Ruhr-Universität Bochum
Lehrstuhl Kommunikationssicherheit
Seminar: IT-Sicherheit
Wintersemester 2002/2003

COMP128 – Kollisionsattacke

Von: Marc Dingfelder
Matrikel-Nr.: 108097232769
Betreut durch: Thomas Wollinger

Inhaltsverzeichnis

1 Einleitung
1.1 Geschichtlicher Hintergrund
1.2 Motivation

2 Der Algorithmus COMP128
2.1 Authentifikationsablauf im GSM-Netz
2.2 Aufbau des COMP128
2.2.1 Ein- und Ausgabeparameter des COMP128
2.2.2 Rundenstruktur
2.2.3 Hashing-Algorithmus / Butterfly-Kompression
2.2.4 Permutation
2.2.5 Berechnungsergebnis des COMP128

3 Kollisionsattacke auf COMP128
3.1 Konstruktion der Attacke
3.1.1 Kollisionen
3.1.2 Auftreten der Kollision beim COMP128-Algorithmus
3.1.3 1. Konstruktion der Kollision
3.1.4 2. Konstruktion von K_i
3.2 Stochastische Untersuchung
3.3 Simulation der Kollisionsattacke
3.4 Alternative Attacken

4 Schlussbetrachtung

5 Literaturverzeichnis

6 Anhang
6.1 Quelltext des Simulationsprogramms
6.2 Herleitung der Geburtstagsproblem-Formel

Abbildungsverzeichnis

Abbildung 1: Anzahl und Wachstum der GSM-Teilnehmer 1992-2002.....04
Abbildung 2: Schematischer Aufbau des GSM-Netzes.....06
Abbildung 3: Authentifikationsablauf zwischen SIM-Karte und Basisstation.....07
Abbildung 4: Ein-/Ausgabeparamter des COMP128-Algorithmus.....08
Abbildung 5: Ablaufdiagramm der COMP128-Rundenstruktur..... 08
Abbildung 6: Schema Butterfly-Kompression.....10
Abbildung 7: Kollision beim Abbilden.....12
Abbildung 8: 1. Konstruktion der Kollision..... 13
Abbildung 9: 2. Konstruktion von K_i..... 14

Tabellenverzeichnis

Tabelle 1: Struktur der Tabellen T_i.....09

Abkürzungsverzeichnis

AC	: Authentication Center (verwaltet K_i, generiert RAND, SRES und K_c für VLR)
BS	: Base Station (hier geht Anruf zuerst ein, leitet an MSC weiter)
CEPT	: Conference Européenne des Administration des postes et des télécommunications die Groupe Speciale Mobile
DPA	: Differential Power Analysis
EIR	: Equipment Identity Register (weitere Datenbank: gestohlene Telefone, durch Betrug aufgefallene Telefon ID-Nummern, fehlerhafte Geräte
GSM	: Global System for Mobile Communication
HLR	: Home Location Register (Datenbank, 1x pro GSM-Netzwerkbetreiber, speichert u.a aktuelle Positionen der Teilnehmer, IMSI, Rechnungs-informationen)
IMSI	: International Mobil Subscriber Identity
ISDN	: Integrated Services Digital Network
MSC	: Mobile Switching Centre (folgt nach BS)
OMC	: Operations and Maintenance Center (überwacht Netzwerk)
PIN	: Personal Identification Number
PSTN	: Public Switched Telephone Network (Festnetz)
SIM(-Karte)	: Subscriber Identifikation Module
SRES	: Signed RESponse
TMSI	: Temporary Mobil Subscriber Identity
VLR	: Visitor Location Register (versorgt HLR mit Informationen über die in der zugehörigen Funkzelle angemeldeten Teilnehmer)

1 Einleitung

1.1 Geschichtlicher Hintergrund

Vor der Einführung des GSM-Standards waren bereits analoge mobile Kommunikationslösungen im Einsatz (z.B. C-Netz im Westen Deutschlands, NMT 450 in Nordeuropa und den Benelux-Ländern,. TACS in England, Radiocom 2000 in Frankreich und RTMI/RTMS in Italien). Diese hatten allesamt den Nachteil der fehlenden Kompatibilität untereinander, so dass eine Kommunikation nach Überschreiten der eigenen Landesgrenze mit den vorhandenen Geräten nicht mehr möglich war.
Das erste Mobilfunknetz gab es in den USA seit 1946, in Deutschland wurde 1958 von der Bundespost das per Hand vermittelte A-Netz (150MHz) eingeführt. Nachteile waren der hohe Preis 8000-15000 Mark für die Geräte, blieb deswegen nur wohlbetuchten vorbehalten (ca. 10000), und der hohe Platzbedarf (halber Kofferraum). Der Betrieb wurde bis 1977 aufrecht erhalten. Das B-Netz (150MHz) folgte 1972 mit Selbstwahl, Roaming in Nachbarländer, ähnliche Kosten wie beim A-Netz, allerdings wuchsen die Benutzerzahlen bis auf 27000 und lief bis 1994. Nachteil war eine Regionenvorwahl. Das C-Netz (450MHz) wurde von 1985 – 2000 betrieben. Bundeseinheitliche Rufnummer, ein zelluläres Netz, das geringere Sendeleistungen erforderte sowie Fax- und Datenverbindungen waren Vorteile, die 803000 Benutzer zur Folge hatten. [1]
Im Jahr 1982 gründet das CEPT (Conference Européenne des Administration des postes et des télécommunications die Groupe Speciale Mobile), initiiert durch 26 Länder.
Es sollte ein einheitlicher Standard für ein europäisches Mobilkommunikationsnetz festgelegt werden. Heute wird GSM in über 100 Ländern weltweit eingesetzt. Details waren digitales Netz, 900MHz, später zusätzlich 1800MHz (Europa) und 1900MHz (USA), effizientere Nutzung der Frequenzen, höhere Übertragungssicherheit, verbesserte Sprachqualität, Kostensenkung bei den Endgeräten, ausgeweitete Infrastruktur und die vollständige Kompatibilität mit dem ISDN Netz. Nachdem Tests durchgeführt wurden, die die spätere technische Realisierung betrafen (z.B. Breit- oder Schmalband), konnte im Juli 1991 ein Testbetrieb in verschiedenen europäischen Ländern beginnen. 1992 schließlich wurde GSM offiziell eingeführt, auch in Deutschland, und 1993 gab es bereits 36 GSM-Netze in 22 Ländern [1], [2]. Bis heute hat sich die Zahl der Teilnehmer auf 824.7 Millionen Nutzer in über 194 Länder erhöht (Stand: 05.03.2003) [3].

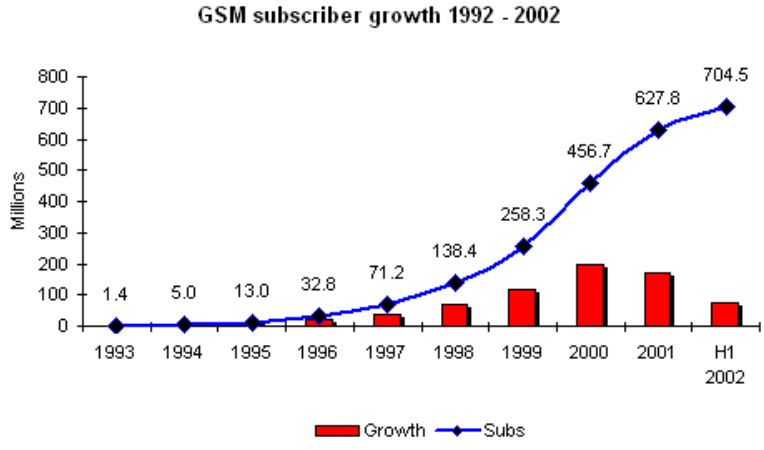

Abbildung 1: Anzahl und Wachstum der GSM-Teilnehmer 1992-2002 [4]

1.2 Motivation

GSM-Telefone haben auf Grund ihrer Handlichkeit, der erlaubten Mobilität beim Telefonieren und nicht zuletzt durch gesunkene Preise, wie bereits erwähnt, eine große Verbreitung in allen sozialen Schichten und Altersgruppen gefunden. Die dabei erzeugten und durch die Luft übertragenen Daten sind jeglichen Abhörversuchen ausgesetzt. Die überwiegende Mehrheit der Gespräche ist für andere von deutlicher Belanglosigkeit geprägt (z.B. zwischen Privatpersonen), doch werden natürlich auch geschäftliche oder nicht für Dritte gedachte Gespräche geführt. Ein mögliches Angriffsbeispiel wäre eine Ausschreibung, bei der das Wissen um die Gebote der Konkurrenz, wenn nicht von existenzieller, so doch von nicht zu vernachlässigender finanzieller Bedeutung ist. Andere Beispiele wären Wirtschafts-, Forschungsspionage oder Privatgeheimnisse von Berühmtheiten. Auch müssen die genau zugeordnete Abrechnung der Telefongespräche (A3/A8 = COMP128), die Geheimhaltung der Identität und des Aufenthaltsortes des Teilnehmers (IMSI = International Mobile Subscriber Identity, TMSI = Temporary Mobile Subscriber Identity) und Schutz gegen unbefugtes Mithören (A5) und Telefonieren auf Kosten anderer sichergestellt (A3) werden. Zur Lösung dieser Probleme sieht die GSM-Norm digitale Authentifikations- und Verschlüsselungsmethoden vor:
Die Sicherheit hängt von der Geheimhaltung der verwendeten Schlüssels ab. Der COMP128-Algorithmus, auch A3/A8 genannt, arbeitet nach dem Challenge-Response Verfahren und benutzt einen geheimen Schlüsse K_i., der auf der SIM-Karte gespeichert ist. Der Algorithmus dient der Authentifikation des Benutzers gegenüber dem Netz, nachdem sich der Benutzer mit seiner PIN (max. 8 Zeichen) gegenüber der SIM-Karte identifiziert hat. Ein Teil des Berechnungsergebnisses wird für den Session Key des A5 benutzt. Der A5 verschlüsselt die Sprach- und Steuerdaten, die über die Luftschnittstelle übertragen werden. Der möglichst seltene Gebrauch der IMSI, die bei der Authentifikation an den Provider gesendet wird, sichert die Anonymität des Benutzers und die Geheimhaltung des Aufenthaltsortes. Nach der aller ersten Anmeldung erhält die SIM-Karte eine verschlüsselt übertragene TMSI, die bei der nächsten Netzanmeldung verwendet wird. Dann wird wieder eine neue TMSI übermittelt.

