A Comparison and Combination of Theory Generation and Model Checking for Security Protocol Analysis

Nicholas J. Hopper Sanjit A. Seshia Jeannette M. Wing
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{hopper+,sanjit+,wing+}@cs.cmu.edu

Abstract

This paper reviews two relatively new tools for automated formal analysis of security protocols. One applies the formal methods technique of model checking to the task of protocol analysis, while the other utilizes the method of theory generation, which borrows from both model checking and automated theorem proving. For purposes of comparison, the tools are both applied to a suite of sample protocols with known flaws, including the protocol used in an earlier study [4] to provide a baseline. We then suggest a heuristic for combining the two approaches to provide a more complete analysis than either approach can provide alone.

1 Introduction

Security protocols based on cryptographic primitives are used today to protect computer systems and network transactions from malicious attacks. These protocols have been known to be notoriously hard to design due to their complexity. Many subtle attacks have been demonstrated that are difficult to catch by manual analysis alone. There is therefore a need for formal automated tools to assist in the design of security protocols.

Many researchers have recently applied automated and semi-automated formal techniques to analyze security protocols. These tools fall into roughly two classes: those based on theorem proving (e.g., [4, 10, 5]) and those based on model checking (e.g., [8, 9, 7]). Tools differ in the degree of automation and expressiveness; while some tools are more automated than others, they often make simplifying assumptions that restrict the class of errors they can catch. These differences indicate that we might find tools that complement each other.

1.1 Model Checking and Theory Generation

In this paper, we compare two recently developed automated tools: Brutus [8], which is a model checker, and RVChecker, which is based on Kindred's theory-generation approach [6].

Brutus is a model checker specialized to analyze security protocols, with a built-in model of the adversary. It uses standard state space analysis techniques to check if the model satisfies the specification. If the model does not satisfy the specification, Brutus comes up with a counterexample showing where the specification breaks. Although model checking is almost entirely "push-button" in nature, Brutus sometimes comes up short in that the protocol might be far too complex to handle due to state-space explosion, or that the counterexample might be too complicated to understand exactly which protocol assumption led to a breach. It could also happen that the protocol breaks only when a certain configuration of multiple runs occurs, and there is no way to automatically figure out which runs are more important for analyzing properties of interest.

RVChecker is a theory generation tool based on REVERE [5] which is based on *belief logics*. The core idea in this approach is to produce a finite representation of the set of all the facts derivable from a protocol specification. Verifying a particular property of interest then simply becomes testing for set membership. The advantage of this approach lies in its automation, and in the high level handling of security protocol properties, thereby helping the designer fix incorrect assumptions about the system and its environment. The obvious disadvantage of this approach lies in the difficulty of seeing how failure of certain assumptions can translate into a real attack on the system. For a fairly complicated interaction between assumptions, it might be very difficult for a human designer to figure out exactly what is going wrong.

We apply these two tools to a suite of two protocols with known flaws, and compare their performance. The comparison shows the strengths and weaknesses of these two approaches. Based on the result of this comparison, we suggest a new method of analysis that uses a complementary combination of the two tools. This method combines the counterexample generation ability of a model checking approach with the high level intuition obtained by use of a belief logic approach, to yield a novel and interesting contribution to security protocol analysis.

1.2 The Protocol Suite

We ran each of BRUTUS and RVChecker on a suite of two protocols, both of which are known to have flaws. This suite includes the Tatebayeshi-Matsuzaki-Newman (TMN) protocol used by Kemmerer, Meadows and Millen to compare three analysis tools - the Interrogator, InaJo and the NRL Protocol Analyzer [4]. Using this protocol thus serves to provide a baseline comparison between the two tools investigated here and the three systems investigated in that article. Our suite also includes the buggy namestamping protocol presented by Dolev and Yao (DY) in their paper on algebra-based analysis of security protocols [3]. The chosen protocols help to illustrate the relative strengths and weaknesses of the two verification approaches. The known attack on the Dolev-Yao protocol is achieved by syntactic manipulation of the messages, which indicates that a model checking approach that explores all possible operations on messages might catch this bug. On the other hand, in the TMN protocol, parties perform key exchange based on beliefs about the authenticity of the other party, so it seems that a belief logic based tool such as RVChecker might catch errors in this protocol.

1.2.1 Tatabayeshi-Matsuzaki-Newman Key Exchange Protocol

The Tatabayeshi-Matsuzaki-Newman (TMN) protocol for key exchange [12], features a server S with a public key and two network nodes A and B who wish to exchange a session key through the server. The protocol consists of four messages:

$$\begin{array}{rrcl}
1. & A \longrightarrow S & : & [S.A.B.\{Ra\}_{Ks}] \\
2. & S \longrightarrow B & : & [B.S] \\
3. & B \longrightarrow S & : & [S.B.A.\{Rb\}_{Ks}] \\
4. & S \longrightarrow A & : & [A.S.B.Ra \oplus Rb]
\end{array}$$

In the above protocol description, the notation $A \longrightarrow S : [S.A.B.\{Ra\}_{Ks}]$ means that the composite message constructed by concatenating S, A, B and Ra encrypted under Ks is sent by principal A to principal S. Rx denotes a large random number generated by principal X, and $\oplus$ denotes bitwise exclusive-or.

After a run of this protocol, A and B use Rb as a session key. The protocol falls victim to several attacks. One attack results from properties of the public-key and symmetric encryption algorithms used (RSA and XOR), in which a second set of users can capture some of the messages from the first protocol instance and trick the server into revealing the key. A second class of attacks results from the fact that no authentication takes place in the protocol, so it is possible for an intruder to masquerade as any of the other principals in the protocol and disrupt the network by, for example learning the session keys of principals or convincing A that he is securely communicating with B when in fact he is communicating with Z. These attacks are more clearly outlined in our analyses in Sections 2 and 3.

1.2.2 Dolev-Yao Namestamp Protocol

Dolev and Yao [3] introduce several simple message transfer protocols with the intent of proving them secure or insecure. All of the protocols involve the transmission of a message M from party A to party B. The protocol we will analyze here, which has a multiple-run attack, has two messages (where Kx indicates the public key of principal X):

$$\begin{aligned} A \longrightarrow B &: [A.\{\{M\}_{Kb}.A\}_{Kb}.B] \\ B \longrightarrow A &: [B.\{\{M\}_{Ka}.B\}_{Ka}.A] \end{aligned}$$

This protocol is intended to securely transfer message M from A to B, but the use of double-encryption allows a three-run attack [3]:

1. Z listens to the first run of the protocol, remembering $\{\{M\}_{Ka}.B\}_{Ka}$ from B's response.

2. Z initiates a second run of the protocol with A, sending

$$[Z.\{\{\{M\}_{Ka}.B\}_{Ka}.Z\}_{Ka}.A]$$

3. A responds, sending Z:

$$[A.\{\{\{M\}_{Ka}.B\}_{Kz}.A\}_{Kz}.Z]$$

4. Z now knows $\{M\}_{Ka}$, so he initiates a third run of the protocol with A, sending:

$$[Z.\{\{M\}_{Ka}.Z\}_{Ka}.A]$$

5. A responds with:

$$[A.\{\{M\}_{Kz}.A\}_{Kz}.Z]$$

6. Z now has the message M.

Section 2 describes Brutus and the analysis of the two protocols using Brutus. Section 3 describes RVChecker and the analysis performed with it. In Section 4 we discuss the results of our analyses and make a comparison between the two tools. Based on this comparison, we propose some heuristics in Section 5 to combine the two approaches. Finally, in Section 6 we offer concluding remarks and suggest avenues for further research.

