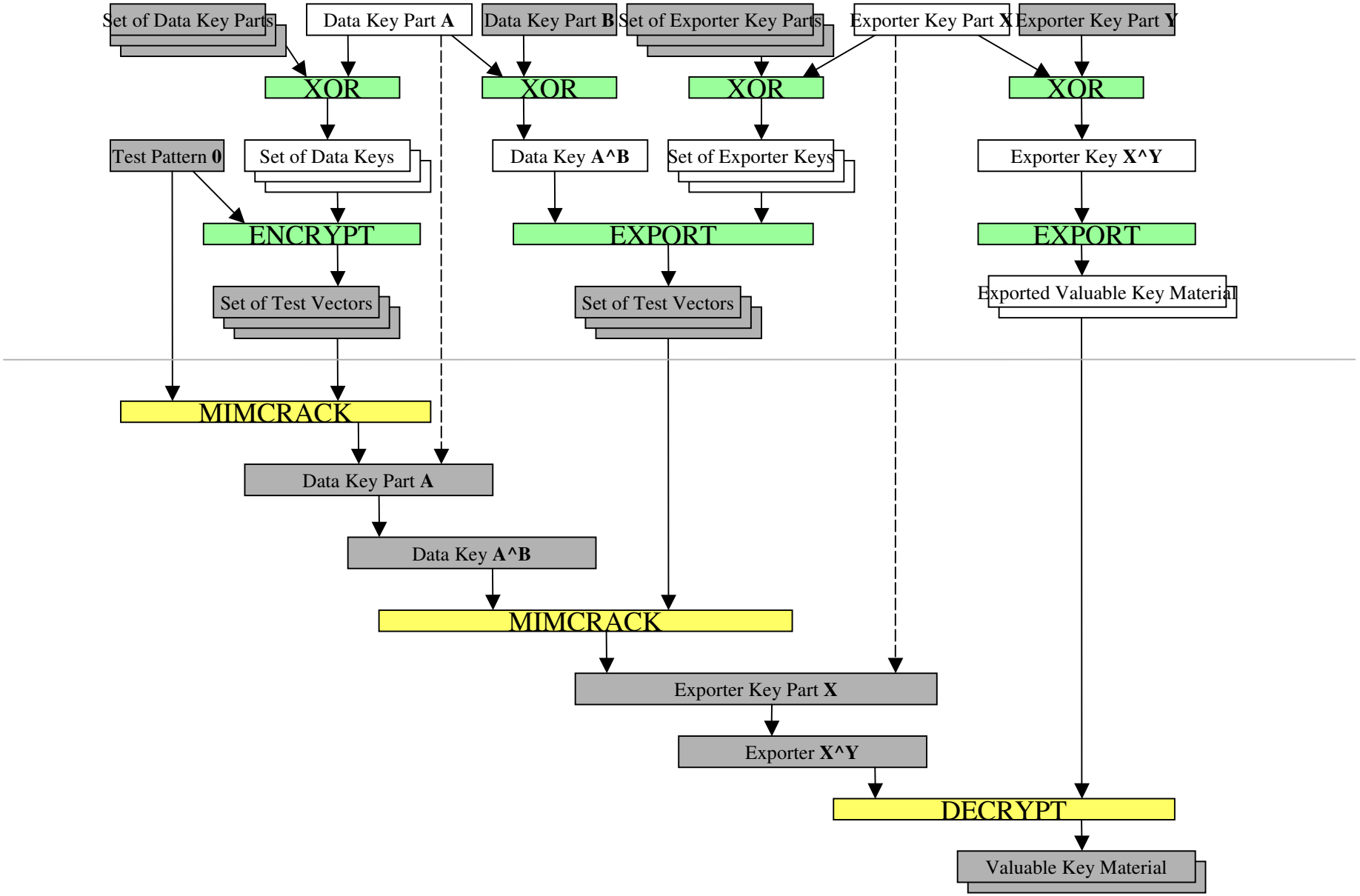
$C \rightarrow U : \{ PAN \}_{TC}$

$U \rightarrow C : \{ PAN \}_{TC} , \{ PMK1 \}_{TMK}$

$C \rightarrow U : \{ PAN \}_{PMK1}$



Not So Simple ?



The Luna CA3

- PCMCIA token, for secure storage of private keys for Certification Authorities
- Manufactured by Chrysalis-ITS (Toronto), acquired by Rainbow, aquired by SafeNet
- Became popular during the rise of PKIs in the dot com boom (Verisign exclusively uses Chrysalis kit for key storage)
- Uses the PKCS#11 API (through an internal proprietary ‘Luna API’)

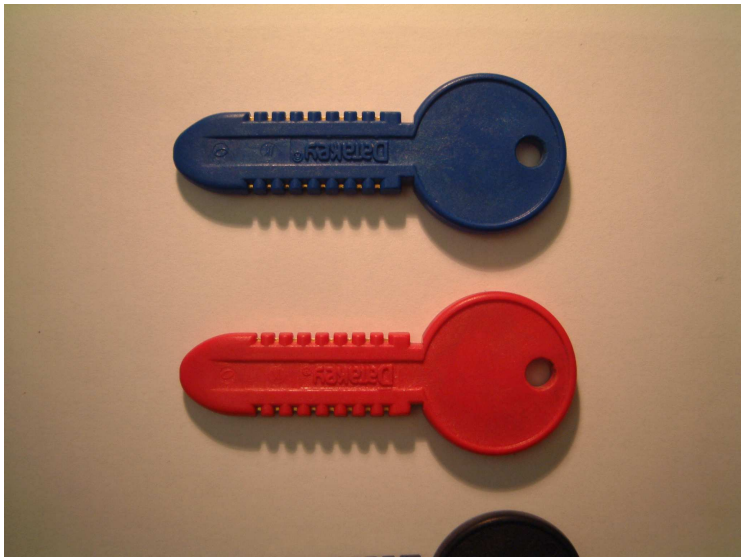
Luna CA3 – Front View

Luna Dock

The Cloning Protocol

- Used for backup and availability
- Initialise a new token into the same domain (you need the RED key)
- Log on to source and destination tokens (with BLUE security officer key)
- Select an object and call `CA_ClonePrivateKey` to transfer between source and destination. The devices exchange public keys then set up a session key for the transfer.

Luna CA3 – Pin Entry Device (PED)



Luna CA3 – Datakeys

Primary Goal

Develop a way to extract all PKCS#11 keys in the clear from the Luna token, with the co-operation of the security officer

Motivations

- Break customer lock-in – help the market
- Learn about internal HSM architecture
- Find implementation faults (buffer overflows?)
- Find new Security API attacks?
- Learn useful skills along the way
  - Reverse-engineering
  - Assembler
  - Particular disassembly tools

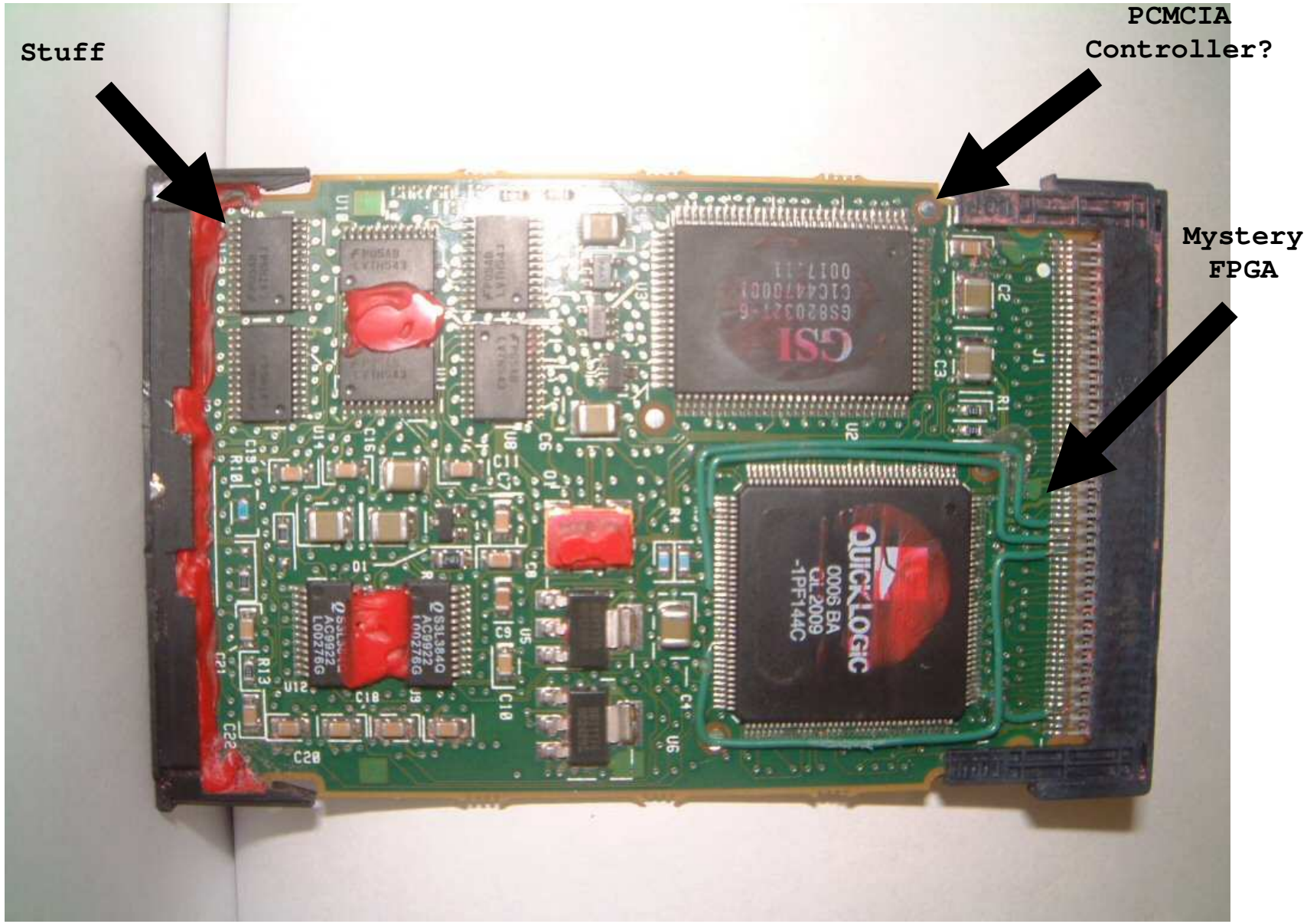
A Simple Plan

- Open up the card
- Reverse-engineer the flash chip
- Discover the cloning protocol
- Extract device keys
- Use keys to impersonate token in cloning protocol

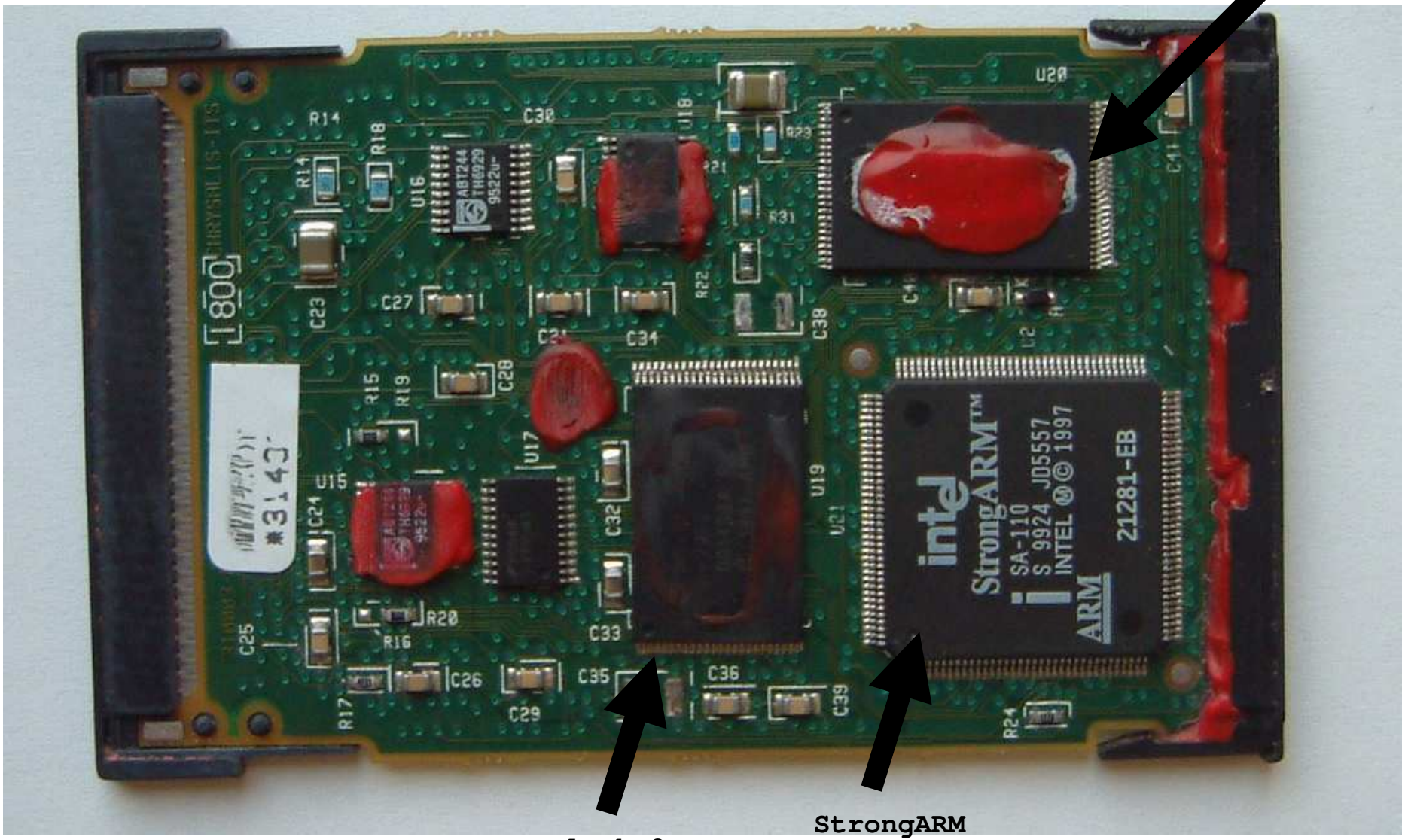
Stage 1 : Finding the Protocol

- Get the ARM code
- Get a reverse engineering tool
- Familiarise and Mark-up ARM code
- Identify Command Despatcher
- Annotate Commands
- Intercept and Decode PCMCIA Bus
- Locate and Decode Cloning Protocol

