Kryptoanalyse PowerVu TV-Verschlüsselung

Es wird gezeigt dass der Security-Chip Schwachstellen enthält die das Ermitteln des Schlüssels ermöglichen.

10.03.2010 (Version 1.0) aktuelle Version auf http://colibri.de.ms/ => PowerVu

Colibri <colibri_dvb@lycos.com>

INHALTSVERZEICHNIS

KRYPTOANALYSE POWERVU TV-VERSCHLÜSSELUNG

EINLEITUNG

Einige TV und Radio Programme werden mit dem Kryptosystem PowerVu verschlüsselt über Satellit ausgesendet. Über Hot Bird 13.0°E sendet beispielsweise „American Forces Network“ (AFN) über Frequenz 10775 H & 11096 H und Bundeswehr TV (BW TV) über Frequenz 11054 H. Aber auch über andere Satelliten (28.2°E, 16.0°E, 4.8°E, 0.8°W, 4.0°W, 15.0°W, 22.0°W, 27.5°W, 30.0°W, 37.5°W, 40.5°W, ...) werden Programme über PowerVu verschlüsselt abgestrahlt.

Hier wurde der PowerVu Receiver D9234 (bzw. der integrierte Security-Chip) der zum Entschlüsseln der „American Forces Network“ (AFN) Programme verwendet werden kann auf Schwachstellen untersucht. Ob bei den anderen Providern die auch PowerVu verwenden der gleiche Security-Chip (mit den gleichen Schwachstellen) zum Einsatz kommt konnte ich nicht prüfen.

Es wurde herausgefunden dass der Security-Chip gleich mehrere Schwachstellen enthält, die kombiniert ausgenutzt das Ermitteln des Entschlüsselungsschlüssels ermöglicht. Die Schwachstellen werden hier beschrieben, wie der ermittelte Schlüssel lautet aber nicht.

Im Folgenden wird beschrieben was bereits in der Vergangenheit über das PowerVu System herausgefunden wurde. Dann wird gezeigt wie der Entschlüsselungsalgorithmus funktioniert. Anschließend wird auf die Schwachstellen eingegangen. Zum Schluss werden noch Möglichkeiten aufgezeigt wie sich die Schwachstellen beheben lassen.

PowerVu-System Grobübersicht

Zuerst mal grob die Info wie die PowerVu Entschlüsselung funktioniert.

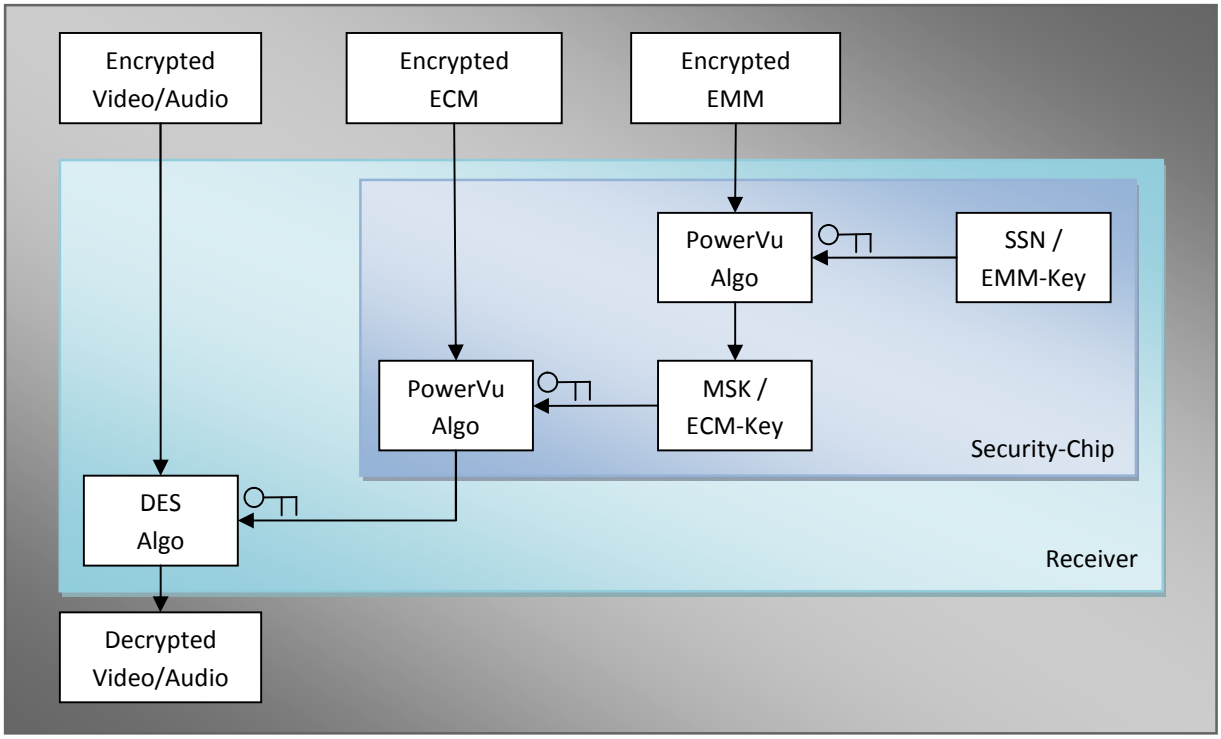

ABBILDUNG 1: SCHLÜSSELHIRARCHIE

Das Programm (u.a. Video und Audio) wird vor dem Aussenden über Satellit mit dem DES-Algorithmus (ECB-Modus) verschlüsselt. Der DES-Schlüssel wird ca. alle 10 Sekunden geändert. Der verschlüsselte DES-Schlüssel wird über einen separaten Stream „Entitlement Control Message“ (ECM) mit übertragen. Der Receiver empfängt die verschlüsselten ECM Pakete und leitet Sie an den Security-Chip weiter. Dieser entschlüsselt mit dem „Multi Session Key“ (MSK) (also ein ECM-Key) dann das ECM Paket und gibt den entschlüsselten DES-Schlüssel an den Receiver zurück (falls zusätzlich eine Freischaltung für das Programm vorliegt) der damit das Programm entschlüsselt. Der MSK ist dabei langlebiger. Er wird vermutlich nur alle paar Monate (oder Jahre) gewechselt. Um dem Security-Chip einen neuen MSK zukommen zu lassen wird über einen separaten Stream „Entitlement Management Message“ (EMM) Pakete übertragen. Die EMM Pakete enthalten verschlüsselte Infos für den Security-Chip der diese nach der Entschlüsselung in seinen nichtflüchtigen Speicher (EEPROM) schreibt. Im Unterschied zu den ECMs die für alle bestimmt sind, enthalten die EMMs immer als Adresse die unverschlüsselte Seriennummer des Security-Chips für den sie bestimmt sind. Entschlüsselt wird ein EMM mit dem EMM-Key. Jeder Security-Chip besitzt bei der Auslieferung einen anderen EMM-Key und er kann anschließend nicht mehr geändert werden. Mit einem EMM kann auch die Freischaltung (Tiers) der Programme geändert werden. Es können also gezielt einzelne Programme gesperrt und andere freigeschaltet werden.