2 Brutus

Brutus is a model checker specialized for analyzing security and electronic commerce protocols [8]. Like most model checkers, Brutus is based on an operational description of the behavior of participating agents in the protocol. Properties can be expressed in a modal logic that is based on this operational description. These properties can be checked against the model using state space analysis. However, unlike a "general-purpose" model checker in which the user must specify a finite state model of the intruder, Brutus has a built-in model of the intruder.

In this section, we first give a brief overview of how Brutus models a protocol and how it defines and checks properties of interest. Then, we use Brutus to model and analyze the TMN and Dolev-Yao protocols.

2.1 The Model

The set $\mathcal{M}$ of possible messages in the Brutus model of protocols can be defined inductively as follows:

1. If $a \in \mathcal{A}, a \in \mathcal{M}$, where $\mathcal{A}$ is the set of atomic messages, i.e., *keys, principal names, nonces* or *data messages*.
2. If $m_1 \in \mathcal{M}$ and $m_2 \in \mathcal{M}$, then $m_1.m_2 \in \mathcal{M}$ (*pairing*)
3. If $m \in \mathcal{M}$ and key $k \in \mathcal{A}$, then $\{m\}_k \in \mathcal{M}$ (*encryption*)

In Brutus, the principals need to be explicitly modeled in the protocol, but the intruder need not. Brutus instead defines a *derivability relation*, "$\vdash$", which defines the way an intruder can derive a message from some initial set of information I.

1. If $m \in I$ then $I \vdash m$
2. If $I \vdash m_1$ and $I \vdash m_2$ then $I \vdash m_1 \cdot m_2$ (pairing)
3. If $I \vdash m_1 \cdot m_2$ then $I \vdash m_1$ and $I \vdash m_2$ (projection)
4. If $I \vdash m$ and $I \vdash k$ for key k, then $I \vdash \{m\}_k$ (encryption)
5. If $I \vdash \{m\}_k$ and $I \vdash k^{-1}$ then $I \vdash m$ (decryption)

$\overline{I}$ denotes the closure of I under application of the above rules.

A protocol is modeled as an asynchronous composition of communicating processes that model the honest principals as well as the intruder. The intruder process can eavesdrop on all communication and can interfere by dropping, changing or adding messages generated from its set of information; thus the communication channel can be thought of as passing through the intruder. The model is made finite by imposing a bound on the number of times ("sessions") a principal is allowed to participate in the protocol.

The formal model of an individual principal in a single session is called an *instance*. Each *instance* H of an honest principal is modeled as a 5-tuple $\langle N, S, I, B, P \rangle$ where:

- N is the name of the principal.
- S is a unique *instance ID* for this instance.

- B: $vars(N) \to \mathcal{M}$ is a set of bindings for $vars(N)$, the set of variables appearing in principal N, which are bound for a particular instance as it receives messages.
- $I \subseteq \mathcal{M}$ is the set of messages known to the principal executing the instance S.
- P is a process description given as a sequence of actions to be performed. These actions include pre-defined actions send and receive actions (written as **send** and **receive**), as well as user defined internal actions (written as **internal**). Internal actions can be used to keep track of the internal state of an instance.

The intruder, Z, is not bound to follow the protocol and the intruder process neither includes a sequence of actions P_Z nor a set of bindings B_Z. Instead, at any time, the intruder can receive any message or it can send any message it can generate from its set of known messages I_Z.

The global model is the asynchronous composition of the models of each instance, including I_Z. Let Σ be the set of global states of the system. The transition relation for the system is written as $\rightarrow$, where $\rightarrow \subseteq \Sigma \times S \times A \times \mathcal{M} \times \Sigma$. Here S is the set of instance IDs, A is the set of action names (which includes **send** and **receive**), and $\mathcal{M}$ is the set of all possible messages. A transition of the system from state σ to state σ' is represented as $\sigma \stackrel{s \cdot a \cdot m}{\longrightarrow} \sigma'$.

2.2 The Logic

The property of interest is specified in a first order logic with finite quantification and the past-time temporal operator.

We now give the syntax of the logic. Atomic propositions in the logic include principal names, instance IDs, instance variables, message constants and variables, and combinations of these as per the message construction rules defined in Section 2.1. In addition, atomic propositions can be formed by combining these with two predefined relations: "=" and "**Knows**", as well as with user-defined relations. A well-formed formula in the logic is defined as follows:

- if f is an atomic proposition, then f is a wff.
- if f is a wff, then $\neg f$ is a wff.
- if f_1 and f_2 are wffs, then $f_1 \wedge f_2$ is a wff.
- if f is a wff and s is an instance variable, then $\exists s.f$ is a wff.
- if f is a wff, then $\diamond_P f$ is a wff. [1]

The semantics of wffs in the logic are defined over the trace π of states (σ_i) and actions(α_i) of the system, given by $\pi = \sigma_0 \alpha_1 \sigma_1 \ldots \sigma_n$. The semantics of atomic propositions is given by the following recursive definition of the satisfaction relation $\models$:

- $\langle \pi, i \rangle \models m_1 = m_2$ iff $\sigma_i(m_1) = \sigma_i(m_2)$, i.e., the interpretations of the two messages in the ith state must be the same.
- The formula $\langle \pi, i \rangle \models s$ **Knows** m iff $\sigma_i(m) \in \overline{I_j}$ for some instance H_j in σ_i such that $S_j = s$
- $\langle \pi, i \rangle \models s\ A\ m$ for some user defined action A iff $\sigma_{i-1} \stackrel{s \cdot A \cdot m}{\longrightarrow} \sigma_i$.

[1] As in propositional temporal logic, $\diamond_P f$ is true at state σ_i, if $\exists$ some prior state $\sigma_j, j < i$ where f holds.

The usual semantics of $\models$ apply to boolean and temporal connectives.

A formula f is said to be true in a trace π iff f is true in every state of π. If a formula is not true for the model, BRUTUS outputs a counterexample trace showing the sequence of internal actions and message exchanges between principals.

A detailed treatment of model checking can be found in the book by Clarke, Grumberg and Peled [2].

2.3 Analysis

We now describe how we used BRUTUS to model and analyze the TMN and Dolev-Yao protocols. Each analysis of a property described in this section ran in under a minute on an Intel Pentium II 300 MHz machine running Linux. BRUTUS identified some of the flaws in both the protocols under consideration, however, it did not identify all of the known flaws due to limitations in its expressiveness.

In the analysis that follows, I stands for the intruder and all other letters stand for honest principals or trusted servers.

2.3.1 Dolev-Yao analysis

The initiating and responding instances of each honest principal are modelled as shown below (each corresponds to one session):

$$\begin{aligned}
INITIATOR \;=\;& \underline{\textbf{internal}}\ (\text{``}begin - initiate'', b) \\
& \underline{\textbf{send}}\ \langle a, b, \{\{M\}_{K_b}, a\}_{K_b}\rangle \\
& \underline{\textbf{receive}}\ \langle b, a, \{\{M\}_{K_a}, b\}_{K_a}\rangle \\
& \underline{\textbf{internal}}\ (\text{``}end - initiate'', b) \\
RESPONDER \;=\;& \underline{\textbf{receive}}\langle a, b, \{\{M\}_{K_b}, a\}_{K_b}\rangle \\
& \underline{\textbf{internal}}(\text{``}begin - respond'', a) \\
& \underline{\textbf{send}}\langle b, a, \{\{M\}_{K_a}, b\}_{K_a}\rangle \\
& \underline{\textbf{internal}}(\text{``}end - respond'', a)
\end{aligned}$$

In the above descriptions, a and b represent the names of the two communicating parties.