Luna CA3 – Depackaged



Luna CA3 – Depackaged



The Luna Flash File – AM29.BIN

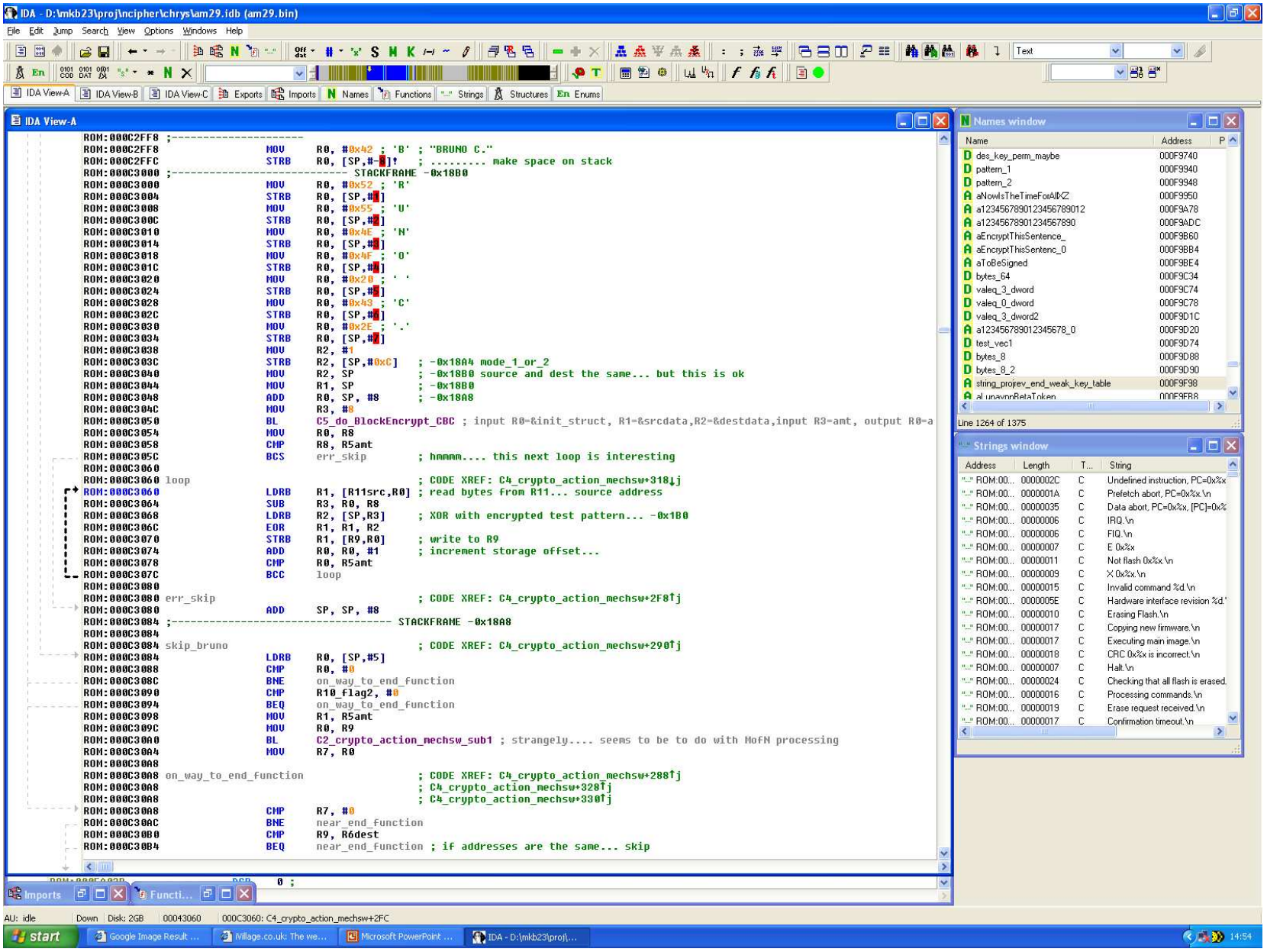
- Two 1/2MB flash chips, holding half words
  - ~300KB code
  - ~500KB data
  - ~200KB blank

- Complexity
  - 1035 subroutines
  - ~1700 pages of assembler (on this screen)

IDA – The Interactive Disassembler

- Made by ‘Datarescue’ – one man consultant went commercial with the tool he developed for himself. Cost ~$700 for 2 year licence.
- Beautiful windows GUI and navigation system. Rename functions and variable names on-the-fly and the new information propagates through the disassembly listing

IDA


IDA - D:\mkb23\proj\ncipher\chrys\am29.idb (am29.bin)
File Edit Jump Search View Options Windows Help
IDA View-A IDA View-B IDA View-C Exports Imports Names Functions Strings Structures Enums
IDA View-A
ROM:000C2FF8 ;----------
ROM:000C2FF8 MOV R0, #0x42 ; 'B' ; "BRUNO C."
ROM:000C2FFC STRB R0, [SP,#-4]! ; .... make space on stack
ROM:000C3000 ;---------- STACKFRAME -0x18B0
ROM:000C3000 MOV R0, #0x52 ; 'R'
ROM:000C3004 STRB R0, [SP,#4]
ROM:000C3008 MOV R0, #0x55 ; 'U'
ROM:000C300C STRB R0, [SP,#5]
ROM:000C3010 MOV R0, #0x4E ; 'N'
ROM:000C3014 STRB R0, [SP,#6]
ROM:000C3018 MOV R0, #0x4F ; 'O'
ROM:000C301C STRB R0, [SP,#7]
ROM:000C3020 MOV R0, #0x20 ; ' '
ROM:000C3024 STRB R0, [SP,#8]
ROM:000C3028 MOV R0, #0x43 ; 'C'
ROM:000C302C STRB R0, [SP,#9]
ROM:000C3030 MOV R0, #0x2E ; '.'
ROM:000C3034 STRB R0, [SP,#A]
ROM:000C3038 MOV R2, #1
ROM:000C303C STRB R2, [SP,#0xC] ; -0x18A4 mode_1_or_2
ROM:000C3040 MOV R2, SP ; -0x18B0 source and dest the same... but this is ok
ROM:000C3044 MOV R1, SP ; -0x18B0
ROM:000C3048 ADD R0, SP, #8 ; -0x18A8
ROM:000C304C MOV R3, #8
ROM:000C3050 BL C5_do_BlockEncrypt_CBC ; input R0=&init_struct, R1=&srcdata,R2=&destdata,input R3=amt, output R0=a
ROM:000C3054 MOV R0, R8
ROM:000C3058 CMP R8, R5amt
ROM:000C305C BCS err_skip ; hmmmm.... this next loop is interesting
ROM:000C3060
ROM:000C3060 loop ; CODE XREF: C4_crypto_action_mechsw+318↓j
ROM:000C3060 LDRB R1, [R11src,R0] ; read bytes from R11... source address
ROM:000C3064 SUB R3, R0, R8
ROM:000C3068 LDRB R2, [SP,R3] ; XOR with encrypted test pattern... -0x1B0
ROM:000C306C EOR R1, R1, R2
ROM:000C3070 STRB R1, [R9,R0] ; write to R9
ROM:000C3074 ADD R0, R0, #1 ; increment storage offset...
ROM:000C3078 CMP R0, R5amt
ROM:000C307C BCC loop
ROM:000C3080
ROM:000C3080 err_skip ; CODE XREF: C4_crypto_action_mechsw+2F8↑j
ROM:000C3080 ADD SP, SP, #8
ROM:000C3084 ;---------- STACKFRAME -0x18A8
ROM:000C3084
ROM:000C3084 skip_bruno ; CODE XREF: C4_crypto_action_mechsw+290↑j
ROM:000C3084 LDRB R0, [SP,#5]
ROM:000C3088 CMP R0, #0
ROM:000C308C BNE on_way_to_end_function
ROM:000C3090 CMP R10_flag2, #0
ROM:000C3094 BEQ on_way_to_end_function
ROM:000C3098 MOV R1, R5amt
ROM:000C309C MOV R0, R9
ROM:000C30A0 BL C2_crypto_action_mechsw_sub1 ; strangely.... seems to be to do with MofN processing
ROM:000C30A4 MOV R7, R0
ROM:000C30A8
ROM:000C30A8 on_way_to_end_function ; CODE XREF: C4_crypto_action_mechsw+288↑j
ROM:000C30A8 ; C4_crypto_action_mechsw+328↑j
ROM:000C30A8 ; C4_crypto_action_mechsw+330↑j
ROM:000C30A8 CMP R7, #0
ROM:000C30AC BNE near_end_function
ROM:000C30B0 CMP R9, R6dest
ROM:000C30B4 BEQ near_end_function ; if addresses are the same... skip
Names window
Name Address
des_key_perm_maybe 000F9740
pattern_1 000F9940
pattern_2 000F9948
aNowIsTheTimeForAllXZ 000F9950
a123456789012345678901 2 000F9A78
a12345678901234567890 000F9ADC
aEncryptThisSentence_ 000F9B60
aEncryptThisSentenc_0 000F9BB4
aToBeSigned 000F9BE4
bytes_64 000F9C34
valeq_3_dword 000F9C74
valeq_0_dword 000F9C78
valeq_3_dword2 000F9D1C
a12345678901234567 8_0 000F9D20
test_vec1 000F9D74
bytes_8 000F9D88
bytes_8_2 000F9D90
string_projrev_end_weak_key_table 000F9F98
Line 1264 of 1375
Strings window
Address Length T... String
ROM:00... 0000002C C Undefined instruction, PC=0x%x
ROM:00... 0000001A C Prefetch abort, PC=0x%x.\n
ROM:00... 00000035 C Data abort, PC=0x%x, [PC]=0x%
ROM:00... 00000006 C IRQ.\n
ROM:00... 00000006 C FIQ.\n
ROM:00... 00000007 C E 0x%x
ROM:00... 00000011 C Not flash 0x%x.\n
ROM:00... 00000009 C X 0x%x.\n
ROM:00... 00000015 C Invalid command %d.\n
ROM:00... 0000005E C Hardware interface revision %d.'
ROM:00... 00000010 C Erasing Flash.\n
ROM:00... 00000017 C Copying new firmware.\n
ROM:00... 00000017 C Executing main image.\n
ROM:00... 00000018 C CRC 0x%x is incorrect.\n
ROM:00... 00000007 C Halt.\n
ROM:00... 00000024 C Checking that all flash is erased.
ROM:00... 00000016 C Processing commands.\n
ROM:00... 00000019 C Erase request received.\n
ROM:00... 00000017 C Confirmation timeout.\n
Imports
Functi...
AU: idle Down Disk: 2GB 00043060 000C3060: C4_crypto_action_mechsw+2FC
start
Google Image Result ...
Village.co.uk: The we...
Microsoft PowerPoint ...
IDA - D:\mkb23\proj\...
14:54

Reverse-Engineering Golden Rules

Conventional wisdom is one rule...

- Figure everything out for yourself!

Reverse-Engineering Golden Rules

My wisdom...

- If you don't know what to do, instead, **do what you can**.