WAS BISHER GESCHAH

Bereits 2005 habe ich unter http://colibri.de.ms in der PowerVu Sektion meine Erkenntnisse veröffentlicht. Speziell das Kapitel 6 „PowerVu Conditional Access system“ im PowerVuSecrets.PDF sollte vor dem Weiterlesen durchgelesen werden. Hier wurde speziell die Firmware des Receivers und die physikalische Schnittstelle vom Security-Chip, der bei PowerVu auch als „Internal Security Element“ (ISE) bezeichnet wird, analysiert. Der ISE-Chip ist kein simples EEPROM, sondern er hat die gleichen Anschlüsse wie der Chip einer Smartcard (VCC, GND, Reset, Takt und Daten). Es wurden die ISE-Kommandos aufgelistet die in der Firmware gefunden wurden. Es wurde gezeigt wie ein ECM in zwei ISE-Kommandos gesplittet werden muss (GetBaseCW und GetSeeds) und wie die Antworten zu einem DES-Key zusammengesetzt werden müssen. Es wurde auch ein Schaltplan für ein „PC to ISE Chip interface“ und das passende Tool „PowerVu-ISE-Tool.exe“ zur Verfügung gestellt mit der man verschiedenen Kommandos zum ISE schicken und die Antworten beobachten kann. Das ISE unterstützt mehr Kommandos (die aber im normalen Gebrauch nicht benötigt werden) als in der Receiver-Firmware vorhanden sind. Mit all den Infos konnte man bereits z.B. bei einem defekten Receiver aber intakten und freigeschaltenen ISE das ISE an einen PC anschließen und ein mit einer DVB-s Satellitenempfangskarte aufgezeichnetes Video offline entschlüsseln. Das ist jedoch kein Sicherheitsproblem da immer noch ein freigeschaltetes ISE benötigt wird.

Am 28.02.2006 hat dann der Benutzer Cinosana in einem Forum die Datei „pvufull.zip“ gepostet. Sie enthält die ISE-Firmware für die 6805 CPU und Daten also das interne EEPROM und ROM. Die EMM und ECM Schlüssel hat er entfernt. Angeblich kam der an diese internen Daten in dem der ein kleines Programm das den kompletten Speicher ausließt und über die serielle ISE-Schnittstelle ausgibt als Kommando dem ISE übergeben hat. Im ISE-RAM liegt hinter dem Speicher in dem ein empfangenes Kommando zur späteren Auswertung abgelegt wird der Stack. Das ISE prüft im Normalfall das erste Kommandobyte das die Länge der folgenden Daten enthält. Ist die Längenbyte grösser als 23h dann bricht das ISE mit einem Fehler ab, damit ein Buffer overflow der den Stack überschreibt verhindert wird.

```
cmp    #023h        ; 4434  A1 23   (2)  If LEN > 0x23, error
bhi    mot4478      ; 4436  22 40   (3)  <error>
```

Um diesen overflow Schutz zu umgehen hat er einen modifizierten T911 Glitcher verwendet und direkt nach dem Sendes des Längenbytes einen Glitch ausgelöst. Das ISE überspringt bei einem Glich einen Befehl in diesem Fall den obigen bhi-Befehl (branch if higher) der einen Fehler ausgibt, statt weitere Daten zu empfangen. Ein Glitch ist hier eine kurzzeitige Änderung in der ISE-Spannungsversorgung bei gleichzeitiger Änderung der Taktfrequenz, dadurch kommt es zum Überspringen des Befehls. Wie er den Glitcher modifiziert hat hat er aber nicht verraten. Die Schwachstellen um die es in diesem Dokument geht sind aber Andere. Der ECM-Key lässt sich nämlich auch ermitteln ohne ein Exploit in das ISE zu laden.

Durch den Einsatz eines Disassemblers (z.B. unterstützt IDA – The Interactive Disassembler die 6805 CPU) konnten weitere Erkenntnisse gewonnen werden.

DER ENTSCHLÜSSELUNGSALGORITHMUS IM DETAIL

Im Folgenden werden der Stream cipher, die ECM und die EMM Entschlüsselung beschrieben.

STREAM CIPHER

Der PowerVu Algorithmus verwendet als Kern folgenden Stream cipher.

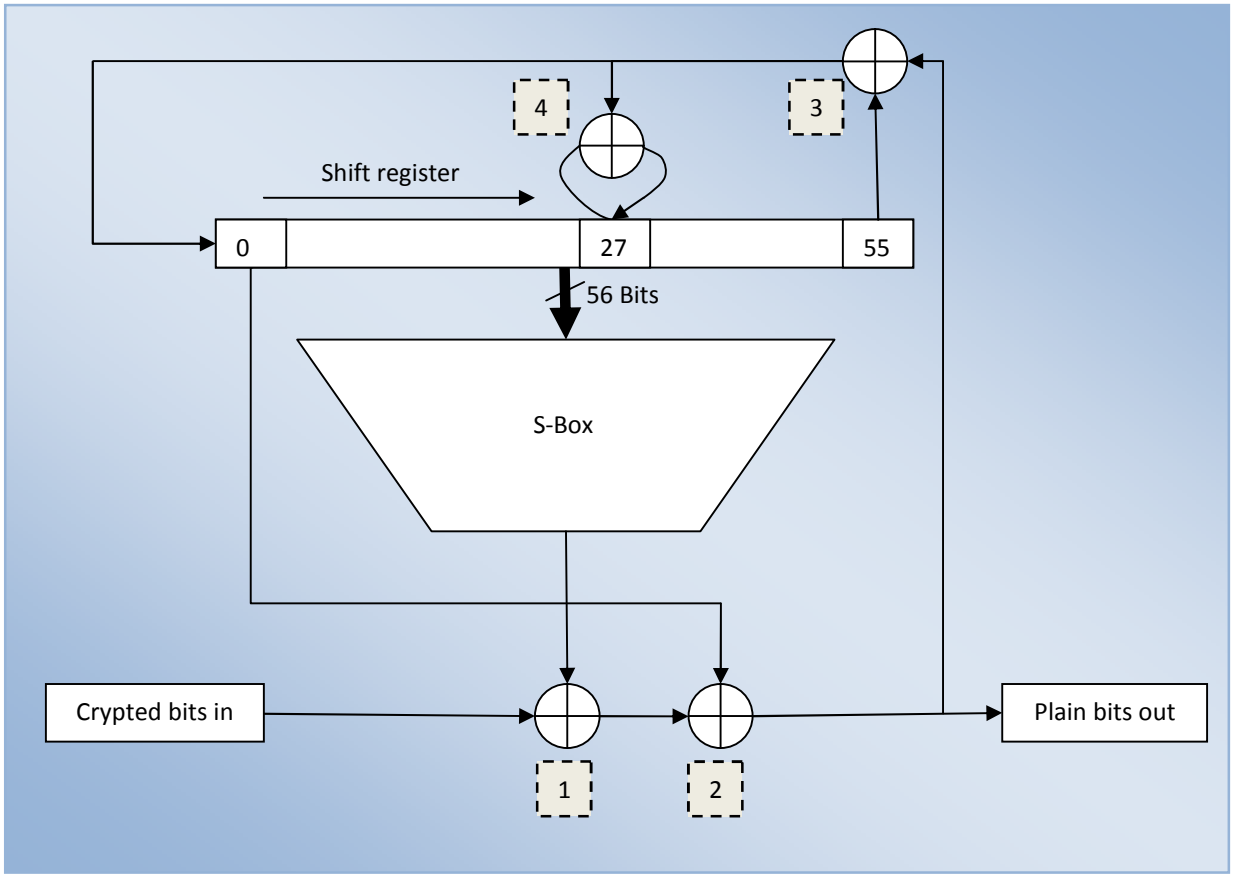

ABBILDUNG 2: STREAM CIPHER

Zur Initialisierung wird der Schlüssel (also ECM oder EMM-Key) in das 56 Bit lange Schieberegister geladen.