Diese Arbeit soll die Schwächen des COMP128 aufzeigen, die offenbar vorhanden sind. Trotzdem wurde der Algorithmus für den höchst sensiblen Bereich der Authentifikation im GSM-Netz eingesetzt. Durch die Geheimhaltung des Algorithmus sollte die Sicherheit zusätzlich erhöht werden, dies hat sich allerdings als Trugschluss herausgestellt.

Zur Aufzeigung der Angreifbarkeit des Algorithmus, wird die Funktionsweise von diesem zuerst beschrieben und, darauf aufbauend, eine Attacke gegen den Algorithmus konstruiert und ausgeführt, um an den geheimen Schlüssel, der auf der SIM-Karte gespeichert ist, zu gelangen.

2 Der Algorithmus Comp128

2.1 Authentifikationsablauf im GSM-Netz

Nach erfolgter Benutzerauthentifikation des Nutzers gegenüber der SIM-Karte mit der PIN, muss sich die SIM-Karte gegenüber dem Netzbetreiber authentifizieren. Der Authentifikationsalgorithmus COMP128 (A3) ist deshalb auf der SIM-Karte sowie im „Authentication Center“ (AC) implementiert. Es läuft zwischen SIM-Karte und dem „Home Location Register“ (HLR)/(AC) eine Methode nach dem Challenge-Response Verfahren ab. Beim Challenge-Response-Verfahren wird eine Herausforderung (Challenge) an den zu Authentifizierenden geschickt, der ein Ergebnis (Response) aus einer Berechnung durch Verwendung eines Geheimnisses zurücksendet und vom Authentifizierer überprüft wird. Zur Sicherheit trägt bei, dass niemals der geheime Schlüssel K_i übertragen wird, sei es verschlüsselt noch im Klartext, sondern nur ein Ergebnis, zu dessen Berechnung der Schlüssel mit herangezogen wurde. Der Schlüssel besitzt eine Länge von 128 bit und ist nur der SIM-Karte und dem „Authentication Center“ (AC) bekannt. Vermittelt werden die Anfragen zwischen beiden Seiten vom „Visitor Location Register“ (VLR), das zuständig für die Funkzelle ist, in der sich der Netzteilnehmer gerade befindet.

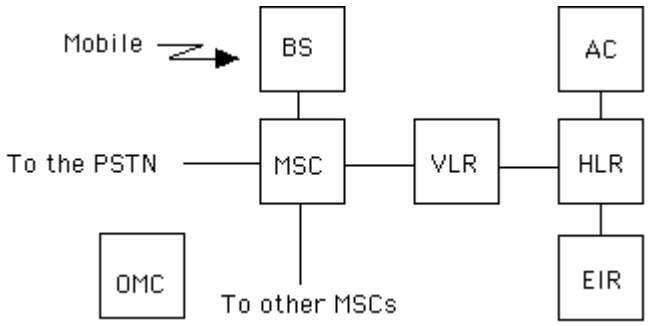

Abbildung 2: Schematischer Aufbau des GSM-Netzes [5]

Meldet sich nun ein Teilnehmer nach erfolgreicher Eingabe der PIN im Netz an, so fordert das VLR die IMSI oder TMSI von der SIM-Karte zur Identifizierung an. Anzumerken ist, dass auf der Teilnehmerseite alles auf der SIM-Karte abläuft, die selber ein eigenständiger Mikrocomputer ist, mit Mikroprozessor, der die Programme ausführt, ROM (Read-Only Memory), in dem das Betriebssystem gespeichert ist und RAM (Random Access Memory), in dem Programme und Algorithmen abgearbeitet und Werte zwischengespeichert werden. Zu der IMSI wird auf der Providerseite die dazugehörigen geheimen Daten K_i gesucht . Es wird eine nicht vorhersagbare 128 bit Zufallszahl RAND generiert und an die SIM-Karte gesendet, dies stellt die Herausforderung („Challenge“) dar. Der Algorithmus Comp128 (A3) berechnet aus dem empfangenen RAND und dem geheimen Schlüssel K_i die Antwort „Signed RESponse“ (SRES), die zurück zum VLR gesendet wird. Dort findet dann der Vergleich mit dem vom AC/HLR ebenfalls berechneten Ergebnis statt, das die gleichen Werte K_i und RAND, des sich anfangs vorgestellten Teilnehmers, zur Berechnung herangezogen hat. Wird eine Übereinstimmung zwischen den beiden SRES-Werten festegestellt, so hat sich die behauptete Identität des Teilnehmers bewahrheitet und das VLR gewährt nun Zugriff auf das Mobilfunknetzwerk, da jetzt bewiesen ist, dass der richtige Schlüssel K_i von der originalen (oder geklonten) SIM-Karte verwendet wurde.

Das Ziel der Kollisionsattacke ist die Rekonstruktion des geheimen Schlüssels K_i, der nicht einfach aus der SIM-Karte ausgelesen werden kann, wie z.B. die IMSI.

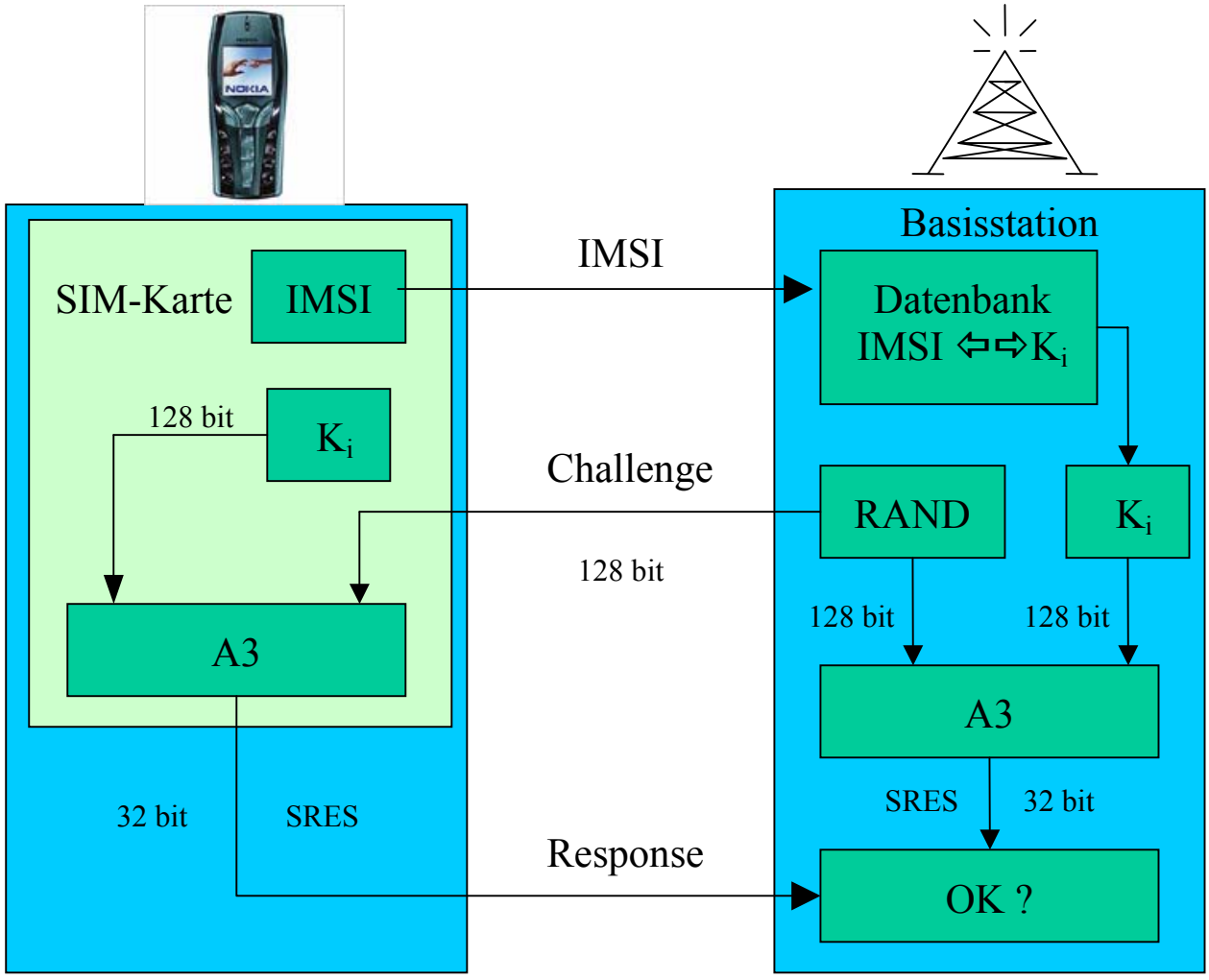

Abbildung 3: Authentifikationsablauf zwischen SIM-Karte und Basisstation [6]

2.2 Aufbau des COMP128

2.2.1 Ein- und Ausgabeparameter des COMP128

Auf der Eingabeseite sind K_i und RAND zu sehen, beide mit jeweils 128 bit, wobei K_i in die unteren 128 bit und RAND in die oberen 128 bit geladen wird. Nach der Verarbeitung durch den Comp128-Algorithmus steht ein 128 bit Ergebnis zur Ausgabe an, daher kommt die „128“ in „Comp128“. Die unteren 32 bit stellen das Ergebnis des A3-Algorithmus dar, die 54 bits von 74-127 bilden mit 10 angehängten 0-bits das Ergebnis des A8, dies ist somit 64 bit lang ist und hat die hier nicht näher untersuchte Funktion des Sitzungsschlüssels für den Sprach- und Steuerdatenverschlüsselungs-algorithmus A5. Anzumerken ist die anscheinend von den Entwicklern gewollte Schwächung des Sitzungsschlüssels um den Faktor 2^{10}, hervorgerufen durch die 10 0-bits, die nicht mehr in einer „brute-force“ Attacke bestimmt werden müssen. Da durch ein bit zwei Zustände ausgedrückt werden können (binär), multipliziert jedes einzelne 0-bit die benötigte Anzahl an Versuchen mit 2.