- Give **everything** a name.

```
if you get stuck…
http://www.babycakesinternational.com/100topbabnam.html
or use movies, friends, books
```

Markup and Annotation

- Make every letter in a name count!

```
C5D2_30SER_BLEV_JANE_do_something_sub1
```

- Group C1 type functions into larger clumps
- Pay special attention to most called functions

  `memcpy` 327 calls
- Start propagating type information
  - (memcpy arg 2 is length, args 0 and 1 pointers)

Finding the Command Despatcher

- Search for the biggest case switches…
  - 45 switch statements in total
  - ranging between 0x17 and 0x5 ways
  - no idea what the command encoding was

```
ADDLS   PC, PC, R0,LSL#2 ; switch 0xC ways
```

- Two pages from back of policy document listing the Luna API commands categorised by module was all we had.

Finding the Command Despatcher

COPY

APPENDIX C. Session And Login States Required For Luna Token Commands

Command To Token	No Session Open	Session Open, No Login	SO Logged On	User Logged On
Token Main Module Commands 7				
LUNA_ZEROIZE	√			
LUNA_INIT_TOKEN			√	
LUNA_GET	√			
LUNA_GET_USV			√	
LUNA_SET_TPV			√	
LUNA_FW_UPDATE			√	
LUNA_CONFIGURE_SP	√			
Session Manager Commands				
LUNA_OPEN_ACCESS 10	√			
LUNA_CLEAN_ACCESS	√			
LUNA_CLOSE_ACCESS	√			
LUNA_GET_ALL_ACCESSES	√			
LUNA_OPEN_SESSION 24	√			
LUNA_CLOSE_SESSION 26		√		
LUNA_CLOSE_ALL_SESSIONS	√			
LUNA_GET_SESSION_INFO 27		√		
LUNA_EXTRACT_CONTEXTS		√		
LUNA_INSERT_CONTEXTS		√		
User Module Commands 8				
LUNA_GET_USER_LIST		√		
LUNA_GET_USER_NAME		√		
LUNA_LOGIN OD OE		√		
LUNA_LOGOUT				√
LUNA_SET_PIN				√
LUNA_INIT_PIN			√	
LUNA_CREATE_USER			√	
LUNA_DELETE_USER			√	
Object Management Module 8				
LUNA_CREATE_OBJECT		√		
LUNA_COPY_OBJECT		√		
LUNA_DESTROY_OBJECT		√		
LUNA_GET_OBJECT_SIZE		√		
LUNA_GET_ATTRIBUTE_VALUE		√		
LUNA_GET_ATTRIBUTE_LENGTH		√		
LUNA_MODIFY_OBJECT		√		
LUNA_FIND_OBJECTS 16?		√		
Random Number Generator Module 2				
LUNA_GET_RANDOM		√		
LUNA_SEED_RANDOM		√		
Key Management Module 9 BIGSWITCH MEN=4				
LUNA_GENERATE_KEY 17				√
LUNA_GENERATE_KEY_W_VALUE				√
LUNA_GENERATE_KEY_PAIR				√
LUNA_WRAP_KEY				√
LUNA_UNWRAP_KEY				√
LUNA_UNWRAP_KEY_W_VALUE				√
LUNA_DERIVE_KEY				√
LUNA_DERIVE_KEY_W_VALUE				√

(one more over page)

Overview — Luna CA3 Security Policies Document #802509 V2.00

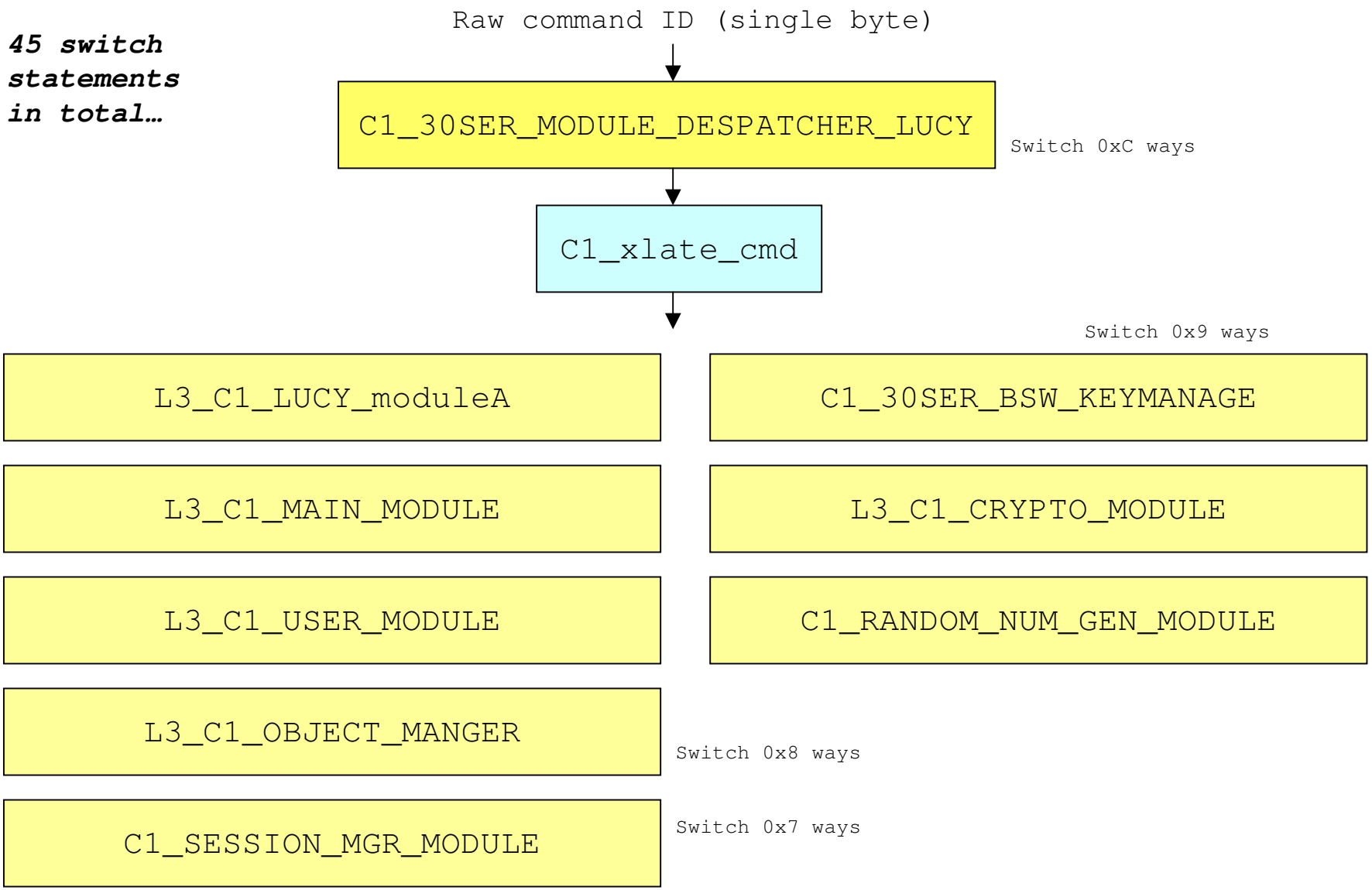
APPENDIX C. Session And Login States Required For Luna Token Commands

(one more over page)

