The Interrogator

Matthias Killat

27. 12. 2002

Inhaltsverzeichnis

1 Einleitung

Allein die Sicherheit von Kryptoalgorithmen vermag es in der Zeit, in der Datenverkehr über das Internet eine wesentliche Rolle spielt, nicht mehr die Sicherheit von Daten zu gewährleisten. Oftmals bestehen die Sicherheitslücken im Versagen der benutzten kryptographischen Protokolle, was unmittelbar die Frage aufwirft, wie man denn die Sicherheit solcher Protokolle nachweisen, oder wenigstens Sicherheitslücken aufdecken kann.
Zu diesem Zweck wurden verschiedene Techniken entwickelt, wobei sich drei grundsätzliche Richtungen unterscheiden lassen. Da wäre zum einen der Ansatz mit modalen Logiken des Glaubens, zu dem die BAN-Logik und etwaige Weiterentwicklungen zählen. Zum anderen existieren Ansätze wie das Dolev-Yao Modell, die versuchen Nachrichtenmodifikationen und -übertragungen algebraisch zu formalisieren. Zu guter Letzt gibt es noch die Kategorie der Zustandsautomaten, in die zum Beispiel der NRL-Analyzer einzuordnen ist.
Der 1984 in einer ersten Version von J.K. Millen entwickelte Interrogator kombiniert diesen Zustandsautomatenansatz mit algebraischen Aspekten der Nachrichtenmanipulation. Das Besondere am Interrogator ist, dass auch der Angreifer in das Modell mit aufgenommen wird. Dabei wird, neben der Möglichkeit das ihm vertrauliche Daten in die Hände fallen können, auch berücksichtigt, dass er von anderen gesendete Daten verändern kann. Desweiteren unterstützt der Interrogator eine Vielzahl von algebraischen Operationen zur Nachrichtenmanipulation, die in späteren Versionen, wie der von 1987, noch erweitert wurde.
Hier soll nun die Funktionsweise des Interrogator-Modells erläutert werden.

2 Grundlagen des Modells

Um die Arbeitsweise des Interrogators formal zu erklären, muss zunächst die Repräsentation der Protokollteilnehmer und Nachrichtenübertragung festgelegt werden. Im Folgenden sei X eine Menge von Datenelementen mit der entsprechend zugehörigen Menge X^* aller Worte über X.

2.1 Protokollteilnehmer und Nachrichten

Definition 1 (Teilnehmer) *Ein Teilnehmer* (X, S, i, t) *in einem Protokoll ist ein Automat, der empfangene Nachrichten in gesendete Nachrichten umwandelt.*
Dabei ist :

- X *eine Menge von Datenelementen*
- $S \subseteq X^*$ *eine Zustandsmenge*
- $i \in S$ *der Startzustand*
- $t : S \times X^* \to S \times X^*$ *eine partiell definierte Transitionsfunktion*

Eine Nachricht $m \in X^*$ ist eine Folge von Datenelementen. Die Nachricht m wird in Zustand $s \in S$ genau dann akzeptiert, wenn t für (s, m) definiert ist. Eine solche Transition wird in der Form

$$(s, m) \to (s', m')$$

notiert.

Ein durch ein Protokoll gegebenes Netzwerk ist nun bestimmt durch eine Menge von Teilnehmern, die mit P bezeichnet wird. Einer dieser Teilnehmer wird in der Regel der Angreifer sein.

Definition 2 (Netzwerk) *Ein Netzwerk ist ein aus den Automaten der Menge seiner Teilnehmer P zusammengesetzter Automat. Ein Netzwerkzustand (N, M, K) ist definiert durch:*

- $N : P \rightarrow S$ *eine Funktion, die den gegenwärtigen Zustand eines Teilnehmers liefert*
- $M \in X^*$ *eine gerade gesendete Nachricht*
- $K \subseteq X$ *die Menge der dem Angreifer bekannten Daten*

Für eine Nachricht $M \in X^*$ wird mit $|M|$ die Menge der Datenelemente, die in M vorkommen, bezeichnet:

$$|M| = \{x \in X | M = vxw : \ v, w \in X^*\}$$

Im Startzustand eines Netzwerkes (N_0, M_0, K_0) gilt:

- $\forall p \in P : N_0(p) = i_p, \quad i_p$ Startzustand des zu p gehörigen Automaten
- $|M_0| \subseteq K_0 \subseteq X$

Offensichtlich ist es nach diesem Modell nötig, dass ein Teilnehmer eine Nachricht empfängt, bevor er antworten kann. Daher wird eine spezielle "Start-Nachricht" M_0 definiert, auf die nur der Initiator des Protokollablaufs reagiert, d.h. seine Transitionsfunktion ist für diese Nachricht in seinem Startzustand definiert.