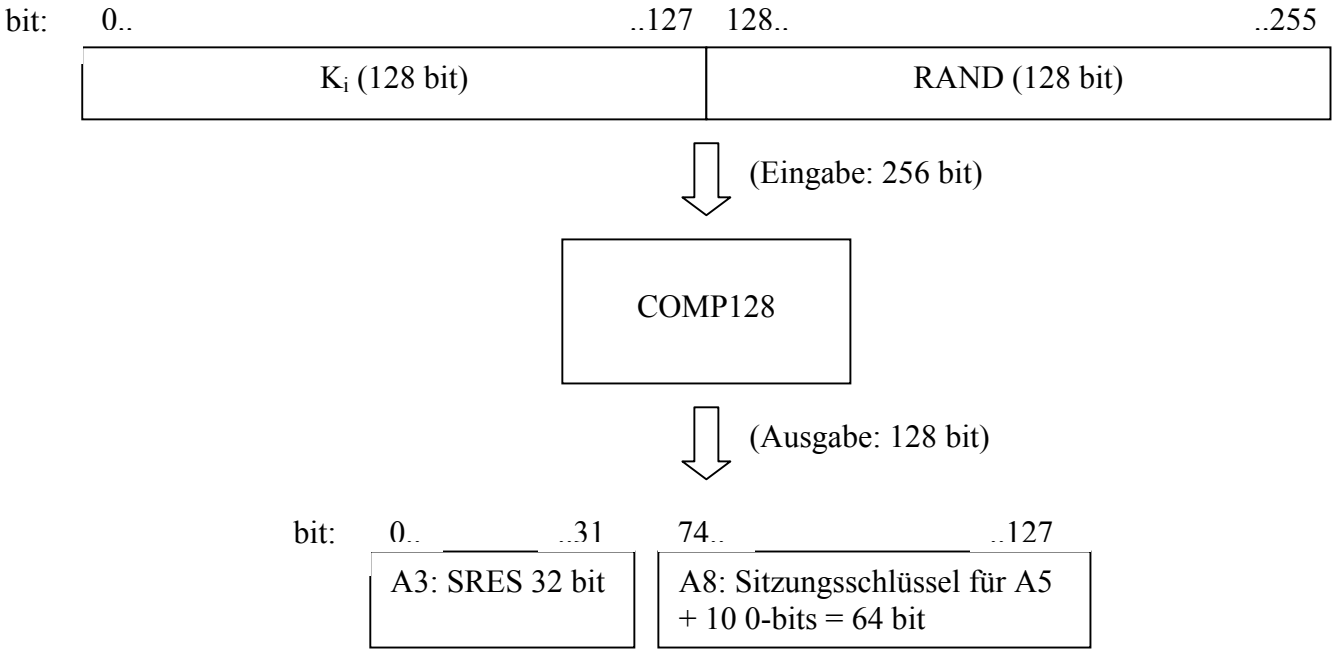

Abbildung 4: Ein-/Ausgabeparamter des COMP128-Algorithmus

2.2.2 Rundenstruktur

Der Aufbau des COMP128-Algorithmus lässt sich als rundenbasiert beschreiben. Nach dem einmaligen Verbinden von K_i und RAND in der Variable x (32 byte), wird die Hashing-Funktion 8 Mal durchlaufen. Nach jedem der 8 Aufrufe der Hashing-Funktion (bis auf den letzten) folgt ein x' mit 32 4-bit Werten (16 byte). Daran schließt ein Permutationsblock an. Vor dem Permutationsblock wird x' zu 16 8-bit Werten zusammengefasst, bei dem die Werte des x' in einer später noch beschriebenen Weise vertauscht werden. Das resultierende x'' (16 byte) muss anschließend noch um das vorangestellte K_i erweitert werden, um den erforderlichen 32 byte der Eingabeseite des Hashing-Algorithmus zu entsprechen. Nach den 8 Durchläufen, bildet das x' (16 byte = 128 bit) die Ausgabe des COMP128-Algorithmus.

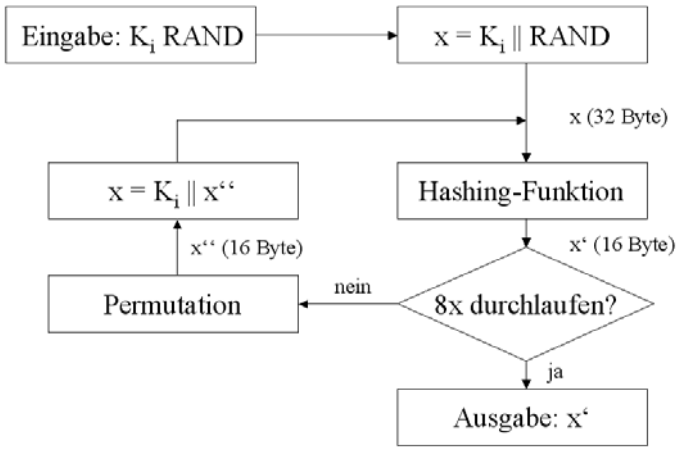

Abbildung 5: Ablaufdiagramm der COMP128-Rundenstruktur

2.2.3 Hashing-Algorithmus / Butterfly-Kompression

Wie aus dem Aufbau des COMP128-Algorithmus ersichtlich ist, besteht dieser aus einer 8-Runden-Struktur, die Hashing-Funktion selber durchläuft intern selber noch mal später erläuterte 5 Runden, im Folgenden zur besseren Unterscheidbarkeit „Level" genannt. Diese 5 Level stellen die „Butterfly"-Kompression dar, die über Tabellen bzw. sogenannte S-Boxen realisiert wird. Für jeden Level gibt es eine eigene Tabelle. Insgesamt gibt es somit 5 Tabellen T_i, für i=0..4, wobei jede aus einer Anzahl von (9-i) Werten besteht (T_0[512], T_1[256], T_2[128], T_3[64], T_4[32]). Gleichzeitig haben Werte in jedem Level eine unterschiedliche Höhe, nämlich (8-i)-bit (T_0: 8bit, T_1 7bit, T_2 6bit, T_3 5bit, T_4 4bit). Die Tabellen sind im Anhang einsehbar.

Tabelle	Anzahl Werte	Wertehöhe
T0	512 (9bit)	256 (8bit)
T1	256 (8bit)	128 (7bit)
T2	128 (7bit)	64 (6bit)
T3	64 (6bit)	32 (5bit)
T4	32 (5bit)	16 (4bit)

Tabelle 1: Struktur der Tabellen T_i (Quelle: Pseudoquellcode s.u.)

Die Hashing-Funktion ist eine surjektive und keine injektive Funktion, da eine Abbildung von 2^{16} Eingabewerten auf einen Ausgabewert stattfindet.

Anhand des Pseudoquellcodes [7] des COMP128-Algorithmus lässt sich die Funktionsweise einfacher darlegen:

```
for j = 0 to 4 do
{  for k = 0 to 2^j-1 do
   {  for l = 0 to 2^(4-j)-1 do
      {
         m = l + k·2^(5-j);
         n = m + 2^(4-j);
         y = (X[m] + 2·X[n]) mod 2^(9-j);
         z = (2·X[m] + X[n]) mod 2^(9-j);
         X[m] = T[j][y];
         X[n] = T[j][z];
      }
   }
}
```

Der Pseudocode ist relativ simpel gehalten und besteht im wesentlichen aus drei For-Schleifen. Startet man bei j=k=l=0, so werden m=0 und n=16 gesetzt. Damit werden in der untenstehenden Abbildung, ganz oben die Stellen 0 und 16 angesprochen. Im Level 0 wird dann in einer Hilfsvariable y das Ergebnis der Addition des einfachen Wertes aus Stelle 0 mit dem zweifachen Wert aus Stelle 16 gespeichert. In Hilfsvariable z wird der zweifache Wert aus Stelle 0 mit dem einfachen Wert aus Stelle 16 addiert und gespeichert. Durch diese Überkreuzverknüpfung hat die Butterfly-Struktur ihren Namen erhalten, sie erinnert an einen Schmetterling. Nun wird die Schleifenvariable l als erstes

erhöht, in Level 0 erreicht l schließlich 15, so dass alle Stellen in dem Level abgedeckt werden und einmal in die Berechnung einbezogen wurden, d.h. die letzte Verknüpfung betrifft die Stellen 15 und 31.