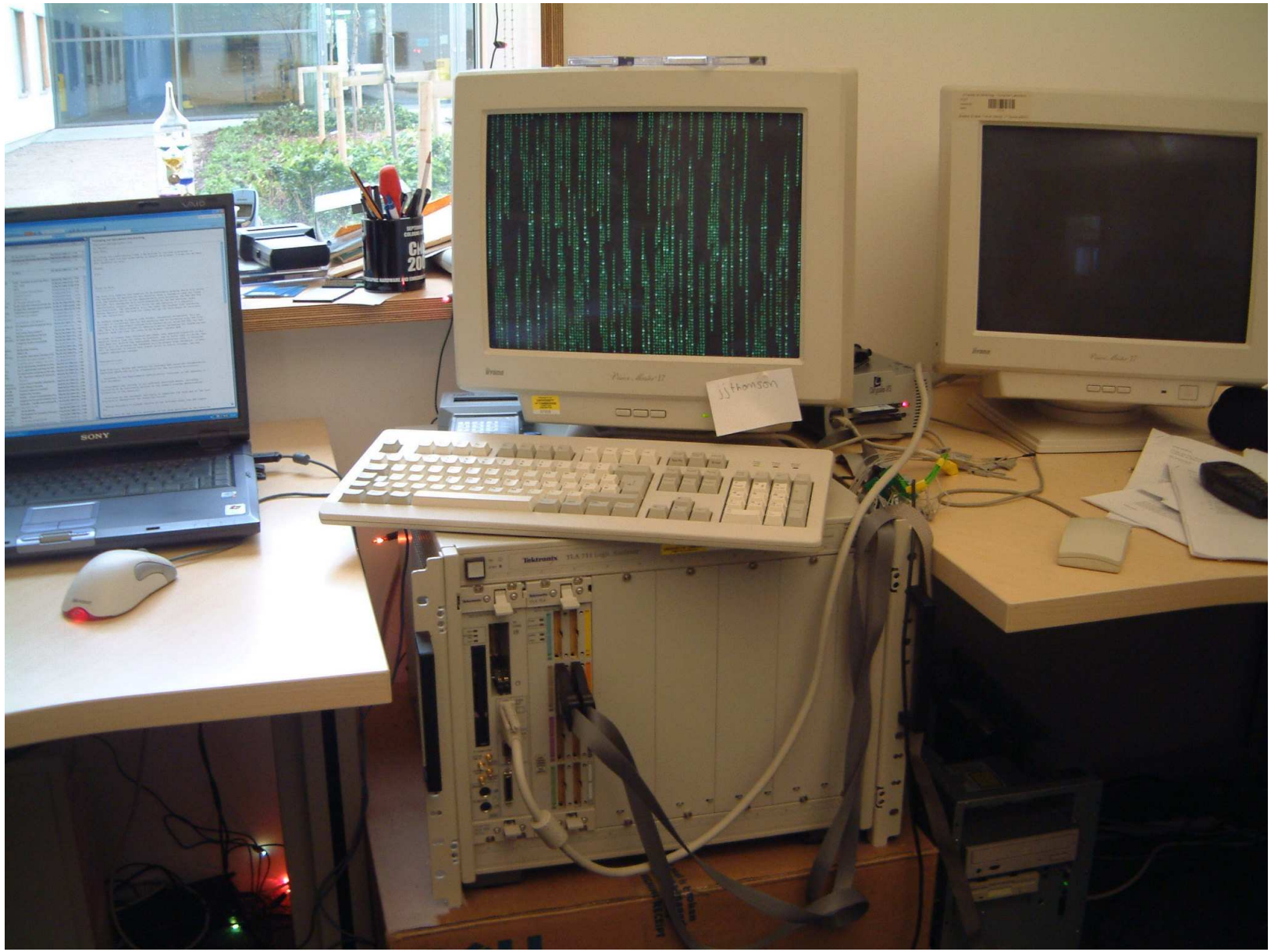
Unrestricted Chrysalis-ITS 14

The Command Despatcher

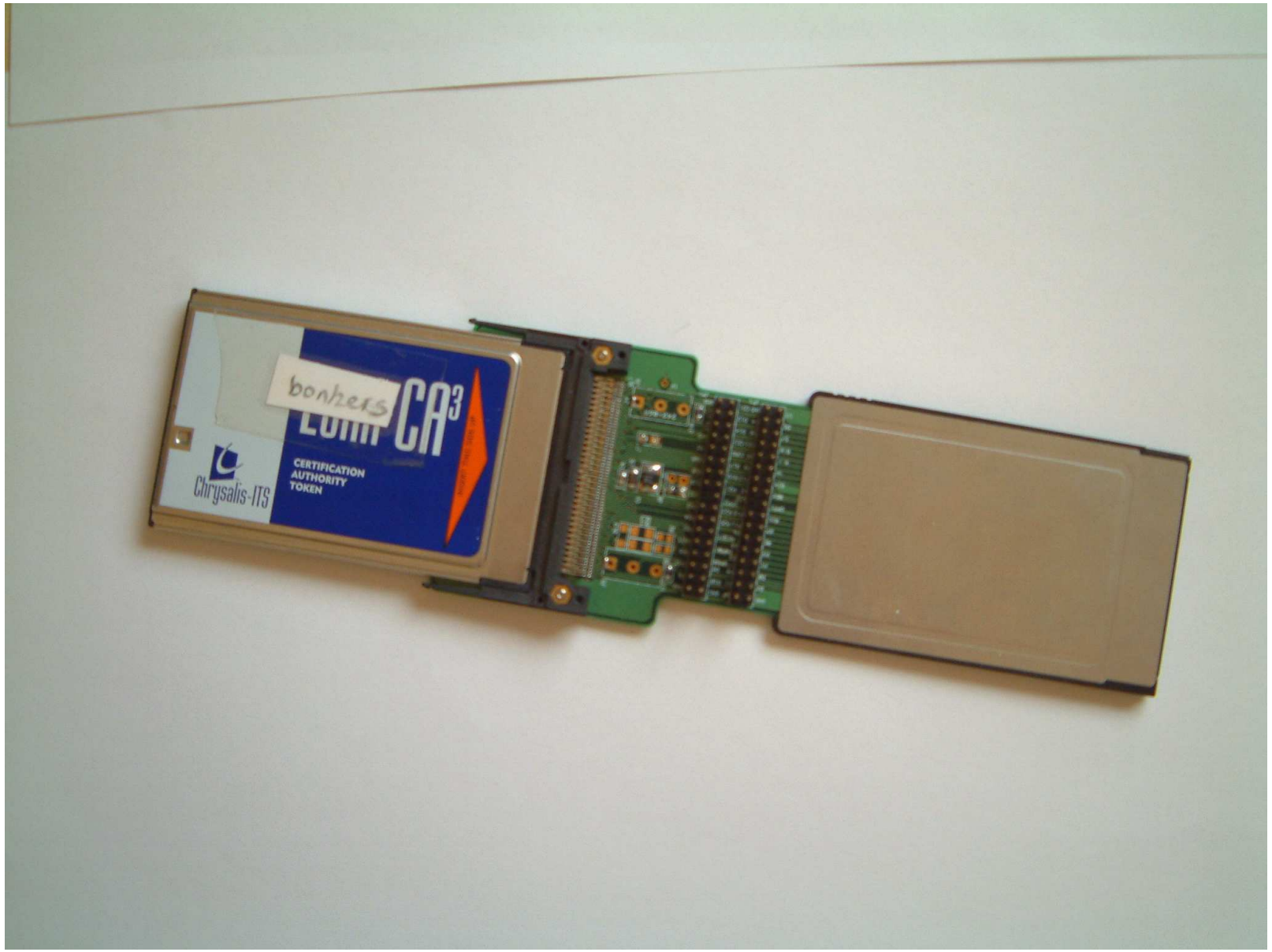
45 switch statements in total…



Intercepting the PCMCIA Bus



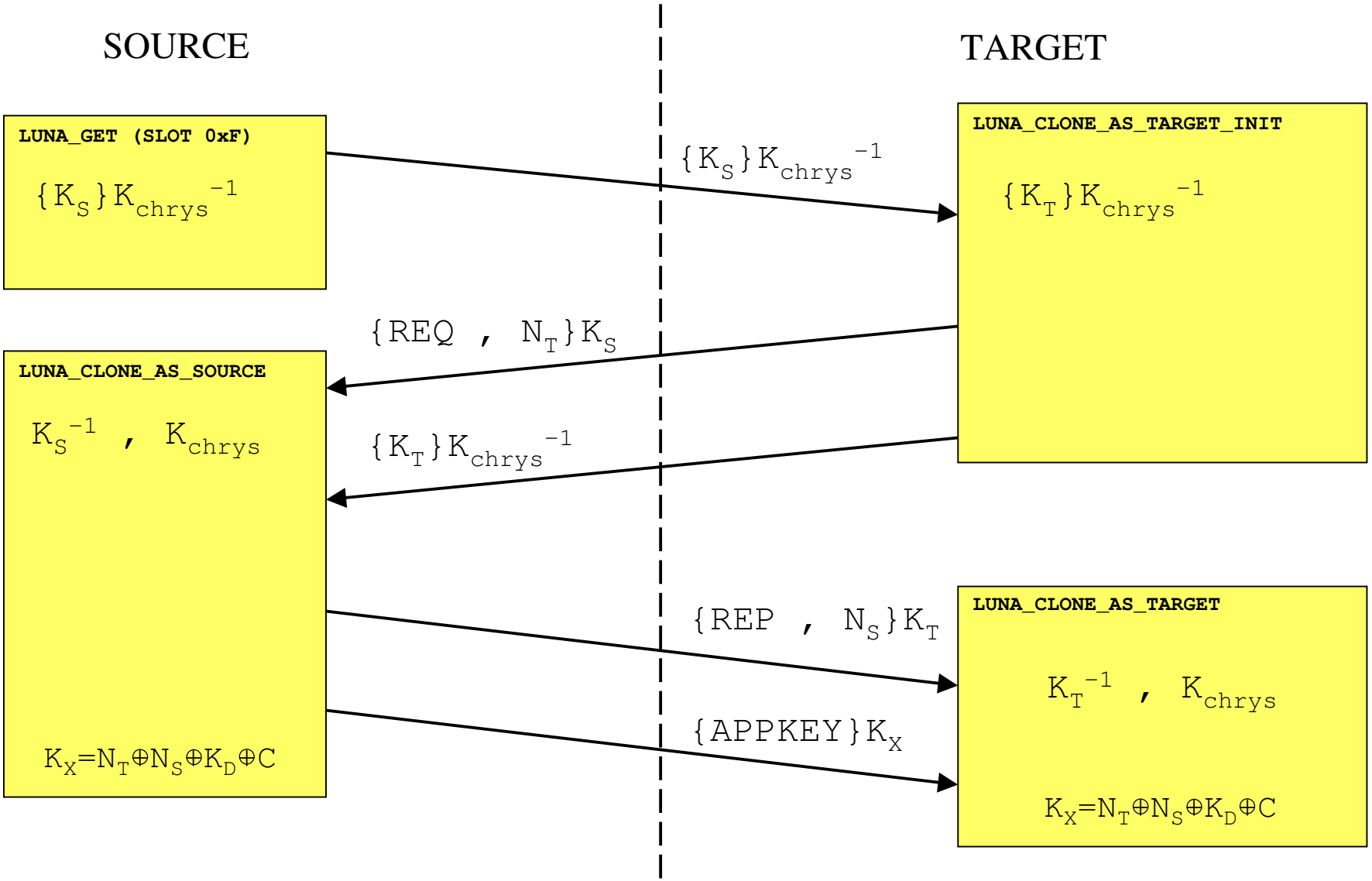
Intercepting the PCMCIA Bus



Bus Intercepts : Cloning Protocol

SOURCE	TARGET
LUNA_FIND_OBJECTS	LUNA_FIND_OBJECTS
LUNA_GET (SLOT 0xE)	LUNA_DESTROY_OBJECT
LUNA_GET (SLOT 0xF)	LUNA_FIND_OBJECTS
LUNA_CLONE_AS_SOURCE	LUNA_GET (SLOT 0xE)
LUNA_GET (SLOT 0xE)	LUNA_GENERATE_KEY
	LUNA_SET_UP_MASKING_KEY
	LUNA_DESTROY_OBJECT
	LUNA_GENERATE_KEY_W_VALUE
	LUNA_CLONE_AS_TARGET_INIT
	LUNA_CLONE_AS_TARGET
	LUNA_GET (SLOT 0xE)

Luna Key Cloning Protocol



Stage 2 : Understanding the Protocol

- We knew ***what*** the cloning routine did, but not ***where*** the key material came from
- The encrypted key material came from `LEELA`, the decryption key from `JADE`
- We could see encryption and decryption, but not exactly ***how***… had to mark-up the crypto routines called by the cloning code
  - Identify which algorithms are used
  - Identify algorithm parameters, key lengths
  - What about IVs?

The Luna Mysteries

- To understand the protocols we needed to discover the purpose of some puzzling functions
  - `C4_crypto_action_mechsw`
  - `LEELA`
  - `JADE`
  - `'EDAFLU'`

C4_crypto_action_mechsw

- Seemed to be the central function for symmetric crypto – called by…
  `C25_C_ACTION_0_ENCRYPT`
  `C27_C_ACTION_0_DECRYPT`
- Called `C5_do_BlockEncrypt_CBC` , and called lots of crypto-like routines, but the two seemed unlinked.
- Evidence of software DES was found (key-schedule), but the block encrypt function called HIFN (a DES accelerator manufacturer) IO functions. Yet there was no HIFN chip in the token. How and where was the DES done?

C4_crypto_action_mechsw (2)

- Solution: a well hidden table jump inside the CBC loop, once discovered made the code make sense
- There were 3 function tables – one for preparing key schedule, one for encrypt and one for decrypt
- DES key schedule was calculated in software, then uploaded into accelerator chip (this upload was mistaken for the full DES calculation)
- Why was DES done as a composite in H/W and S/W? To claim ‘hardware accelerated DES in marketing brochure’? Space was too limited in FPGA?

Hunting LEELA

- Official name: `C68_LEELA_load` and `C35_LEELA_save`
- The token private key came from LEELA slot 0xF, but where did the slot live? The code used `memcpy` to pluck it from unusual address, but we only had rough idea of the memory map. Could they be special secure memory inside FPGA?
- Eventually: discovered that LEELA slot save code looked like flash file update code: became convinced that slots lived on 1MB flash image.
- Wrote script to scan flash for linked list of pointers as theorised from reader code. Success! Found LEELA slots at 0x88000 in AM29.BIN

Finding JADE

- JADE, officially:

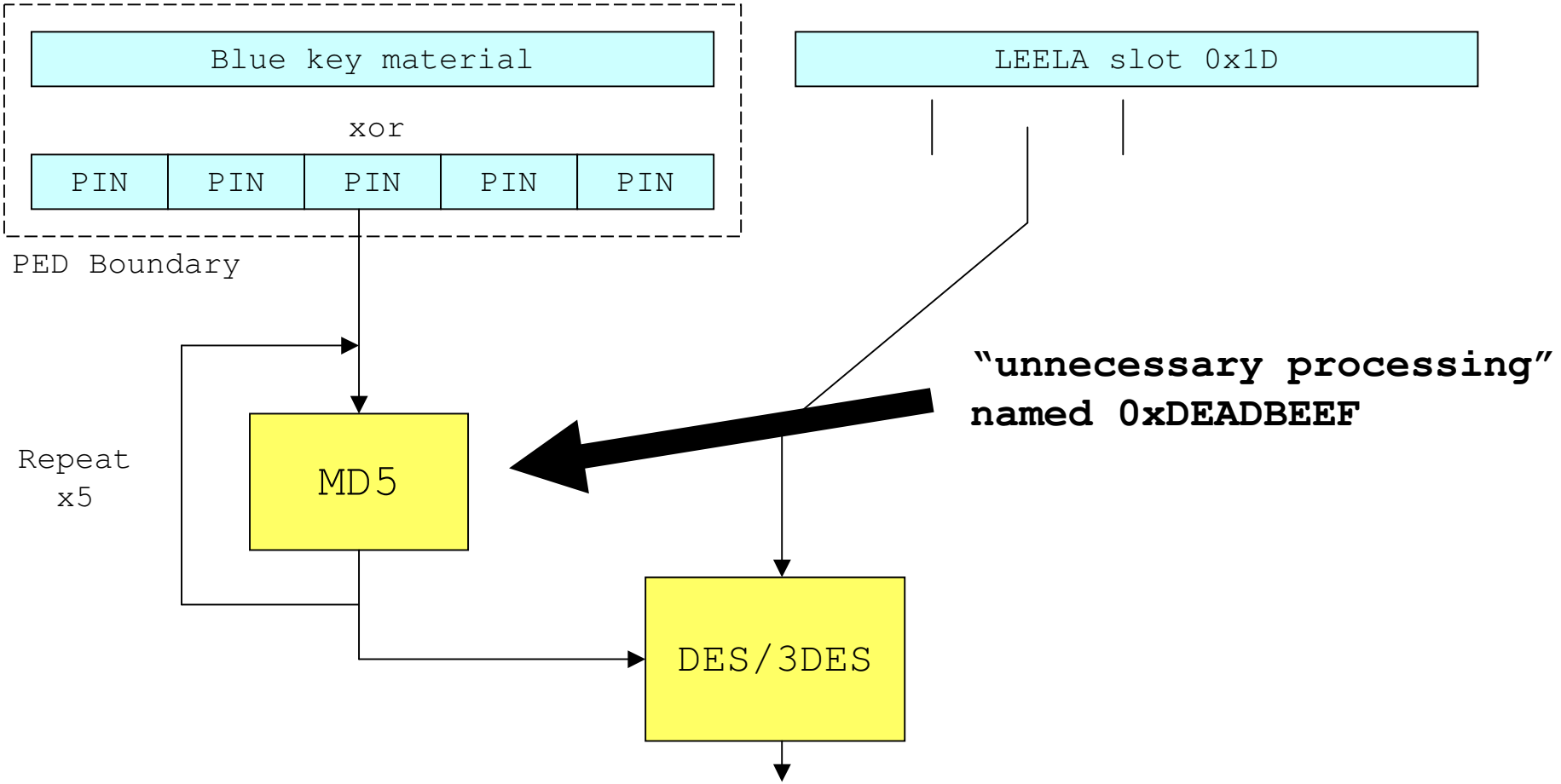
```
C12_JADE_prep_crypto1struct_entryA
C4_JADE_entryB
```

- JADE **takes no arguments**, and returns a crypto1struct , containing a DES key or a 3DES key used for decrypting the contents of a LEELA slot.
- Problem: JADE walks through data structure in RAM to find keys – how can we locate code that set up keys in data structure?

