DAS LONGLEY UND RIGBY-TOOL

STEFFEN BUHLE

Offenbar ist es in jedem Dokument über Analysemittel für kryptographische Protokolle am Anfang üblich, ein bis zwei Worte über die Notwendigkeit dieser Hilfsmittel zu verlieren. Ich möchte mit dieser (schönen?) Tradition nicht brechen:
Im Laufe der Entwicklung der Kryptographie entstanden sehr viele mehr oder weniger sichere Algorithmen, die Daten verschlüsseln. Vom heutigen Stand der Entwicklung aus sind diese Algorithmen so ausgefeilt, dass der Beruf des Hackers wohl an Attraktivität verloren haben dürfte, zumindest für jene, die keinen Hochleistungs-Parallel-Rechner mit extremen MegaFLOPS - Werten ihr eigen nennen dürfen. Trotzdem passieren immer wieder Hack's, und das nicht nur Normaluser, die ihr Kennwort unter das Mousepad schreiben. Also muß es offensichtlich doch Möglichkeiten geben, trotz mathematisch durchdachter Algorithmen und komplexer Verschlüsselungstools, chiffrierte Daten in einem Netzwerk sichtbar zu machen, ohne $'Viele * 10^{(noch\ viele\ mehr)}'$ verschiedene Schlüssel ausprobieren zu müssen. Es ist die Übertragung der Schlüssel.
In der Kette der Aktionen, angefangen von der Kontaktaufnahme zweier User im Netz, über die Schlüsselverteilung und Authentifizierung, bis zum Versenden der verschlüsselten Daten, ist das wirklich schwache Glied genau die Schlüsselverteilung zwischen Absender und Empfänger. Solche kryptographischen Protokolle, also Abarbeitungsfolgen, die vorschreiben wie und wann Schlüssel übertragen werden, und welche Funktionen dafür benutzt werden, sind für Protokolldesigner schwer auf Sicherheitslücken analysierbar. Bei vielen komplexen Protokollen (auch bei dem, dass für das Longeley und Rigby Tool als Beipiel dienen soll) sieht man sozusagen den Wald vor lauter Bäumen nicht mehr, und es benötigt viel Konzentration, wenn man jede mögliche dem Angreifer bekannte Information mit den kryptographischen Funktionen kombinieren soll, in der Hoffnung, dass sich keine Sicherheitslücke (zB. das Extrahieren eines Schlüssels) auftut. Bei solchen Arbeiten stößt der Mensch schnell an seine Grenzen. Desweiteren neigt der Protokolldesigner dazu, sein Protokoll schnell als sicher abzutun, weil es ja im Grunde auch "verwirrend ist, nach einer Sicherheitslücke zu suchen, wenn man davon überzeugt ist, dass eine solche Sicherheitslücke gar nicht existiert"([1], Seite 76, trans.). Genau dies ist die Stelle, an der die (theoretische) Informatik ins Spiel kommt.
Es gibt einige Ansätze zur formalen Analyse von kryptographischen Protokollen:

(1) Logiken: die BAN-Logik
(2) Algebren: das Dolev-Yao Modell, Interrogator
(3) Zustandsautomaten: NRL-Analyser, Interrogator

Das hier im Weiteren beschriebene Longley und Rigby Tool, baut auf der Idee des Interrogators auf. Zur Struktur dieses Scripts: Nach einer kurzen Einführung wird die Arbeitsweise des Longley und Rigby Tools vorgestellt, einige Sonderfälle bei der Implementierung besprochen und zum Schluss das Werkzeug auf ein komplexes Protokoll angewandt.

Key words and phrases. DES, Suchbaum, Interrogator, Kryptography, Protokoll, Tagged Key Scheme.

1. Einführung

Bei diesem Werkzeug von Prof. Dennis Longley und S. Rigby handelt es sich um ein Package für PROLOG, dass 1995 entwickelt wurde um Fehler (Sicherheitslücken) in kryptographischen Protokollen ausfindig zu machen. Die Idee von Longley und Rigby ist schnell erklärt:
Wenn der Angreifer eine Sicherheitslücke sucht, dann fragt er sich: 'Wenn ich diese und jene Information besitze, ist es dann auch ausreichend Information, um (z.B.) einen Schlüssel zu erlangen, der PIN's codiert [...]?' ([1], Seite 77, trans.)
Das Longley und Rigby Tool stellt sich die umgekehrte Frage: Wenn ich (z.B.) eine PIN dekodieren möchte, welche Daten benötige ich dazu und kann ich *jemals* diese Daten erlangen?
Diese Idee verbirgt sich auch hinter dem Interrogator (siehe [2]), jedoch unterscheidet sich die Umsetzung doch stark: Der Interrogator arbeitet mit Zustandsmaschinen, die jeden Teilnehmer (auch den Angreifer) modellieren, mit Netzwerkszustandsfolgen, die den eigentlichen Ablauf (zeitlich) der Protokollschritte simulieren und mit abstrakten kryptographischen Funktionen, die von der Semantik einer kryptographischen Funktion abstrahieren. Die Funktionen werden auf eine reine Syntaxveränderung der Eingabedaten auf Ausgabedaten heruntergebrochen.
Ausser dem Ansatz der abstrakten kryptographischen Funktionen hat das Longley und Rigby Tool nichts weiter mit der Umsetzung des Interrogator gemein - es arbeitet mit beschrifteten Suchbäumen.