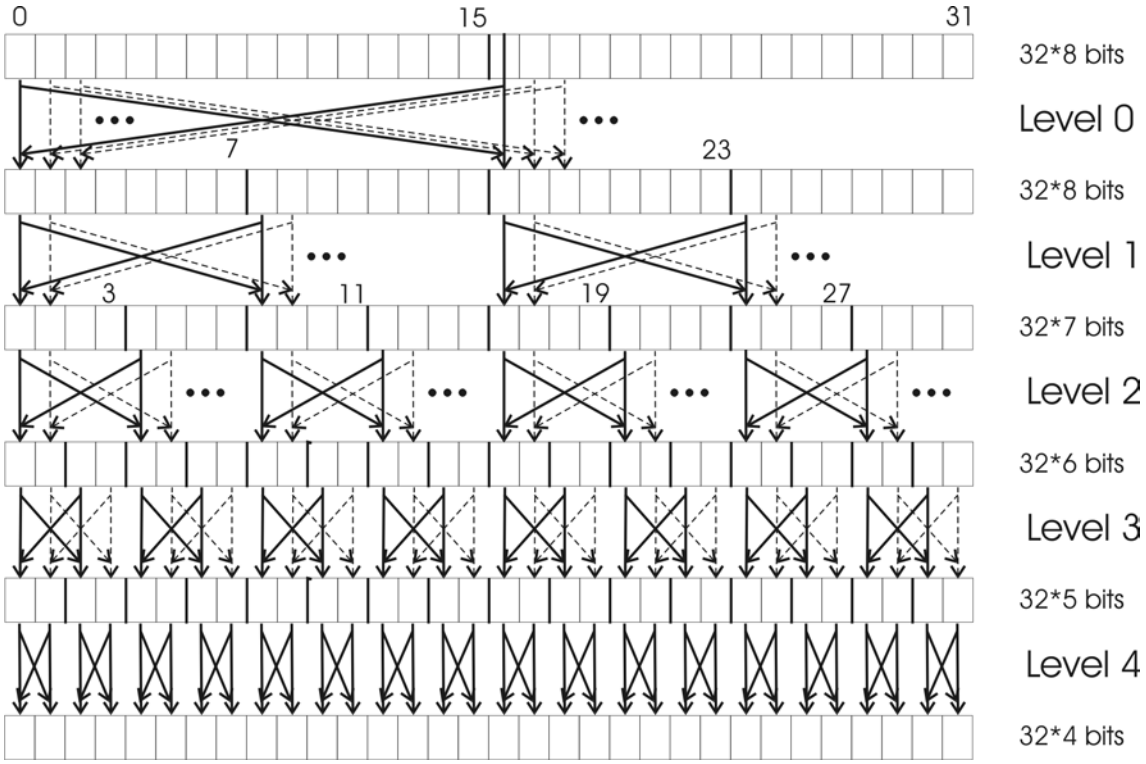

Abbildung 6: Schema Butterfly-Kompression

In den weiteren Levels wird das gleiche Verfahren angewandt, jedoch werden die miteinanderverknüpften Bereiche halbiert und die Anzahl der Bereichspaare verdoppelt, bis schlussendlich in Level 4 16 nebeneinaderstehende Paare miteinander verknüpft werden. Der Zählindex i beschreibt die Positionen, die mit der gleichen Anzahl (übriger) direkt nachfolgender Positionen verknüpft werden, z.B. Level 0: 0..15 mit 16..31, Level 1: 0..7 mit 8..15, 16..23 mit 24..31, usw.

Level 0:	$y = (X[i] + 2 \cdot X[i+16]) \bmod 512$ $z = (2 \cdot X[i] + X[i+16]) \bmod 512$	*i=0..15*
Level 1:	$y = (X[i] + 2 \cdot X[i+8]) \bmod 256$ $z = (2 \cdot X[i] + X[i+8]) \bmod 256$	*i=0..7, 16..23*
Level 2:	$y = (X[i] + 2 \cdot X[i+4]) \bmod 128$ $z = (2 \cdot X[i] + X[i+4]) \bmod 128$	*i=0..3, 8..11, 16..19, 24..27*
Level 3:	$y = (X[i] + 2 \cdot X[i+2]) \bmod 64$ $z = (2 \cdot X[i] + X[i+2]) \bmod 64$	*i= 0..1, 4..5, 8..9, 12..13,* *16..17, 20..21, 24..25, 28..29*
Level 4:	$y = (X[i] + 2 \cdot X[i+1]) \bmod 32$ $z = (2 \cdot X[i] + X[i+1]) \bmod 32$	*i= 0, 1, 2, ... , 30, 31*

2.2.4 Permutation

Sofern die 8 Runden des COMP128 noch nicht durchlaufen wurden, folgt im Blockdiagramm der Hashing-Funktion der Permutationsblock. Das Ziel der Permutation ist die Umordnung des 128 bit langen x' aus der Hashing-Funktion. Dabei wird jedes bit einzeln verschoben, in dem jede Position k an die Position (k·17) mod 128 gesetzt wird. Damit jedes bit von x' im Ergebnis der Permutation x'' genau einmal vorkommt, wurde mit 17 ein primitives Erzeugendes der Menge $\mathbf{Z}_{128}$ gewählt. Ein primitives Erzeugendes liegt genau dann vor, wenn, wie in diesem Beispiel, die Zahl 17 mit allen Potenzen von 0 bis 127 potenziert wird, anschließend Modulo 128 gerechnet, alle Elemente zwischen 0 und 127 genau einmal erzeugt. Sind alle bits vertauscht, wird das natürlich auch 128 bit lange x'' gebildet, in dem die gerade erzeugte bit-Folge rückwärts eingelesen wird.

2.2.5 Berechnungsergebnis des COMP128

Wie schon erwähnt, stellt x' nach 8 Runden das 128 bit lange Ergebnis des COMP128-Algorithmus dar, welches aus 32 4-bit Werten gebildet wird.

3 Kollisionsattacke auf COMP128

Die Kollisionsattacke wurde erstmals im Jahre 1998 an der Berkeley-Universität durch Marc Briceno, Ian Goldberg und David Wagner durchgeführt. Sie machten sich eine Schwachstelle des Algorithmus zu Nutze, durch die es ihnen ermöglicht wurde, innerhalb von 7-12 Stunden, den geheimen Benutzerschlüssel K_i zu rekonstruieren. Der hohe Zeitaufwand kommt durch die begrenzte Geschwindigkeit der SIM-Karte zu Stande, die je nach Typ, bei knapp 6.5 Anfragen pro Sekunde liegt. Ausgeführt werden müssen insgesamt, wie später noch berechnet wird, ca. 165000 Anfragen an die SIM-Karte, d.h. so viele RAND-Werte werden an sie gesendet und so viele SRES-Werte von ihr empfangen und ausgewertet. Benötigt werden, neben dem hohen Zeitbedarf, die PIN und die IMSI. Letztere lässt sich einfach aus der SIM-Karte auslesen, zu der PIN und zu der SIM-Karte sollte sich natürlich nicht unrechtens Zugang verschafft werden. Eine Demonstration einer realen Attacke zu Anschauungszwecken, hat der Chaos Computer Club im Jahre 2001 präsentiert. Eine erfolgreiche Attacke hat zur Folge, dass mit dem erlangten Wissen von K_i auf Kosten anderer telefoniert und jedes Gespräch desjenigen passiv belauscht werden kann. Dazu wird ein Kartenleser, ein GSM-Kartenemulator (Software) und ein Season-Interface (Hardware), auch Inverse-Reader genannt, vorausgesetzt [8].
Da der COMP128-Algorithmus nach dem Prinzip „security-by-obscurity“, also Sicherheit durch Geheimhaltung, geschützt war, wurde es nur durch den Umstand ermöglicht die Unsicherheit des Algorithmus zu beweisen, in dem in einer „newsgroup“ Unterlagen auftauchten. Diese waren allerdings unvollständig, so dass mit „reverse-engineering“ die fehlenden Bausteine gewonnen wurden.

3.1 Konstruktion der Attacke

3.1.1 Kollisionen

Eine Kollision bei der hier durchgeführten Kollisionsattacke tritt dann auf, wenn die Ergebnisse $SRES_1$ und $SRES_2$ des COMP128-Algorithmus, für zwei verschiedene Werte $RAND_1$ und $RAND_2$, gleich sind. Auftreten kann eine Kollision nur dann, wenn eine Menge durch eine Funktion auf eine kleinere Menge abgebildet wird. Dabei muss zwangsläufig mehr als ein Punkt der ersten Menge auf einen Punkt der zweiten Menge abgebildet werden, wodurch dann Kollisionen ermöglicht werden.

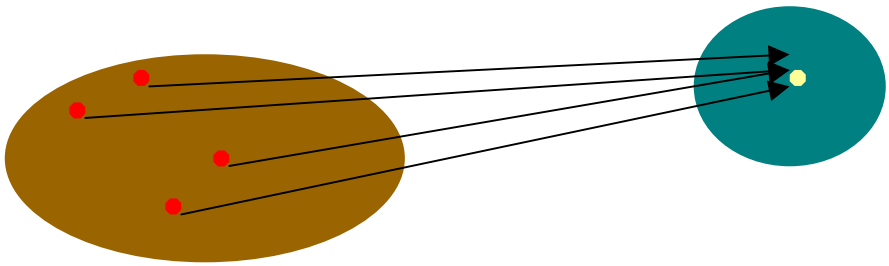

Abbildung 7: Kollision beim Abbilden

3.1.2 Auftreten der Kollision beim COMP128-Algorithmus

Die Frage, an welcher Stelle die Kollision im COMP128-Algorithmus auftreten kann, soll im Folgenden beantwortet werden. Bei der Betrachtung des 1.Levels der Hashing-Funktion fällt auf, dass sich dort die Stellen $x_0[i]$ (von K_i) und $x_0[i+16]$ (von RAND) beeinflussen. Es findet hier eine Abbildung von 2^8 auf 2^8 x 2^8 statt, d.h. es handelte sich um eine bijektive Funktion und es kann keine Kollision in dem 1. Level auftreten.
Im 2. Level hingegen findet eine Abbildung von 2^8 x 2^8 auf 2^7 x 2^7 statt. Diese Reduktion des Ausgabewertebereiches ist die Voraussetzung für eine Kollision. Es beeinflussen sich hier die Stellen $x_1[i]$, $x_1[i+8]$ (von K_i), $x_1[i+16]$, $x_1[i+24]$ (von RAND). Davon sind $x_1[i+16]$, $x_1[i+24]$ bei der Attacke frei wählbar sind. Die Länge des Ergebnisses $x_2[i]$, $x_2[i+8]$, $x_2[i+16]$, $x_2[i+24]$ beträgt 4·7 bit = 28 bit. Ist eine Kollision nach der 2. Runde eingetreten, so zieht sich diese durch die weiteren Levels. Dies führt schlussendlich zu einer Kollision am Ausgang des COMP128-Algorithmus. Für die Kollisionsattacke wird hier nur versucht werden, eine Kollision nach der 2. Runde zur Rekonstruktion von K_i zu benutzen. Nach den weiteren Levels wäre dies auch möglich, würde den Aufwand aber deutlich (quadratisch) steigern für jeden weiteren Level. Die Kollisionsattacke lässt sich in zwei Teile auftrennen:

3.1.2 1. Gewinnung der Kollisions-RAND-Paare

Die Eingaben des COMP128-Algorithmus auf der SIM-Karte sind K_i und RAND. K_i ist bekanntlich fest vorgegeben, so dass nur RAND verändert werden kann bei der Gewinnung der 8 Kollisions-RAND-Paare. Da nach der 2. Runde nur die Stellen i+16, i+24 relevant sind, werden alle anderen Stellen von RAND auf 0 gesetzt, während die Stellen i+16 und i+24 jeweils solange von 0 bis 255 durchgezählt werden, bis zwei RAND-Werte gefunden sind, für die sich, in Verbindung mit dem festen K_i, das gleiche SRES-Ergebnis ergibt. Dies ist nach maximal 256·256 = 65536 Tests gelungen, danach hat man das erste RAND-Paar gefunden. Es werden allerdings 8 RAND-Paare benötigt, da der Schlüssel K_i und RAND aus je 16 Stellen bestehen. Das obige wiederholt sich also für die entsprechend erhöhten Positionsindizes i=1..15.