Für jedes zu entschlüsselndes Bit wird folgendes gemacht:

Die 56 Bits des Schieberegisters werden durch die S-Box auf 1 Bit reduziert. Dieses Bit wird dann mit dem nächsten verschlüsselten Bit XOR (Nr.1) verknüpft. Das Ergebnis wird dann mit dem Bit 0 des Schieberegisters XOR (Nr.2) verknüpft. Das Resultat ist das entschlüsselte Bit. Dieses Bit wird dann mit dem Bit 55 des Schieberegisters XOR (Nr.3) verknüpft. Das Ergebnis wird dann mit Bit 27 des Schieberegisters XOR (Nr.4) verknüpft und das Resultat wieder an die gleiche Stelle ins Schieberegister zurückgeschrieben. Zuletzt wird dann noch das Schieberegister um eine Stelle nach rechts geschoben. Der alte Inhalt von Bit 55 geht dabei verloren und ins Bit 0 wird das Resultat der XOR (Nr.3) Verknüpfung geschrieben.

Wie folgendes Bild zeigt ist die große S-Box die 56 Bits in 1 Bit umwandelt aus Speicherplatzgründen aus kleineren S-Boxen (mit unterschiedlichen Inhalten) aufgebaut.

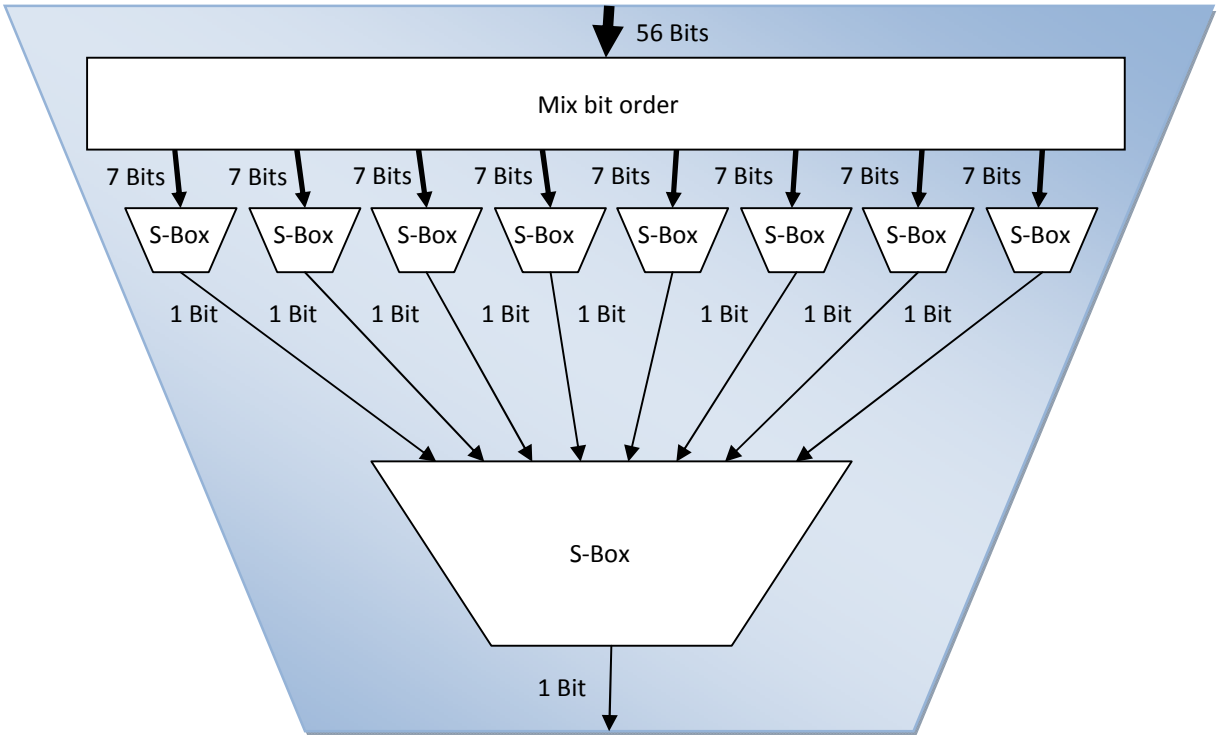

ABBILDUNG 3: AUFBAU DER GROSSEN S-BOX

Die S-Boxen die 7 Bits in 1 Bit umwandeln benötigen nur jeweils 16 Bytes. 4 Eingangsbits bestimmen den Index des 16 Byte Arrays und die verbleibenden 3 Eingangsbits bestimmen das Bit des Ergebnisbytes das verwendet werden soll.

Die eine S-Box die 8 Bits in 1 Bit umwandelt benötigt 32 Bytes. 5 Eingangsbits bestimmen den Index des 32 Byte Arrays und die verbleibenden 3 Eingangsbits bestimmen das Bit des Ergebnisbytes das verwendet werden soll.

ECM Entschlüsselung

Die folgende Beschreibung setzt Kenntnisse des oben erwähnten PowerVuSecrets.PDF voraus.

Der Receiver teilt das ECM in zwei Teile auf. Den ersten Teil sendet er über das ISE Kommando 0 an das ISE. Das ISE gibt darauf das Base CW zurück. Dann sendet der Receiver den zweiten Teil über das Kommando 1 an das ISE. Das gibt darauf hin die vom aktuellen TV-Programm benutzten Seed Werte zurück. Möglich sind dabei Seeds für Video, Audio1, HSD, Audio2, Utility und VBI. Benötigt der Receiver noch Seeds für Audio 3 und Audio 4 kann er sie vom ISE mit einem leeren Kommando 8 anfordern.

Der Receiver verknüpft dann das Base CW mit dem Video Seed um den DES-Schlüssel für die Video-Entschlüsselung (oder Base CW mit Audio 1 Seed um den DES-Schlüssel für die Audio 1-Entschlüsselung) zu bekommen.

Kommando 0 (Get Base CW)

Der ECM Teil im Kommando 0 ist 10h Bytes lang.

Die ersten beiden Bytes ECM[0..1] sind unverschlüsselt. Der Rest ECM[2..0xF] ist verschlüsselt.

Die unverschlüsselten Bytes enthalten Infos mit welchem Schlüssel das ECM entschlüsselt werden muss.

Das kann entweder ein statischer (nicht über ein EMM updatebarer) ROM/EEPROM Schlüssel oder ein EVEN/ODD Schlüssel sein der über ein EMM geändert werden kann.