We checked the following classes of properties:

- Secrecy: The following property was checked:

  $$\neg(I\ \textbf{Knows}\ M)$$

  This property comes out to be true when we model less than three sessions of the initiator (A) and less than two sessions of the responder (B). However, when we model three sessions of A and one of B, BRUTUS finds the exact same attack as is described by Dolev and Yao [3]. Moreover, we find alternative ways of attacking the protocol if the relative number of sessions of A and B are changed. One such variation occurs when B listens for two sessions and A initiates using one session; the intruder is able to obtain the message as shown in Figure 1.

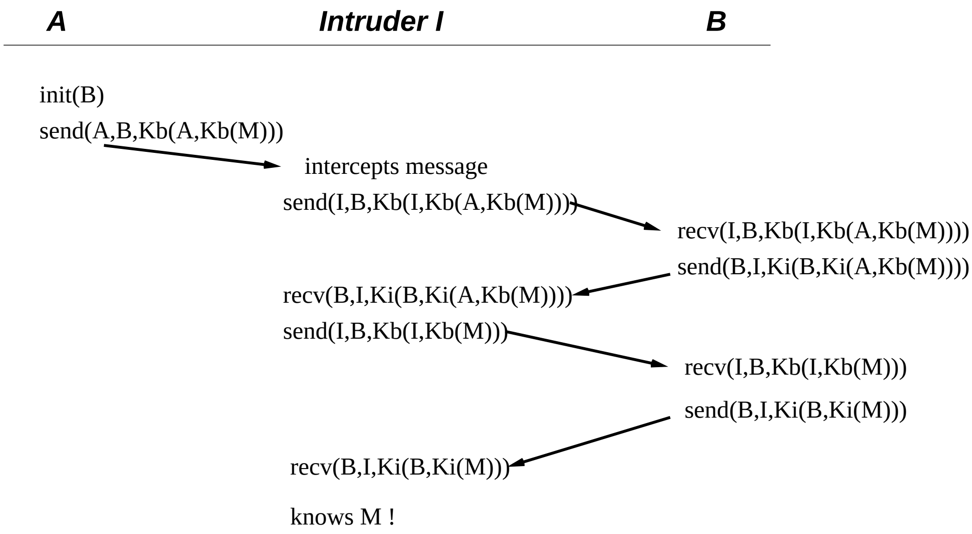

Figure 1: Counterexample showing secrecy flaw in Dolev-Yao

- Authentication: To check authentication, we checked the following two properties :

$$(1)\ \forall a.\{a\ \underline{\mathbf{internal}}\ (\text{“}end - initiate'', B) \quad \Rightarrow$$
$$(\exists S_b.(S_b.P = B) \wedge \quad \Diamond_P \quad (S_b.P\ \underline{\mathbf{internal}}\ (\text{“}end - respond'', a)))\}$$
$$(2)\ \forall b.\{b\ \underline{\mathbf{internal}}\ (\text{“}end - respond'', A) \quad \Rightarrow$$
$$(\exists S_a.(S_a.P = A) \wedge \quad \Diamond_P \quad (S_a.P\ \underline{\mathbf{internal}}\ (\text{“}begin - initiate'', b)))\}$$

We claim these properties are "authentication" properties – the first claims that if A finishes initiating a message to a honest principal B, then there is a session in which B responded to this message at some earlier point of time. The second claims that if B finished responding to a message apparently from A, then it was indeed A who initiated the send of that message in some session at some time in the past. Note that these are rather weak conditions of authenticity which rely on nametags attached to messages rather than something more secure. This arises from the fact that there are no signatures or shared secrets involved in the protocol. Properties of this kind have also been referred to as *correspondence* properties by Woo and Lam [13].

For a single run of A and B, both properties check out to be true. This happens because I is unable to get M through a replay attack, and so if A or B get the message they expected, it must have been legally generated.

However, if either A or B is allowed to have more than one session, then the intruder I can replay the message and make sure that A or B completes a session thinking that the message which originated from I actually came from an honest principal. One such attack is shown in Figure 2. Here we have the same model as in the previous attack, viz., two sessions of B and initiation by A. In this case, A thinks that B is talking to her, while B thinks he is talking to I, so the authentication property does not hold.

Since Brutus considers all principals to be honest, we cannot express honesty properties in the protocol. This is a limitation in the expressiveness of the specification logic.

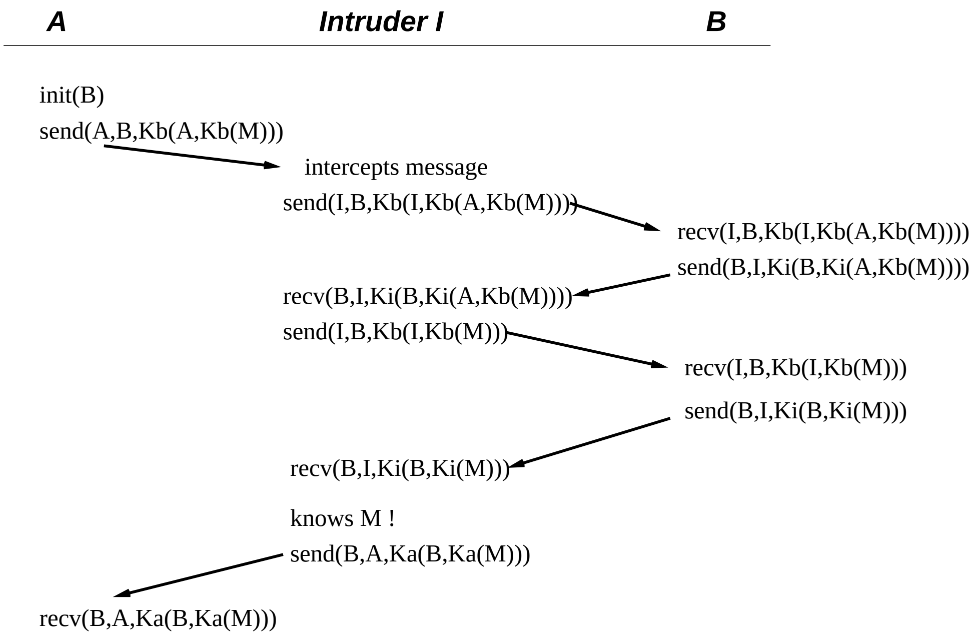

Figure 2: Counterexample showing authentication flaw in Dolev-Yao

2.3.2 TMN analysis

In TMN, we have three honest principals: the initiator A, the server S, and the responder B. We model these as shown below:

$$
\begin{aligned}
INITIATOR &= \underline{\textbf{internal}}\ (\text{“}begin - initiate'', B)\\
& \underline{\textbf{send}}\ \langle A, S, B, \{R_A\}_{K_S}\rangle\\
& \underline{\textbf{receive}}\ \langle s, A, b, \{R_b\}_{R_a}\rangle\\
& \underline{\textbf{internal}}\ (\text{“}end - initiate'', b)\\
RESPONDER &= \underline{\textbf{receive}}\langle s, B, a\rangle\\
& \underline{\textbf{internal}}(\text{“}begin - respond'', a)\\
& \underline{\textbf{send}}\langle B, s, a, \{R_B\}_{K_s}\rangle\\
& \underline{\textbf{internal}}(\text{“}end - respond'', a)\\
SERVER &= \underline{\textbf{receive}}\langle a, S, b, \{R_a\}_{K_S}\rangle\\
& \underline{\textbf{internal}}(\text{“}begin - initiate'', b)\\
& \underline{\textbf{send}}\langle b, S, a\rangle\\
& \underline{\textbf{receive}}\langle b, S, a, \{R_b\}_{K_S}\rangle\\
& \underline{\textbf{internal}}(\text{“}end - initiate'', b)\\
& \underline{\textbf{internal}}(\text{“}begin - respond'', a)\\
& \underline{\textbf{send}}\langle S, a, b, \{R_a\}_{R_b}\rangle\\
& \underline{\textbf{internal}}(\text{“}end - respond'', a)
\end{aligned}
$$

In the above model, small letters (such as a) are used in place of principal names (such as A) to indicate that the party represented by the letter p may not be the principal P, but some other party Q playing the role of P.