2.2 Nachrichtenübertragung

Es existieren zwei verschiedene Arten der Nachrichtenübertragung, die als Zustandsübergänge des Netzwerkes definiert werden. Zum einen gibt die normale Übertragung der Nachricht, bei der ein Teilnehmer die erhaltene Nachricht mit einer neuen beantwortet, zum anderen besteht die Möglichkeit, dass der Angreifer die derzeitige Nachricht durch eine eigene ersetzt.

Definition 3 (normale Nachrichtenübertragung) *Eine Netzwerktransition $(N', M', , K') \rightarrow (N, M, K)$ ist eine normale Nachrichtenübertragung, wenn ein Empfänger $p \in P$ existiert, so dass:*

- $N(q) = N'(q) \quad \forall q \in P, q \neq p$
- $t_p((N'(p), M') = (N(p), M)$
- $K = K' \cup |M'|$

Die erste Festlegung bedeutet nur, dass die Zustände aller anderen Parteien erhalten bleiben. Als zweites wird gefordert, dass für den Teilnehmer p eine entsprechende Antwort M auf M' durch seine Transitionsfunktion definiert ist. Die letzte Bedingung hat zur Folge, dass die in M übertragenen Datenelemente dem Angreifer nun bekannt sind. Das bedeutet also, dass ein Datenelement dem Angreifer genau dann bekannt ist, wenn es in einer Nachricht übertragen wurde.

Definition 4 (Nachrichtenmodifikation) *Eine Netzwerktransition* $(N', M', K') \to (N, M, K)$ *ist eine Nachrichtenmodifikation durch Nachricht* $M' \in X^*$, *wenn:*

- $N(p) = N'(p) \quad \forall p \in P$
- $K = K'$
- $|M| \subseteq K$

Das bedeutet, dass die Zustände aller Teilnehmer und das Wissen des Angreifers erhalten bleiben und die ursprüngliche Nachricht M durch eine Nachricht M' aus dem Angreifer bekannten Daten ersetzt wird. Dabei ist hier zu beachten, dass der Angreifer bei dieser Art der Übertragung die Daten aus der ursprünglichen Nachricht nicht erfährt. Dies kann aber durch Kombination mit einer normalen Nachrichtenübertragung erreicht werden.

Definition 5 (Net History) *Eine Net History, die in den Zustand* (N, M, K) *führt, ist eine Folge von Netzwerkzuständen* $H = ((N_i, M_i, K_i)|i = 0, \ldots, n)$, *so dass*

$$(N_i, M_i, K_i) \to (N_{i+1}, M_{i+1}, K_{i+1}), \quad i = 1, \ldots, n-1$$

und $(N_n, M_n, K_n) = (N, M, K)$.

Die Net History zu einem Zustand (N, M, K) gibt den Protokollverlauf an, der vom Startzustand zu diesem Zustand führte, mit anderen Worten ein Pfad vom Startzustand in den Zustand (N, M, K).

Definition 6 (Message History) *Ist* $H = ((N_i, M_i, K_i)|i = 0, \ldots, n)$ *eine Net History, so heißt die durch* H *induzierte Folge* (M_i) *Message History.*

Eine Message History ist also der Nachrichtenverlauf, der bei Netzwerktransitionen gemäß einer in den Zustand (N, M, K) führenden Net History entsteht.

2.3 Bemerkungen zum Modell

In diesem Modell wird nicht das genaue Aussehen einer übertragenen Nachricht festgelegt, da dies für die Analyse nicht notwendig ist. Es wird jedoch angenommen, dass Sender und Empfänger der Nachricht in dieser an den ersten beiden Positionen auftauchen. Desweiteren wird davon ausgegangen, dass eine von einem Teilnehmer gesendete Nachricht allen anderen zugänglich ist, theoretisch also jeder antworten könnte. Die Zustände der Teilnehmer und damit auch des Netzwerks sind ebenfalls nur abstrakt definiert und hängen wesentlich vom zu analysierenden Protokoll ab.