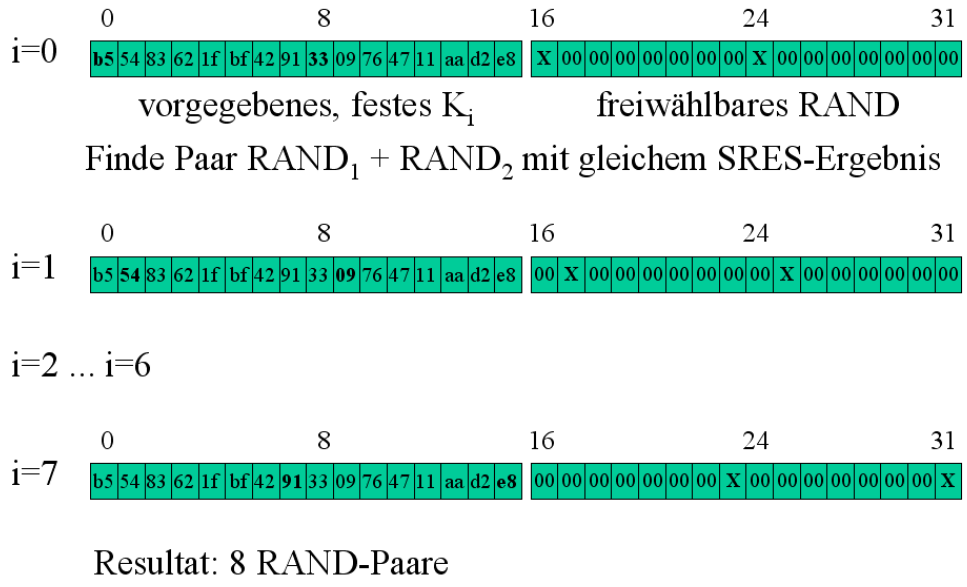

Abbildung 8: 1. Konstruktion der Kollision

3.1.3 2. Konstruktion von K_i

Nun wird umgekehrt verfahren, die beiden RAND-Werte sind für jeden Positionsindex i=0..15 vorgegeben. Bei i=0 werden alle nicht relevanten Stellen des K_i (alle außer i und i+8) auf 0 gesetzt und die Stellen i und i+8 von 0 bis 255 durchgezählt. Dies wird solange durchgeführt, bis SRES für das aktuelle K_i und beide RAND-Werte gleich ist. War dies erfolgreich, wurden bereits zwei Schlüsselbytes gefunden. Also ist es notwendig, diese Suche sieben Mal zu wiederholen für i=1..7. Danach ist das Ziel erreicht, alle 16 Bytes des geheimen Schlüssels K_i wurden gefunden.

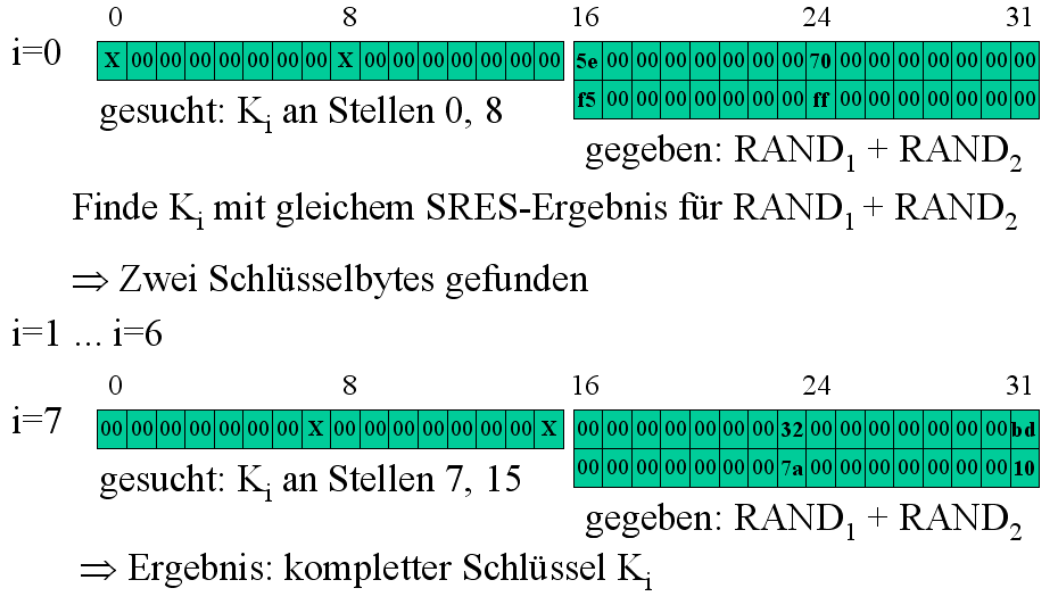

Abbildung 9: 2. Konstruktion von K_i

3.2 Stochastische Untersuchung

Der Grund, warum für die Kollisionsattacke auf Kollisionspaare nach dem zweiten Level getestet wird, soll nun dargelegt werden. Die Wahrscheinlichkeit, mit der eine Kollision nach dem zweiten Level auftritt, lässt sich mit der bekannten Formel zum Geburtstagsproblem lösen (Herleitung s. Anhang 6.2), die normalerweise dazu dient, die Wahrscheinlichkeit zu berechnen, mit der in einer Gruppe zwei Personen am gleichen Tag Geburtstag haben. Bei einer Gruppe von >=23 Personen liegt die Wahrscheinlichkeit bei >50%. Die Anzahl der Tests ist mit $n=2^{16}$ gegeben, da dies die Anzahl der verschiedenen RAND-Werte ist. Des weiteren wird noch $m = 4 \cdot 7 = 28$ gesetzt, wodurch die Anzahl der möglichen Ergebnisse ausgedrückt wird. Zu Stande kommt dies durch die Länge des Ergebnisses der sich beeinflussenden Stellen nach dem zweiten Level, dies sind 4 7-bit Werte.

$$Prob_{Koll} = 1 - e^{\frac{-n^2}{2 \cdot m}} = 1 - e^{\frac{-2^{16^2}}{2 \cdot 2^{28}}} \approx 0.997$$

Die Wahrscheinlichkeit ist fast 1, weshalb eine Ausrichtung der Kollisionsattacke auf den zweiten Level mehr als gerechtfertigt ist.

Der Erwartungswert für das Auftreten der ersten Kollision lässt sich mit der nachstehenden Formel berechnen:

$$E(n) = \sqrt{\pi \cdot \frac{m}{2}} \approx 2^{14.326} \approx 20538$$

Da mit einem Kollisionspaar allerdings erst 2 Schlüsselbytes abgedeckt werden, sind insgesamt also 8·20538 = 164303 Tests, d.h. verschiedene RAND-Werte, die zum Testen an die SIM-Karte geschickt werden müssen, notwendig. Dadurch kommt der Zeitaufwand von >7 Stunden zur Rekonstruktion von K_i zu Stande.

Die Anzahl der insgesamt vorhandenen Kollisionen beträgt 2^{16}, da dies der Anzahl der Eingabewerte auf einen Ausgabewert entspricht.

Weiter ist von Interesse, wie groß die Wahrscheinlichkeit ist, mit der eine Kollision in einer der späteren Levels (3 – 5) auftreten kann. Zur Berechnung wird wieder die Formel des Geburtstagsproblems benutzt, allerdings mit anderen Parameterwerten m und n. n ist in dem Fall $2^{14.326}$, entspricht also dem oben berechneten Erwartungswert - der Anzahl der Tests, die nötig sind, bis zum Auftreten der ersten Kollision. Mit $m = 2^{32}$ ist wieder die Anzahl der Ergebnisse bestimmt, diesmal von den 4·8 bit der SRES stammend.

$$Prob_{ZufallsKoll} = 1 - e^{\frac{-n^2}{2 \cdot m}} = 1 - e^{\frac{-2^{14.326^2}}{2 \cdot 2^{32}}} \approx 0.0479 \approx 5\%$$

Die hier verwendete Methode, um Ki zu erhalten, ist nur für Kollisionen nach dem zweiten Level ausgelegt. Nach höheren Levels ist, wie bereits erwähnt, unter in Kaufnahme eines größer werdenden Aufwandes möglich. Findet nun eine Zufallskollision in den Levels 3-5 statt, so können für dieses RAND-Paar keine Schlüsselbytes gefunden werden, da dann andere Stellen des Ki oder RAND mitverantwortlich sind (s. Butterflykompressionsschema), die aber beim Rekonstruieren immer, da als nicht-relevant angenommen, auf 0 gesetzt sind. In einem solchen Fall muss, wenn alle 256·256 Kombinationen des K_i erfolglos getestet wurden, ein weiteres RAND-Paar für den aktuellen Positionsindex i gesucht werden (Positionsindex i hat nichts zu tun mit dem i in K_i). In besonderen Fällen kann es auch notwendig sein, mehrmals ein neues RAND-Paar zu suchen.

