TitaniumHack by Colibri

19.08.2003

Die Titanium Card ist eine laut Aufdruck „high security access smartcard“ (Ich dachte zwar immer hochsichere Smartcards könnte man nicht hacken, aber ich habe mich halt getäuscht.)

Die Titanium Card hat 32 kB Flash, 32 kB Eeprom, 1 kB RAM, unterstützt RSA, DES, 3DES, SHA und lässt sich mit Assembler und C programmieren.

Files für die Titanium Card werden vor dem veröffentlichen im Internet fast immer kryptografisch stark (mit einem Hybridverfahren aus 1024 Bit RSA und 168 Bit 3DES) verschlüsselt. Beim Programmieren werden die Crypted-Files zur Smartcard transferiert und kartenintern entschlüsselt bevor diese ins Flash/Eeprom gebrannt werden.

Da man dadurch die Anwendungen nicht disassemblieren und selbst weiterentwickeln kann oder auch nicht auf wesentlich billigere Smartcards (z.B. FunCard) portieren kann und so auf die teure Titanium Card angewiesen ist, habe ich ein Entschlüsselungsprogramm geschrieben. Ihr findet das Tool „TitaniumHack.exe“ mit der man die Crypted-Files in Plain-Files umwandeln kann auf meiner Homepage <http://colibri.de.ms/>

Im folgenden ist der Ver- und Entschlüsselungsvorgang und danach das Dumpen des Titanium Card Betriebssystem beschrieben.

Verschlüsselungsvorgang:

Card->PC: **ATR** wird ausgegeben (3B EC 00 00 40 32 54 49 54 41 4E 49 55 4D 00 11 01 03)

PC->Card: **Authenticate** - Der PC authentisiert sich gegenüber der Card (80 82 00 00 08 43 52 59 50 54 4F 53 00)

Card->PC: Authentisierung erfolgreich (90 00)

PC->Card: **Get CryptKey** - Der PC fordert einen crypt key an (80 84 00 00 80)

Die Smartcard generiert daraufhin einen PlainKey, der aus einer festen Bytefolge „88 00 00 00 00 00 00 00“ und einer 0x78 Byte langen Zufallszahl zusammengesetzt ist. In der 0x78 Byte langen Zufallszahl ist ein 0x18 langer 3DES Key und ein 8 Byte langer Initialisierungsvektor (IV) enthalten mit dem später das Plain Flash/Eeprom File verschlüsselt wird.

Die Smartcard gibt aber nicht den PlainKey an den PC zurück, sondern verschlüsselt ihn über den Public-Key-Algo RSA.

PlainKey Beispiel (MSB..LSB): 88 00 00 00 00 00 00 00 00 9A 82 31 B5 5F DA B4 86 15
FD AD 16 24 5D CE D1 A0 4D 70 DD 7A 70 F9 8B C6 69 AF D0 C5 F2 BE 7C CF D8 60 1D
C1 2A 87 82 DA 1B 4F 62 29 63 0A F8 8E 74 C8 D7 B4 89 88 3B 9F 3E 8C 7D F0 CC 1B
89 E1 22 CE 69 A7 93 CF 7E 7D 88 22 A3 1C AA B2 F3 1C FC 6D C8 A2 F5 AB 35 99 52
5F F0 09 BB 35 8D 9E 6B FF 36 FF EC 44 08 19 0B 2A EF 33 43 E1 2D C8 AF 03 51 DC
F0 3B D5

Der CryptKey wird wie folgt berechnet:
$$\text{CryptKey} = \text{PlainKey}^{\text{PublicKey}} \bmod \text{Modulus}$$

PublicKey: 0x010001

Modulus:

0xB831414E0B4613922BD35B4B36802BC1E1E81C95A27C958F5382003DF646154CA92FC1CE02C3BE
047A45E9B02A9089B4B90278237C965192A0FCC86BB49BC82AE6FDC2DE709006B86C7676EFD5976
26FAD633A4F7DC48C445D37EB55FCB3B1ABB95BAAA826D5390E15FD14ED403FA2D0CB841C6506095
24EC555E3BC56CA957

Die Berechnung ergibt folgenden CryptKey:

0x7BF975C91AC637F8B9115B1F3EF1E09A4DEB27D31B43EAC752F8581BAC8E34AB43AB1FFA532269
22FF2A7D8A2821322EF5F8D6D0A009CF27C872A746303626823B98119DAC5F5D843B34B261F83E29
3C036D47EB903DECDEAF587A67672FF1309754B781C03F04DBA2EB0AB32E32B5E4C5960D5F55170B
A85823BB56ECFB6FB8

Card->PC: Die Smartcard antwortet mit dem CryptKey. Das LSB wird zuerst übertragen

(B8 6F FB EC 56 BB 23 58 A8 0B 17 55 5F 0D 96 C5 E4 B5 32 2E B3 0A EB A2 DB 04
3F C0 81 B7 54 97 30 F1 2F 67 67 7A 58 AF DE EC 3D 90 EB 47 6D 03 3C 29 3E F8 61
B2 34 3B 84 5D 5F AC 9D 11 98 3B 82 26 36 30 46 A7 72 C8 27 CF 09 A0 D0 D6 F8 F5
2E 32 21 28 8A 7D 2A FF 22 69 22 53 FA 1F AB 43 AB 34 8E AC 1B 58 F8 52 C7 EA 43
1B D3 27 EB 4D 9A E0 F1 3E 1F 5B 11 B9 F8 37 C6 1A C9 75 F9 7B 90 00)

Die Datei CryptKey.Hex enthält anschließend folgendes:

:10000000B8FFBEC56BB2358A80B17555F0D96C5
:10001000E4B5322EB30AEB A2DB043FC081B75497
:1000200030F12F67677A58AFDEEC3D90EB476D03
:100030003C293EF861B2343B845D5FAC9D11983B
:100040008226363046A772C827CF09A0D0D6F8F5
:100050002E3221288A7D2AFF22692253FA1FAB43
:10006000AB348EAC1B58F852C7EA431BD327EB4D
:100070009AE0F13E1F5B11B9F837C61AC975F97B
:00000001FF

PC->Card: **Set CryptKey** - Der PC setzt den CryptKey. Das ist der gleiche den die
Smartcard gerade für uns generiert hat. Das LSB wird zuerst übertragen. (80 86 00 00 80

B8 6F FB EC 56 BB 23 58 A8 0B 17 55 5F 0D 96 C5 E4 B5 32 2E B3 0A EB A2 DB 04 3F
C0 81 B7 54 97 30 F1 2F 67 67 7A 58 AF DE EC 3D 90 EB 47 6D 03 3C 29 3E F8 61 B2
34 3B 84 5D 5F AC 9D 11 98 3B 82 26 36 30 46 A7 72 C8 27 CF 09 A0 D0 D6 F8 F5 2E
32 21 28 8A 7D 2A FF 22 69 22 53 FA 1F AB 43 AB 34 8E AC 1B 58 F8 52 C7 EA 43 1B
D3 27 EB 4D 9A E0 F1 3E 1F 5B 11 B9 F8 37 C6 1A C9 75 F9 7B)

Die Smartcard berechnet nun wie folgt aus dem CryptKey den PlainKey:

$$\text{PlainKey} = \text{CryptKey}^{\text{PrivateKey}} \bmod \text{Modulus}$$

PrivateKey:

0xAA8FB9C85A3A2EFF4923F3C30719D2EB3B94E37B50B68B0BE8A9562E0A724560F2BE2D79E6277A
3ACD3B1635B2849B6FC56E5AEF897BEC799C9DA9199F2337CAC74CC109E3A285EA8CB70852BA565
707526D52FE280B62F3EB4FB087D10DDE7C58BFA61CAB12B8F0DB79634D97CB925082B8DCD139500
7E0DBCD91F5C2D2E39

Modulus:

0xB831414E0B4613922BD35B4B36802BC1E1E81C95A27C958F5382003DF646154CA92FC1CE02C3BE
047A45E9B02A9089B4B90278237C965192A0FCC86BB49BC82AE6FDC2DE709006B86C7676EFD5976
26FAD633A4F7DC48C445D37EB55FCB3B1ABB95BAAA826D5390E15FD14ED403FA2D0CB841C6506095
24EC555E3BC56CA957

Die Berechnung ergibt wieder den ursprünglichen PlainKey:

0x8800000000000000009A8231B55FDAB48615FDAD16245DCED1A04D70DD7A70F98BC669AFD0C5F2BE
7CCFD8601DC12A8782DA1B4F6229630AF88E74C8D7B489883B9F3E8C7DF0CC1B89E122CE69A793CF
7E7D8822A31CAAB2F31CFC6DC8A2F5AB3599525FF009BB358D9E6BFF36FFEC4408190B2AEF3343E1
2DC8AF0351DCF03BD5

Jetzt extrahiert die Smartcard den 0x18 Byte langen Key und den 8 Byte langen IV für die nachfolgenden 3DES Operationen.