2.4 Modellierung der Verschlüsselung

Bisher wurde nur der Fall betrachtet, das der Angreifer Nachrichten beliebiger Parteien abfangen kann, er also die Daten in dieser Nachricht erfährt. Nun ist aber bei derartigen Protokollen anzunehmen, das Daten verschlüsselt übertragen werden, und der Angreifer nur die Chiffre erhält. Wie bei allen derartigen Analysemethoden werden die Verschlüsselungsalgorithmen selbst als sicher angesehen, d.h. der Angreifer kann den Klartext nur lesen, wenn er den richtigen Schlüssel besitzt und den angewendeten Algorithmus kennt. Dabei wird davon ausgegangen, dass Ver- bzw. Entschlüsselungsalgorithmen allgemein bekannt sind.

Ein Entschlüsselungsalgorithmus wird als ein Teilnehmer definiert, der bei Empfang einer Nachricht, die aus Schlüssel und Chiffre besteht, den entsprechenden Klartext sendet. Das heißt der Angreifer kann eine empfangene Nachricht und ein Datenelement aus seiner Wissensbasis K an diese Blackbox senden. Falls dieses Datenelement der richtige Schlüssel war, hat der Angreifer Kenntnis vom Klartext und die Vertraulichkeit dieser Nachricht wurde verletzt. Welche Ver- bzw. Entschlüsselungsalgorithmen für die Analyse von Bedeutung sind und somit in die Beschreibung des Protokolls aufgenommen werden müssen, hängt vom Protokoll ab. Im allgemeinen werden dies jedoch die von Sender und Empfänger verwendeten Algorithmen sein.

3 Arbeitsweise des Interrogators

Die Arbeitsweise des Interrogators besteht darin, dass man, nachdem das Protokoll spezifiziert wurde, einen unsicheren Zustand vorgibt. Der Interrogator versucht dann eine Net History, d.h. einen Pfad vom Startzustand in diesen Zustand, zu finden. Wird der Zustand erreicht, so ist eine Sicherheitslücke in diesem Protokoll vorhanden.

Ein Zustand gilt als unsicher wenn die Vertraulichkeit (Confidentiality) der Daten verletzt wurde oder nicht sichergestellt ist, dass ein Datenelement zu diesem Zeitpunkt eine gewünschte Eigenschaft hat (Authentication). Diese Eigenschaft kann zum Beispiel darin bestehen, dass ein Empfänger sicher sein will, dass die Nachricht von einem speziellen anderen Teilnehmer gesendet wurde.
Es läßt sich zeigen, dass Confidentiality und Authentication äquivalent sind, d.h. die jeweiligen Problemstellungen lassen sich ineinander überführen.

Will man feststellen ob ein Zustand erreichbar ist, so kann man ein neues Datenelement und eine Zustandüberführung hinzufügen, so dass dieses Datenelement genau dann gesendet wird, wenn der Zustand erreicht wurde.
Umgekehrt fügt man bei der Fragestellung ob ein Datenelement dem Angreifer bekannt werden kann einen weiteren Teilnehmer hinzu, der auf Nachrichten die dieses Datenelement enthalten antwortet, und daraufhin in einen ausgezeichneten Zustand übergeht. Es ist dann zu prüfen, ob dieser Zustand erreichbar ist.

Diese Äquivalenz läßt es zu, nur eines dieser Probleme zu betrachten. Um den Suchvorgang nach Pfaden in derartige Zustände zu automatisieren, wurde der Interrogator mit Prolog implementiert.

3.1 Die Such-Logik

Definition 7 *Die Vertraulichkeit (Confidentiality) eines Datenelementes $x \in X$ ist verletzt, wenn eine Menge K bekannter Daten derart existiert, so dass ein Zielzustand (N, M, K) vom Startzustand erreichbar ist, in dem $x \in K$ ist.*

Um diesen Sachverhalt in der Programmiersprache Prolog auszudrücken, wird ein Prädikat pKnows(x, H, N, M) in rekursiver Weise unter Zuhilfenahme weiterer Prädikate definiert, wobei H die zum Zustand (N, M, K) gehörige Message History ist. Dieses Prädikat soll genau dann wahr sein, wenn x dem Angreifer im Zustand (N, M, K) bekannt ist.