3.3 Simulation der Kollisionsattacke

Das Programm scan_collision_md.exe basiert auf den verfügbaren Quellen von Marc Briceno, Ian Goldberg, and David Wagner und dem Chaos Computer Club. Eine Verbesserung gegenüber scan_collision des CCC stellt die automatische Suche nach weiteren RAND-Paaren dar, falls eine Kollision nicht im zweiten Level stattgefunden hat und K_i nicht dafür an der Stelle bestimmt werden konnte. Aufgerufen wird das Programm mit K_i als Parameter in hexadezimaler Schreibweise, bestehend aus 32 Ziffern mit vorangestelltem „0x“:

Beispiel 1: (einfacher Durchlauf)
scan_collision_md.exe 0x01234567890123456789012345678901

Beispiel 2: (für i=5 ein weiteres RAND-Paar berechnet)
scan_collision_md.exe 0x12345678901234567890123456789012

Dabei stellen zwei Ziffern eine 8-byte Zahl dar. Somit ergeben sich die 16 byte des K_i.
Es werden die beiden Stufen 1. Gewinnung der Kollisions-RAND-Paare und 2. Rekonstruktion von K_i durchgeführt und am Ende das Ergebnis K_i ausgegeben, zusätzlich auch die Anzahl der benötigten Tests, sowie die real benötigte Zeit, wäre eine richtige SIM-Karte eingesetzt worden, zu statistischen Zwecken. Auf einem aktuellen PC dauert eine Simulation zwischen 30-50 Sekunden, je nach K_i und der Anzahl der benötigten erneuten Suchen nach RAND-Paaren.
Benötigt wird „cygwin1.dll“, da dass Programm unter CygWin, einer Linux-Shell ähnlichen Umgebung, kompiliert wurde. Dies ermöglicht die Lauffähigkeit unter Windows.

3.4 Alternative Attacken

Eine sehr vielversprechende alternative Attacke auf den COMP128-Algorithmus wurde von Josyula R. Rao, Pankaj Rohatgi und Helmut Scherzer vom IBM Watson Research Center im Mai 2002 präsentiert [7]. Bei der sogenannten „partition attack“, die einer „differential power analysis“ (DPA) entspricht, wird ein Seitenkanal-Angriff durchgeführt, d.h. es werden Stromverbrauch und elektromagnetische (EM-) Strahlung überwacht. Dadurch ist es möglich, mit nur 8 an die SIM-Karte gesendeten, verschiedenen, adaptiv gewählten RAND-Werten den Schlüssel K_i zu erlangen. Dies stellt einen beträchtlichen Zeitgewinn dar, im Vergleich zur Kollisionsattacke, wenn auch höhere Kosten für die dafür benötigten Messgeräte zu vermuten sind. Wieder muss man auch hier physikalisch im Besitz der SIM-Karte sein.

Weiterhin erwähnenswert ist „Partial Collision Search by Side Channel Analysis“ von A. Wiemers vom BSI [9], präsentiert beim Workshop „Smartcards and Side Channel Attacks“ im Januar 2003 am Horst Goertz Institut in Bochum. Dabei werden teilweise Kollisionen nach dem ersten Level schon gesucht, dies geschieht durch den Vergleich der Leistungsaufnahmen. Der Vorteil dieser Methode liegt darin, dass nur ungefähr 2000 Versuche durchgeführt werden müssen.

4 Schlussbetrachtung

Wie die Kollisionsattacke gezeigt hat, ist der COMP128-Algorithmus alles andere als sicher zu bezeichnen. In Deutschland war/ist dieser Algorithmus nicht sehr verbreitet, er fand nur bei D2 Verwendung, allerdings erfreute er sich in anderen der über 190 Länder auf der Welt, in denen das GSM-Netz existiert, größerer Beliebtheit. Anzumerken ist, dass der COMP128 nur eine Beispielimplementation war, aber dennoch in der Praxis eingesetzt wurde. Ein noch größeres Sicherheitsrisiko entsteht durch die kurz erwähnte „partitioning attack“, wodurch noch schneller Telefonieren auf Kosten anderer und Abhören möglich ist. Beachtung sollte auch die Attacke über die Luftschnittstelle finden, bei der mit einer falschen Basisstation, die nur die wichtigsten Funktionen einer echten beherrscht, der Authentifizierungsablauf, anstatt nur einmal, tausende Male durchlaufen werden kann, dazu werden die RAND-Werte und die SRES-Werte lediglich durch die Luft übertragen. Dann muss ein Angreifer auch nicht im Besitz der SIM-Karte sein, sondern nur sicherstellen, dass er in der Nähe des Handys ist und die stärkste Sendeleistung im Umkreis besitzt für das angegriffene Handy.
In der Zwischenzeit wurden einige Maßnahmen durchgeführt, um die Sicherheit zu erhöhen: Ob die Einführung des COMP128-2 eine Verbesserung darstellt, ist nicht bekannt, da auch hier wieder die Sicherheit durch Geheimhaltung erreicht werden soll. Ein weiterer Schritt zum Schutz gegen das SIM-Karten-Kopieren ist eine Beschränkung der möglichen Anfrageversuche bei neueren SIM-Karten auf z.B. 50000, eine Zahl, die im Normaleinsatz einer SIM-Karte wohl nie erreicht werden wird, allerdings lässt sich dadurch eine komplette Kollisionsattacke gar nicht mehr durchführen. Für die „Partitioning Attack“ oder die „Partial Collision Search by Side Channel Analysis“ stellt dies dagegen kein großes Hindernis dar. Auch machen Mobilfunkprovider inzwischen ein doppeltes Eingebuchtsein einer SIM-Karte unmöglich.

5 Literaturverzeichnis

[1] Geschichte des Mobilfunks („Geschichte des Mobilfunks.htm“)
http://www.handy-seiten.de/10-Geschichte/10-geschichte.html

[2] GSM: ein wenig Geschichte („GSMBOX_DE - GSM ein wenig Geschichte.htm“)
http://de.gsmbox.com/gsm/storia.gsmbox

[3] www.gsm.org: http://www.gsmworld.com/news/statistics/index.shtml

[4] http://www.gsmworld.com/news/statistics/graph7.html („gsm_benutzerstatistik.gif“)

[5] www.privateline.com/pcs/structure.html
(„www_privateline_com_pcs_structure.gif“)

[6] Seminar: Topics in Cryptography and Security, Vortrag:
Sicherheitsprobleme in GSM-Netzen von Thomas Kleintje
Ruhr-Universität-Bochum, 08.02.2002, Lehrstuhl für Kommunikationssicherheit
(Seite 6 von 19)

[7] „gsm[1] 1000 255 8.ps“

[8] http://www.ccc.de/gsm/

[9] A. Wiemers (BSI). Partial Collision Search by Side Channel Analysis. Presentation at the Workshop: Smartcards and Side Channel Attacks, January 2003. Horst Goertz Institute, Bochum, Germany. („wiemers.pdf“)

[10] Biljana Cubaleska, Andrea Frank, Bernhard Löhlein, Nour-Eddine Ourahou, Sonja Schaup, Oliver Stutzke, „Foundations of Cryptology” Kapitel 6, Fern-Universität Hagen

[11] Erik Zenner, Kryptographische Protokolle im GSM-Standard: Beschreibung und Kryptanalye, Diplomarbeit Universität Mannheim, Juli 1999