The current version of BRUTUS requires secret keys generated by A and B to be represented as symmetric keys, with a symmetric key for I. Moreover, it requires I's key to replace that of A or B wherever I replaces A or B. Representing secret keys in this manner ties the key to the principal name within a given message. Thus, if I attempts an attack in which it replaces a principal's name with its own (masquerading as that principal), it also effectively changes the key, often rendering the attack ineffective.

We checked the following classes of properties:

- Secrecy: We want to check if the shared secret R_B remains a secret from the Intruder I. More formally, we want to check the following two properties:

$$\begin{aligned}(1) &\ : \ \neg(I\ \textbf{Knows}\ R_B)\\ (2) &\ : \ \neg(I\ \textbf{Knows}\ R_B) \wedge \neg(I\ \textbf{Knows}\ R_A)\end{aligned}$$

  The first property means that I cannot read messages passed between A and B The second property means that in addition, I cannot masquerade as S while communicating with A.

  BRUTUS finds both properties to be false. First, if the intruder is allowed to take on the role of the server (i.e., the name "I" can replace "S" in messages), then a very simple attack is possible, as shown in Figure 3. If we assume that the intruder cannot masquerade as the

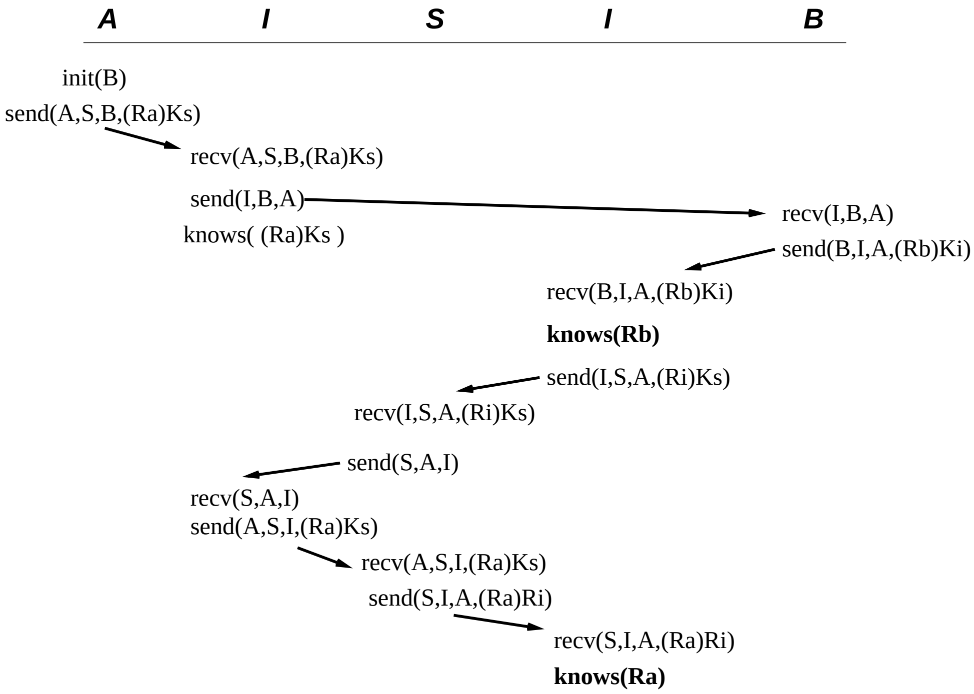

Figure 3: Counterexample 1 showing secrecy flaw in TMN

  server, then we can still find an attack. With a single session of S we are only able to find an attack that allows I to discover R_B, but with two sessions of S, I can also find R_A. This latter attack is shown in Figure 4. This observation brings up a key point: BRUTUS provides no automatic way of finding the number of sessions of S we need to model to find a bug. In this case, a trial-and-error approach worked since the number of additional sessions needed was relatively small, but this approach may not be appropriate for more complicated examples.

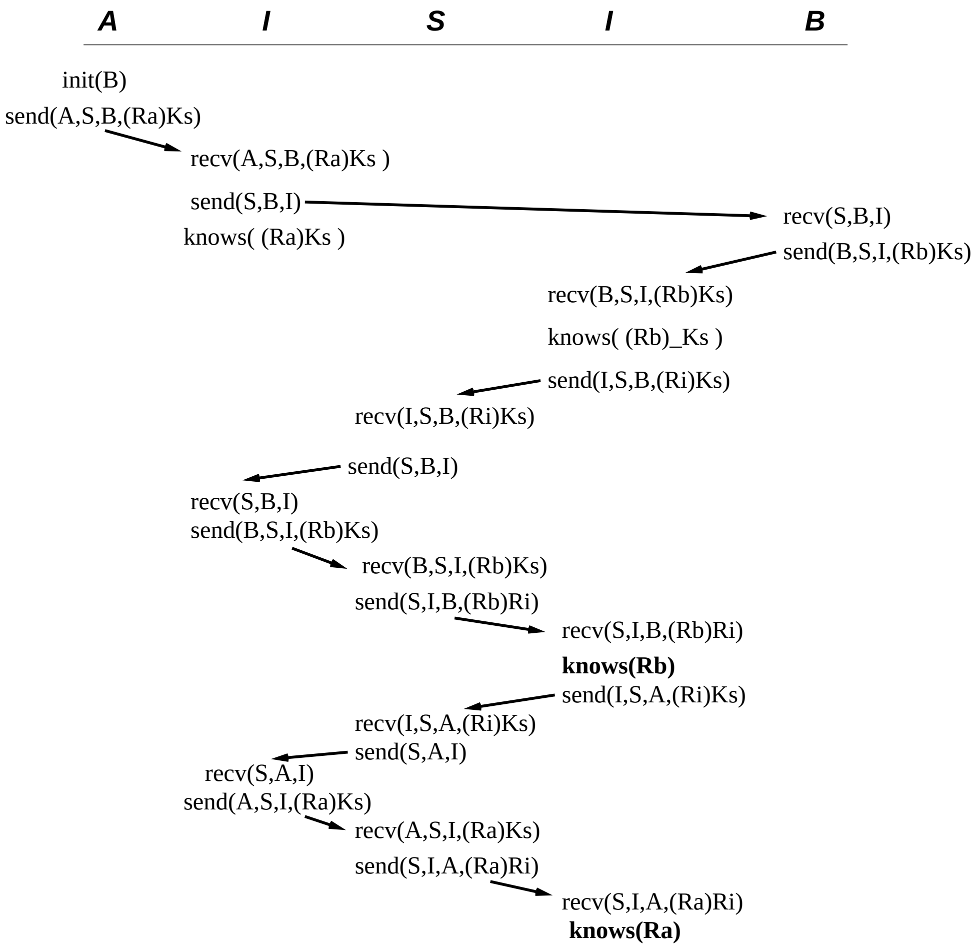

Figure 4: Counterexample 2 showing secrecy flaw in TMN

- Authentication: We check the same authentication properties as for the DY protocol. However, in this case, Brutus deems both properties to be true. This is because it is not possible for Brutus to represent the secret key of I with the principal name of B in the same message as discussed earlier.