Definition 8 (pKnows) *Das Prädikat pKnows ist definiert durch:*

$$
\begin{aligned}
&\mathit{pKnows}(x, H, N, M) \iff \\
&(H = [], N = N_0, M = M_0, x \in K_0) \lor \\
&(x \in |M|,\ \mathit{reachable}(H, N, M)) \lor \\
&(\exists H', N', M' : \mathit{net\_\ trans}(H', N', M', N, M) \land \mathit{pKnows}(x, H', N', M') \land H = H'[M])
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{reachable}(H, N, M) \iff \\
&(H = [], N = N_0, M = M_0) \lor \\
&((\exists H', N', M' : H = H'[M]) \land \mathit{net\_\ trans}(H', N', M', N, M) \land \mathit{reachable}(H', N', M'))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{net\_\ trans}(H', N', M', N, M) \iff \\
&((\exists p \in P : \mathit{transition}(N'(p), M', N(p), M)) \land (q \neq p \Longrightarrow N'(q) = N(q)) \lor \\
&(N = N', \forall x \in |M| : \mathit{pKnows}(x, H', N', M'))
\end{aligned}
$$

Der erste Fall von pKnows tritt ein, wenn x schon in der Ausgangswissensbasis K_0 enthalten war. Im zweiten Fall ist x ein Datenelement der letzten Nachricht und der Zustand ist erreichbar, sonst muss x in einer Nachricht eines vorhergehenden Zustandes enthalten gewesen sein.
Ein Zustand ist gemäß reachable erreichbar, wenn er Startzustand ist oder eine Transition von einem erreichbaren Zustand in diesen Zustand existiert.
Durch net_ trans wird schließlich eine Netzwerktransition, also entweder eine normale Übertragung oder eine Nachrichtenmodifikation, definiert. Die vom Protokoll abhängige Transitionsfunktion muss durch das Prädikat transition vom Benutzer vorgegeben werden. Dabei existiert eine Transition von (s', m') in (s, m) genau dann, wenn transition(s', m', s, m) wahr ist.

Theorem 1 *$\mathit{pKnows}(x, H, N, M)$ ist genau dann wahr, wenn eine Net History existiert, die in den Zustand (N, M, K) führt bei der $x \in K$ ist.*

Der Beweis dieses für die Analyse zentralen Satzes soll hier nicht geführt werden, er ist jedoch in der angegebenen Literatur nachzulesen. Dieses Theorem ist deshalb von so großer Bedeutung, weil es im Grunde die Korrektheit des Modells bzw. Algorithmus zeigt.

Die oben definierten Prädikate werden dann in eine Prolog-Form gebracht und um einige weitere für den Programmablauf wichtige, Prädikate ergänzt, die unter anderem Mengen- und Listenoperationen ermöglichen. Desweiteren werden noch einige Vorkehrungen getroffen, um unendliche Schleifen bei der Suche zu vermeiden, wie zum Beispiel per Konvention festgelegt, dass nur maximal eine Nachrichtenmodifikation in Folge auftreten kann, was am Endresultat nichts ändert.

Der Benutzer gibt dann nach Spezifikation des Protokolls mittels des Transitionsprädikates einen unsicheren Zustand vor, der auf Erreichbarkeit getestet werden soll. Das Programm versucht nun einen Pfad zum Startzustand zu finden. Gibt es mehrere Transitionsmöglichkeiten, muss der Benutzer eine von diesen wählen. Wenn sich damit kein Pfad zum Startzustand konstruieren läßt und noch Alternativen existieren, muss der Benutzer erneut eine Wahl treffen, bis alle Auswahlmöglichkeiten ausgeschöpft wurden, da das Programm nur über eingeschränktes Backtracking verfügt.
Wurde ein Pfad in einen unsicheren Zustand gefunden, so ist das gesamte Protokoll unsicher. Im Fall, dass das Programm abbricht und keinen Pfad findet, stellt dieser Zustand keine Sicherheitslücke dar. Andernfalls läßt sich keine Aussage treffen.

3.2 Lösen von Gleichungen

Dadurch, dass bei der Angabe der Transitionsfunktionen der einzelnen Teilnehmer im allgemeinen Funktionssymbole (der Ver- und Entschlüsselungsfunktionen) auftauchen, ist der Interrogator gezwungen, bei der Suche nach vorhergehenden Zuständen verschiedene Gleichungen zu lösen.