Wird aktuell der EVEN-Key verwendet und es soll auf einen neuen Schlüssel gewechselt werden dann wird über ein EMM der ODD-Key geändert. Nach ein paar Wochen sollte dann jedes ISE den neuen Schlüssel bekommen haben. Dann werden die ECMs statt mit dem EVEN-Key mit dem ODD-Key verschlüsselt. Bei AFN wird von den 4 vom ISE unterstützen „Bouquet IDs“ (BIDs) anscheinend nur BID 0 verwendet.

Bei AFN habe ich nur die beiden Werte A0 00 (EVEN-Key BID 0) und B0 00 (ODD-Key BID 0) gesehen.

ECM[0]								ECM[1]								ECM-Schlüssel	
Bit								Bit									
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Typ	Adresse
		0	0													ROM-Key	ROM:4094
		0	1													EEPROM-Key	EE:10A4
		1	**0**											**0**	**0**	**EVEN-Key (BID 0)**	**EE:057C**
		1	0											0	1	EVEN-Key (BID 1)	EE:05E8
		1	0											1	0	EVEN-Key (BID 2)	EE:0654
		1	0											1	1	EVEN-Key (BID 3)	EE:06C0
		1	**1**											**0**	**0**	**ODD-Key (BID 0)**	**EE:0584**
		1	1											0	1	ODD-Key (BID 1)	EE:05F0
		1	1											1	0	ODD-Key (BID 2)	EE:065C
		1	1											1	1	ODD-Key (BID 3)	EE:06C8

TABELLE 1: WELCHER SCHLÜSSELTYP VERWENDET WERDEN SOLL KANN DAS ISE AN EINIGEN BITS VOM ECM[0..1] ERKENNEN

ECM[2..5]	ECM[6]	ECM[7]	ECM[8..E]	ECM[F]
SeedBase	Byte muss mit dem nächsten Kommando 0 ECM[5] identisch sein	Signaturbyte muss mit ECM[0] identisch sein	Base CW	Signaturbyte muss mit ECM[8] identisch sein

TABELLE 2: NACH DER ENTSCHLÜSSELUNG

Nach der Entschlüsselung prüft das ISE ob die Signatur passt. ECM[7] und ECM[0] müssen identisch sein. ECM[F] und ECM[8] müssen identisch sein. Passt die Signatur nicht, dann wird kein Base CW zurückgegeben.

Außerdem merkt sich das ISE das ECM[6] Byte. Kommt eine neues Kommando 0 wird das aktuelle ECM[5] Byte mit dem vorherigen ECM[6] Byte verglichen. Sind sie identisch dann wurde die Reihenfolge der ECMs nicht verändert und es wurde auch kein ECM ausgelassen.

ECM[2..5] wird für die Seed-Berechnung im Kommando 1 benötigt.

Der Zustand (die 56 Bits) des Schieberegisters **nach** dem Entschlüsseln des ECMs wird gesichert und später vor dem Entschlüsselungsvorgang im Kommando 1 und 8 jeweils wiederhergestellt.

KOMMANDO 1 (GET SEEDS)

Der ECM Teil im Kommando 1 ist 1Fh Bytes lang.

Die ersten beiden Bytes ECM[0..1] die die Channel-ID enthalten sind unverschlüsselt. Der Rest ECM[2..1F] ist verschlüsselt.

Für die Entschlüsselung des Kommando 1 ECMs wird der Zustand des Schieberegisters **nicht** mit dem ECM-Key initialisiert, sondern der Zustand verwendet wie er nach der Entschlüsselung des ECMs beim Kommando 0 war.

Nach der Entschlüsselung prüft das ISE ob die Signatur passt. ECM[1Dh] und ECM[0] müssen identisch sein. ECM[1Eh] und ECM[1] müssen identisch sein. Falls die Signatur nicht passt werden keine Seeds zurückgegeben.

Zur Berechnung eines Seeds das fügt das ISE zwei Teile zusammen.

Der linke Teil ist der IV Wert. Er hat immer eine feste Länge von Ah Bits (ggf. links mit 0 auffüllen) und er lässt sich wie in der folgenden Tabelle dargestellt berechnen.

IV Typ	IV Wert (Ah Bits lang)
Video	((ECM[10h] & 1Fh) <<3) | 0
Audio 1	((ECM[11h] & 3Fh) <<3) | 1
HSD	((ECM[12h] & 1Fh) <<3) | 2
Audio 2	((ECM[13h] & 3Fh) <<3) | 1
Util	((ECM[14h] & 0Fh) <<3) | 4
VBI	((ECM[15h] & F8h)>>3)<<3) | 5
Audio 3	((ECM[19h] & 3Fh) <<3) | 1
Audio 4	((ECM[1Ah] & 3Fh) <<3) | 1

TABELLE 3: BERECHNUNG DER IV WERTE

Der rechte Teil ist die SeedBase vom Kommando 0 (4 Bytes bzw. 32 Bits lang).

Folgende Abbildung zeigt wie die verschiedenen Seeds berechnet werden.

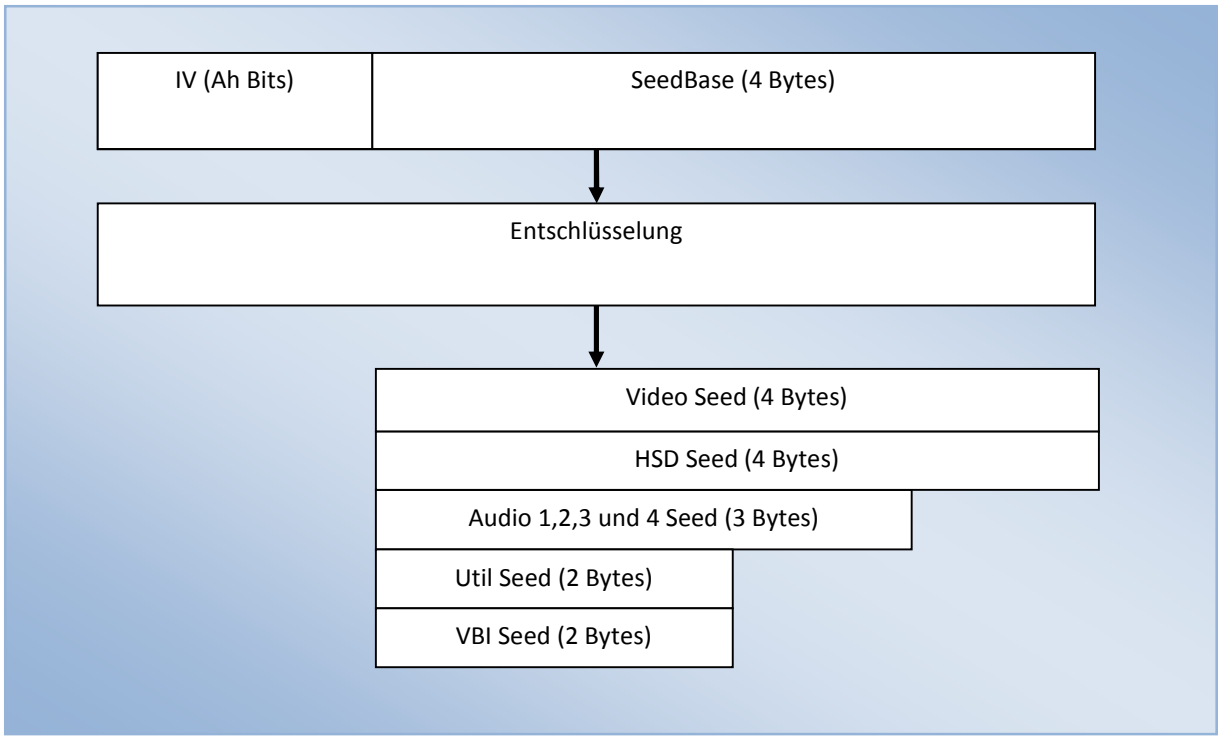

ABBILDUNG 4: ENTSCHLÜSSELUNG DER SEEDS

EMM Entschlüsselung

Der EMM Teil im Kommando 3 ist 1Bh Bytes lang.