2. Arbeitsweise des Longley und Rigby Tools

2.1. **Formalisierung der zu untersuchenden Daten.**

2.1.1. *Der Protokollablauf:* In einem 'normalen' Protokollablauf wird z.B. zu einem Zeitpunkt t eine spezifizierte Nachricht M vom Teilnehmer X zum Teilnehmer Y übertragen. Der Teilnehmer Y reagiert zum Zeitpunkt $t+1$ mit einer Antwort, einem Funktionsaufruf oder ähnlichem. Das wird im Longley und Rigby Tool so nicht modelliert.
Das Tool benötigt lediglich *alle* Nachrichten, die in einem kompletten Protokollablauf zwischen X und Y (und wen auch immer noch...) versendet wurden - es ist irrelevant zu welchem Zeitpunkt irgendeine Nachricht gesendet wurde. Dies kann man sich in der Praxis folgendermaßen vorstellen: Der Angreifer speichert jede Nachricht, die über das Netzwerk zwischen X und Y ausgetauscht wurde, und versucht erst die Nachrichten zu dechiffrieren, wenn die eigentliche Kommunikation schon lange vorbei ist (er aber die maximal verfügbare Information zum Dechiffrieren hat). Eine Nachricht ist dabei wie folgt zu verstehen: Eine Nachricht ist eine Folge von Termen. Wenn eine Nachricht aus mehreren Termen besteht, z.B.: M='k, $enc(k,x)$' (wobei k ein Schlüssel ist und enc(k,x) ein mit k verschlüsselter Text x ist), dann soll der Angreifer in der Lage sein, diese Terme auch aus der gesamten Nachricht extrahieren zu können. Das bedeutet, er kennt k und $enc(k,x)$. Allerdings könnte er aus M='$enc(k2,(k,enc(k,x)))$' auch nur $enc(k2,(k,enc(k,x)))$ herausfiltern.
Das Longley und Rigby Tool bekommt desweiteren alle Funktionen (seien es kryptographische Funktionen oder auch Hilfsfunktionen) als abstrakte Formel der folgenden Art übergeben:
Beispiel: Sei ENCRYPT eine Funktion, die aus einem Klartext X und einem Schlüssel K den chiffrierten Text $enc(K,X)$ erzeugt, dann ergibt sich für ENCRYPT folgende Formel:

$$ENCRYPT: \qquad enc(K,X) \leftarrow X\ AND\ K$$

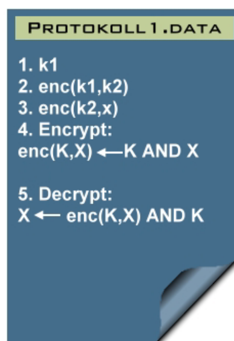
Es ist vielleicht gewöhnungsbedürftig, das Ergebnis links und die Eingabe rechts stehen zu haben, aber da das Werkzeug bei der Abarbeitung auch versucht, zu einem Wert als ein Ergebnis einer Funktion, die passenden Eingangswerte zu finden, ist die Schreibweise sehr viel angenehmer beim Verfolgen des Suchpfades.
Noch ein Hinweis zur Notation in diesem Skript: Grossbuchstaben bedeuten Variablen (also: 'K' steht für einen beliebig einsetzbaren Schlüssel) und Kleinbuchstaben bedeuten Instanzen von Variablen (also: '$k1$' wäre dann z.B. ein 'wirklicher' Schlüssel, mit dem man K belegen könnte).

2.1.2. *Modellierung des Angreifers:* Der Angreifer besitzt, wie bereits erwähnt, die gesamte Information, die während eines Protokolldurchlaufes versendet wird. Darüberhinaus hat er Zugriff auf jede der verwendeten Funktionen, da er als berechtigter Benutzer des Netzwerkes modelliert wird, und so auch um die Art der Verschlüsselungstechniken (z.B. DES) weiß. Es wird nicht angenommen, dass der Angreifer über Superuser-Rechte verfügt oder Zugriff auf netzwerkinterne höhere Schlüssel (diese werden vom Netzwerk zur Verschlüsselung tieferer Schlüssel benutzt, aber nie separat übertragen) hat.

2.1.3. *Modellierung einer Sicherheitslücke:* Das Werkzeug bekommt eine zu untersuchende Sicherheitslücke mitgeteilt, indem der Protokolldesigner aus der gegebenen Menge von übertragenen Nachrichten, den Teil einer Nachricht markiert, der auf keinen Fall durch einen Angreifer extrahiert werden soll. Zum Beispiel bei bei der Nachricht $M = enc(k, x)$ würde der Protokolldesigner den durch k codierten Klartext x markieren. Könnte der Angreifer x aus der Nachricht M durch Anwenden der kryptographischen Funktionen herausfiltern (z.B. wenn er k besäße), so wäre das eine Sicherheitslücke.

2.2. **Komponenten des Longley und Rigby Tools.**

2.2.1. *Die Wissensbasis:* Das System benötigt eine Datei, in der alle Daten stehen, die der Angreifer zur Verfügung hat. Zu diesen Daten gehören, wie bereits erwähnt, die kryptographischen Funktionen, alle übertragenen Nachrichten und einige Daten, von denen der Designer annehmen kann, dass sich der Angreifer sie sich leicht besorgen/generieren kann. Diese Datei enthält praktisch alle Ausgangsinformationen für einen Hack - sie sei als Wissensbasis bezeichnet.



2.2.2. *Der beschriftete Suchbaum.* Die eigentliche Suche nach Sicherheitslücken findet in der wichtigsten Datenstruktur dieses Werkzeuges statt: dem Suchbaum. Die Blätter dieses Baumes repräsentieren alle Daten, die der Angreifer zum Hacken zur Verfügung hat (also eine Teilmenge der Wissensbasis) und die Wurzel wird mit der durch den Protokolldesigner spezifizierten Sicherheitslücke initialisiert. Jeder Knoten des Baumes wird beschriftet: Alle Knoten, die Informationen enthalten, die in der Wissensbasis stehen, werden als 'Available' beschriftet und alle Knoten, deren Informationen noch nicht vorhanden sind, werden als 'Required' beschriftet. Daraus folgt, dass die Wurzel vor dem Suchen mit 'Required' beschriftet ist und dass jedes Blatt mit 'Available' markiert ist.