Finding JADE (2)

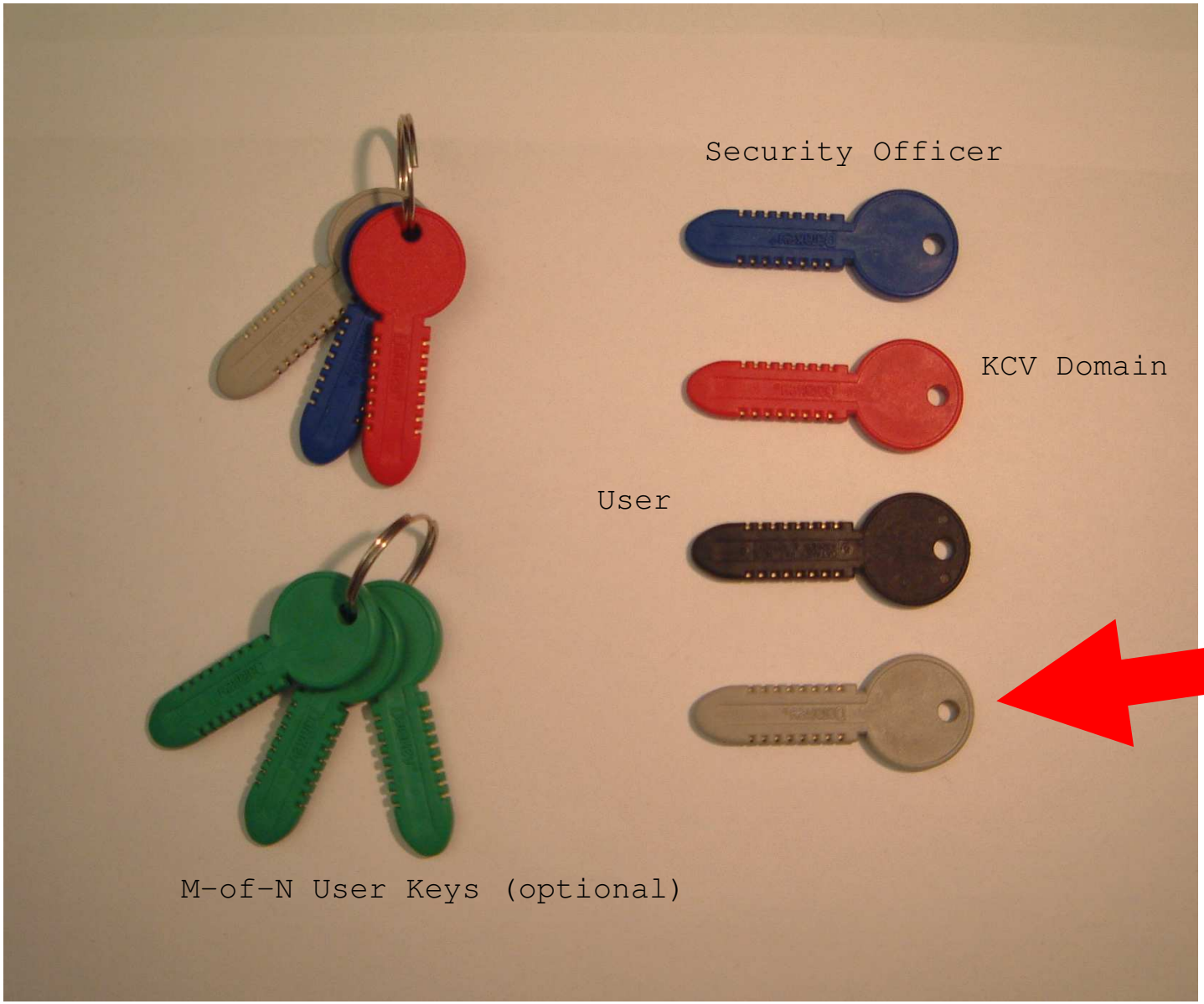
- Solutions:
  - Take a guess. Look in login routines – maybe JADE keys come from physical datakeys
  - Observe class of error code in JADE functions, and search for functions exhibiting similar error codes
- Success: `C3_LOGINOUT_setup_auth_contexts_JADE` was found. In fact, key material in JADE slots came from a decrypted version of the data structure inside a LEELA slot.
- But where did the encryption key come from? The datakey? And if so, which?

Finding JADE (3)



- Problem: So how can the keys be stored in encrypted form when the token is uninitialised? – there is no blue key

Datakeys Revisited



The Luna PED Protocol

- PED talks to token be reusing high address lines from PCMCIA spec as bidirection communications channels
- Three lines: RESET, DATA, and DATA_VALID
- However, DATA_VALID was clocked in an unpredictable erratic way. Reason: Luna token implements serial communications protocol in software, and cycle time of loop was data dependent.
- Used a datakey reader to make an independent observation of data on keys, and try to observe this on the bus.

The ‘EDAFLU’ Story

- During initialisation of a token, there is a special requirement: insert the mystery ‘grey key’
- Grey key not mentioned at all in documentation, or release notes
- Contained 64 bytes, mainly zeroes, save for one interesting constant… more 0xDEADBEEF?

```
00 00 01 00 00 30 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 65 64 61 66 6C 75 00 74 00 00 00
```

The ‘EDAFLU’ Story (2)

```
00 00 01 00 00 30 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 65 64 61 66 6C 75 00 74 00 00 00
```

e d æa f åa u l t \0

Datakey reader had wrong half-word endian!

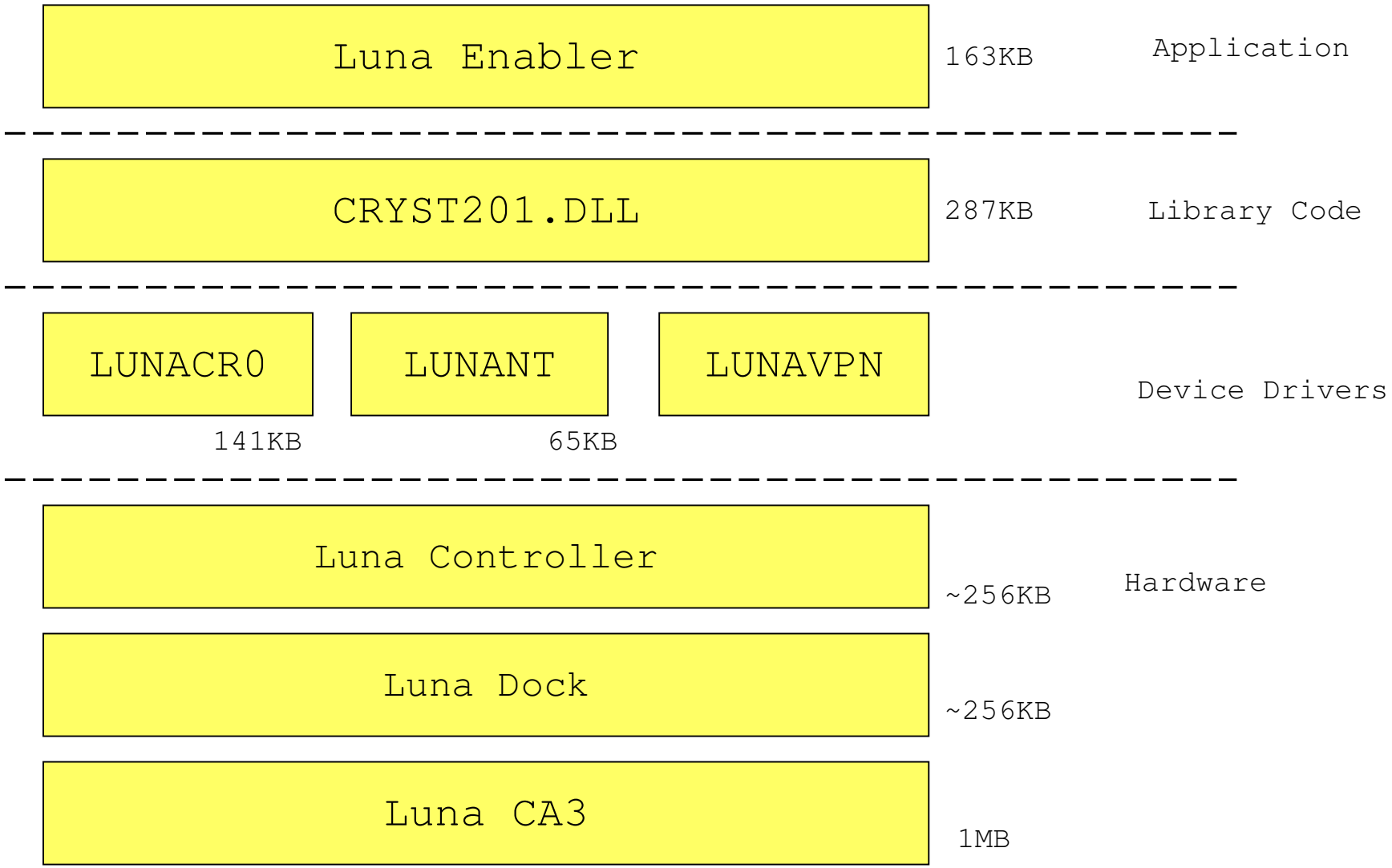
Extracting the Token Private Key

- LEELA slot contained encrypted private key of token, in two forms, encrypted under grey key and under current blue key.
- Key material from data key retrieved
- JADE decrypts slot and puts clear keys in RAM
- We re-implemented decryption of LEELA slot using hash of ‘default’ key.
- Unfortunately…FAILURE
- Need to emulate ARM code and try again, or switch to another plan

Stage 3 : Breaking the Protocol

- Find the protocol in the code stack
- Familiarisation and mark-up of PKCS#11 DLL code in CRYST201.DLL
- Follow data flow inside DLL
- Intercept and change data flow
- A change of plan: CVKs

The Luna Code Stack



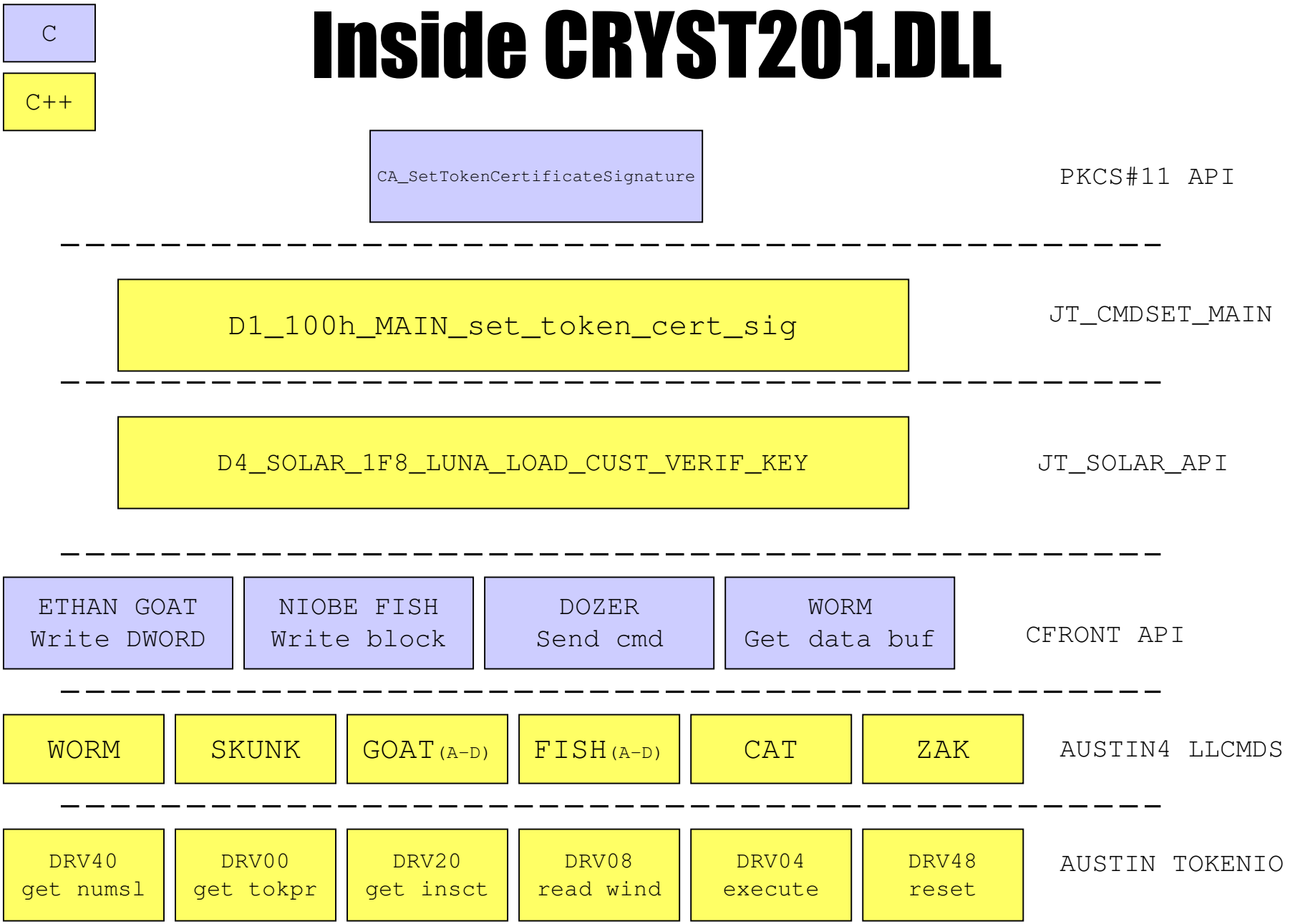
Inside CRYST201.DLL

- Usual PKCS#11 entry points exported, but some extra vendor-specific ones of interest