Das erste Byte EMM[0] ist unverschlüsselt. Der Rest EMM[1..1Ah] ist verschlüsselt.

Die BID berechnet sich aus (EMM[0] & 0x0F). Die BID gibt an welcher der 4 EMM Schlüssel für die Entschlüsselung verwendet werden soll. Bei AFN wird anscheinend immer nur BID 0 verwendet.

BID Nr.	EMM-Schlüsseladresse im ISE
0	EE:0544

1	EE:0552
2	EE:0560
3	EE:056E

Nach der Entschlüsselung wird die Signatur geprüft. EMM[18h..1Ah] muss identisch mit den letzten 3 Bytes der ISE-Seriennummer (siehe Kommando 20h) sein. Passt die Signatur nicht wird das EMM ignoriert.

Der EMM-Typ entspricht (EMM[2] & 7Fh). Er gibt an welche Informationen im EMM enthalten sind.

EMM-Typ	Enthält
0	Even Tiers und Even ECM-Key
1	Odd Tiers und Odd ECM-Key
2	Even Extended Tiers und Even Blackout codes
3	Odd Extended Tiers und Odd Blackout codes
6	Blackout codes, Location und LatComp

Kommando 3 schreibt nur einen Teil sofort ins EEPROM. Der andere Teil wird im RAM zwischengespeichert.

Mit dem anschließenden Kommando 4 wird dann der zwischengespeicherte Teil ebenfalls ins EEPROM geschrieben.

Vermutlich darf ein Kommando nicht beliebig lange dauern, sondern das ISE muss innerhalb einer kurzen Zeit Antworten. Max. ein 4 Byte Block kann gleichzeitig gelöscht (alle Bits auf 0 setzen) oder programmiert (einige Bits auf 1 setzen) werden. Nach dem Löschen oder Programmieren muss laut Datasheet jeweils 10 ms gewartet werden. Durch das Aufteilen der EEPROM Updates auf zwei Kommandos 3 & 4 kann ein Kommando schneller abgeschlossen werden.

SCHWACHSTELLEN

Im Folgenden sind zwei Angriffe beschrieben die jeder für sich genommen zwar kein Sicherheitsproblem darstellt, jedoch kombiniert das Ermitteln des ECM-Schlüssel ermöglichen.

TIMING ATTACK

Eine Schwachstelle beim ISE ist dass die Zeit die es benötigt um ein ECM zu entschlüsseln nicht konstant ist, sondern neben dem verschlüsselten ECM auch vom ECM-Schlüssel abhängt. Um die Schwachstelle auszunutzen benötigt man z.B. eine elektronische Schaltung die die Anzahl der Taktimpulse zwischen dem letzen Byte des Kommandos und dem Startbit des ersten Bytes der Antwort ermitteln kann.

Im Stream cipher wird z.B. folgende Befehlsfolge verwendet den Zustand von Bit 0 von Byte 51h auf Bit 2 von Byte 5Ch zu kopieren.

```
EE:07EF          brclr  0, byte_51, loc_7F4
EE:07F2          ora    #4          ; benötigt 2 Takte zusätzlich
EE:07F4 loc_7F4:
EE:07F4          sta    byte_5C
```

Nur wenn das Bit gesetzt ist wird zusätzlich der Befehl “ora #4” ausgeführt und die Verarbeitung benötigt somit 2 Takte länger als wenn das Bit gelöscht ist. Folgende Tabelle zeigt welches gesetzte Bit im Stream cipher welche Verzögerung um x Takte verursacht.

Welches Bit	Taktverzögerung wenn Bit gesetzt ist
Ergebnis der XOR Nr.3 Verknüpfung	11
Jedes der 56 Bits im Schieberegister	2
Jedes Eingangsbit der 8zu1 S-Box	5
Ausgangsbit der 8zu1 S-Box	2
Crypted Eingangsbit	6

Key Change Interruption Attack

Der Algorithmus ist bekannt. Bei einem freigeschaltenen ISE kann man sich auch ein ECM über Kommando 0 zu einem BaseCW entschlüsseln lassen.

Nur theoretisch kann man jetzt am Computer alle möglichen ECM-Keys durchprobieren bis das gleiche BaseCW zurückgegeben wird.

Angenommen ein Computer kann 65536 Keys (also 16 Bits) pro 18 Sekunden durchprobieren.

Dann würde er für 56 Bits in der Praxis zu lange brauchen (siehe Tabelle), da jedes Bit mehr die benötigte Zeit verdoppelt.

Schlüssellänge in Bits	Benötigte Zeit
16	18 Sekunden
17	36 Sekunden
18	72 Sekunden
24	76 Minuten
32	14 Tage
40	10 Jahre
48	2.451 Jahre
54	156.894 Jahre
55	313.787 Jahre
56	627.575 Jahre

Die Key Change Interruption Attack ermöglicht es statt einen langen 56 Bit Key (=627.575 Jahre) durchprobieren zu müssen das paktisch nicht möglich ist, nur jeweils einen 32 Bit Key (=14 Tage) und einen 24 Bit Key (=76 Minuten) durchprobieren zu müssen.

Wenn ein neuer ECM-Key der 7 Byte (56 Bits) lang ist in das EEPROM geschrieben wird, dann werden die 7 Bytes nicht gleichzeitig programmiert. Ins ISE-EEPROM kann man max. 4 Byte gleichzeitig schreiben. Es werden also statt 7 Bytes, erst 4 Bytes und dann nochmal 3 Bytes geschrieben. Die Bits im EEPROM lassen sich außerdem nur setzen, sind also nur von 0 auf 1 programmierbar. Hat ein EEPROM Byte z.B. den Wert 30h kann man es direkt auf 35h ändern, aber nicht auf 05h. Das ISE vergleicht also erst mal ob auch Bits gelöscht werden müssen, falls ja dann löscht das ISE erst einen EEPROM Block und programmiert dann den neuen Wert, falls nein dann entfällt der Löschbefehl und der neue Wert wird direkt programmiert.

In der Praxis sind alter und neuer Schlüssel total unterschiedlich, es wird also immer auch ein Löschbefehl verwendet werden.

Folgendes Bild zeigt wie sich intern der Schlüssel stufenweise ändert.