2.2.3. *Die erfolgreiche Suche:* Eine allgemeine Beschreibung des Suchbaumaufbaues ist zu umständlich, weswegen ich es gleich an einem Beispiel demonstrieren möchte: Sei unsere Wissensbasis mit folgenden Informationen gefüllt:

$k1$, $enc(k1, k2)$, $enc(k2, x)$, sowie mit den Funktionen:
$ENCRYPT:\quad enc(K, X) \leftarrow K\ AND\ X$
$DECRYPT:\quad X \leftarrow enc(K, X)\ AND\ K$

Unser Angreifer hat also bei der Übertragung der Nachrichten einen Schlüssel $k1$, einen mit $k1$ chiffrierten Schlüssel $k2$ und einen mit $k2$ chiffrierten Klartext x abgefangen und hat Zugriff auf die Ver- und Entschlüsselungsfunktion. Unser Angreifer hat sein Ziel erreicht, wenn er den Klartext x enthüllen kann (was in diesem Beispiel nicht wirklich schwer ist :-]).
Beim Aufbau des Suchbaumes passiert nun folgendes: Die Wurzel wird erstellt und bekommt als Wert die Information, die vor dem Angreifer gschützt werden soll (Also hier: x). Die Wurzel wird nun mit 'Required' beschriftet, da diese Information *so* nicht in der Wissensbasis enthalten ist. Da nun diese Information nicht in der Wissensbasis steht, muß sie also mit Hilfe einer kryptographischen Funktion entstanden sein. Da unsere Information kein Chiffrat ist (also nicht mit '$enc(...$' beginnt), kann nur die DECRYPT - Funktion das Mittel der Wahl sein. Die Variable X in DECRYPT wird nun also mit x belegt (unifiziert) und die Wurzel bekommt die zwei Eingangsdaten der Funktion als Nachkommen: Also $enc(K, x)$ (Bemerke: K konnte noch nicht belegt werden)und K. Nun wird die Wissensbasis erneut nach Übereinstimmungen durchsucht und es wird tatsächlich auch einen Term $enc(k2, x)$ gefunden. Nun wird K mit $k2$ unifiziert und der Knoten mit dem Wert $enc(k2, x)$ erhält die Beschriftung 'Available'. Der

andere neue Knoten K wird, weil es sich ja um den gleichen Schlüssel handelt, auch mit $k2$ unifiziert. Unglücklicherweise befindet sich aber in der Wissensbasis kein Schlüssel $k2$ - dieser Knoten muss also als 'Required' beschriftet werden. Nun wird, da nicht alle neuen Knoten als 'Available' deklariert sind, in den 'Required' Knoten fortgefahren. Jetzt betrachten wir also den Knoten mit den Wert $k2$: Dieser Wert muss wiederum durch eine kryptographische Funktion entstanden sein und das ist, unter der gleichen Begründung wie oben, die DECRYPT-Funktion. Der Knoten bekommt also wiederum die Nachfahren $enc(K, k2)$ und K. In der Wissensbasis ist nun ein Term $enc(k1, k2)$ zu finden, was dazu führt, dass K mit $k1$ unfiziert wird und der Knoten als 'Available' gilt. Auch der Geschwisterknoten K wird mit $k1$ unifiziert, ist in der Wissensbasis auffindbar(!) und wird somit als 'Available' bezeichnet. Da nun alle Nachkommen von $k2$ 'Available' sind, wird nun auch $k2$ 'Available' und da jetzt wiederum alle Nachkommen von x 'Available' sind, folgt x wird 'Available' $\Rightarrow$ die Suche nach einer Sicherheitslücke war erfolgreich (Darstellung des Suchbaumes in Abbildung 1a bis 1c).

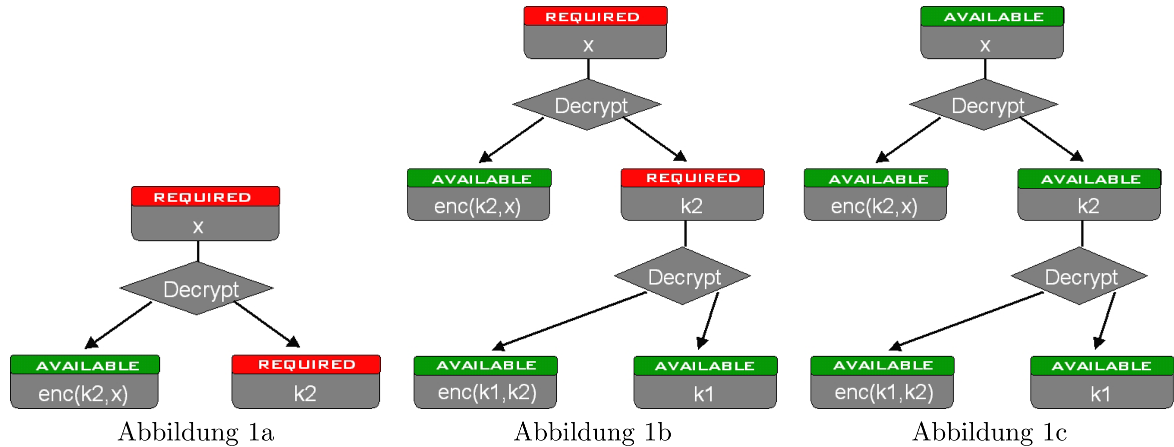


Abbildung 1a Abbildung 1b Abbildung 1c

Ausgegeben wird nach erfolgreicher Suche die Abarbeitungsvorschrift, die ein potentieller Angreifer vollziehen muß, um an die sicheren Daten zu gelangen. Also in unserem Fall:

(1) Decrypt: enc(k1,k2) with k1 giving k1.
(2) Decrypt: enc(k2,x) with k2 giving x.

2.2.4. *Die erfolglose Suche:* Sei unsere Wissensbasis mit folgenden Informationen gefüllt:

$enc(k1, k2)$, $enc(k2, x)$, sowie mit den Funktionen:
$ENCRYPT:\quad enc(K, X) \leftarrow K\ AND\ X$
$DECRYPT:\quad X \leftarrow enc(K, X)\ AND\ K$

Die Wissensbasis hat sehr viel Ähnlichkeit mit dem vorherigen Beispiel, enthält jedoch nicht $k1$ und wird somit auch nicht zum Erfolg des Angreifers führen. Die Erstellung des Wurzelknotens sowie deren Nachkommen ist identisch mit denen des vorangegangenen Beispieles. Der Term $enc(k2, x)$ wurde erfolgreich gefunden, und somit bekommt die Schlüsselvariable K eine Unifikation auf $k2$. Nun wird genauso wie vorher die DECRYPT-Funktion als potentieller Erzeuger von $k2$ ermittelt, und die Nachkommen $enc(K, k2)$ und K werden erzeugt. Da ein Term $enc(k1, k2)$ in der Wissensbasis existiert wird nun K mit $k1$ unifiziert. Der Knoten mit $enc(k1, k2)$ kann als 'Available' gekennzeichnet werden, jedoch der Knoten mit dem Wert $k1$ bleibt 'Required'.
Also wird erkannt, dass wiederum $k1$ nur aus der DECRYPT-Funktion entstanden sein kann, weil $k1$ nicht verschlüsselt vorliegt (das Fehlen von '$enc()$'). Es werden abermals Nachkommen erzeugt, welche die Werte $enc(K, k1)$ und K tragen. Nun gibt es keinen Term, der auf $enc(K, k1)$ matchen würde - d.h. K kann *nicht* unifiziert werden. Nun muss das Longley und Rigby Tool alles versuchen, die Variablen auf irgendeine Weise zu unifizieren, auf das die Terme irgendwann in der Wissensbasis zu finden sind. Dieses 'alles versuchen' umfasst das Unifizieren mit allen Informationen der Wissensdatei (also mit $enc(k1, k2)$ und $enc(k2, x)$) und mit Mehrfachanwendung der kryptographischen Funktionen (auch ohne unifizierte Variablen). Das alles führt leider zu einem sehr aufwendigen 'Probieren' mit einigen Spezialfällen, das ich, des großen Umfangs wegen, erst in 2.3 darstellen werde.