- Multiple Intruder Collusion: Simmons describes an attack on the TMN protocol (described in [12]) in which the homomorphic property of the keys is used. In essence, the key used in a single valid run of the protocol is replayed by two colluding intruders (who could be two different sessions of the same intruder). Since this attack depends on the homomorphic property of keys, which cannot be expressed in Brutus, Brutus fails to catch this flaw. Note however, that such algebraic properties could be expressed in other model checkers, e.g., $Mur\phi$ [9].

In a context where Brutus was exclusively used for protocol analysis, we will fail to catch the Simmons attack on the TMN protocol; however, the use of another tool in conjunction with Brutus might enable us to catch the flaw.

3 RVChecker

RVChecker is a protocol analysis tool inspired by the tool REVERE developed by Kindred [5]. REVERE features two new techniques: *theory generation* and the RV logic, an extended BAN-style belief logic [1]. Combining these features results in a tool that is fully automatic; given a protocol specification no further action is required by the user to analyze the protocol. Whereas REVERE is written in SML, the paper's first author developed RVChecker entirely in Java, so it is also portable to any platform for which there is a JVM available.

3.1 Theory Generation

RVChecker employs Kindred and Wing's theory generation for reasoning about small theories in limited logics [6]. Theory generation is a syntactic method of theorem proving based on a saturation approach. The concept is to produce a finite representation of a (possibly infinite) theory generated by a set of rules and assumptions.

Their approach divides the rules of a logic into "shrinking" and "growing" rules (as decided by any well-defined measure) and repeatedly applies only the shrinking rules, essentially guaranteeing that the process will come to a halt. Then to test whether a specific formula is a consequence of the rules and assumptions, a backward-chaining search over the growing rules is applied, which will also guarantee termination. The sufficient condition for termination provided by Kindred and Wing is the restriction that a shrinking rule must still shrink even when all of its premises that can be met by the conclusion of a growing rule are removed.

A consequence of this condition is that given a set of rules and a candidate measure, it is possible to automate the process of checking a logic for termination under theory generation. This allows the development of a general-purpose theory generation algorithm that can automatically prove any judgement of any logic that satisfies these termination conditions. Both REVERE and RVChecker implement this algorithm, formally described in Kindred's thesis [5].

In addition to being fully automatic, this algorithm has the potential to be quite fast; in his thesis, Kindred gives figures for the analysis of several security protocols under several different logics; the longest analysis took REVERE 50 seconds to complete on a 500 MHz processor. RVChecker was not designed with processing speed as a primary consideration; the protocols analyzed here took on the order of 5 minutes to check on a 500 MHz Pentium III.

3.2 RV logic

RV is a belief logic developed by Kindred [5] with a core similar to the Burrows-Abadi-Needham (BAN) logic of authentication [1]. The innovations of RV lie in its rules for *responsibility* and a technique of *explicit interpretations*. The belief rules of RV are essentially the same as those of BAN and will not be discussed here; the responsibility properties attempt to account for the possibility of irresponsible behavior on the part of the principals of the protocol, while explicit interpretations represent an attempt to address the dangerous idealization step necessary to analyze a protocol using the BAN logic. In this section we will describe RV rules in terms of the symbols listed in Table 1, for consistency with the notation of Kindred [5]. (So, for example, the first rule in Figure 5 reads "if P sees M, then P sees M'".)

Notation	Meaning
$P \triangleleft M$	P sees M
$P \mid\approx M$	P says M
$P \mid\sim M$	P said M
$P \mid\equiv M$	P believes M
$Q \stackrel{K}{\longleftrightarrow} R$	K is a shared key between Q and R
$\stackrel{K}{\mapsto} Q$	K is Q's public key
$\{X\}_K$	Message X, encrypted under key K
$Q \stackrel{Y}{\rightleftharpoons} R$	Y is a shared secret between Q and R
P **controls** X	P has jurisdiction over X
$\sharp(X)$	X is "fresh", that is, "random" or "new"

Table 1: RV symbols

$$\frac{P \triangleleft M}{P \triangleleft M'} \quad \frac{P \mid\equiv Q \mid\sim M}{P \mid\equiv Q \mid\sim M'} \quad \frac{P \mid\equiv Q \mid\approx M}{P \mid\equiv Q \mid\approx M'}$$

Figure 5: RV Interpretation rule patterns. Here M matches all or part of a concrete message, while M' represents its intended meaning

3.2.1 Explicit Interpretation

BAN-style protocol analysis necessarily involves generating idealized versions of the messages in a protocol and doing analysis on these idealized protocols rather than the concrete protocols intended. The risk is that the idealization might contain hidden assumptions about the relative safety of a message; BAN fails to consider other possible interpretations of the same concrete message.

RV addresses this by introducing a syntax for concrete protocol messages; the protocol is analyzed using the actual concrete messages intended for implementation, along with "interpretation" rules that allow for the explicit idealization of the protocol. Kindred severely restricts these interpretation rules to prevent invalidating the rules of the logic [5].

An interpretation rule must match one of the patterns shown in Figure 5. In addition, the concrete message M cannot contain any variables specific to one instance of the protocol, and may not contain any functions other than concatenation; the idealized meaning M' may not contain protocol instance variables or the **encrypt** function; and no concrete message may match more than one interpretation rule for a given protocol. This approach allows the explicit statement of how the protocol is being idealized, and placing these restrictions on the interpretation rules means that they can be automatically checked, pointing out dubious idealizations to the user.

Since nonces are not always used merely for freshness or randomness, but are also used to stand in for longer pieces of information, RV also allows nonce-binding operations to be included in the premises of the interpretation rules; we do not use this feature of the logic for either of the protocols evaluated here.

3.2.2 Responsibility

The RV logic also introduces the notion of responsibility in protocol analysis. Here the concept is that RV analysis allows one to determine whether or not the principals involved in a protocol are acting responsibly, as defined by two properties: *honesty* and *secrecy*. Roughly speaking, the honesty property requires that a principal believe any possible interpretation of a message he sends, while the secrecy property requires a principal to believe that it is safe for any intruder to see all of the data sent in a concrete message. RV derives these properties through the **legit** and **maysee** operators and rules described here.