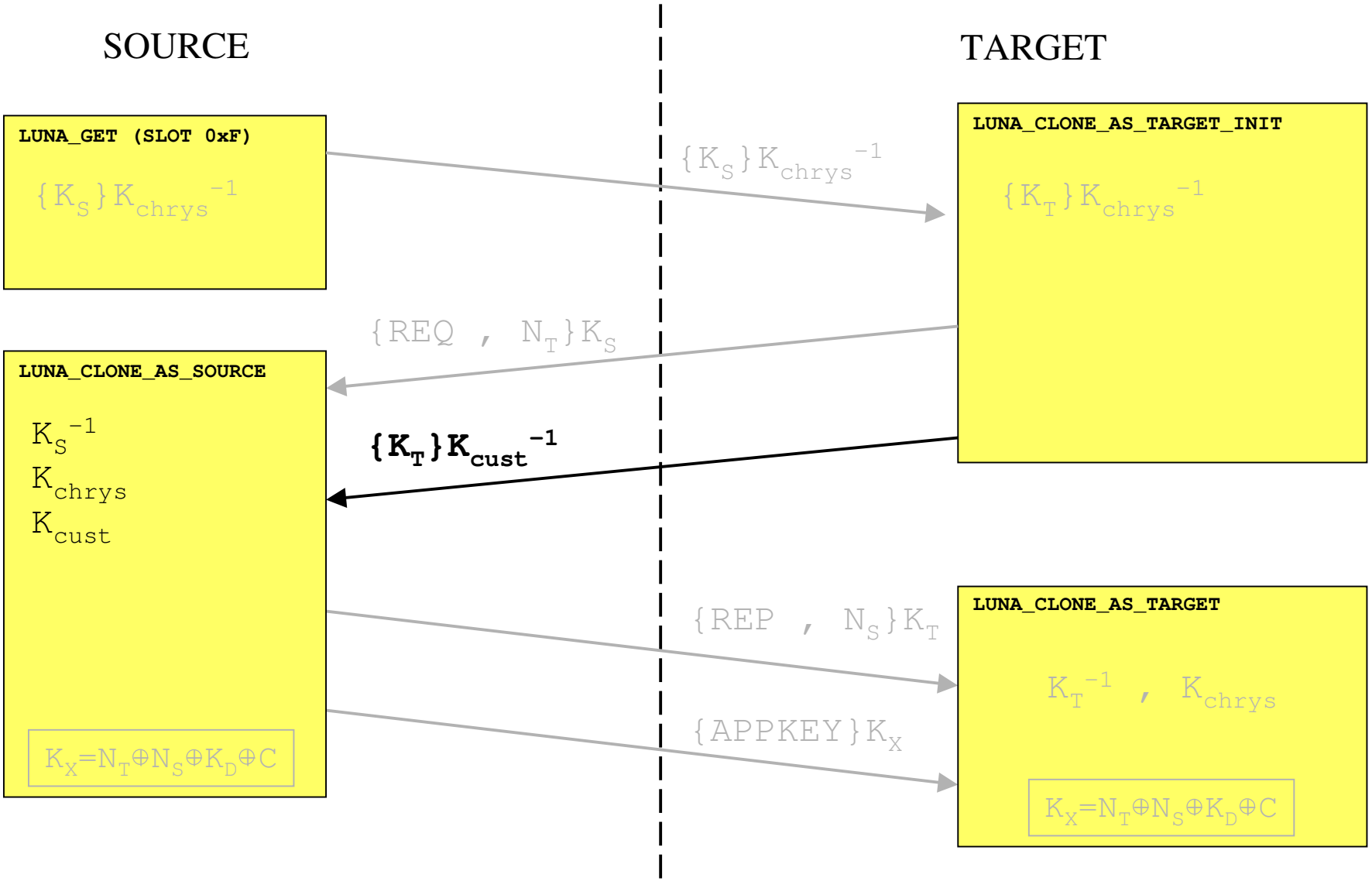
```
CA_SetCloningDomain
CA_SetTokenCertificateSignature
CA_ClonePrivateKey                    (and many more…)
```

- DLL written in mix of C++ and C. PKCS#11 entry points called C++ methods of object hierarhcy representing different models of Luna token (Luna 1, Luna 2, Luna CA3, Luna RA etc.)
- These methods called ‘SOLAR API’, which corresponded closely (but not exactly) to Luna API intercepted on PCMCIA bus. SOLAR API called C stub functions, which called I/O methods of C++ class hierarchy representing different device drivers.
- To summarise: a real mess inside