6 Anhang

6.1 Quelltext des Simulationsprogramms

```
#include <map.h>
#include <stdio.h>
#include <ctype.h>

#undef TEST
#define TEST
#define VERBOSES
#undef VERBOSES

typedef unsigned char Byte;


struct ltsim
{
    bool operator()(const Byte* b1, const Byte* b2) const
    {
      return memcmp(b1, b2, 12) < 0;
    }
};

typedef map<const Byte*, const Byte*, ltsim> simoutput_map;

/* An implementation of the GSM A3A8 algorithm.  (Specifically,
   COMP128.)
 */

/* Copyright 1998, Marc Briceno, Ian Goldberg, and David Wagner.
 * All rights reserved.
 */

/*
 * For expository purposes only.  Coded in C merely because C is a
 * much more precise, concise form of expression for these purposes.
 * See Judge Patel if you have any problems with this...
 * Of course, it's only authentication, so it should be exportable for
 * the usual boring reasons.
 */

/* The compression tables. */

static const Byte
table_0[512] = {102,177,186,162,  2,156,112, 75, 55, 25,  8, 12,251,
193,246,188,109,213,151, 53, 42, 79 ,191,115,233,242,164,223,209,148,
108,161,252, 37,244, 47, 64,211,6,237,185,160,139,113, 76,138, 59,70,
67, 26, 13,157, 63,179,221, 30,214, 36,166,69,152,124,207,116,247,194,
41, 84, 71,  1, 49, 14, 95, 35,169, 21, 96, 78, 215, 225,182,243, 28,
92,201,118,  4, 74,248,128, 17, 11,146,132,245, 48,149, 90,120, 39,
87,230,106,232,175, 19,126,190,202,141,137,176,250, 27,101, 40,219,
227, 58, 20, 51,178, 98,216,140, 22, 32,121, 61,103, 203, 72, 29,110,
85, 212,180,204,150,183, 15, 66,172,196, 56,197,158,  0,100, 45,153,
7,144,222,163,167, 60,135,210,231,174,165, 38,249,224, 34,220,229,
217,208,241, 68,206,189,125,255,239, 54,168, 89,123,122,73, 145,117,
234,143, 99,129,200,192, 82,104,170,136,235, 93, 81, 205, 173,236,
94,105, 52, 46,228,198,  5, 57,254, 97,155,142,133,199,171, 187, 50,
65,181,127,107,147,226,184,218,131, 33, 77, 86, 31, 44, 88, 62,238,
18, 24, 43,154, 23, 80,159,134,111,  9,114,  3, 91, 16,130, 83, 10,
```

```
195,240,253,119,177,102,162,186,156,  2, 75,112, 25, 55, 12,  8, 193,
251,188,246,213,109, 53,151, 79, 42,115,191,242,233,223,164,148, 209,
161,108, 37,252, 47,244,211, 64,237,  6,160,185,113,139,138, 76, 70,
59, 26, 67,157, 13,179, 63, 30,221, 36,214, 69,166,124,152, 116,207,
194,247, 84, 41,  1, 71, 14, 49, 35, 95, 21,169, 78, 96,225, 215,243,
182, 92, 28,118,201, 74,  4,128,248, 11, 17,132,146, 48,245, 90,149,
39,120,230, 87,232,106, 19,175,190,126,141,202,176,137, 27, 250, 40,
101,227,219, 20, 58,178, 51,216, 98, 22,140,121, 32,103, 61, 72,203,
110, 29,212, 85,204,180,183,150, 66, 15,196,172, 197, 56,  0, 158, 45,
100,  7,153,222,144,167,163,135, 60,231,210,165,174,249, 38, 34,224,
229,220,208,217, 68,241,189,206,255,125, 54,239, 89,168,122, 123,145,
73,234,117, 99,143,200,129, 82,192,170,104,235,136, 81, 93,173,205,
94,236, 52,105,228, 46,  5,198,254, 57,155, 97,133,142, 171,199, 50,
187,181, 65,107,127,226,147,218,184, 33,131, 86, 77, 44, 31, 62, 88,
18,238, 43, 24, 23,154,159, 80,111,134,114,  9, 91,  3, 130, 16, 10,
83,240,195,119,253},

table_1[256] = { 19, 11, 80,114, 43,  1, 69, 94, 39, 18,127,117, 97,
3, 85, 43, 27,124, 70, 83, 47, 71, 63, 10, 47, 89, 79,  4, 14, 59, 11,
5, 35,107,103, 68, 21, 86, 36, 91, 85,126, 32, 50,109, 94,120,  6, 53,
79, 28, 45, 99, 95, 41, 34, 88, 68, 93, 55,110,125,105, 20, 90, 80,
76, 96, 23, 60, 89, 64,121, 56, 14, 74,101,  8, 19, 78, 76, 66,104,
46,111, 50, 32,  3, 39,  0, 58, 25, 92, 22, 18, 51, 57, 65,119,116,
22,109,  7, 86, 59, 93, 62,110, 78, 99, 77, 67, 12,113, 87, 98,102,
5, 88, 33, 38, 56, 23,  8, 75, 45, 13, 75, 95, 63, 28, 49,123,120,
20,112, 44, 30, 15, 98,106,  2,103, 29, 82,107, 42,124, 24, 30, 41,
16,108,100,117, 40, 73, 40,  7,114, 82,115, 36,112, 12,102,100, 84,
92, 48, 72, 97,  9, 54, 55, 74,113,123, 17, 26, 53, 58,  4,  9, 69,
122, 21,118, 42, 60, 27, 73,118,125, 34, 15, 65,115, 84, 64, 62, 81,
70,  1, 24,111,121, 83,104, 81, 49,127, 48,105, 31, 10,  6, 91, 87,
37, 16, 54,116,126, 31, 38, 13,  0, 72,106, 77, 61, 26, 67, 46, 29,
96, 37, 61, 52,101, 17, 44,108, 71, 52, 66, 57, 33, 51, 25, 90,  2,
119,122, 35},

table_2[128] = { 52, 50, 44,  6, 21, 49, 41, 59, 39, 51, 25, 32, 51,
47, 52, 43, 37,  4, 40, 34, 61, 12, 28,  4, 58, 23,  8, 15, 12, 22,
9, 18, 55, 10, 33, 35, 50,  1, 43,  3, 57, 13, 62, 14,  7, 42, 44, 59,
62, 57, 27,  6,  8, 31, 26, 54, 41, 22, 45, 20, 39,  3, 16, 56, 48,
2, 21, 28, 36, 42, 60, 33, 34, 18,  0, 11, 24, 10, 17, 61, 29, 14, 45,
26, 55, 46, 11, 17, 54, 46,  9, 24, 30, 60, 32,  0, 20, 38,  2, 30,
58, 35,  1, 16, 56, 40, 23, 48, 13, 19, 19, 27, 31, 53, 47, 38, 63,
15, 49,  5, 37, 53, 25, 36, 63, 29,  5,  7},

table_3[64] = {  1,  5, 29,  6, 25,  1, 18, 23, 17, 19,  0,  9, 24,
25,  6, 31, 28, 20, 24, 30,  4, 27,  3, 13, 15, 16, 14, 18,  4,  3,
8,  9, 20,  0, 12, 26, 21,  8, 28,  2, 29,  2, 15,  7, 11, 22, 14, 10,
17, 21, 12, 30, 26, 27, 16, 31, 11,  7, 13, 23, 10,  5, 22, 19},

table_4[32] = { 15, 12, 10,  4,  1, 14, 11,  7,  5,  0, 14,  7,  1,
2, 13,  8, 10,  3,  4,  9,  6,  0,  3,  2,  5,  6,  8,  9, 11, 13, 15,
12},

*table[5] = { table_0, table_1, table_2, table_3, table_4 };

/*
 * rand[0..15]: the challenge from the base station
 * key[0..15]: the SIM's A3/A8 long-term key Ki
 * simoutput[0..11]: what you'd get back if you fed rand and key to a
 * real SIM.
 *
 *   The GSM spec states that simoutput[0..3] is SRES,
 *   and simoutput[4..11] is Kc (the A5 session key).
 *   (See GSM 11.11, Section 8.16.  See also the leaked document
```

```
 *   referenced below.)
 *   Note that Kc is bits 74..127 of the COMP128 output, followed by
 *   10 zeros.
 *   In other words, A5 is keyed with only 54 bits of entropy. This
 *   represents a deliberate weakening of the key used for voice
 *   by a factor of over 1000.
 *
 * Verified with a Pacific Bell Schlumberger SIM.  Your mileage may
 * vary.
 * Marc Briceno <marc@scard.org>, Ian Goldberg <iang@cs.berkeley.edu>,
 * and David Wagner <daw@cs.berkeley.edu>
 */

/*
 * This code derived from a leaked document from the GSM standards.
 * Some missing pieces were filled in by reverse-engineering a working
 * SIM. We have verified that this is the correct COMP128 algorithm.
 *
 * The first page of the document identifies it as
 *      _Technical Information: GSM System Security Study_.
 *      10-1617-01, 10th June 1988.
 * The bottom of the title page is marked
 *      Racal Research Ltd.
 *      Worton Drive, Worton Grange Industrial Estate,
 *      Reading, Berks. RG2 0SB, England.
 *      Telephone: Reading (0734) 868601   Telex: 847152
 * The relevant bits are in Part I, Section 20 (pages 66--67).  Enjoy!
 *
 * Note: There are three typos in the spec (discovered by
 * reverse-engineering).
 * First, "z = (2 * x[n] + x[n]) mod 2^(9-j)" should clearly read
 * "z = (2 * x[m] + x[n]) mod 2^(9-j)".
 * Second, the "k" loop in the "Form bits from bytes" section is
 * severely botched: the k index should run only from 0 to 3, and
 * clearly the range on "the (8-k)th bit of byte j" is also off
 * (should be 0..7, not 1..8, to be consistent with the subsequent
 * section).
 * Third, SRES is taken from the first 8 nibbles of x[], not the last
 * 8 as claimed in the document.  (And the document doesn't specify
 * how Kc is derived, but that was also easily discovered with reverse
 * engineering.)
 * All of these typos have been corrected in the following code.
 */

void A3A8(/* in */ Byte rand[16], /* in */ Byte key[16],
     /* out */ Byte simoutput[12])
{
     Byte x[32], bit[128];
     int i, j, k, l, m, n, y, z, next_bit;

     /* ( Load RAND into last 16 bytes of input ) */
     for (i=16; i<32; i++)
          x[i] = rand[i-16];

     /* ( Loop eight times ) */
     for (i=1; i<9; i++) {
          /* ( Load key into first 16 bytes of input ) */
          for (j=0; j<16; j++)
               x[j] = key[j];
          /* ( Perform substitutions ) */
          for (j=0; j<5; j++)
               for (k=0; k<(1<<j); k++)
                    for (l=0; l<(1<<(4-j)); l++)
```

```
                {
                            m = l + k*(1<<(5-j));
                            n = m + (1<<(4-j));
                            y = (x[m]+2*x[n]) % (1<<(9-j));
                            z = (2*x[m]+x[n]) % (1<<(9-j));
                            x[m] = table[j][y];
                            x[n] = table[j][z];
                        }
            /* ( Form bits from bytes ) */
            for (j=0; j<32; j++)
                  for (k=0; k<4; k++)
                        bit[4*j+k] = (x[j]>>(3-k)) & 1;
            /* ( Permutation but not on the last loop ) */
            if (i < 8)
                  for (j=0; j<16; j++)
            {
                        x[j+16] = 0;
                        for (k=0; k<8; k++)
                {
                            next_bit = ((8*j + k)*17) % 128;
                            x[j+16] |= bit[next_bit] << (7-k);
                        }
                  }
      }

      /*
       * ( At this stage the vector x[] consists of 32 nibbles.
       *   The first 8 of these are taken as the output SRES. )
       */

      /* The remainder of the code is not given explicitly in the
       * standard, but was derived by reverse-engineering.
       */

      for (i=0; i<4; i++)
            simoutput[i] = (x[2*i]<<4) | x[2*i+1];
      for (i=0; i<6; i++)
            simoutput[4+i] = (x[2*i+18]<<6) | (x[2*i+18+1]<<2)
                        | (x[2*i+18+2]>>2);
      simoutput[4+6] = (x[2*6+18]<<6) | (x[2*6+18+1]<<2);
      simoutput[4+7] = 0;
}

int hextoint(char x)
{
  x = toupper(x);
  if (x >= 'A' && x <= 'F')
    return x-'A'+10;
  else if (x >= '0' && x <= '9')
    return x-'0';
  fprintf(stderr, "bad input.\n");
  exit(1);
  return 0;
}

int main(int argc, char **argv)
{
  //Byte pin[8] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
  Byte key[16], simoutput[12], simoutput2[12];
  int i,j,k,l,mcounter,mcounter2,collisions_counter;
  int j2,k2;
  float rcounter;
  simoutput_map* simmap;
```

```
  simoutput_map* simmap2;
  Byte* collision_map[8][2];
  Byte* collision_map2[8][2];

#ifdef TEST
  if (argc != 2 || strlen(argv[1]) != 34
      || strncmp(argv[1], "0x", 2) != 0)
  {
    fprintf(stderr, "Usage: %s 0x<key>\n", argv[0]);
    exit(1);
  }

  for (i=0; i<16; i++)
    key[i] = (hextoint(argv[1][2*i+2])<<4)
      | hextoint(argv[1][2*i+3]);
#else
/*  if (argc != 3) {
    fprintf(stderr, "Usage: %s <PIN> <timeout>\n", argv[0]);
    exit(1);
  }

  for (i=0; i<strlen(argv[1]) && i<8; i++) {
    pin[i] = argv[1][i];
  }

  set_timeout(atoi(argv[2]));
*/
#endif

//  init_card("/dev/ttyS1", pin);

  mcounter = 0;
  mcounter2= 0;

  printf("\n\n\n");
  printf("-------------------------------------------------------------
------------------\n");
  printf("Präsentation COMP128-Kollisionsattacke von Marc Dingfelder
(w)2003\n");
  printf("-------------------------------------------------------------
------------------");

  printf("\n\n");
  printf("-------------------------------------------------------------
------------------\n");
  printf("1. Teil: Suche nach Kollision-RAND-Paar, Ki fest,\n");
  printf("         variiere Stellen i, i+8 von RAND, finde RAND1,
RAND2 mit gleichem SRES\n");
  printf("-------------------------------------------------------------
------------------\n\n\n");
  for(i=0; i < 8; i++)

  {
    simmap = new simoutput_map;

    collision_map[i][0] = NULL;
    collision_map[i][1] = NULL;
    for(j=0; j<256; j++)
    {
      if ((j % 4) == 0)
        fprintf(stderr, ".");
      for(k=0; k<256; k++)
      {
```

```
            Byte* newrand = new Byte[16];
          memset(newrand, 0, 16);
          newrand[i+8] = k;
          newrand[i] = j;

          Byte* newoutput = new Byte[12];

        mcounter++;

#ifdef TEST
          A3A8(newrand, key, newoutput);
#else
//    get_response(newrand, newoutput);
#endif
          if(simmap->count(newoutput))
        {
            printf("\ni=%i:1.Kollision",i);
            for (l=0; l<16; l++)
              printf("%02X", newrand[l]);
            printf(" ");
            for (l=0; l<16; l++)
              printf("%02X", (*simmap)[newoutput][l]);
            j=256; k=256;
          collision_map[i][0] = new Byte[16];
          collision_map[i][1] = new Byte[16];
            memcpy(collision_map[i][0], newrand, 16);
            memcpy(collision_map[i][1], (*simmap)[newoutput], 16);

          }
        else
        {
#ifdef VERBOSES
            for (l=0; l<16; l++)
              printf("%02X", newrand[l]);
            printf(" ");
            for (l=0; l<12; l++)
              printf("%02X", newoutput[l]);
            printf("\n");
#endif
             (*simmap)[newoutput]=newrand;
        }
          fflush(stdout);
      }
    }
    printf("\n");
    for(simoutput_map::iterator I=simmap->begin(); I!=simmap->end();
I++)
    {
      delete (*I).first;
      delete (*I).second;
      //      simmap.erase(I);
    }
    delete simmap;
  }
// Ende der regulären Kollisionssuche

  printf("\n\n\n");
  printf("-------------------------------------------------------------
------------------\n");
  printf("2. Teil: Bilde Schlüssel Ki, variiere Stellen i, i+8 von Ki,
\n");
  printf("         SRES muss für gefundene RAND1, RAND2 gleich
sein\n");
```

```
  printf("-------------------------------------------------------------
-------------------\n\n\n");

// Start Schlüsselsuche
  Byte newkey[16];
  Byte resultkey[16];
  Byte res1[16], res2[16];

  memset(resultkey, 0, 16);

  for(i=0; i<8; i++)
  {
    memset(newkey, 0, 16);
    for(j=0; j<256; j++)
    {
      newkey[i]=j;
      if ((j % 4) == 0)
        fprintf(stderr, ".");
      for(k=0; k<256; k++)
      {
          newkey[i+8]=k;
          A3A8(collision_map[i][0], newkey, res1);
          A3A8(collision_map[i][1], newkey, res2);
          if(!memcmp(res1, res2, 12))
        {
            resultkey[i]=j;
          resultkey[i+8]=k;
          printf("\ni=%01d:Key bytes gefunden, Ki bisher: ",i);
            for (l=0; l<16; l++)
              printf("%02X", resultkey[l]);
          j = 256; k = 256;
        }
        if (j==255)
        {
              if (k==255)
            {
              printf("\nKeine Key bytes für i=%01d gefunden,
Zufallskollision nach 2.Runde???\n",i);



// 2. Kollision berechnen

              collisions_counter = 0;
              simmap2 = new simoutput_map;
              collision_map[i][0] = NULL;     //2
              collision_map[i][1] = NULL;     //2
              for(j=0; j<256; j++)
              {
                if ((j % 4) == 0)
                  fprintf(stderr, ".");
                for(k=0; k<256; k++)
                {
                    Byte* newrand2 = new Byte[16];
                    memset(newrand2, 0, 16);
                    newrand2[i+8] = k;
                    newrand2[i] = j;

                    Byte* newoutput2 = new Byte[12];

                  mcounter2++;

                    A3A8(newrand2, key, newoutput2);
```

```
                    if(simmap2->count(newoutput2))
                   {
                    collisions_counter++;
//                    printf("\n %01d. Kollision",collisions_counter);

                    //teste, ob dieses 1. Kollision ist:
                    if(collisions_counter == 2)
                    {
                        printf("\ni=%i:2.Kollision",i);
                        for (l=0; l<16; l++)
                        printf("%02X", newrand2[l]);
                        printf(" ");
                        for (l=0; l<16; l++)
                        printf("%02X", (*simmap2)[newoutput2][l]);
                        j=256; k=256;
                      collision_map[i][0] = new Byte[16];     //2
                      collision_map[i][1] = new Byte[16];     //2
                        memcpy(collision_map[i][0], newrand2, 16);
//2
                        memcpy(collision_map[i][1],
(*simmap2)[newoutput2], 16);
                    }
                    }
                  else
                  {
                      (*simmap2)[newoutput2]=newrand2;
                  }
                    fflush(stdout);
                }//for k
              }//for j
//              printf("\n");
              for(simoutput_map::iterator I=simmap2->begin();
I!=simmap2->end(); I++)
              {
                delete (*I).first;
                delete (*I).second;
                //      simmap.erase(I);
              }
              delete simmap2;
              j=0;
              k=0;
            }//if (k==255)
        }//if (j==255)
// Ende Suche 2. Kollision




/*

// Suche Key bytes für 2. Kollision


        memset(newkey, 0, 16);
        for(j2=0; j2<256; j2++)
        {
          newkey[i]=j2;
          fprintf(stderr, ".");
          for(k2=0; k2<256; k2++)
          {
              newkey[i+8]=k2;
```

```
              A3A8(collision_map2[i][0], newkey, res1);
              A3A8(collision_map[i][1], newkey, res2);
              if(!memcmp(res1, res2, 12))
             {
                resultkey[i]=j2;
              resultkey[i+8]=k2;
              printf("\nKey bytes gefunden, Ki bisher: ");
                for (l=0; l<16; l++)
                  printf("%02X", resultkey[l]);
              j2 = 256; k2 = 256;
              }
              if (j2==255)
              {
                if (k2==255)
                {
                printf("\nKeine Key bytes für i=%01d gefunden für
2.Kollision (sehr unwahrscheinlich *g*)\n",i);
                printf("Zufallskollision ab >=3. Runde???\n");
              }
             }
           }//for k
//           printf("\n");
         }//for j2

// Ende Suche nach Key bytes für 2. Kollision

*/


      }//for k
    }//for j
    printf("\n");
  }
  printf("\n------------------------------------------------------------
---------------------");
  printf("\nErmitteltes  Ki: ");
  for (l=0; l<16; l++)
    printf("%02X", resultkey[l]);
  printf("\nEingegebenes Ki: ");
  for (l=0; l<16; l++)
    printf("%02X", key[l]);
  printf("\n------------------------------------------------------------
---------------------");
  printf("\n\nAnzahl Kollisionstests: (1.Kollision) %01d +
(2.Kollision) %01d = %01d",mcounter, mcounter2,mcounter+mcounter2);
  rcounter = (mcounter+mcounter2)/6.25/3600;
  printf("\n\nAttacke hätte real %03.02f Stunden gedauert (bei 6.25
Tests/sek)",rcounter);
  printf("\n\n------------------------------------------------------------
----------------------\n");
  return 0;
}
```