Original interpretation rule	Corresponding **legit** rule
$\dfrac{P \mid\equiv Q \mid\approx M}{P \mid\equiv Q \mid\approx M'}$	$\dfrac{Q \mid\equiv M' \quad Q \mid\equiv \textbf{signed}(M, M_s, P, Q)}{Q \mid\equiv \textbf{legit}(M_s)}$
$\dfrac{P \mid\equiv Q \mid\sim M}{P \mid\equiv Q \mid\sim M'}$	$\dfrac{Q \mid\equiv M' \quad Q \mid\equiv \textbf{signed}(M, M_s, P, Q)}{Q \mid\equiv \textbf{legit}(M_s)}$
$\dfrac{P \triangleleft M}{P \triangleleft M'}$	$\dfrac{Q \mid\equiv M'}{Q \mid\equiv \textbf{legit}(M)}$

Table 2: Honesty rules corresponding to protocol message interpretation rules

The **legit** operator works in conjunction with the interpretation rules introduced in the previous section; when analyzing a protocol, for each interpretation rule, a corresponding rule is introduced, according to the proper rule from Table 2. The RV logic also has rules that allow one to derive that the encryption of a legitimate or honest message is also honest, and the concatenation of two honest messages is honest. It introduces a concept of signed messages, with the idea that any message encrypted under a private or shared key or combined with a shared secret is signed. Then to check the honesty of a principal P in connection with a given concrete message M, it suffices to derive $P \mid\equiv \textbf{legit}(M)$ [5].

The **maysee** operator communicates that it is acceptable for a given principal to see a piece of information. RV's **maysee** rules specify that if information is public, then any principal believes that any other principal may see it; a principal is allowed to see anything he or she has already seen before; a principal may see anything that an intruder may see, and so on. For a complete list of RV's **maysee** rules, see Figure 6. To check secrecy properties in RV, the protocol analyst adds a principal I (the *Intruder*) and checks that for each message M and sender S, $S \mid\equiv I$ **maysee** M [5].

3.3 RVChecker Protocol Analysis

Protocol analysis with RV and theory generation involves several phases:

1. A specification is supplied, consisting of a set of messages between principals, a set of interpretation rules conforming to the restrictions listed above, a list of the principals involved, a set of belief goals for the protocol, and a set of initial assumptions about the beliefs of the principals involved.

2. Theory generation is applied to the set of initial assumptions to generate T_0, the consequence closure of the assumptions under RV.

$$\frac{P \lhd Q \stackrel{K}{\longleftrightarrow} R}{P \mid\equiv Q \textbf{ maysee } K} \qquad \frac{P \lhd Q \stackrel{Y}{\rightleftharpoons} R}{P \mid\equiv Q \textbf{ maysee } Y} \qquad \frac{P \lhd \stackrel{K_1}{\mapsto} Q \quad K_1 = K_2^{-1}}{P \mid\equiv Q \textbf{ maysee } K_2}$$

$$\frac{P \lhd Q \textbf{ maysee } Y}{P \mid\equiv Q \textbf{ maysee } Y} \qquad \frac{P \lhd Q \lhd Y}{P \mid\equiv Q \textbf{ maysee } Y} \qquad \frac{P \mid\equiv Q \textbf{ maysee } X}{P \mid\equiv Q \textbf{ maysee } \{X\}_K}$$

$$\frac{P \mid\equiv Q \textbf{ maysee } (X, Y)}{P \mid\equiv Q \textbf{ maysee } X} \qquad \frac{P \lhd \stackrel{K}{\mapsto} Q}{P \mid\equiv Z \textbf{ maysee } K} \qquad \frac{P \mid\equiv Z \textbf{ maysee } X}{P \mid\equiv Q \textbf{ maysee } X}$$

$$\frac{P \mid\equiv Q \textbf{ maysee } X \quad P \mid\equiv \stackrel{K}{\mapsto} Q}{P \mid\equiv Z \textbf{ maysee } \{X\}_K} \qquad \frac{P \mid\equiv Q \textbf{ maysee } X \quad P \mid\equiv Q \textbf{ maysee } Y}{P \mid\equiv Q \textbf{ maysee } [X.Y]}$$

$$\frac{P \mid\equiv Q \textbf{ maysee } X \quad P \mid\equiv R \textbf{ maysee } X \quad P \mid\equiv Q \stackrel{K}{\longleftrightarrow} R}{P \mid\equiv Z \textbf{ maysee } \{X\}_K}$$

Figure 6: RV's **maysee** rules

3. For each message M_i with sender S_i and recipient R_i, the formula $R_i \lhd M_i$ is added and theory generation is re-applied to compute the theory T_i.
4. Secrecy is checked: $S_i \mid\equiv I$ **maysee** $M_i \in T_{i-1}$.
5. Honesty is checked: $S_i \mid\equiv$ **legit** $(M_i) \in T_{i-1}$.
6. For each belief goal g, check that $g \in T_n$, where n is the number of messages.

If not all of the belief goals are met, the protocal analyst may proceed by adding and removing statements from the set of initial assumptions until the resulting specification satisfies all of the above steps; the extra assumptions can be examined for validity. The analyst may also remove an assumption from the initial set to determine whether the assumption is necessary or merely sufficient for a protocol to be secure. In this section, we give the results of protocol analysis for the selected suite.

RVChecker identified flaws in both the DY and TMN protocols; here we present the analysis of each. The suite nicely points out some weaknesses of the RV approach, as well as its strengths; for example, we were able to find flaws through an inability to meet belief goals, but discovering these flaws required extensive familiarity with the RV logic to isolate the necessary (or flawed) additional assumptions. In addition, knowing that some dubious assumptions are necessary does not necessarily indicate how an attacker can take advantage of these assumptions – while in these cases we knew of the attacks *a priori*, in actual analysis using RVChecker on other protocols this will not in general be the case.

3.3.1 Dolev-Yao analysis

RVChecker can quickly identify one fault in the DY protocol: it provides no authentication. One might assume that desirable belief goals for DY would be "B believes it is A who says M," and "A believes B says that it is A who says M" (formally, $A \mid\equiv B \mid\approx (A \mid\approx M)$), however, since there is no signature and no previous shared secret of any sort, this belief-oriented set of goals cannot be achieved. In any case, the main goal of the protocol as introduced by Dolev and Yao is secrecy, so the belief goals can be weakened to the "seeing" goals "B sees that A said M" and "A sees that B sees A said M", while we analyze the honesty and secrecy properties of the protocol. Even these further weakened goals cannot be reached without breaking an interpretation rule restriction: because RV and RVChecker do not allow compositional rules, there is no way for a principal to derive that when she sees $[X.\{Y\}_K]$ and has the key K, then she sees X and Y together in the same message. Thus our interpretation rules include encryption and specify protocol principals (so that we may connect the encryption key with the principal who may decrypt it):

$$\frac{B \triangleleft [\{X\}_{Kb}.Q]}{B \triangleleft Q \mid\sim X} \quad \frac{A \triangleleft [\{X\}_{Ka}.Q]}{A \triangleleft (Q \triangleleft (A \mid\sim X))}$$

With these interpretation rules and the proper public key belief set:

$$\begin{array}{ll} A \mid\equiv \overset{Ka}{\mapsto} A & A \mid\equiv \overset{Kb}{\mapsto} B \\ B \mid\equiv \overset{Ka}{\mapsto} A & B \mid\equiv \overset{Kb}{\mapsto} B \end{array}$$

RVChecker finds that the "seeing" goals are satisfied.

The honesty goals require additional assumptions to be met. The first required assumption is that $A \mid\equiv A \mid\sim M$, which is necessary to assume because only the recipient sees the message in RVChecker's analysis. We also require the assumptions:

$$\begin{array}{ll} A \mid\equiv \textbf{legit } (A) & A \mid\equiv \textbf{legit } (B) \\ B \mid\equiv \textbf{legit } (A) & B \mid\equiv \textbf{legit } (B) \end{array}$$

in order to get around the fact that namestamps rather than keys are used to provide authentication. With these assumptions, it is easy to derive the appropriate **legit** belief for the first message. After the first message, $B \triangleleft A \mid\sim M$ can be used to derive

$$B \mid\equiv B \triangleleft A \mid\sim M$$

which allows us to derive the **legit** properties for the second message.