Auf jeden Fall sei bemerkt, dass das Werkzeug nicht in der Lage ist, solche Knoten als 'Available' zu kennzeichnen. In diesem Fall muss der nicht-erfüllbare Knoten mit seinem Schwesterknoten aus dem Suchbaum entfernt werden. Nun muss in der Wissensbasis nach einer neuen Funktion gesucht werden, durch die $k1$ entstanden sein könnte. Da es eine solche Funktion nicht mehr gibt, folgt, dass $k1$ 'Required' bleibt. Also muss auch $k1$ zusammen mit dem Schwesterknoten $enc(k1, k2)$ entfernt werden. Es ist aber wiederum keine neue Funktion zu dem Term $k2$ in der Wissensbasis auffindbar, also wird auch diese Knoten-Ebene entfernt, und x wird betrachtet. Da auch x durch keine andere als die $DECRYPT$-Funktion entstanden sein kann, bleibt x 'Required' $\rightarrow$ die Suche wird mit einer Meldung des Mißerfolgs abgebrochen (Darstellung des Suchbaumes in Abb. 2a bis 2d).

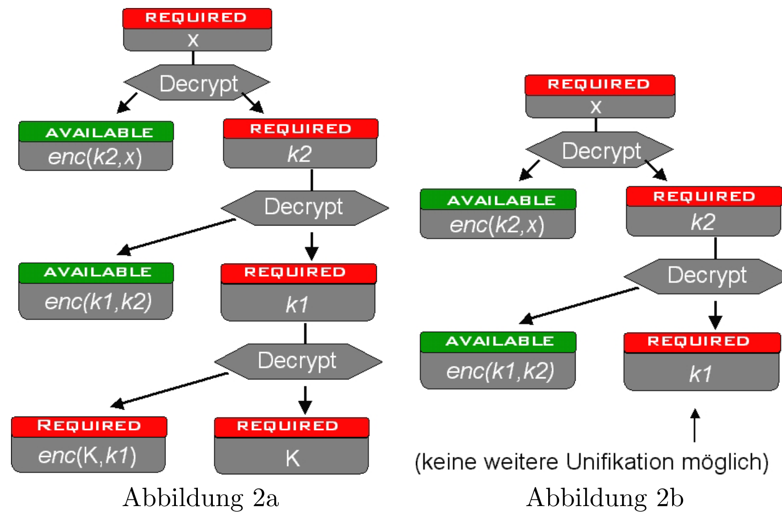


Abbildung 2a Abbildung 2b

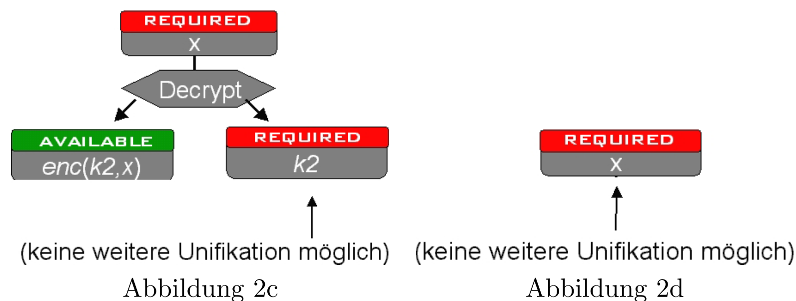


Abbildung 2c Abbildung 2d

Ausgegeben werden nach erfolgloses Suche alle die Knoten, die nur 'Required' - Nachkommen hatten. Also in unserem Fall: '$k1$ has failed.'

2.3. **Sonderfälle der Implementation des Packages.**

2.3.1. *Kreise:* Wenn ein Knoten einen identischen Vorfahren hat, dann entsteht ein Kreis. Das heißt, bei der Abarbeitung des Algorithmus wird immer wieder eine identische Schleife durchlaufen die nur beendet wird, wenn ein in der Schleife entstandener Term in der Wissensbasis gefunden werden kann und alle Variablen im Term unifiziert werden können. Sollte es so einen Term in der Wissensbasis nicht geben, so entsteht ein unendlicher Pfad $\Rightarrow$ das Programm bricht nicht ab. In diesem Ausnahmefall muß das Longley und Rigby Tool immer eine Schutzabfrage durchführen, bevor Knoten-Nachkommen erzeugt werden. Diese Schutzabfrage bezieht sich auf die Länge des Termes: Wenn so ein Kreis entsteht, dann werden die Terme immer länger, weil eine Funktion immer und immer wieder aufgerufen wird (z.B.: $enc(K, enc(K, enc(K, enc(K, x))))$). Um dem vorzubeugen, wird in der Wissensbasis die Länge l der längsten Funktions-Mehrfachausführung ermittelt. Ist die Länge des Termes im aktuellen Knoten $\geq l$, und der Term kann nicht in der Wissensbasis gefunden werden, so werden auch alle seine Nachkommen nicht in der Wissensbasis sein.