3DES-Key: 9A 82 31 B5 5F DA B4 86 15 FD AD 16 24 5D CE D1 A0 4D 70 DD 7A 70 F9
8B

IV: C6 69 AF D0 C5 F2 BE 7C

Card->PC: CryptKey akzeptiert (90 00)

PC->Card: **Select Flash** - Der PC wählt das Flash aus (80 70 80 00 00)

Card->PC: Befehl erfolgreich (90 00)

Als Beispiel soll unsere PlainFlash.Hex folgendes enthalten:

:10200000000102030405060708090A0B0C0D0E0F
:10201000101112131415161718191A1B1C1D1E1F
:00000001FF

PC->Card: **Crypt Flash** - Der PC überträgt die Nutzdaten der .Hex-Datei die verschlüsselt werden soll. Die Startadresse in unseren Fall 0x2000 wird in P1P2 übertragen (80 88 20 00 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F)

Die Daten werden nun mit dem 3DES Algo im CBC Mode verschlüsselt.

Zu Beachten ist, dass unmittelbar vor und nach dem 3DES Crypt eines 8 Byte Blocks ein spezieller 8 Byte Block per XOR-Verknüpfung beigemischt wird.

Dieser spezielle 8 Byte Block ist wie folgt aufgebaut:

Die beiden linken Bytes enthalten die Offsetadresse der 8 Byte Nutzdaten. Das dritte Byte enthält den Wert 0x80 da es Daten für das Flash sind (Bei Eeprom Daten würde es 0x40 enthalten). Die 5 rechten Bytes enthalten nur Nullen und verändern dadurch das Ergebnis nicht.

PlainBlock1 wird mit dem IV XOR verknüpft.

Dieses Ergebnis wird mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 80 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann 3DES verschlüsselt.

Diese verschlüsselten Daten werden wieder mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 80 00 00 00 00 00 falls Offset = 0x2000).

Das Ergebnis ist der CryptBlock1. Dieser CryptBlock1 wird jetzt zum aktuellen IV für den nächsten Block.

Card->PC: Verschlüsselung war erfolgreich (90 00)

PC->Card: **Get Crypt Response** - Der PC fordert die verschlüsselten Daten von der Smartcard an (80 B2 00 00 20)

Card->PC: Die verschlüsselten Daten werden zurückgeschickt (DF 96 CB A1 2F 82 DD 82 25 EF 72 0C 3D 47 C4 C1 75 FB E6 9D D2 52 FE A4 D5 ED F5 C4 BD 71 35 A9 90 00)

Die CryptedFlash.Hex enthält anschliessend folgendes:

```
:10200000DF96CBA12F82DD8225EF720C3D47C4C1
:1020100075FBE69DD252FEA4D5EDF5C4BD7135A9
:00000001FF
```

PC->Card: **Select Eeprom** - Der PC wählt das Eeprom aus (80 70 00 00 00)

Card->PC: Befehl erfolgreich (90 00)

Als Beispiel soll unsere PlainEep.Eep folgendes enthalten:

```
:10200000000102030405060708090A0B0C0D0E0F
:10201000101112131415161718191A1B1C1D1E1F
:00000001FF
```

PC->Card: **Crypt Eeprom** - Der PC überträgt die Nutzdaten der .Eep-Datei die verschlüsselt werden soll. Die Startadresse in unseren Fall 0x2000 wird in P1P2

übertragen (80 88 20 00 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F)

Die Daten werden nun mit dem 3DES Algo im CBC Mode verschlüsselt.

Zu Beachten ist, dass unmittelbar vor und nach dem 3DES Crypt eines 8 Byte Blocks ein spezieller 8 Byte Block per XOR-Verknüpfung beigemischt wird.