The secrecy goals are also met, requiring only two assumptions: $A \mid\equiv B$ **maysee** M, which seems obvious since A is sending M to B; and $B \mid\equiv A$ **maysee** M, which seems reasonable once B sees that $A \mid\sim M$. With these assumptions, RVChecker easily derives that the secrecy properties hold.

The DY protocol, however, was designed to have a secrecy flaw; thus our results represent either a shortcoming of RV's secrecy rules (meaning they are unable to account totally for multiple-run attacks) or an invalid assumption. The assumption that $B \mid\equiv A$ **maysee** M allows the secrecy check to succeed for the second message; without it, B cannot derive that I **maysee** $\{M\}_{Ka}$. Thus it might be tempting to conclude that this assumption is invalid; however, Dolev and Yao also give an example of a protocol with perfect secrecy, for which this assumption is required for RV to derive the secrecy properties. This shortcoming is a result of a conscious design decision by RV's creator which results in a conservative approach to secrecy; thus sometimes these rules will lead to the conclusion that a secrecy flaw is present when in fact none is. Here the secrecy flaw comes both from lack of authentication and double encryption, but RV's rules do not allow us to isolate the double encryption as a source of the flaw. Thus in a context where RVChecker was being used exclusively for protocol analysis, we would conclude that it had missed the flaw; if, however, another tool is involved we could use the additional tool to investigate this flaw, as we do in Section 5.

3.3.2 TMN analysis

Through analysis with RVChecker we were able to find belief and responsibility errors in TMN. Since TMN is a key-exchange protocol, we start with the belief goals. Generally, we would like to have both A and B believe that Rb is a shared key between them, and further we would like both principals to believe that the other principal shares their belief, resulting in the goals:

$$
\begin{array}{cc}
A \mid\equiv A \overset{Rb}{\longleftrightarrow} B & B \mid\equiv A \overset{Rb}{\longleftrightarrow} B \\
A \mid\equiv B \mid\equiv A \overset{Rb}{\longleftrightarrow} B & B \mid\equiv A \mid\equiv A \overset{Rb}{\longleftrightarrow} B
\end{array}
$$

Since there is no direct authentication in this protocol, however, the bottom two goals are impossible, so we will settle for the goals:

$$
A \mid\equiv A \overset{Rb}{\longleftrightarrow} B \quad B \mid\equiv A \overset{Rb}{\longleftrightarrow} B \quad S \mid\equiv A \overset{Rb}{\longleftrightarrow} B
$$

We also have a problem in that, as specified in Section 1.2.1, any interpretation rule which matches message 1 or 3 will match both, yet they have different interpretations; to solve this problem we add tags `REQ` and `RSP` to these messages (which simulate the state that S would have to maintain in the concrete protocol). If we try to write the interpretation rules now, we find one more problem: as with DY, RV is unable to derive that if a principal sees $[X.\{Y\}_K]$, then that principal sees (X, Y). Thus in our idealization step we have already identified one potential attack: an attacker can change the names in the messages being passed around without changing the keys; thus he could change a message from A to S to say $[S.A.I.\ldots]$ rather than $[S.A.B.\ldots]$. To fix this problem, we encrypt the entire contents of messages 1 and 3 rather than just the keys:

$$
\begin{array}{rcl}
A \longrightarrow S & : & \{[\mathtt{REQ}.S.A.B.Ra]\}_{Ks} \\
B \longrightarrow S & : & \{[\mathtt{RSP}.S.B.A.Rb]\}_{Ks}
\end{array}
$$

Then our interpretation rules for these messages are relatively straightforward:

$$
\frac{P \lhd [\mathtt{REQ}.P.Q.R.K]}{P \lhd P \overset{K}{\longleftrightarrow} Q} \quad \frac{P \lhd [\mathtt{RSP}.P.Q.R.K]}{P \lhd Q \mid\approx Q \overset{K}{\longleftrightarrow} R}
$$

However, since the specification of TMN calls for bitwise exclusive-or encryption in the final message, we cannot modify it to encrypt the entire contents. Instead, we will break the restriction that an interpretation rule cannot include any protocol instance variables, which is reasonable because in any single run, A only expects to get a key for one person:

$$
\frac{P \mid\equiv S \mid\approx K}{P \mid\equiv S \mid\approx P \overset{K}{\longleftrightarrow} B}
$$

Note that in this rule, S refers to a protocol constant, the server S.

To check our belief goals, we require several assumptions. Besides the jurisdiction assumptions (involving which principals are allowed to control the keys used by other principlas) necessary, the interesting assumptions are

$$A \mid\equiv A \stackrel{Ra}{\longleftrightarrow} S$$

and

$$B \mid\equiv A \stackrel{Rb}{\longleftrightarrow} B$$

since A and B are responsible for generating these keys. The second of these assumptions satisfies our second belief goal. To satisfy our first belief goal, however, we require an additional assumption, which points out a second attack. For $A \mid\equiv A \stackrel{Rb}{\longleftrightarrow} B$, we must infer the following sub-goals:

$$A \mid\equiv S \textbf{ controls } A \stackrel{Rb}{\longleftrightarrow} B$$
$$A \mid\equiv S \mid\approx A \stackrel{Rb}{\longleftrightarrow} B$$

The first sub-goal is the obvious jurisdiction assumption. The second sub-goal can be derived through our interpretation rule, if we can infer $A \mid\equiv S \mid\approx Rb$. This inference requires the assumption $A \mid\equiv\ \sharp(Rb)$; that is, A must assume that Rb is freshly generated. In fact, this points to the attack found by Simmons [12] in which two collaborating intruders use keys which are not fresh to find A's key.

The final belief goal is also a source of difficulty: to derive $S \mid\equiv A \stackrel{Rb}{\longleftrightarrow} B$, we need to infer that, for some P:

$$S \mid\equiv P \textbf{ controls } A \stackrel{Rb}{\longleftrightarrow} B$$

$$S \mid\equiv P \mid\approx A \stackrel{Rb}{\longleftrightarrow} B$$

Given the definition of the protocol, it is a reasonable assumption that $P = B$, since B generates Rb. We end up with an authentication problem, however: while we can derive that $S \triangleleft B \mid\approx A \stackrel{Rb}{\longleftrightarrow} B$, the protocol does not have sufficient public key architecture for authentication. Thus there is no way for S to become convinced that $B \mid\approx A \stackrel{Rb}{\longleftrightarrow} B$. This seems to correspond again to a masquerading attack, in which the intruder I poses as B and convinces A that he shares a key with B rather than I.

When checking responsibility, we find that under reasonable assumptions the first three messages (as modified) will satisfy honesty, while because of the belief problem above, the fourth message cannot satisfy the honesty property, since A will interpret S's message as meaning $S \mid\approx A \stackrel{Rb}{\longleftrightarrow} B$, and S does not believe this. Thus RV affords two opportunities to find this error (although we admittedly added the third belief goal only because we suspected that its failure lead to the failure of the honesty check for the final message). Similarly, the secrecy property fails for the final message, because since S is not convinced that Rb is really a shared key for A and B, the **maysee** rules do not allow us to infer that $S \mid\equiv A$ **maysee** Rb.