2.3.2. *Offene Instanzierung.* Dieser Sonderfall trat bereits im Beispiel der erfolglosen Suche auf: Es kann sein, dass für den aktuellen Knotenwert kein Term in der Wissensbasis gefunden werden kann. In diesem Fall darf aber nicht abgebrochen werden, da es vielleicht nach nochmaliger Ausführung mit einer

(oder einigen) kryptographischen Funktion(en), einen Term gibt, der auf den Knotenwert matched. Ein Beispiel dafür wäre folgende Wissensbasis:
$k1$, $enc(k1, k2)$, $enc(k2, x)$, $enc(t, enc(k1, enc(k2, x))$, t, sowie folgende Funktionen:
$Encrypt : enc(K, X) \leftarrow K\ AND\ X$
$Decrypt : X \leftarrow enc(K, X)\ AND\ K$

Es ergibt sich der folgende Suchbaum in der Abbildung 3a bis 3c:

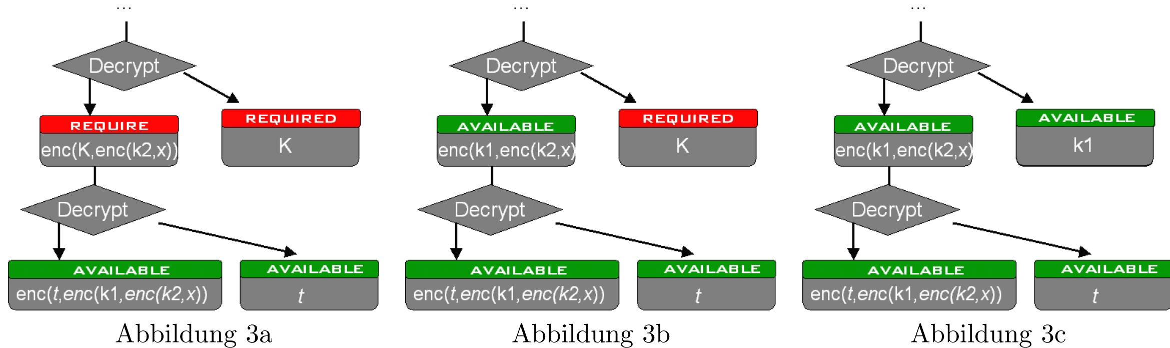


Abbildung 3a Abbildung 3b Abbildung 3c

Bemerke: In den vorherigen Beispielen habe ich einfach immer für offene Variablen Ausdrücke wie "K"geschrieben. Tatsächlich müsste es aber K_i ($i \in \mathbb{N}$) heißen. Warum? Nun die Mehrfachausführung der DECRYPT-Funktion liefert Terme wie z.B. $enc(K, enc(K, x)$. Dies suggeriert aber, dass der erste Schlüssel K in $enc(K, enc(..))$ identisch sein muss mit dem K in $..., enc(K, x))$ - was natürlich nicht stimmt. Korrekt wäre: $enc(K_2, enc(K_1, x))$.

2.3.3. *Synchrone Instanzierung.* Aus der Abbildung 3 ist gut ersichtlich, dass der Term $enc(K, enc(k2, x))$ dadurch, dass in seinem Nachfahrenknoten K auf $k1$ unifiziert wurde, sich nun auch in $enc(k1, enc(k2, x))$ ändern muss. Es ist dabei zu beachten, dass sein Geschwisterknoten auch K enthält und *synchron* dazu unifiziert werden muss. Dies mag trivial klingen, ist aber bei komplexeren (und tieferen) Bäumen kein leichtes Unterfangen, denn es darf nicht zuviel und nicht zu wenig Freiraum für die Suche gegeben werden.

3. Analyse vom Tagged Key Management Scheme

Ein 'Tagged Key Management Scheme' (Verwaltungsvorschrift für beschriftete Schlüssel) ist ein kryptographisches Protokoll, das auf hierarchischer Schlüsselverwaltung basiert. Existiert z.B. ein Schlüssel kt für die Übertragung von Daten, ein Schlüssel ks für die aktuelle Kommunikationssitzung (Session) und ein Masterschlüssel km, der einzig zur Verschlüsselung der beiden vorherigen Schlüsseln da ist und nie übertragen wird (er ist auf jeder Workstation im Netzwerk praktisch fest und unveränderbar gespeichert), dann spricht man von einer hierarchischen Schlüsselverwaltung (kt ist über ks und km ist über kt). Dabei werden Daten folgender Art im Netzwerk übertragen:

$$enc(km,\ kt) \qquad enc(kt,\ ks) \qquad enc(ks,\ 'Klartext')$$

3.1. **Definition des Netzwerkes.** Das hier zu untersuchende Protokoll wurde entwickelt für ein 'electronic funds transfer point of sale' (EFTPOS) Netzwerk, also einem Netzwerk, das primär für Transaktionen von Gelder gedacht ist und dementsprechend sicher sein muß. Dieses Protokoll wurde kurz vor der Implementation Longley und Rigby zur Untersuchung vorgelegt und es zeigte sich, dass es leider nicht den Sicherheitsüberprüfungen standhalten konnte.
Daraus folgte natürlich, dass es zu keiner Implementation kam und das gesamte Konzept überarbeitet werden musste. Offensichtlich scheinen aber schon Teile des Protokolles in anderen Netzwerken umgesetzt zu sein, da bei der Anfrage zur Veröffentlichung des Protokolles als Beispiel für das Longley und Rigby Tool, nur erlaubt wurde, die minimal mögliche Darstellung des Schema's zur Erläuterung des Fehlers zu veröffentlichen (siehe [1], Seite 83). Deshalb kann ich leider keine Abarbeitungsvorschrift für 2 Teilnehmer im Netz (...mit 'Wenn X *bla* und *blub* sendet, dann sendet Y...') präsentieren, da die Veröffentlichungen dazu nicht existieren. Das Protokoll liegt nur in der entsprechend formatierten Eingabe für des Longley und Rigby Tool vor.

Weil die Umsetzung dieses Schema's auf einem speziellen Netzwerk, nämlich einem EFTPOS - Netzwerk, laufen sollte, und dies auch im Protokoll ein wesentlicher Bestandteil ist, möchte ich an dieser Stelle ein paar Worte darüber verlieren: Beim EFTPOS wird stets eine Verbindung zwischen einem USER, also zum Beispiel einem Bankkunden, und einem HOST, zum Beispiel der Deutschen Bank hergestellt und verwaltet.