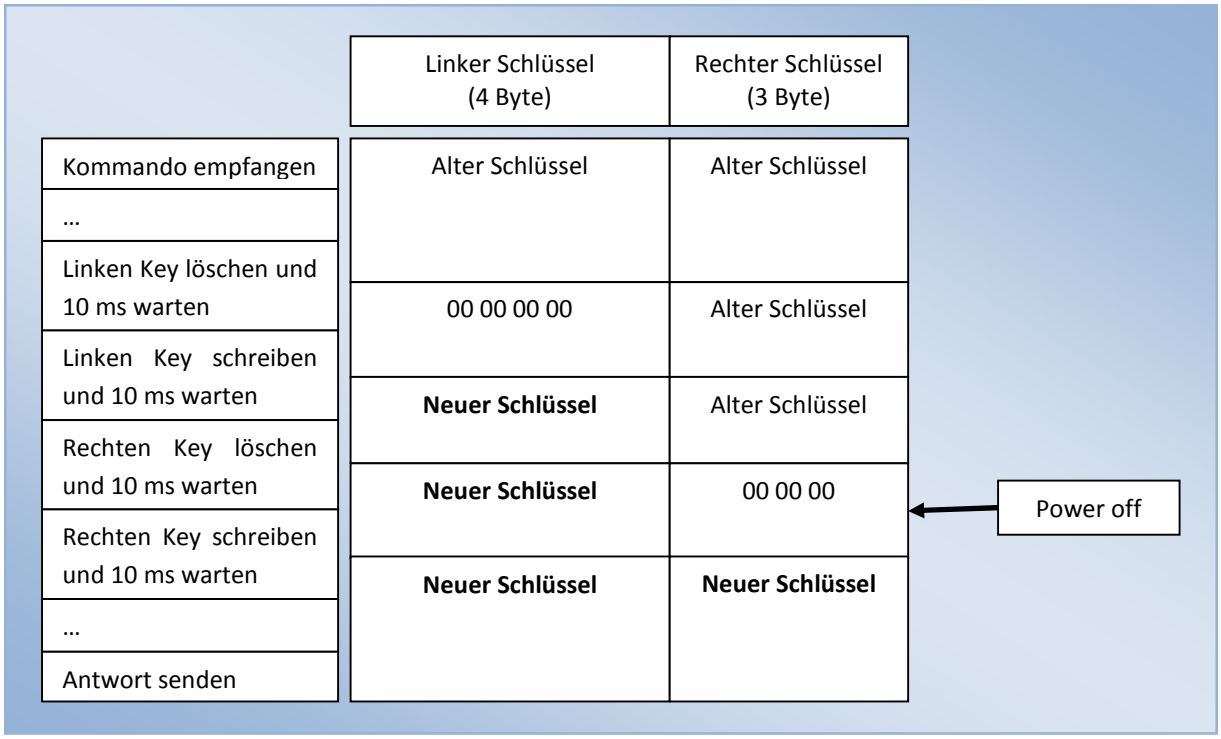

ABBILDUNG 5: MÖGLICHER ZEITPUNKT FÜR DIE "KEY CHANGE INTERRUPTION ATTACK" UM DEN LINKEN TEIL DES NEUEN SCHLÜSSELS ZU ERMITTELN

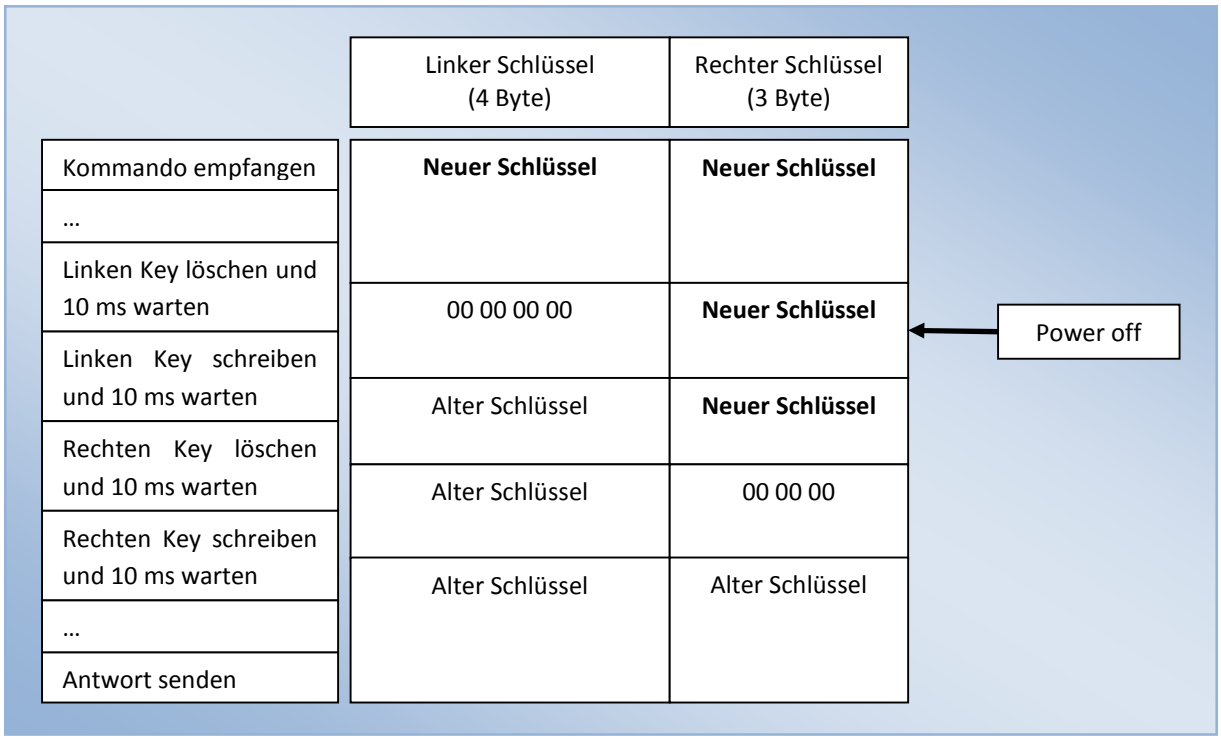

ABBILDUNG 6: MÖGLICHER ZEITPUNKT FÜR DIE "KEY CHANGE INTERRUPTION ATTACK" UM DEN RECHTEN TEIL DES NEUEN SCHLÜSSELS ZU ERMITTELN (DAZU WIRD DER KEY VON NEU NACH ALT GEWECHSELT)

Unterbricht man die Versorgungsspannung an der in Abbildung 5 gezeigten Stelle, besteht der Schlüssel nicht mehr aus 56 unbekannten Bits (was das Ermitteln in der Praxis unmöglich macht), sondern nur noch aus 32 unbekannten Bits (die rechten 24 Bits sind gezielt auf 0 geändert worden).

Für diesen Angriff muss man 2 EMM aufgezeichnet haben. Man kann dazu entweder direkt am ISE mit einem PC-Programm die übertragenen Kommandos über die serielle Schnittstelle aufzeichnen oder man kann die EMMs mit einer Satellitenempfangskarte (DVBs-Karte) am PC aufzeichnen. Direkt am ISE wird man nur die EMMs abgreifen können die auch für diese ISE Seriennummer bestimmt sind. Bei einer DVBs-Karte wird man dagegen Updates für alle ISE sehen. Diese werden in einer Endlosscheife ausgestrahlt. Ich habe dazu immer wieder mal z.B. 1 Stunde die EMMs aufgezeichnet und gleich anschießend auch 1 Minute den kompletten Transponder (also die Video und Audio Streams der PowerVu Programme und auch die zugehörigen ECMs pro Programm) um Testdaten zur Verfügung zu haben.