6.2 Herleitung der Geburtstagsproblem-Formel [10]:

P(n, k) : Wahrscheinlichkeit, dass etwas doppelt vorkommt (Geburtstag) bei k (k ≤ n) Teilen (Schüler), wobei es n gleichwahrscheinliche Werte (1 bis n, hier n=365 Tage) gibt.

Einfacher:
Q(n, k) : Wahrscheinlichkeit für nicht doppeltes Vorkommen
N : Anzahl unterschiedliche Weisen, um k Werte ohne doppeltes Vorkommen zu haben.

$$N = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

Die maximale Anzahl an Möglichkeiten ist n^k.

$$\Rightarrow Q(n,k) = \frac{N}{n^k} = \frac{n!}{(n-k)! \cdot n^k}$$

$$\Rightarrow P(n,k) = 1 - Q(n,k) = 1 - \frac{n!}{(n-k)! \cdot n^k}$$

Für k = 23 und n = 365 erhält man einen Wert von 0.507, also ist die Wahrscheinlichkeit 50.7%, dass zwei Schüler in einer Klasse mit 23 Schülern am gleichen Tag Geburtstag haben.

$$P(n,k) = 1 - \frac{n!}{(n-k)! \cdot n^k} = 1 - \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{n^k} = 1 - \left[\frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ldots \cdot \frac{n-k+1}{n} \right]$$

$$P(n,k) = 1 - \left[\left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdot \ldots \cdot \left(1 - \frac{k-1}{n}\right) \right]$$

Mit $(1-x) \le e^{-x}$ für alle $x \ge 0$ folgt:

$$P(n,k) > 1 - \left[\left(e^{\frac{-1}{n}} \right) \cdot \left(e^{\frac{-2}{n}} \right) \cdot \ldots \cdot \left(e^{\frac{-(k-1)}{n}} \right) \right] = 1 - e^{-\left[\frac{1}{n} + \frac{2}{n} + \ldots + \frac{k-1}{n} \right]} = 1 - e^{-\frac{1}{n} \cdot (1+2+\ldots+(k-1))} = 1 - e^{-\frac{1}{n} \cdot \frac{k \cdot (k-1)}{2}}$$

Also:

$$P(n,k) > 1 - e^{-\frac{k \cdot (k-1)}{2 \cdot n}}$$

Für grosse k lässt sich die Formel auf die im Text verwendete Formel annähern:

$$P(n,k) > 1 - e^{-\frac{k^2}{2 \cdot n}}$$