Nachfolgend stehen, ähnlich zu Prolog, Großbuchstaben für Variablen, Kleinbuchstaben für Konstanten, und eckige Klammern kennzeichnen Listen bzw. Sequenzen von Daten. Angenommen ein Teilnehmer p empfängt im Zustand s eine Nachricht $[K, A]$, entschlüsselt A mit K und sendet das Resultat. Eine derartige Transition sähe folgendermaßen aus:

$$(s, [K, A]) \rightarrow (s', [d(K, A)])$$

Hierbei ist d die zur Verschlüsselungsfunktion e gehörige Entschlüsselungsfunktion.

Beim Aufruf von

$$\text{pKnows}(x, H, N, M) \quad \text{mit}\, N(p) = s' \,\text{und}\, M = [b]$$

versucht der Interrogator im nichttrivialen Fall eine Menge von Vorgängerzuständen zu bestimmen, von denen $(s', [b])$ mit einem Transitionsschritt erreichbar ist. Dazu muss folgende Gleichung gelöst werden:

$$b = d(K, A)$$

Die Lösung $A = e(K, b)$ kann gefunden werden, indem man von der Tatsache das $d = e^{-1}$ ist Gebrauch macht:

$$X = d(Y, e(Y, X))$$

Man benutzt nun den Unifikator $\{Y = K, A = e(K, X)\}$, um beide rechten Seiten zu unifizieren. Dadurch wird die Ausgangsgleichung auf

$$b = d(K, e(K, X)) = X$$

reduziert. Dies liefert als Lösung

$$A = e(K, b)$$

Das Lösen der Gleichungen durch Unifikation und Termersetzung funktioniert leider nicht immer. Sind zum Bespiel kommutative Operationen vorhanden, so ist offensichtlich Unifikation nicht ohne weiteres allgemein möglich. Ein Beispiel dafür wäre eine Transition mit bitweisem XOR:

$$(s, [K, A]) \rightarrow (s', [\text{xor}(K, A)])$$

In diesem Fall existieren zwar ebenfalls Termersetzungsregeln, sie sind jedoch wesentlich komplizierter. Da die Theorie des allgemeinen Lösens solcher Gleichungen hier nicht Gegenstand sein soll, sei hier nur Folgendes gesagt. Der Interrogator besitzt für diese Fälle kommutativer Operatoren spezielle Routinen, die zum Beispiel prüfen, ob Argumentlisten durch Permutationen ineinander übergehen und wendet dementsprechende Regeln an.

Noch komplizierter wird es, wenn solche Operatoren kombiniert auftreten, d.h. durch Komposition verknüpft werden, wie es in diesem Beispiel der Fall ist:

$$e(\text{xor}(a, b), d(\text{xor}(b, A), c)) = c$$

Die Regel $e(Y, d(Y, X))$ läßt sich hier nicht ohne weiteres anwenden, denn $\text{xor}(a, b)$ und $\text{xor}(b, A)$ sind nicht unifizierbar. Der Interrogator benutzt nun die bereits erwähnten Regeln für kommutative Operationen und findet die Substitution $\{A = a, X = c, Y = \text{xor}(a, b)\}$ und erhält damit:

$$e(\text{xor}(a, b), d(\text{xor}(b, a), c)) = c$$

Abschließend bleibt dazu zu sagen, dass dies nicht immer notwendigerweise zu Lösungen führt, da die Ersetzungsregeln zum Teil auf Heuristiken basieren. Allerdings wird damit ein Großteil praxisrelevanter Fälle abgedeckt.

4 Anwendungsbeispiel Purdy-Simmons-Studier

Im Jahre 1982 wurde ein Protokoll zur Verteilung von Software von Purdy, Simmons und Studier veröffentlicht. Mit diesem Protokoll kann ein Softwarehändler seine Software in verschlüsselter Form an Kunden senden und nur denjenigen zugänglich machen, die sich registrieren lassen bzw. sie bezahlen.
Dazu erhält jeder Kunde ein "Software Protection Module" (SPM), dass einen speziellen, für jeden Kunden einzigartigen Schlüssel enthält. Um die Software zu entschlüsseln, bekommt der Benutzer einen weiteren Schlüssel, der nur mit diesem SPM zusammenarbeitet und ihn in die Lage versetzt, die Software zu decodieren.
Das Protokoll hat jedoch eine Schwäche, die zuerst von Simmons entdeckt wurde. Hier soll nun gezeigt werden, wie der Interrogator diese Schwachstelle findet, genauer gesagt von dem kritischen Zustand zeigt, dass er erreichbar ist.