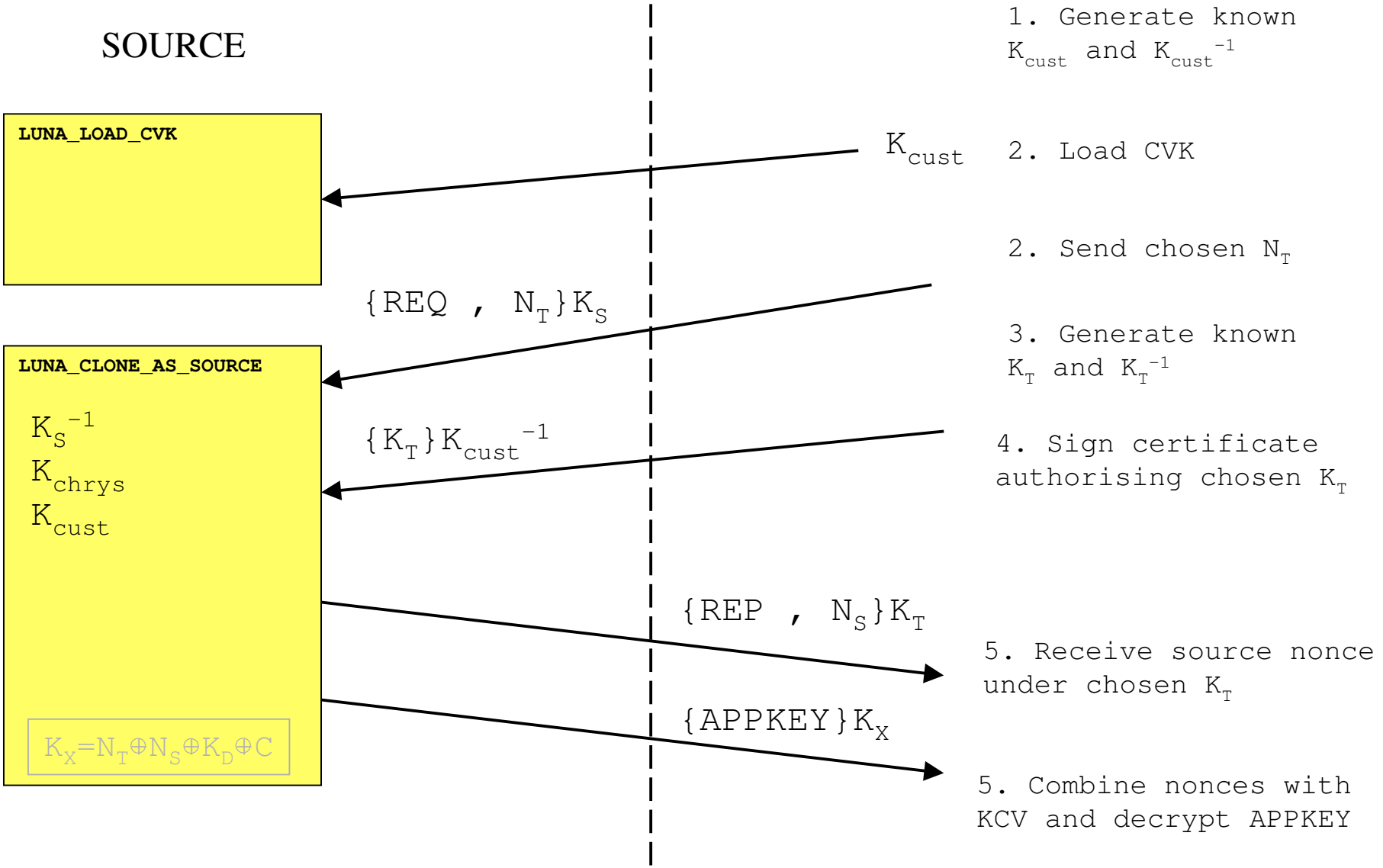
Inside CRYST201.DLL



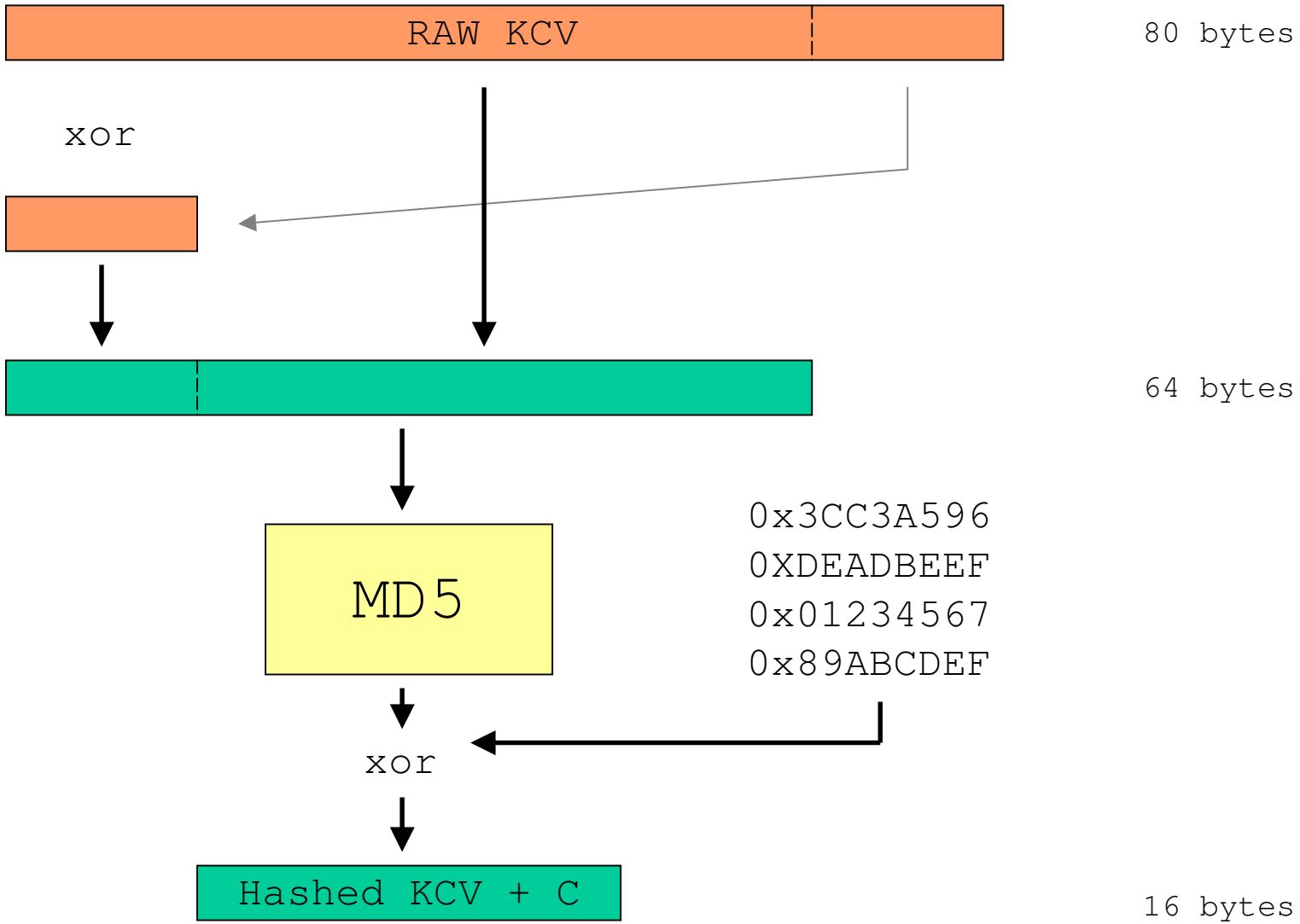
Customer Verification Keys



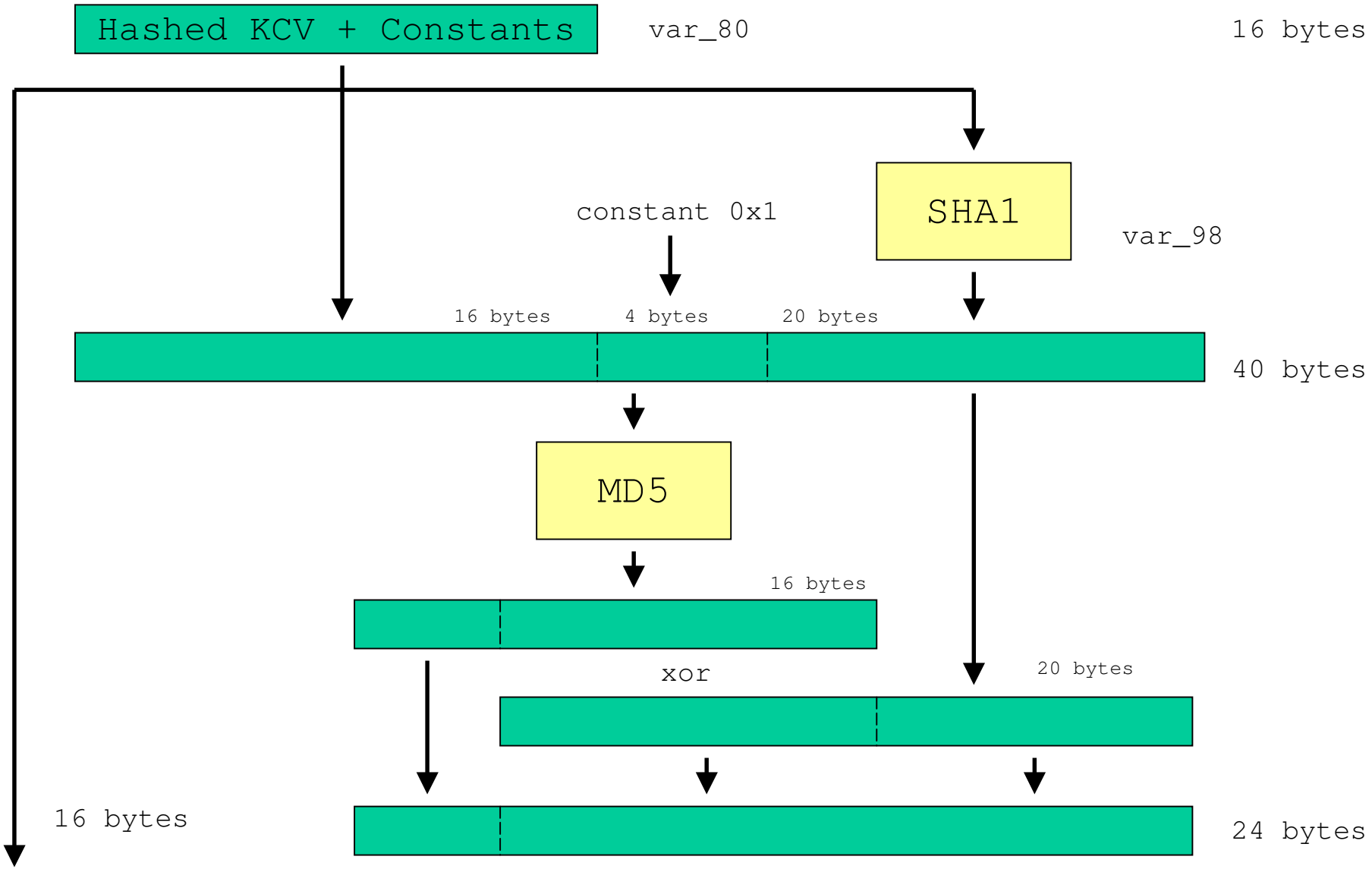
Cloning to Clear



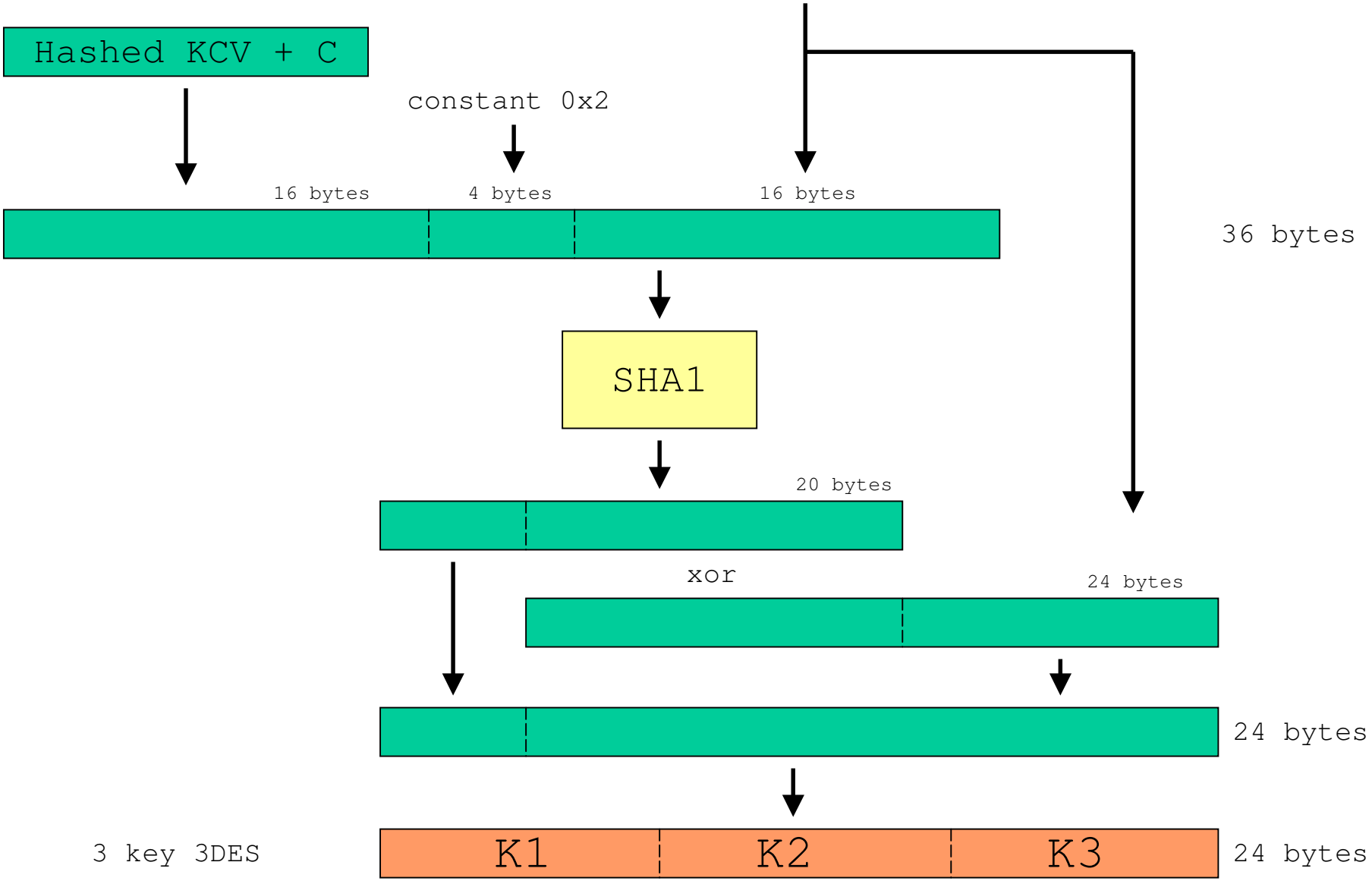
Making the Key Cloning Vector



Making the Key Cloning Vector (2)



Making the Key Cloning Vector (3)



Lessons Learned

- Going in the front door (reverse-engineering) is tough, but it is a skill that can be learned, and done again much more quickly
- Choice of tools, and knowledge of tools is vital to chances of success
- It's easy to drown in a sea of maybes and unknowns and give up. The golden rules of reverse engineering can help
  - "do what you can", and "name everything"

Lessons Learned (2)

- Legacy code is much better camouflage than obfuscation to slow reverse engineering.
- 0xDEADBEEF hinders reimplementation of crypto code as it has to bit-for-bit perfect
- A new defence – stupidity! If the programmer understands his task poorly, the reverse engineer will have an even worse time.
- Beware of undocumented features in your API. Chrysalis didn't let on about the CVK, what are other manufacturers hiding?

Lessons Learned (3)

- The Luna CA3 API *is* secure, but the architecture has accumulated too much baggage – if it is pushed much further, it may break completely.
- If the Luna CA3 is anything to go by, HSM code is no better than O/S code.
- Even if your architecture is not exploited by a Security API attack, it may still be used in an *unexpected way*.

IDA Strengths

- Excellent navigation interface design, once familiarisation done
- Excellent cross-referencing comment system
- Good auto-analysis and support for standard libraries
- Strong use of colours and graphics to help spot patterns
- Good extensibility, supporting scripts and plugins

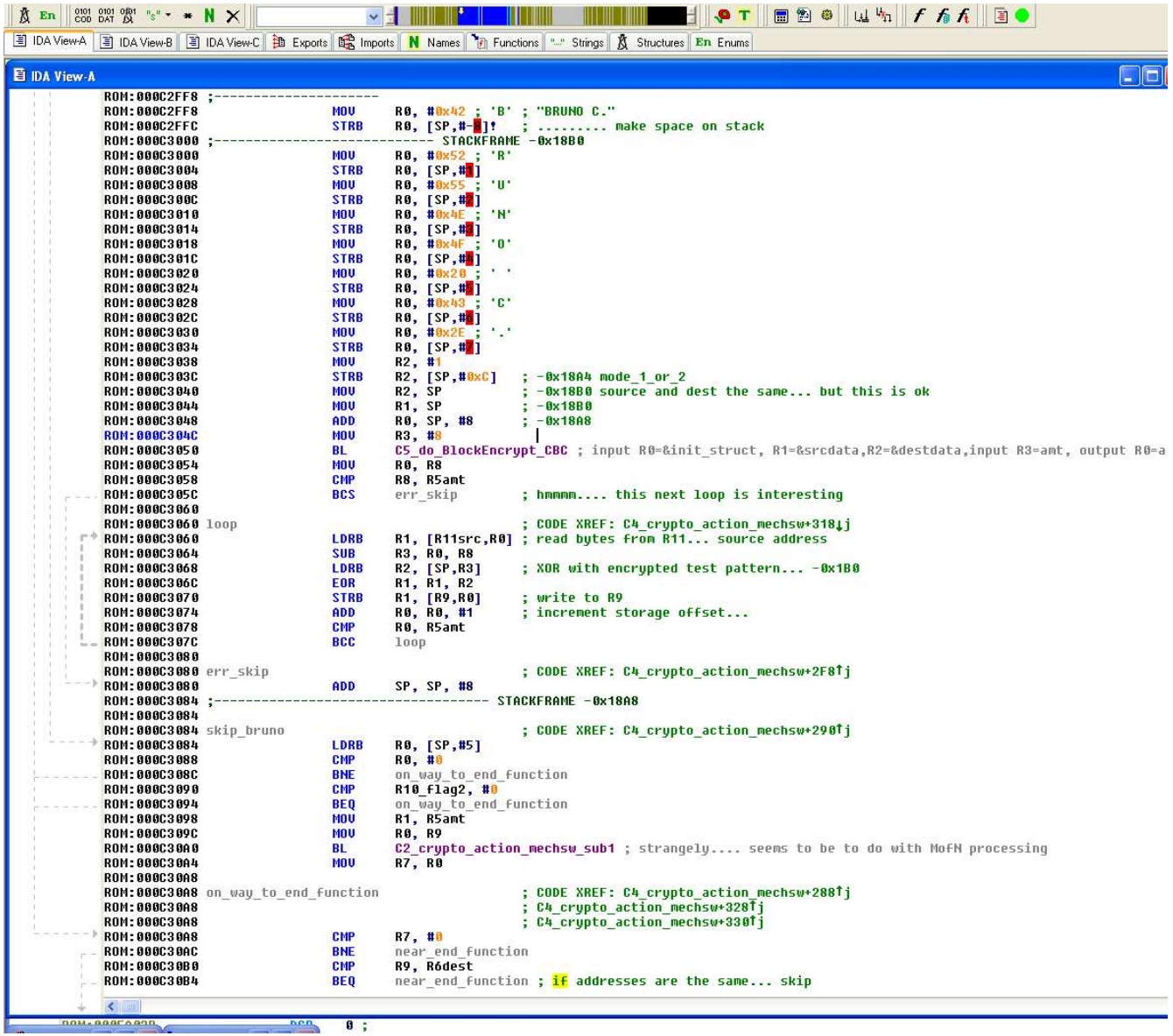
IDA Weaknesses

- No graphing of conditional jumps or calculated jumps
- Poor support for stack variables on ARM
- Poor documentation – many features discovered late
- Non-standard look and feel
- Some cosmetic defects