Man benötigt also ein EMM das den zu ermittelnden Schlüssel enthält (hier als neuer Schlüssel bezeichnet) und ein weiteres EMM das einen andern Schlüssel enthält (hier als alter Schlüssel bezeichnet), das kann tatsächlich ein alter Schlüssel sein, aber auch ein ganz neuer ungültiger Schlüssel sein der bei PowerVu ausgesendet wird wenn ein Abo ausgelaufen ist. Ich hatte mit dem EMM loggen per DVBs-Karte angefangen bevor ich mir einen abgelaufenen nicht mehr freigeschaltenen PowerVu Receiver gebraucht gekauft habe. Ich habe also ein altes EMM (incl. Testdaten mit ECM, Video und Audio) und ein aktuelles EMM das einen ungültigen Schlüssel enthält. Ich kann mit dem ermittelten ehemaligen EMM Schlüssel nur den damaligen kurzen mit aufgezeichneten Videostream entschlüsseln, aber keine aktuellen Programme. Wichtig ist dass diese beiden unterschiedlichen EMMs vom gleichen Schlüsseltyp sind (also beide EVEN-Key oder beide ODD-Key) und somit im gleichen EEPROM Bereich gespeichert werden.

Man kann also jetzt beliebig zwischen alten und neuen Schlüssel wechseln und wie folgt einen Schlüsselteil des neuen Schlüssels auf 0 ändern. Zum Abschalten braucht man eine Elektronik die in der Lage ist nach x Takten oder nach x ms nach dem letzten Kommando den VCC Pin vom ISE auf 0 setzt.

Rechten Schlüsselteil auf 0 setzen (linker Teil enthält neuen Schlüssel):

- altes EMM schicken
- neues EMM schicken und nach dem Löschen der rechten Seite die Spannung ausschalten

Linken Schlüsselteil auf 0 setzen (rechter Teil enthält neuen Schlüssel):

- neues EMM schicken
- altes EMM schicken und nach dem Löschen der linken Seite die Spannung ausschalten

Doch wie ermittelt man den richtigen Abschaltzeitpunkt? Man kann ja von außen einzelnen Phasen der schrittweisen Schlüsseländerung nicht sehen.

Jetzt kommt die zuvor beschriebene Timing Attack ins Spiel. Mit ihr lässt sich feststellen ob sich der Schlüssel nach dem Unterbrechen der Spannung während der Schlüsseländerung geändert hat. Mit unterschiedlichen ECM Schlüsseln dauert ja die ECM-Entschlüsselung (Kommando 0) unterschiedlich lange.

Man kann also für das Bestimmen des linken Schlüsselteils in einer Schleife ein altes EMM schicken und dann das neue EMM[1] (zum Ermitteln des rechten Schlüsselteils erst neues EMM dann Altes) nach x ms abbrechen (x erhöht man bei jedem Schleifendurchgang um 1 ms. Bei x = 1000 ms beendet man die Schleife). Nach jedem Abbruch des EMMs schickt man ein ECM mit immer gleichen Inhalts und bestimmt über die Timing Attack die Bearbeitungszeit. Nur auf die beiden ECM Header Bytes muss man achten (A000 für EVEN-Key bzw. B000 für ODD-Key), als folgende verschlüsselte ECM Daten kann man z.B. lauter Nullen verwenden.

[1] Ein EMM wird über Kommando 3 zum ISE geschickt dabei wird ein Teil der Daten wie Tiers bereits ins EEPROM geschrieben. Das Kommando 4 muss immer anschließend folgen dabei wird der ECM Schüssel ins EEPROM geschrieben. Die erwähnte Unterbrechung erfolgt immer im Kommando 4.

Beim 1ms Raster braucht man 1000 Versuche. Um die EEPROM Umprogrammierungen möglichst gering zu halten kann man zuerst mit einem 100 ms Raster probieren. Die (3 * 10 ms) Zwischenzustände liegen eng beieinander, dagegen vergeht vorne und vor allem hinten raus viel mehr Zeit ohne das sich der Schlüssel ändert. Die Zeit die vor der Schlüsseländerung vergeht ist bei jedem ISE anders, da jedes ISE individuelle EMM Schlüssel hat und die Zeit die die EMM Entschlüsselung benötigt vom Schlüssel abhängt.

Die folgende Tabelle enthält fiktive Beispielwerte. Die ECM Taktanzahl Wert bei 0 ms entspricht immer dem alten Schlüssel (da bei 0 ms noch keine Schlüsseländerung stattgefunden haben kann). Man sieht das bei 100 ms und auch bei 200 ms die gleiche Taktanzahl ermittelt wurde, somit ist bei 200 ms immer noch der alte Schlüssel aktiv. Die ECM Taktanzahl bei 1000 ms entspricht immer dem neuen Schlüssel (da es ausreichend Zeit war die Schlüsseländerung komplett durchzuführen). Man sieht das bei 900 ms, 800 ms, …, 300 ms die gleiche Taktanzahl ermittelt wurde, somit ist bei 300 ms bereits erstmals der neue Schlüssel aktiv. Die Schlüsseländerung hat also zwischen 200 ms und 300 ms stattgefunden. Nur diesen Bereich braucht man jetzt noch mit 1 ms Schritten feinabtasten um die 3 etwa 10 ms grossen Zwischenblöcke zu finden. Man notiert sich dann die Mitte von Zwischenblock 1 bzw. 3 für das spätere gezielte auf 0 setzen eines Schlüsselteils.

Zeit im ms vor der EMM Unterbrechung	Taktanzahl die die anschließende ECM Entschlüsselung benötigt	Schlüssel
0	155A7h	Alter Key
100	155A7h	Alter Key
200	155A7h	Alter Key
300	1560Fh	Neuer Key
400	1560Fh	Neuer Key
500	1560Fh	Neuer Key
600	1560Fh	Neuer Key
700	1560Fh	Neuer Key
800	1560Fh	Neuer Key
900	1560Fh	Neuer Key
1000	1560Fh	Neuer Key

ABBILDUNG 7: FIKTIVE BEISPIELWERTE (MS UND TAKTANZAHL)

Die Zeit für die EMM Entschlüsselung ist nicht nur vom ISE individuellen EMM Schlüssel, sondern auch vom verschlüsselten EMM selbst abhängig. Man muss die Tabellen also zweimal ermitteln. Einmal für den Wechsel vom alten EMM aufs Neue und einmal für den Wechsel vom neuen EMM aufs Alte um später beide Schlüsselteile ermitteln zu können.

Wir sind jetzt soweit vom neuen Schlüssel den linken 4 Byte Schlüsselteil oder den rechten 3 Byte Schlüsselteil gezielt auf Null setzten zu können.

Jetzt portiert man den PowerVu Algorithmus auf den PC.

Man schreibt sich eine Funktion die die ECM Entschlüsselung von Kommando 0 (Get BaseCW) beherrscht.