4.1 Spezifikation des Protokolls

Das SPM besteht aus den drei Funktionen generate, encrypt und decrypt. Die generate-Funktion kombiniert zwei Schlüssel zu einem neuen und die encrypt- und decrypt-Funktionen ver- bzw. entschlüsseln angegebene Daten mit einem Schlüssel. In allen Funktionen wird auch der jeweilige, dem SPM eigene, einzigartige Schlüssel verwendet.

- $\text{generate}(A, B) = e(m, xor(q, d(m, A), d(m, B)))$
- $\text{encrypt}(A, B) = e(xor(q, d(m, A)), B)$
- $\text{decrypt}(A, B) = d(xor(q, d(m, A)), B)$

Dabei ist m ein globaler Schlüssel und q der Schlüssel des SPM's.

Der Händler verschlüsselt zuerst die Software s mit seinem SPM, das den Schlüssel qv besitzt, mit Schlüssel j und sendet die so entstandene Chiffre $s' := e(\text{xor}(qv, d(m, j)), s)$an alle Kunden:

vendor $\rightarrow$ customers: $e(\text{xor}(qv, d(m, j)), s)$

Wenn ein Kunde die Software nutzen möchte, muss er $e(m, qc)$ an den Händler zurücksenden (und die Software bezahlen), wobei qc der einzigartige Schlüssel des SPM's des Kunden ist.

customer $\rightarrow$ vendor: $e(m, qc)$

Der Händler generiert mit der generate-Funktion seines SPM aus $e(m, qc)$ und j

$$e(m, \text{xor}(qv, d(m, e(m, qc)), d(m, j))) = e(m, \text{xor}(qv, qc, d(m, j))) =: K$$

den Schlüssel zum decodieren der Software mit dem SPM des Kunden und sendet ihn zurück.

vendor $\rightarrow$ customer: $e(m, \mathrm{xor}(qv, qc, d(m, j)))$

Der Kunde wendet nun die decrypt-Funktion seines SPM's mit diesem Schlüssel auf die Software s an, und erhält so den Klartext der Software.
Um das zu sehen, setzt man diesen Schlüssel und die codierte Software s' in die decrypt-Funktion des Kunden ein:

$$
\begin{aligned}
& decrypt(K, s') \\
= \ & d(xor(qc, d(m, K)), s') \\
= \ & d(xor(qc, d(m, e(m, \mathrm{xor}(qv, qc, d(m, j))))), e(\mathrm{xor}(qv, d(m, j)), s)) \\
= \ & d(xor(qc, d(m, e(m, \mathrm{xor}(qc, \mathrm{xor}(qv, d(m, j)))))), e(\mathrm{xor}(qv, d(m, j)), s)) \\
= \ & d(xor(qc, d(m, e(m, \mathrm{xor}(qc, U)))), e(U, s)) \\
= \ & d(xor(qc, \mathrm{xor}(qc, U)), e(U, s)) \\
= \ & d(xor(U, \mathrm{xor}(qc, qc)), e(U, s)) \\
= \ & d(xor(U, 0), e(U, s)) \\
= \ & d(U, e(U, s)) \\
= \ & s
\end{aligned}
$$

Dabei wurde Kommutativität, Assoziativität, 0 als neutrales Element des bitweisen XOR und $d(Y, e(Y, X)) = X$ benutzt.

4.2 Spezifikation für den Interrogator

Als Angreifer wird hier der Kunde x betrachtet, der versucht die Software s ohne Autorisierung zu benutzen. Um die Schwachstelle des Protokolls zu finden, wird angenommen, dass der Angreifer alle angegebenen Übertragungen zu einem anderen Kunden c abhören konnte und somit gilt:

$$\{e(\mathrm{xor}(qv, d(m, j)), s),\ e(m, qc),\ e(m, \mathrm{xor}(qv, qc, d(m, j)))\} \subset K_0$$

Als nächstes muss die Transitionsfunktion für die SPM Funktionen des Angreifers in Prolog Form angegeben werden. Dabei hat das SPM genau einen Zustand z und das Prädikat solve(S, T) versucht Lösungen für die Gleichung

$$S = T \quad S,\ T \text{ Terme mit Variablen}$$

zu finden, d.h. die in S und T vorkommenden Variablen an feste Werte zu binden, die die Gleichung erfüllen.