Dieser spezielle 8 Byte Block ist wie folgt aufgebaut:

Die beiden linken Bytes enthalten die Offsetadresse der 8 Byte Nutzdaten. Das dritte Byte enthält den Wert 0x40 da es Daten für das Eeprom sind (Bei Flash Daten würde es 0x80 enthalten). Die 5 rechten Bytes enthalten nur Nullen und verändern dadurch das Ergebnis nicht.

PlainBlock1 wird mit dem IV XOR verknüpft.

Dieses Ergebnis wird mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 40 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann 3DES verschlüsselt.

Diese verschlüsselten Daten werden wieder mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 40 00 00 00 00 00 falls Offset = 0x2000).

Das Ergebnis ist der CryptBlock1. Dieser CryptBlock1 wird jetzt zum aktuellen IV für den nächsten Block.

Card->PC: Verschlüsselung war erfolgreich (90 00)

PC->Card: **Get Crypt Response** - Der PC fordert die verschlüsselten Daten von der Smartcard an (80 B2 00 00 20)

Card->PC: Die verschlüsselten Daten werden zurückgeschickt (83 0C 5F B6 7D 4A 94 5D 57 98 4E E5 9D 55 7D C8 2D C9 43 EC B9 12 4A 20 FA 54 74 93 50 D4 93 FE 90 00)

Die CryptedEEp.Eep enthält anschliessend folgendes:

:10200000830C5FB67D4A945D57984EE59D557DC8
:102010002DC943ECB9124A20FA54749350D493FE
:00000001FF

Entschlüsselungsvorgang:

Card->PC: **ATR** wird ausgegeben (3B EC 00 00 40 32 54 49 54 41 4E 49 55 4D 00 11 01 03)

PC->Card: **Authenticate** - Der PC authentisiert sich gegenüber der Card (80 82 00 00 08 43 52 59 50 54 4F 53 00)

Card->PC: Authentisierung erfolgreich (90 00)

Als Beispiel soll unsere CryptKey.Hex folgendes enthalten:

:10000000B86FFBEC56BB2358A80B17555F0D96C5
:10001000E4B5322EB30AEB2DB043FC081B75497
:1000200030F12F67677A58AFDEEC3D90EB476D03
:100030003C293EF861B2343B845D5FAC9D11983B
:100040008226363046A772C827CF09A0D0D6F8F5
:100050002E3221288A7D2AFF22692253FA1FAB43
:10006000AB348EAC1B58F852C7EA431BD327EB4D
:100070009AE0F13E1F5B11B9F837C61AC975F97B
:00000001FF

PC->Card: **Set CryptKey** - Der PC setzt den crypt key. Das LSB wird zuerst übertragen

(80 86 00 00 80 B8 6F FB EC 56 BB 23 58 A8 0B 17 55 5F 0D 96 C5 E4 B5 32 2E B3 0A EB A2 DB 04 3F C0 81 B7 54 97 30 F1 2F 67 67 7A 58 AF DE EC 3D 90 EB 47 6D 03 3C 29 3E F8 61 B2 34 3B 84 5D 5F AC 9D 11 98 3B 82 26 36 30 46 A7 72 C8 27 CF 09 A0 D0 D6 F8 F5 2E 32 21 28 8A 7D 2A FF 22 69 22 53 FA 1F AB 43 AB 34 8E AC 1B 58 F8 52 C7 EA 43 1B D3 27 EB 4D 9A E0 F1 3E 1F 5B 11 B9 F8 37 C6 1A C9 75 F9 7B)

Die Smartcard berechnet nun wie folgt aus dem CryptKey den PlainKey:

PlainKey = CryptKey^{PrivateKey} mod Modulus

PrivateKey:

0xAA8FB9C85A3A2EFF4923F3C30719D2EB3B94E37B50B68B0BE8A9562E0A724560F2BE2D79E6277A
3ACD3B1635B2849B6FC56E5AEF897BECD799C9DA9199F2337CAC74CC109E3A285EA8CB70852BA565
707526D52FE280B62F3EB4FB087D10DDE7C58BFA61CAB12B8F0DB79634D97CB925082B8DCD139500
7E0DBCD91F5C2D2E39