Die Funktion soll als Input folgende Daten übernehmen:

- einen 7 Byte langen ECM-Key
- das verschlüsselte ECM

Die Funktion soll als Output folgende Daten zur Verfügung stellen:

- das entschlüsselte ECM
- eine relative Taktverzögerung

Wie unter Timing Attack beschrieben ist die Anzahl der Takte die die ECM Entschlüsselung benötigt abhängig vom verschlüsselten ECM und dem ECM Key. Diese Taktverzögerung lässt sich auch in einer Variable in unserer Funktion emulieren. Pro zu entschlüsselnden Bit werden die zusätzlichen Takte die ein gesetztes Bit an bestimmten Stellen verursacht aufsummiert. Die Stellen und die zusätzlichen Takte sind unter Timing Attack in der Tabelle dargestellt.

Jetzt erzeugt man ein ECM mit dem gewünschten EVEN oder ODD Header und zufälligen (nicht nur Nullen) Inhalt z.B.

A0 00 49 DC 56 5D D1 2C F0 E5 96 44 CA 58 7A 8A

Dann lässt man sich man sich von der Funktion die relative Taktverzögerung für dieses ECM und den Schlüssel „00 00 00 00 00 00 00“ berechnen.

Jetzt benötigen wir noch die Taktanzahl die das ISE für dasselbe ECM und ebenfalls den Schlüssel „00 00 00 00 00 00 00“ braucht. Dazu müssen wir beide Schlüsselteile auf 0 setzen.

Zuerst die rechte Seite auf 0 setzen:

- Altes EMM senden
- Neues EMM senden und unterbrechen wenn die linke Seite den neuen Schlüssel hat und die rechte Seite „00 00 00„ ist

Dann die linke Seite auf 0 setzen:

- Altes EMM senden und unterbrechen wenn die linke Seite „00 00 00 00„ ist

Jetzt ist der Schlüssel komplett 0 und die Taktanzahl die das ISE für dasselbe ECM benötigt kann ermittelt werden.

Jetzt kann die Differenz zwischen der absoluten Taktanzahl die das ISE gebraucht hat und der relativen Taktanzahl die unsere Funktion ermittelt hat berechnet werden. Die Differenz zieht man ab jetzt immer automatisch von der vom ISE ermittelten Taktanzahl ab.

Den Vorgang wiederholt man mit ein paar anderen zufälligen Werten für das EMM und lässt es wieder vom ISE und der Funktion mit den Schlüssel „00 00 00 00 00 00 00“ entschlüsseln. Die Taktanzahl vom ISE und der Funktion muss identisch sein.

Jetzt geht’s ans Bestimmen der beiden Schlüsselteile.

Dazu ändert man den Schlüssel im ISE so das der linke Schlüssel „00 00 00 00“ ist und der rechte Schlüsselteil dem neuen Schlüssel entspricht.

Wir schicken dann ein zufälliges ECM ans ISE und merken uns die Taktanzahl.

Dann entschlüsseln wir mit unserer Funktion das gleiche ECM mit allen in Frage kommenden Schlüsseln:

- 00 00 00 00 00 00 00
- 00 00 00 00 00 00 01
- 00 00 00 00 00 00 02
- 00 00 00 00 00 00 03
- …
- 00 00 00 00 FF FF FC
- 00 00 00 00 FF FF FD

- 00 00 00 00 FF FF FE
- 00 00 00 00 FF FF FF

Die linke 4 Byte lange Seite ist dabei immer 0. Nur die Rechte variieren wir.

Immer wenn die Taktanzahl unser Funktion mit der zuvor ermittelten ISE Taktanzahl übereinstimmt schreiben wir den möglichen Schlüsselkandidaten in eine Datei.

Dann wählen wir eines neues zufälliges ECM schicken es zum ISE und notieren uns wieder die Referenztaktanzahl. Dann probieren wir nicht mehr alle möglichen (256 * 256 * 256) Schlüssel, sondern nur die Schlüsselkandidaten die wir zuvor in eine Datei geschrieben haben. Nur bei den Schlüsseln die die gleiche Taktanzahl wie das ISE haben, landen in einer weiteren Schlüsselkandidatendatei. Bei jedem Durchgang nimmt die Anzahl der verbleibenden Schlüssel rapide ab. Nach ein paar Durchgängen bleibt nur noch ein Schlüssel übrig. Das ist der rechte Teil von dem gesuchten Schlüssel.

Für den linken Teil verfährt man entsprechend. Da der linke Teil aus (256 * 256 * 256 * 256) möglichen Schlüsseln besteht wird es länger als beim rechten Teil dauern.

Setzt man die beiden ermittelten Teile zusammen hat man den gesuchten Schlüssel.[2]

Als Abschlusstest kann man ein neues EMM ohne Unterbrechung ans ISE schicken und sich vom einem echten ECM über Kommando 0 das BaseCW geben lassen. Anschließend lässt man sich das ECM mit dem ermittelten Schlüssel auch von der Funktion entschlüsseln. Die beiden BaseCWs müssen identisch sein.

Theoretisch kann man anschließend sogar ein aufgezeichnetes Video einem PC Programm und dem ermittelten Schlüssel ohne dem Security Chip entschlüsseln.

[2] Generell ist ja die Timing Attack bereits seit geraumer Zeit bekannt. Uralte PayTV Smartcards brauchten für den Vergleich der Signatur im Kommando und der berechneten Signatur unterschiedlich lange. Es wurde Byte für Byte verglichen und bei dem ersten Unterschied das Kommando abgebrochen. So konnte man Stück für Stück die richtigen Signaturbytes bestimmen und das Kommando wurde letztendlich von der Smartcard angenommen und ausgeführt.
Bei der Key Change Interruption Attack was es anders, die kannte ich noch nicht und musste sie erst selbst entwickeln. Falls jemand sie bereits aus dem Internet kannte (ggf. auch unter einem anderen Namen) würde ich mich über einen Link freuen.

EMPFEHLUNGEN

Um das Ermitteln des Schüssels über den hier beschriebenen Weg zu verhindern genügt es eigentlich nur eine der beiden Schwachstellen zu beheben, trotzdem sollte man natürlich beide beheben.

Um die Timing Attack ins Leere laufen zu lassen muss der Entschlüsselungsalgorithmus immer die gleiche Zeit benötigen.

Benötigt man aber if Anweisungen wie folgende

```
if(variable1 == 0)
{
  variable2 = 5;
}
```

Dann sollte man einen else Zweig verwenden der exakt die gleiche Verarbeitungszeit benötigt wie der if Zweig.

```
if(variable1 == 0)
{
  variable2 = 5;
}
else
{
  dummy_variable = 0;
}
```

Um die Key Change Interruption Attack ins leere Laufen zu lassen könnte man z.B. folgendes tun:

Im EEPROM ein Flag „KeyIsValid“ mitführen. Vor Schlüsselupdates wird es auf false gesetzt. Nach dem Schlüsselupdate wieder auf true.

```
KeyIsValid, false
Schlüssel löschen und schreiben
KeyIsValid, true
```

Vor jeder Benutzung des Schlüssels wird dann das Flag z.B. wie folgt ausgewertet:

```
If(KeyIsValid)
{
   //Entschlüsselung durchführen
}
else
{
   //Kommando mit Fehler abbrechen
}
```

Falls Cinosana das ISE wirklich mit Hilfe von einem Glitch ausgelesen hat, sollte man das ISE natürlich auch resistent dagegen machen. Also wenn Versorgungsspannung oder Takt außerhalb des erlaubten Bereichs sind, dann intern einen Reset auslösen.