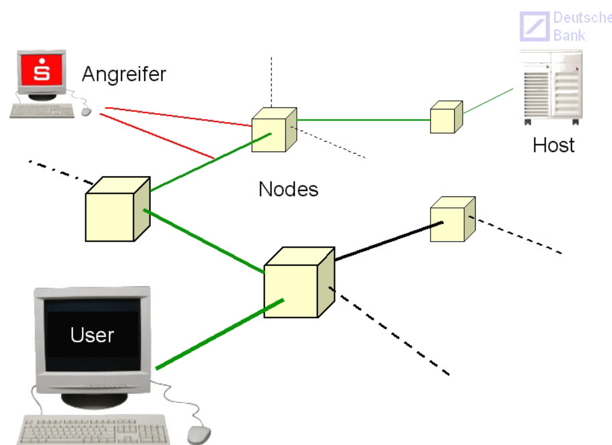


Abbildung 4

Zwischen dem USER und dem HOST befinden sich sogenannte NODES, also in der Praxis einfache Server, welche die physikalische Verbindung zwischen den Teilnehmer routen (In der Abbildung sei eine Verbindung die grünen Linien). Im USER-Rechner, in den NODES und im HOST-Server gibt es kryptographische Module, also irgendeine spezielle Hardware (in der Praxis vielleicht eine Steckkarte), die für die Ver- und Entschlüsselung der Daten verantwortlich ist, sowie einige Besonderheiten aufweist:

(1) Diese kryptographischen Module sind manipulationssicher. Das bedeutet: Wenn versucht wird, mit z.B. anderer Hardware die Funktionen und Schlüssel in den Modulen zu extrahieren und/oder Veränderungen daran durchzuführen, dann gehen diese Module eher kaputt als das dies möglich sei.
(2) Alle kryptographischen Funktionen werden von diesem Modulen ausgeführt. Wird ein Text unter einem Schlüssel entschlüsselt, um dann unter einem anderen Schlüssel verschlüsselt zu werden, so stellt das Modul sicher, dass zu keinem Zeitpunkt der Klartext offen liegt.
(3) Sollten neue Schlüssel in dem Netzwerk benutzt werden, so ist jede Node in der Lage, diese Schlüssel zur Nachbarnode weiter zu transferieren.
(4) Im gesamten Netzwerk gibt es *einen* Masterschlüssel, der in den Modulen sicher abgespeichert ist.

3.2. **Definition des Angreifers.** Über den Angreifer werden die Standardannahmen des Longley und Rigby Tools gemacht (siehe 2.1.2). Speziell für dieses Netzwerk hat er natürlich vollen Zugriff auf die Funktionen in den kryptographischen Modulen.

3.3. **Das Protokoll.** Wir haben nun kennengelernt, was eine hierarchische Schlüsselverwaltung von einer normalen Schlüsselverwaltung unterscheidet, jedoch benötigen wir offensichtlich 'Tagged Keys', also beschriftete Schlüssel. Warum?

3.3.1. *Beschriftete Schlüssel:* Hier ein Beispiel: Was auch immer in so eine Node für Funktionen vorhanden sind, es gibt auf jeden Fall eine Funktion der folgenden Form :

$$\text{DECRYPT: DATA} \leftarrow enc(km, \text{K}) \text{ AND } enc(\text{K, DATA})$$

Also eine Entschlüsselungsfunktion, die ein Chiffrat DATA unter einem Schlüssel K, welcher wiederum mit dem Masterschlüssel km chiffriert ist, entschlüsselt.
Gedacht ist diese Funktion für folgende Aufgabe:

$$'Klartext' \leftarrow \; enc(km, ks) \; AND \; enc(ks,' Klartext')$$

Der 'Klartext' wird unter dem Session-Schlüssel ks (so eine Art Sitzungsschlüssel, der bei jeder neuen Verbindung geändert wird) verschlüsselt übergeben, sowie das Chiffrat von ks unter dem Masterschlüssel km. Da alle notwendigen Daten korrekt sind, kann die Funktion den Klartext zurückgeben. Jedoch kann man leider diese Funktion auch missbrauchen:

$$ks \leftarrow enc(km, kt)\ AND\ enc(kt, ks)$$

Es wird nun der Session-Schlüssel als Chiffrat unter dem Übertragungsschlüssel kt und das Chiffrat vom Übertragungsschlüssel unter dem Master-Schlüssel übergeben. Jetzt kann der Angreifer sich ganz bequem den Session-Schlüssel ausgeben lassen. Die Grundidee der hierarchischen Schlüsselsysteme ist aber, das nie ein Schlüssel offen übertragen werden soll - jeder Schlüssel ist zumindest mit einem anderen Schlüssel chiffriert. Das führt nun zu einem Problem: Die Funktion ist wichtig und kann nicht entfernt werden, aber sie muss erkennen, ob sie ein Chiffrat mit ihrem Masterschlüssel entschlüsseln darf oder eben nicht....es könnten z.B. mehrere Masterkeys benutzt werden. Würde der obige Term nämlich nicht $enc(km, kt)$ lauten, sondern $enc(km2, kt)$, würde die DECRYPT - Funktion eine Müll-Ausgabe der folgenden Art werfen:

$$decrypt(enc(km2, kt), enc(kt, ks))$$

Ein anderer Ansatz stammt von R. W. Jones ('Some techniques for handling encipherment keys', ICL Tech. J., 3(2)(1982) 175-188): Er bemerkte, dass von den 64 Bit der verwendeten DES-Schlüssel, ja nur 56 Bits als eigentlicher Schlüssel dienen. Die restlichen 8 Bits werden normalerweise für Paritätsüberprüfungen benutzt. Wenn man diese 8 Bits als Beschriftung ('Tag') missbraucht (z.B. der Masterschlüssel hat den Code 0000 0001, der Session-Schlüssel den Code 0000 0010), dann benötigt man nicht mehrere Masterschlüssel. Die Funktion DECRYPT würde dann folgendermaßen implementiert werden:

(1) Entschlüssele $enc(km, K)$ mit km.
(2) Überprüfe, ob die Beschriftung von K den Wert '0000 0010' hat.
    (a) Bei Erfolg: Dann ist K der Sessionschlüssel. Entschlüssele $enc(K, X)$ mit K und gib' X zurück.
    (b) Bei Misserfolg: Gib eine Fehlermeldung aus und brich ab.

Der Angreifer kann nicht kontrolliert die Beschriftung beeinflussen, da jeder Schlüssel außerhalb der kryptographischen Module mit km verschlüsselt ist.
Notation: Im Weiteren werde ich die Beschriftung eines Schlüssels immer in eckigen Klammern hinter dem Chiffrat dieses Schlüssels notieren. Also: Der Schlüssel ks hätte durch Notation von $enc(km, ks)[kis]$ die Beschriftung 'kis'.