Weak Spots in the Luna CA3

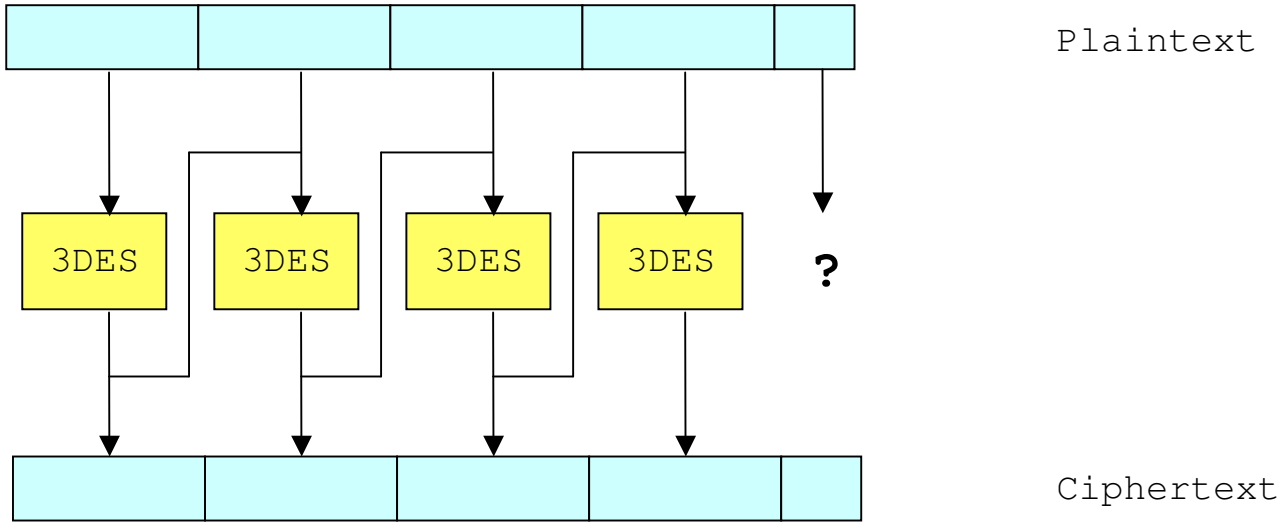
- Application Key Integrity
  - During transport, cipher was 3-Key 3DES in CBC with fixed IV, 32-bit CRC with custom polynomial used for ‘integrity’
- Buffer, integer overflows?
  - Will take a brief look shortly
- Cryptographic Algorithms
  - “BRUNO C.” (to be explained…)

“BRUNO C.”


IDA View-A
IDA View-B
IDA View-C
Exports
Imports
Names
Functions
Strings
Structures
Enums
IDA View-A
ROM:000C2FF8 ;----------------------
ROM:000C2FF8 MOV R0, #0x42 ; 'B' ; "BRUNO C."
ROM:000C2FFC STRB R0, [SP,#-8]! ; .... make space on stack
ROM:000C3000 ;---------------------------- STACKFRAME -0x18B0
ROM:000C3000 MOV R0, #0x52 ; 'R'
ROM:000C3004 STRB R0, [SP,#1]
ROM:000C3008 MOV R0, #0x55 ; 'U'
ROM:000C300C STRB R0, [SP,#2]
ROM:000C3010 MOV R0, #0x4E ; 'N'
ROM:000C3014 STRB R0, [SP,#3]
ROM:000C3018 MOV R0, #0x4F ; 'O'
ROM:000C301C STRB R0, [SP,#4]
ROM:000C3020 MOV R0, #0x20 ; ' '
ROM:000C3024 STRB R0, [SP,#5]
ROM:000C3028 MOV R0, #0x43 ; 'C'
ROM:000C302C STRB R0, [SP,#6]
ROM:000C3030 MOV R0, #0x2E ; '.'
ROM:000C3034 STRB R0, [SP,#7]
ROM:000C3038 MOV R2, #1
ROM:000C303C STRB R2, [SP,#0xC] ; -0x18A4 mode_1_or_2
ROM:000C3040 MOV R2, SP ; -0x18B0 source and dest the same... but this is ok
ROM:000C3044 MOV R1, SP ; -0x18B0
ROM:000C3048 ADD R0, SP, #8 ; -0x18A8
ROM:000C304C MOV R3, #8
ROM:000C3050 BL C5_do_BlockEncrypt_CBC ; input R0=&init_struct, R1=&srcdata,R2=&destdata,input R3=amt, output R0=a
ROM:000C3054 MOV R0, R8
ROM:000C3058 CMP R8, R5amt
ROM:000C305C BCS err_skip ; hmmmm.... this next loop is interesting
ROM:000C3060
ROM:000C3060 loop ; CODE XREF: C4_crypto_action_mechsw+318↓j
ROM:000C3060 LDRB R1, [R11src,R0] ; read bytes from R11... source address
ROM:000C3064 SUB R3, R0, R8
ROM:000C3068 LDRB R2, [SP,R3] ; XOR with encrypted test pattern... -0x1B0
ROM:000C306C EOR R1, R1, R2
ROM:000C3070 STRB R1, [R9,R0] ; write to R9
ROM:000C3074 ADD R0, R0, #1 ; increment storage offset...
ROM:000C3078 CMP R0, R5amt
ROM:000C307C BCC loop
ROM:000C3080
ROM:000C3080 err_skip ; CODE XREF: C4_crypto_action_mechsw+2F8↑j
ROM:000C3080 ADD SP, SP, #8
ROM:000C3084 ;---------------------------------- STACKFRAME -0x18A8
ROM:000C3084
ROM:000C3084 skip_bruno ; CODE XREF: C4_crypto_action_mechsw+290↑j
ROM:000C3084 LDRB R0, [SP,#5]
ROM:000C3088 CMP R0, #0
ROM:000C308C BNE on_way_to_end_function
ROM:000C3090 CMP R10_flag2, #0
ROM:000C3094 BEQ on_way_to_end_function
ROM:000C3098 MOV R1, R5amt
ROM:000C309C MOV R0, R9
ROM:000C30A0 BL C2_crypto_action_mechsw_sub1 ; strangely.... seems to be to do with MofN processing
ROM:000C30A4 MOV R7, R0
ROM:000C30A8
ROM:000C30A8 on_way_to_end_function ; CODE XREF: C4_crypto_action_mechsw+288↑j
ROM:000C30A8 ; C4_crypto_action_mechsw+328↑j
ROM:000C30A8 ; C4_crypto_action_mechsw+330↑j
ROM:000C30A8 CMP R7, #0
ROM:000C30AC BNE near_end_function
ROM:000C30B0 CMP R9, R6dest
ROM:000C30B4 BEQ near_end_function ; if addresses are the same... skip
0 ;

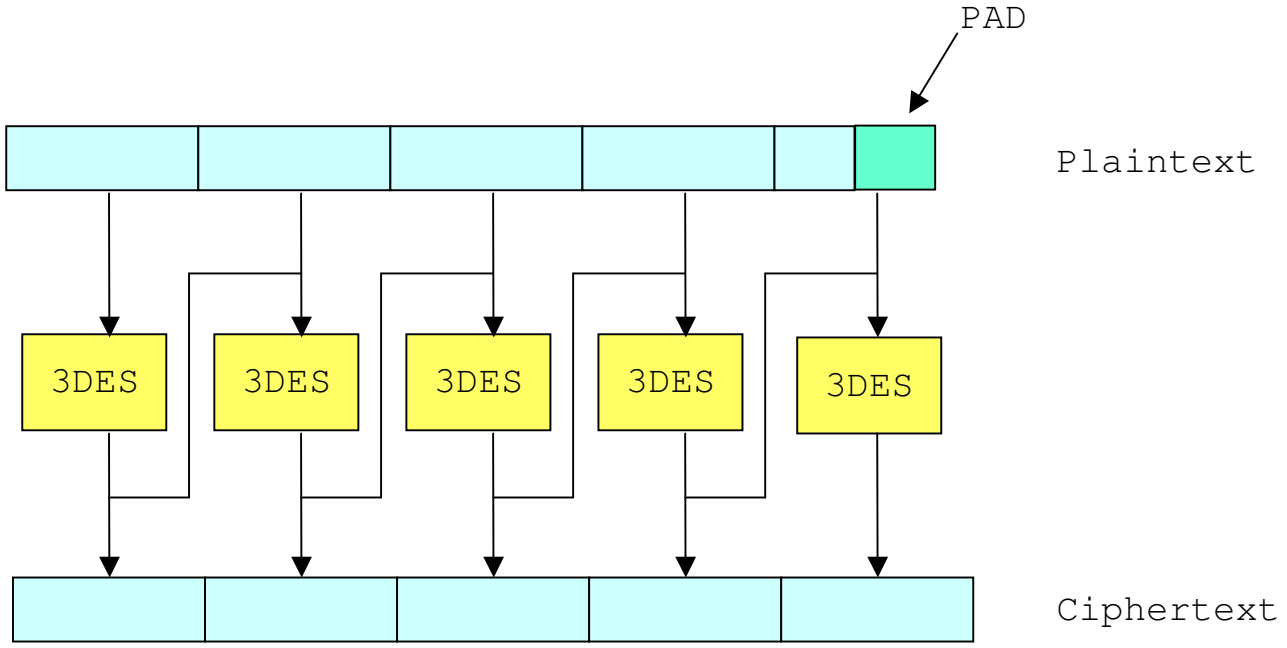
“BRUNO C.”

- Question: How do you encrypt data that doesn’t fit to a block boundary?



“BRUNO C.”

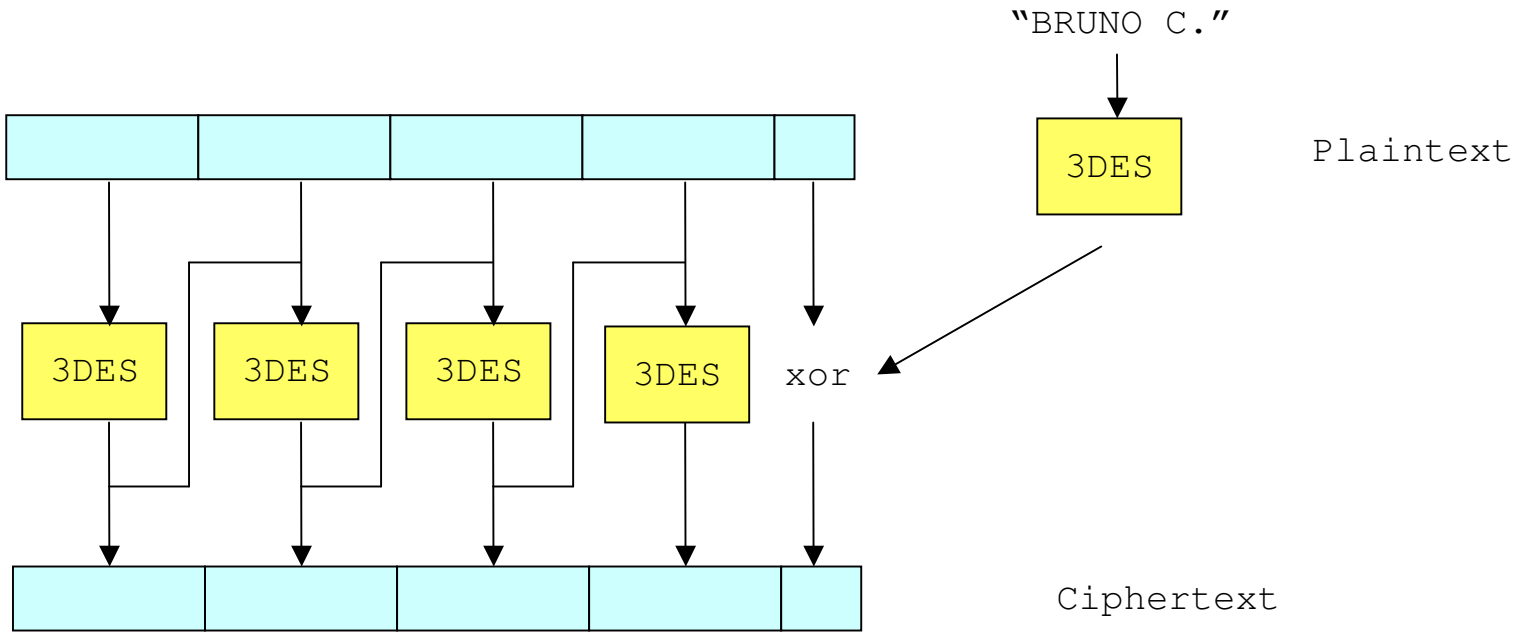
- Question: How do you encrypt data that doesn’t fit to a block boundary?



Problem : Not enough 0xDEADBEEF !

“BRUNO C.”

- Question: How do you encrypt data that doesn’t fit to a block boundary?



Luna CA3 users, don’t worry…

Luna CA3 users, don't worry...

More Information

http://www.cl.cam.ac.uk/~mkb23/research.html

Technical Report coming April 2004

CL: Possible reverse-engineering mini course coming soon