Die Transitionsfunktion sieht dann wie folgt aus:

$\mathrm{transition}(z, [x, \mathrm{spm}, \mathrm{generate}, A, B], z, [spm, x, C]) :- \mathrm{solve}(C, e(m, \mathrm{xor}(qx, d(m, A), d(m, B))))$.

$\mathrm{transition}(z, [x, \mathrm{spm}, \mathrm{encrypt}, A, B], z, [spm, x, C]) :- \mathrm{solve}(C, e(\mathrm{xor}(qx, d(m, A)), B))$.

$\mathrm{transition}(z, [x, \mathrm{spm}, \mathrm{decrypt}, A, B], z, [spm, x, C]) :- \mathrm{solve}(C, d(\mathrm{xor}(qx, d(m, A)), B))$.

4.3 Suche nach einem Pfad

Nun da der Interrogator alle nötigen Daten kennt, stellt sich die Frage, ob der Kunde x die Software s entschlüsseln kann, ohne selbst einen (individuellen) Schlüssel vom Händler erhalten zu haben. Dies geschieht durch den Aufruf von pKnows:

$$\text{pKnows}(s, H, z, [spm, x, s])$$

Da die Betrachtung aller Aufrufe recht zeitraubend ist und vor allem das Suchen der Lösungen mittels des Prädikates solve sehr komplex und rechenintensiv ist, soll sich hier auf die wesentlichen resultierenden Aufrufe, nämlich diejenigen die dazu führen das s bekannt ist, beschränkt werden. Dies genügt, da damit die Existenz eines solchen Pfades bewiesen wird. Dabei wird der Übersicht halber auf die Message History verzichtet.

$\text{pKnows}(s, H, z, [spm, x, s])$

$\quad \text{reachable}(H, z, [spm, x, s])$

$\quad\quad \text{net\_trans}(H', N', M', z, [spm, x, s])$

$\quad\quad\quad \text{transition}(N', M', z, [spm, x, s])$

Die ersten beiden Möglichkeiten führen zu keinem Ergebnis, nur die decrypt-Funktion führt später zu einer Lösung.

$\quad\quad\quad\quad \text{solve}(s, d(\text{xor}(qx, d(m, A)), B)$

$B := e(\text{xor}(qx, d(m, A)), s)$ (Bindung von Variablen)
$N' := z,\ M' := [x, spm, decrypt, A, B]$

$\quad\quad \text{reachable}(H', N', M')$

$\quad\quad\quad \text{net\_ trans}(H'', N'', M'', N', M')$

$\quad\quad\quad\quad \text{pKnows}(B, H'', N'', M'')$

B ist die in K_0 enthaltene verschlüsselte Software, wenn $A := e(\text{xor}(qx, qv, d(m, j)))$ gesetzt wird. Da aber auch A in der Nachricht M'vorkommt, wird pKnows auch für A aufgerufen:

$\quad\quad\quad\quad \text{pKnows}(A, H'', N'', M'')$

$\quad\quad\quad\quad\quad \text{reachable}(H'', N'', M'')$

$A \in |M''|$ testweise annehmen

$\quad\quad\quad\quad\quad\quad \text{net\_ trans}(H''', N''', M''', N'', M'')$

$\quad\quad\quad\quad\quad\quad\quad \text{transition}(N''', M''', N'', M'')$

$C := A$ setzen (Prolog Suchlogik)

$$\text{solve}(C, e(m, \text{xor}(qx, d(m, A'), d(m, B')))))$$

Durch Backtracking wird zuerst die generate-Funktion ausprobiert, was zu den Lösungen $A' = e(m, qc)$ und $B' = e(m, \text{xor}(qv, d(m, j), qc))$ führt.