3.3.2. *Die kryptographischen Funktionen.* Wie bereits erwähnt, kann das Protokoll nur unvollständig erläutert werden. Die Idee hinter der einen oder anderen Funktion wird wohl nur völlig den Designern erschlossen bleiben, aber ich versuche trotzdem soviel wie möglich Zusammenhang zwischen den Funktionen herzustellen.

Funktion $ENCRYPT: \quad enc(KD, DATA) \leftarrow DATA\ AND\ enc(km, KD)[kds]$
Bemerkung: Verschlüsselt die Informationen $DATA$, die geheim bleiben sollen.

Funktion $DECRYPT: \quad DATA \leftarrow enc(KD, DATA)\ AND\ enc(km, KD)[kdr]$
Bemerkung: Entschlüsselt die geheime Information $DATA$ mit KD.

Funktion $GENERATE: enc(km, KG)[kds]\ AND\ enc(K, KG)[kdr] \leftarrow enc(km, K)[kis]$
Bemerkung: Erzeugt einen neuen Schlüssel KG und liefert als Rückgabe $enc(K, KG)[kdr]$ für die Nachbar-Nodes. Die Ausgabe $enc(km, KG)[kds]$ bleibt in der Node für die $ENCRYPT$ - Funktion (K wäre eine Art Übertragungsschlüssel, KG der Session-Schlüssel) .

Funktion $Received: enc(km, K2)[kdr] \leftarrow enc(km, K1)[kir]\ AND\ enc(K1, K2)[kdr]$
Bemerkung: Empfängt einen erzeugten Schlüssel von der Nachbar-Node. Der empfangene Schlüssel $K2$, der mit $K1$ chiffriert ist, wird in ein Chiffrat unter km übersetzt und in der Node abgespeichert (wird für $DECRYPT$ gebraucht).

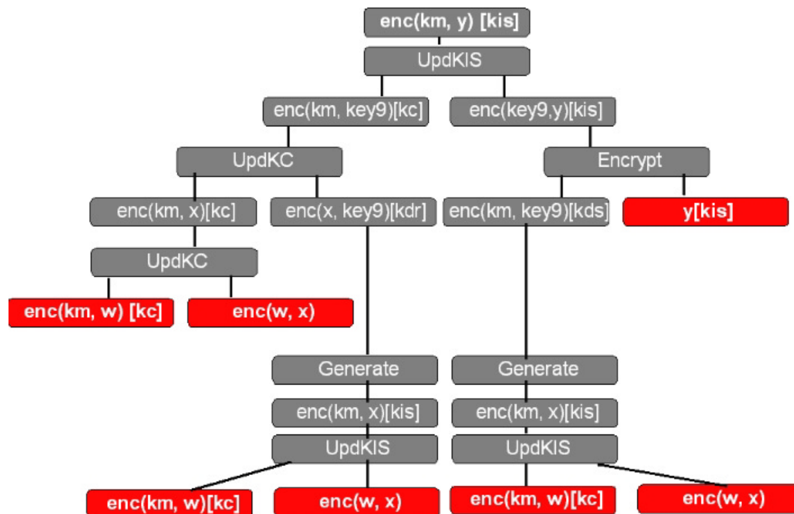
Update-Funktionen:
$UpdKC$: $enc(km, K)[kc] \leftarrow enc(km, KM)[kc]$ AND $enc(KM, K)$
$UpdKIS$: $enc(km, K)[kis] \leftarrow enc(km, KM)[kc]$ AND $enc(KM, K)$
$UpdKIR$: $enc(km, K)[kir] \leftarrow enc(km, KM)[kc]$ AND $enc(KM, K)$
Bemerkung: Diese Funktionen ersetzen einen alten Masterschlüssel KM mit dem neuen km. Treffen Schlüssel in der Node ein, die mit dem alten Masterschlüssel codiert sind, so werden diese mit dem neuen Masterkey chiffriert. Die verschiedenen Funktionen ergeben sich aus der Tatsache, dass die Ausgabe für eine spezielle krypt. Funktion vorgesehen ist: z.B. $UpdKIS$ liefert einen Schlüssel, der $GENERATE$ übergeben werden kann.

3.4. **Die Fehlersuche.** Nun haben wir alle Informationen, um das Longley und Rigby Tool in die Lage zu versetzen, das Protokoll auf Schwachstellen zu testen. Alle obigen Funktionen werden in die Wissensbasis geschrieben, zusammen mit den unter 2.2.3 beschriebenen elementaren Ver- und Entschlüsselungsfunktionen. Doch auf welche Schwachstelle hin soll das Protokoll untersucht werden? Der triviale Fall wäre $DATA$. Da aber das Protokoll in Bezug auf den Masterschlüssel km relativ robust ist, würde unsere Suche immer scheitern. Es muß also nach einem Term gesucht werden, der uns einen Zugriff auf $DATA$ erlaubt.So ein Term wäre $enc(km, y)[kis]$, wobei y ein Sclüssel ist, der dem Angreifer bekannt ist.
Denn wenn dieser Term erst einmal in einer Node vorhanden ist, dann haben alle danach mit $GENERATE$ erzeugten Schlüssel die Form $enc(y, newkey)[kdr]$. Der Schlüssel $newkey$ wird nun für die Datenverschlüsselung benutzt (ENCRYPT). Da aber der Angreifer den Schlüssel y kennt, er hat ihn ja eigenhändig erzeugt, kann er aus $enc(y, newkey)$ den Schlüssel $newkey$ extrahieren, und alle Nachrichten dechiffrieren. Nun stellt sich nur noch die Frage, wie der Angreifer einen Term $enc(km, y)[kis]$ in eine Node bekommt, von dem er y erzeugt hat aber km nicht kennt.
Genau diese Aufgabe wurde dem Longley und Rigby Tool gestellt: die Wurzel des Suchbaumes ist also $enc(km, y)[kis]$ mit der (Knoten-) Beschriftung 'Required'.