Modulus:

0xB831414E0B4613922BD35B4B36802BC1E1E81C95A27C958F5382003DF646154CA92FC1CE02C3BE
047A45E9B02A9089B4B90278237C965192A0FCC86BB49BC82AE6FDC2DE709006B86C7676EFD5976
26FAD633A4F7DC48C445D37EB55FCB3B1ABB95BAAA826D5390E15FD14ED403FA2D0CB841C6506095
24EC555E3BC56CA957

Die Berechnung ergibt wieder den ursprünglichen PlainKey:

```
0x880000000000000000009A8231B55FDAB48615FDAD16245DCED1A04D70DD7A70F98BC669AFD0C5F2BE
7CCFD8601DC12A8782DA1B4F6229630AF88E74C8D7B489883B9F3E8C7DF0CC1B89E122CE69A793CF
7E7D8822A31CAAB2F31CFC6DC8A2F5AB3599525FF009BB358D9E6BFF36FFEC4408190B2AEF3343E1
2DC8AF0351DCF03BD5
```

Jetzt extrahiert die Smartcard den 0x18 Byte langen Key und den 8 Byte langen IV für die nachfolgenden 3DES Operationen.

3DES-Key: 9A 82 31 B5 5F DA B4 86 15 FD AD 16 24 5D CE D1 A0 4D 70 DD 7A 70 F9
8B

IV: C6 69 AF D0 C5 F2 BE 7C

Card->PC: CryptKey akzeptiert (90 00)

PC->Card: **Select Flash** - Der PC wählt das Flash aus (80 70 80 00 00)

Card->PC: Befehl erfolgreich (90 00)

Als Beispiel soll unsere CryptedFlash.Hex folgendes enthalten:

```
:10200000DF96CBA12F82DD8225EF720C3D47C4C1
:1020100075FBE69DD252FEA4D5EDF5C4BD7135A9
:00000001FF
```

PC->Card: **Write Flash** - Der PC überträgt die verschlüsselten Nutzdaten der .Hex-Datei.

Die Startadresse in unseren Fall 0x2000 wird in P1P2 übertragen (80 D7 20 00 20 DF 96
CB A1 2F 82 DD 82 25 EF 72 0C 3D 47 C4 C1 75 FB E6 9D D2 52 FE A4 D5 ED F5 C4 BD
71 35 A9)

Die Daten werden nun mit dem 3DES Algo im CBC Mode entschlüsselt und anschliessend ins Flash gebrannt.

Zu Beachten ist, dass unmittelbar vor und nach dem 3DES Crypt eines 8 Byte Blocks ein spezieller 8 Byte Block per XOR-Verknüpfung beigemischt wird.

Dieser spezielle 8 Byte Block ist wie folgt aufgebaut:

Die beiden linken Bytes enthalten die Offsetadresse der 8 Byte Nutzdaten. Das dritte Byte enthält den Wert 0x80 da es Daten für das Flash sind (Bei Eeprom Daten würde es 0x40 enthalten). Die 5 rechten Bytes enthalten nur Nullen und verändern dadurch das Ergebnis nicht.

Der CryptBlock1 wird mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 80 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann mit 3DES entschlüsselt.

Diese verschlüsselten Daten werden wieder mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 80 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann mit dem IV XOR verknüpft.

Das Ergebnis ist der PlainBlock1. Der ursprüngliche CryptBlock1 wird jetzt zum aktuellen IV für den nächsten Block.

Card->PC: Schreiben war erfolgreich (90 00)

PC->Card: **Select Eeprom** - Der PC wählt das Eeprom aus (80 70 00 00 00)

Card->PC: Befehl erfolgreich (90 00)

Als Beispiel soll unsere CryptedEep.Eep folgendes enthalten:

```
:10200000830C5FB67D4A945D57984EE59D557DC8  
:102010002DC943ECB9124A20FA54749350D493FE  
:00000001FF
```

PC->Card: **Write Eeprom** - Der PC überträgt die verschlüsselten Nutzdaten der .Eep-

Datei. Die Startadresse in unseren Fall 0x2000 wird in P1P2 übertragen (80 D7 20 00 20 83 0C 5F B6 7D 4A 94 5D 57 98 4E E5 9D 55 7D C8 2D C9 43 EC B9 12 4A 20 FA 54 74 93 50 D4 93 FE)

Die Daten werden nun mit dem 3DES Algo im CBC Mode entschlüsselt und anschliessend ins Eeprom gebrannt.