Analog liefert weiteres Backtracking, dass A' und B' in K_0, also bekannt sind. Es sind genau die Daten, die beim Datenverkehr des Kunden c mit dem Händler v abgefangen wurden. Das bedeutet, dass der anfangs angegebene Zustand erreichbar

ist, d.h. das Protokoll ist unsicher, denn x kann s decodieren, ohne vom Händler den Schlüssel erhalten zu haben.
Der vollständige entstehende Nachrichtenverlauf zwischen dem Kunden x und seinem SPM ist hier dargestellt:

$$x \;\rightarrow\; \text{SPM} \;:\; \text{generate}(e(m, qc), e(m, \text{xor}(qv, d(m, j), qc))$$

$$\text{SPM} \;\rightarrow\; x \;:\; e(m, \text{xor}(qx, qv, d(m, j)))$$

$$x \;\rightarrow\; \text{SPM} \;:\; \text{decrypt}(e(m, \text{xor}(qx, qv, d(m, j))), e(\text{xor}(qv, d(m, j), s))$$

$$\text{SPM} \;\rightarrow\; x \;:\; s$$

Die nachfolgende Rechnung zeigt nochmal, dass Kunde x mit den Daten von Kunden c einen Schlüssel generieren kann, mit dem sein SPM die codierte Software entschlüsselt.

$$\begin{aligned}
&\text{generate}(e(m, qc), e(m, \text{xor}(qv, d(m, j), qc)) \\
&= \; e(m, \text{xor}(qx, d(m, e(m, qc)), d(m, e(m, \text{xor}(qv, d(m, j), qc))))) \\
&= \; e(m, \text{xor}(qx, qc, \text{xor}(qv, d(m, j), qc))) \\
&= \; e(m, \text{xor}(qx, qc, qv, d(m, j), qc)) \\
&= \; e(m, \text{xor}(qx, 0, qv, d(m, j))) \\
&= \; e(m, \text{xor}(qx, qv, d(m, j)))
\end{aligned}$$

Dies ist, wie oben bereits gezeigt wurde, der Schlüssel, den Kunde x benötigt, um die Software mit seinem SPM zu decodieren.

5 Schlussbemerkungen

Wir haben gesehen, wie sich durch Kombination eines Automatenansatz und algebraischen Operationen auf recht einfache Art und Weise ein Modell zur Analyse kryptographischer Protokolle konstruieren lässt. Vorteile bei dieser Vorgehensweise liegen in der klaren Trennung von Zustandsautomat und algebraischem Gleichungsansatz, was eine einfache Erweiterbarkeit durch neue Operatoren ermöglicht. Durch die Art der Simulation des Angreifers, vor allem der Nachrichtenmodifikation, wird ein breites Spektrum von Angriffen abgedeckt.
Die Prolog-Implementation ist allerdings von der Laufzeit nicht sehr effizient, was daran liegt, dass der im schlimmsten Fall vollständig zu durchsuchende Zustandsraum selbst bei kurzen Protokollen schnell sehr groß werden kann. Es besteht trotz Optimierung der Algorithmen in späteren Interrogatorversionen sogar die Möglichkeit, dass das Programm nicht terminiert.
Der Interrogator teilt ebenfalls das Problem aller andereren Ansätze. Es kann zwar unter Umständen nachgewiesen werden, dass bestimmte unsichere Zustände nicht erreichbar sind, das heißt aber noch lange nicht, dass das Protokoll sicher ist, da weitere erreichbare unsichere Zustände existieren können.
In der Praxis ist es mit diesem Tool gelungen eine ganze Reihe bekannter Fehler zu reproduzieren, jedoch konnte kein einziger neuer Fehler gefunden werden. Das, und die Tatsache das die Bedienung des Interrogators recht kompliziert ist, haben dazu geführt, dass bei der Analyse von kryptographischen Protokollen heutzutage anderen Modellen der Vorzug gegeben wird.

Literatur:

J. K. Millen
"The Interrogator model"
IEEE Symposium on Security and Privacy, 1995

Die folgende Literatur enthält weiterführende Informationen zum Interrogator, die hier allerdings nicht benutzt wurden.

J.K.Millen S. C. Clark, S. B. Freedman
"The Interrogator: Protocol Security Analysis",
IEEE Transactions on Software Engineering, SE-13(2), 1987

Richard Kemmerer, Catherine Meadows, Jonathan Millen
"Three Systems for Cryptographic Protocol Analysis",
Journal of Cryptology, 7(2), 1994

J. K. Millen
"The Interrogator: A Tool for Cryptographic Protocol Security",
IEEE Symposium on Security and Privacy, 1984