3.4.1. *Erster Suchdurchlauf:* Es wurde kein Pfad gefunden. Die Ausgabe der Knoten, bei denen keine Unifikation mehr möglich wurde, war: $enc(km, km)[kds]$ und $enc(km, X)[kc]$.
Der Term $enc(km, km)[kds]$ kann für den Angreifer unmöglich konstruierbar sein, aber die zweite Ausgabe brachte schon einen wichtigen Hinweis: Wenn der Angreifer also eine beliebigen Bitfolge (mit der Länge eines DES-Schlüssels) b erzeugt, die mit km decodiert, wieder eine zufällige Folge ergibt, aber im Paritätsteil den Code für die Beschriftung [kc] trägt, dann wäre $b = enc(km, X)[kc]$. Die Wahrscheinlichkeit, dass er genau so ein b erzeugt, ist (wegen der 8 Bit Beschriftung) $\frac{1}{256}$. Ein Angreifer wird sicherlich keine Probleme haben, bei einer Möglichkeit von 1 aus 256, die richtige Folge zu erzeugen. Die Wissensbasis wurde um den Term $enc(km, w)$ erweiter. Die Suche wurde erneut gestartet.

3.4.2. *Zweiter Suchdurchlauf:* Auch der zweite Suchdurchlauf war nicht erfolgreich. Die Anzahl der gescheiterten Knoten war grösser, unter anderen: $enc(km, X)[kc]$ und $enc(w, X)$.
Den ersten Term haben wir schon hinzugefügt. Der zweite Term ist wieder sehr vielversprechend: w ist ja der Schlüssel, der sich hinter unserer Zufallsbitfolge $enc(km, w)$ verbarg. Nun haben wir aber keinen Zugriff auf km - also können wir auch nicht w spezifizieren. Aber das ist auch egal. Denn: in $enc(w, X)$ wird ein beliebiger Text X mit w kodiert, also wenn wir wiederum eine Zufallsbitfolge (muß nicht einmal irgendwelche Beschriftungseigenschaften erfüllen) in das System einführen, dann ist der dechiffrierte Text nach Entschlüsselung mit w auch wieder eine Zufallsfolge - also ausreichend um X zu unifizieren. Wir fügen $enc(w, x)$ in die Wissensbasis ein und beginnen die Suche von vorn.

3.4.3. *Dritter Suchdurchlauf:* Der Suchvorgang vom Longley und Rigby Tool endet erfolgreich mit folgendem Suchbaum:



Der Suchbaum zum Tagged Key Management Scheme

Alle mit rot markierten Knoten stehen für Informationen, die der Angreifer in das System geschleust hat. Im Grunde ist nur der Knoten mit dem Term $y[kis]$ die einzige Information, die für den Angreifer brauchbar ist. Alle anderen roten Knoten enthalten nur Zufallsbitfolgen, die höchstens vielleicht eine Beschriftungseigenschaft erfüllen müssen. Es ist sehr interessant, wie redundant offensichtlich manche Daten sind, wenn $enc(km, y)[kc]$ erzeugt wird - die Zufallswörter x und w fallen einfach aus der Berechnung heraus.
Jetzt ist jeder Angreifer in der Lage, alle Nachrichten, die zwischen zwei Teilnehmern ausgetauscht wurden, zu entschlüsseln.

4. Zusammenfassung

Das Longley und Rigby Tool ist eines von vielen Verifikationsmittel im Bereich der Analyse von kryptographischen Protokollen. Es gehört sicherlich zu den wenigen Werkzeugen, die in dem Bereich auch Erfolge, also gefunde Fehler in Protokollen, vorweisen können. Dennoch hat es nicht wirklich die Welt erobert. Ich denke, dass die weitere Verbreitung der Analyse durch Modale Logiken (BAN) darauf zurückzuführen ist, dass nun einmal bei Softwarepackages wie dem Behandelten ein neues Protokoll mit einem völlig neuem Konzept sofort gewaltige Anpassungen an den Suchcode notwendig machen. Es genügt bei abweichenden Protokollkonzepten einfach nicht immer, die Wissensbasis durch Terme zu erweitern, während bei den Modalen Logiken wirklich nur neue Regeln hinzugefügt werden - die Analyse führt schneller zu Ergebnissen.
Dennoch, so glaube ich, muss man anerkennen, dass dieses Werkzeug einige wunderbare Eigenschaften hat, die jedem Protokolldesigner viel tiefere Einblicke in sein Protokoll verschaffen.

Vorteile	Nachteile
Der Suchpfad gibt (selbst bei Mißerfolg) einen tieferen Einblick in das Protokoll.	Nur vorher spezifizierte Angriffe werden untersucht (wie beim Interrogator)
Es sind 'Was wäre wenn?'-Versuche durchführbar.	Die Ausgangssituation von einem Angreifer ist nur schwer modellierbar.
Die krypt. Funktionen können sehr einfach modelliert werden (von der Arbeitsweise wird abstrahiert auf Eingabe- in Ausgabeumwandlung).	Es gibt keinen Vollständigkeitsbeweis.

Berlin, den 15. Januar 2003.

(Dieses Paper entstand im Rahmen des Seminars '*Analyse kryptographischer Protokolle*' am Institut für Informatik der Humboldt Universität zu Berlin.)

Literatur

(1) D. Longley, S. Rigby, '*An Automatic Search for Security Flaws in Key Management Schemes*', Computers and Security,11(1):75-90, 1992
(2) J.K.Millen, S.C.Clark, S.B. Freedman, '*The Interrogator: Protocol Security Analysis*', IEEE Transactions on Software Engineering, SE-13(2), 1987
(3) Reinhard Wobst, '*Abenteuer Kryptologie Methoden, Risiken und Nutzen der Datenverschlüsselung*', Addison-Wesley (Thema 'DES')
(4) Matthias Killat, '*The Interrogator*', Paper aus dem Seminar 'Analyse kryptographischer Protokolle' der Humboldt Universität zu Berlin: Institut für Informatik , 27. Dezember 2002
(5) Cathrine A. Meadows, '*Formal Verification of Cryptographic Protocols: A Survey*', IEEE, Center of High Assurance Computer Systems, Naval Research Laboratory (Washington DC)
(6) Xiaoyue Gong, Rong Fan, '*Überblick formaler Methoden bei der Protokollverifikation*', Proseminar 'IT Sicherheit', Technische Universität Darmstadt: Fachbereich Informatik: Fachbereich Programmiermethodik, 30 Juli 2001