Zu Beachten ist, dass unmittelbar vor und nach dem 3DES Crypt eines 8 Byte Blocks ein spezieller 8 Byte Block per XOR-Verknüpfung beigemischt wird.

Dieser spezielle 8 Byte Block ist wie folgt aufgebaut:

Die beiden linken Bytes enthalten die Offsetadresse der 8 Byte Nutzdaten. Das dritte Byte enthält den Wert 0x40 da es Daten für das Eeprom sind (Bei Flash Daten würde es 0x80 enthalten). Die 5 rechten Bytes enthalten nur Nullen und verändern dadurch das Ergebnis nicht.

Der CryptBlock1 wird mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 40 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann mit 3DES entschlüsselt.

Diese verschlüsselten Daten werden wieder mit dem speziellen 8 Byte Block XOR-Verknüpft (z.B. 20 00 40 00 00 00 00 00 falls Offset = 0x2000).

Diese Daten werden dann mit dem IV XOR verknüpft.

Das Ergebnis ist der PlainBlock1. Der ursprüngliche CryptBlock1 wird jetzt zum aktuellen IV für den nächsten Block.

Card->PC: Schreiben war erfolgreich (90 00)

Dumpen des Titanium Card Betriebssystem:

Um z.B. an den RSA PrivateKey zu kommen war ein Auslesen des Betriebssystem nötig. Dazu muss ein Dump Programm in die Smartcard geladen werden, das den zu dumpenden Adressbereich (z.B. Startadresse in P1 & P2 und die Anzahl im Len-Byte) entgegennimmt und die Bereiche aus dem Flash an den PC zurückgibt.

Zum Schreiben des Programms kann man sich die Entwicklungsumgebung für die Titanium Card „Development_Titanium_14-05-03“ z.B. von

<http://www.keyteam.de/kteam/download.php?op=viewdownload&cid=12> herunterladen.

Das in der Entwicklungsumgebung enthaltene Beispielprogramm ist dafür bestens geeignet, lediglich eine minimale Änderung ist nötig.

Fügt den folgenden gelb markierten Code in die Datei „TestApplication.c“ ein:

```
// check class
if( CLA != 0xC0 ) {
    // unsupported class
    SendSW(SW_WRONG_CLA);
    continue;
}

//Anfang Dump Code
if( INS == 0xD0 ) {
    // ACK
    SendByte(INS);
    for(i=0; i<LEN; i++)
        SendByte(CommandHeader[0x9C00 + (P1<<8)+P2 + i]); //EE ab 0xA000
= Flash ab 0x0000
    SendSW(SW_OK);
    continue;
}
//Ende Dump Code

// return 8 Bytes RANDOM
if( INS == 0x10 ) {
    if( (P1 | P2) != 0 ) {
        SendSW(SW_WRONG_P1P2);
        continue;
    }
}
```

Lasst den Rest am Besten so wie er ist, denn der Offset der Variablen „CommandHeader“ sollte nach dem compilieren immer noch bei 0x0400 liegen (der CommandHeader-Offset 0x0400 + die Konstante 0x9C00 muss 0xA000 ergeben).

Das Tool zum Kompilieren des Programms könnt Ihr z.B. von <http://www.iar.com/Products/?name=EWAVR> herunterladen („30-day evaluation version“ anklicken).

Startet die IAR Embedded Workbench und öffnet die Datei „TestApplication.prj“.

Wählt im Fenster „TestApplication.prj“ das Target „Release“ aus.

Klickt dann auf Project -> Build All.

Jetzt wurden im Unterverzeichnis „Release\Exe“ die beiden Dateien TestApplication.hex und TestApplication.eep erzeugt.

Diese zwei Dateien könnt Ihr mit dem Titanium Card Progger auf die Smartcard schreiben.

Folgendes WinExplorer-Skript zeigt wie die Kommandos zum Auslesen aufgebaut sind:

```
; WinExplorer File Dump_TC-OS.xpl
; Das Titanium Betriebssystem befindet sich im Flash von 0x0000 - 0x1FFF
; Das User Programm ist ab 0x2000 zu finden
; CLA = C0
; INS = D0
; P1 = Hi-Byte der Flash Adresse
; P2 = Lo-Byte der Flash Adresse
; Len = Anzahl der Bytes

;lese ersten Block (0x0000 - 0x007F) des TC-OS
C0 D0 00 00 80
r01 ;echo INS
```

```

r80 ;Read Data

;lese zweiten Block (0x0080 - 0x00FF) des TC-OS
C0 D0 00 80 80
r01 ;echo INS
r80 ;Read Data

;lese dritten Block (0x0100 - 0x017F) des TC-OS
C0 D0 01 00 80
r01 ;echo INS
r80 ;Read Data

; evtl. weitere Blöcke lesen

;lese letzten Block (0x1F80 - 0x1FFF) des TC-OS
C0 D0 1F 80 80
r01 ;echo INS
r80 ;Read Data

```

Bei einer Titanium Card mit OS Version 1.03 solltet Ihr folgende Daten zurückbekommen:

```

Executing Script: C:\Dump_TC-OS.xpl
TX Data : C0 D0 00 00 80
RX Data : D0
RX Data : 0C 94 AD 01 0C 94 AD 01 0C 94 6C 0A 0C 94 B9 05
          58 98 18 95 50 98 18 95 0C 94 22 00 0C 94 34 00
          3A 00 4E 05 75 05 04 06 AF 05 24 06 43 06 6E 06
          83 00 98 00 A5 00 B7 00 CA 00 DB 00 5F 01 FE 01
          79 01 3B 07 E0 91 FF 04 E2 31 58 F4 FF 27 EE 0F
          E0 5E FF 4F C8 95 10 2C 31 96 C8 95 E1 2D F0 2D
          09 95 FF 27 EE 27 08 95 F0 32 18 F0 F0 38 08 F4
          C8 95 08 95 0E 94 1C 0B 0C 94 3C 00 FF FF FF FF

```

Extra data in receive buffer ... Flushing buffer

```
Flushed : 90 00
```

Hier sind ein paar Offsets, falls jemand das Titanium OS disassemblieren will:

Startadresse nach dem Reset ist Flash:0000

```

Unterprogramme:
Flash:003A SYSCALL_MGMT_RESTORELOADER
Flash:054E SYSCALL_IO_ISOCLOCK
Flash:0575 SYSCALL_IO_SENDBYTE
Flash:0604 SYSCALL_IO_SENDSW
Flash:05AF SYSCALL_IO_RECEIVEBYTE
Flash:0624 SYSCALL_NVM_RAMTOEEPROM
Flash:0643 SYSCALL_NVM_EEPROMSET
Flash:066E SYSCALL_NVM_EEPROMCOPY
Flash:0083 SYSCALL_CRYPT_64BITRANDOM
Flash:0098 SYSCALL_CRYPT_GENRND
Flash:00A5 SYSCALL_CRYPT_CRYPTORESET
Flash:00B7 SYSCALL_CRYPT_CLEARY
Flash:00CA SYSCALL_CRYPT_RANDOMGENERATION
Flash:00DB SYSCALL_CRYPT_LOADMODULUS
Flash:015F SYSCALL_CRYPT_MODEXP
Flash:01FE SYSCALL_CRYPT_DES

```

Flash:0179 SYSCALL_CRYPT_3DES
Flash:073B SYSCALL_CRYPT_SHA

RSA-Konstanten:

Flash:0F40 RSA_Modulus (MSB first) 80 Bytes lang
Flash:0F80 RSA_PrivateKey (MSB first) 80 Bytes lang
Flash:0FC0 RSA_PublicKey (MSB first) 3 Bytes lang

Hinweis: Die (Word) Adressen im Disassembler gehen von Flash:0000 bis Flash:0FFF, die (Byte) Adressen im .BIN-File dagegen von 0000 bis 1FFF. Der RSA_PrivateKey ist im Disassembler also auf Pos Flash:0F80, im .BIN-File aber auf Pos 1F00.

Ihr könnt mich unter folgender Adresse erreichen:

colibri_dvb@lycos.com

Das neueste Dokument könnt Ihr auf der folgenden Seite finden:

<http://colibri.de.ms/>