4 Comparison

In this paper we have highlighted the approaches of two fully-automated protocol analysis tools, Brutus and RVChecker, by presenting their analyses of two cryptographic protocols. Since both tools found flaws in both protocols, we attempt to sort through the advantages and disadvantages of each. In the next section, we suggest a heuristic for their combination that attempts to exploit the advantages of both.

Brutus and RVChecker both represent fully-automated protocol analysis techniques; that is, given a specification, neither tool requires further intervention from the user to complete its analysis, in contrast to the tools examined by Kemmerer et al [4]. The speed of both approaches also allows a more interactive approach in that the protocol and assumptions can be changed and the results analyzed (relatively) quickly – but both tools require significant knowledge of the underlying model used in analysis; Brutus for the specification of security properties and RVChecker for the interpretation of analysis results.

System	Dolev-Yao Flaws		TMN Flaws	
	Auth	Secrecy	Auth/Secrecy	Simmons/Key
RVChecker	YES	NO	YES	YES
Brutus	YES	YES	YES	NO

Table 3: Results of protocol analysis. YES indicates that the system identified the flaw.

Table 3 summarizes the results of our comparison. It is interesting to note that while both Brutus and RVChecker found flaws in the TMN and DY protocols, the type of flaws found by each were somewhat different. For example, while both tools identified the authentication failure in TMN, RV was able to suggest, without any specific knowledge of the encryption schemes used, that it may be possible to attack the protocol with non-fresh session keys. And for DY, RV identified an authentication flaw but even with a reasonable assumption (which holds for a slightly modified but secure version of the protocol) was unable to identify the security flaw that Brutus easily identified. Therefore, while it might be tempting to claim that each tool always identifies certain types of flaws, we can see from this small set of protocols that neither tool consistently identifies the same types of flaws.

5 Combination

In addition to having the advantage of catching more flaws when used independently on the same problem, the theory generation and model-checking approaches present a certain complementarity. Analysis using theory generation and belief logics, as implemented in Revere and RVChecker, allows the identification of critical assumptions which allow or prevent a protocol from meeting its belief goals or from passing honesty or secrecy checks. Without prior knowledge of protocol flaws, however, it may be difficult to see how the failure of an assumption can be marshalled into an attack; further, as in the case of the DY assumptions, it may be difficult to identify which assumptions are questionable. Using model-checking, as in Brutus, yields actual attacks, but especially for lengthy counterexamples, it can be difficult to discern why the attack works, or what assumptions about the protocol are being violated.

5.1 A Motivating Example

As an example, in the RVChecker analysis of DY, we were able to find that the secrecy property fails if B does not believe that A **maysee** M; however knowing that this is a crucial assumption does not give us any knowledge of how an attacker might take advantage of it. On the other hand, with our knowledge of how the attack works, Brutus was quickly able to find an attack which yielded the message; however this does not give us a high level explanation of *why* the protocol might have failed. Using RVChecker and BRUTUS in combination, we see that the protocol fails because $B \mid\equiv A$ **maysee** M and we are given an example of how this flaw can be utilized to yield an error.

Further, the fact that RVChecker needed both assumptions $B \mid\equiv A$ **maysee** M and $A \mid\equiv B$ **maysee** M brings out the symmetry in the roles that might be played by A and B in an attack. This means that in BRUTUS, if we want the intruder to take on the role of B, we can introduce additional runs of A and vice-versa. As discussed in Section 2.3, different combinations of runs of A and B lead to different attacks (from that mentioned in [3]), one of which is presented in Figure 1.

5.2 A General Methodology for Combined Analysis

In general, we can think of the following two ways to combine these approaches:

1. *From Assumptions to Counterexamples:* We can use RVChecker to identify crucial assumptions, and then use our knowledge of these assumptions to search for counterexamples in BRUTUS.

2. *From Counterexamples to Assumptions:* We can search for counterexamples in BRUTUS and use this knowledge to isolate important assumptions in RVChecker.

In this section, we present short heuristics that are suggested by the previous example. In the next section, we show more examples that support these heuristics.

5.2.1 Assumptions to Counterexamples

To see how to move from RVChecker analysis to counterexamples in BRUTUS, observe that if an assumption held by principal A is both dubious and crucial for satisfaction of some property, then it most likely falls under one of two cases:

1. A believes something about her own behavior; in this case, we can use BRUTUS to search for counterexamples in which the intruder plays the role of A by modeling extra sessions of the other principals.

2. A believes something about the behavior of another principal B, in which case we allow the intruder to play the role of B in our analysis with BRUTUS by modeling extra sessions of A.

In the protocols presented here, the counterexamples that were found by BRUTUS can be motivated in this fashion by failures found using RVChecker. For example, the extra assumptions needed for secrecy in Dolev-Yao point to ways in which extra sessions of A or B can be utilized to generate an attack; we apply this heuristic to the TMN protocol in Section 5.3.

5.2.2 Counterexamples to Assumptions

In the second case, we would like to use RVChecker to determine which of a protocol designer's assumptions about a protocol fail in a counterexample generated by BRUTUS. To combine the tools in this direction, we model the entire counterexample trace as a protocol in RVChecker; note that this model will have the intruder I involved as a participating principal. We then proceed to add assumptions about each of the principals until the goal properties (secrecy, honesty, authentication, etc.) become satisfied. Some of these assumptions will be about I. The assumptions held about

I when he plays the role of principal Q will then indicate assumptions that the protocol designer either explicitly or implicitly made about Q and that I has used to subvert the protocol. Thus we can identify the faulty assumptions, which will hopefully lead to a better understanding and possibly a simple fix to the protocol.

5.3 More examples: DY attack and TMN

In this section, we show how the methodology outlined in the previous section can be applied to the two protocols we have studied.

5.3.1 DY Assumptions from Attack

Section 5.1 indicates how RVChecker can be used to guide Brutus by indicating which sessions to model. Going the other way, we can also model the attack produced by Brutus as a protocol in RVChecker and analyze this protocol to determine what assumptions about the intruder are necessary to satisfy the secrecy properties. Thus we model the second attack listed in Section 2 as a protocol with six messages, the first two between A and B and the last four between A and I, and attempt to satisfy the secrecy properties. In addition to the properties listed in Section 3.3.1 and the necessary assumptions about I's ID and public key, we find that these assumptions are necessary to satisfy secrecy:

$$I \mid\equiv I \textbf{ maysee } M \quad I \mid\equiv B \textbf{ maysee } M \quad B \mid\equiv I \textbf{ maysee } M$$

This pinpoints the assumption that both principals, particularly B must believe that it is safe for the recipient to see the message M, when in fact because of the lack of authentication it could potentially be an intruder receiving the message. This gives us some high-level insight into what went wrong with the design of the protocol.

5.3.2 TMN

We also applied both heuristics to the TMN protocol. In our analysis with RVChecker (Section 3.3.2) we found that it was necessary either to change the protocol or allow the server S to use inappropriate interpretation rules to satisfy the goal $S \mid\equiv A \stackrel{Rb}{\longleftrightarrow} B$; and that to satisfy secrecy, we required the additional assumption that $S \mid\equiv A$ **maysee** Rb. So following rule 2 of our heuristic, we can model an extra session of S in Brutus and allow I to play both A and B; the result is the attack shown in Figure 4, in which I manages to retrieve Ra from the server. Thus we arrive at the conclusion that the lack of authentication in this protocol results in inappropriate interpretation of messages and inappropriate assumptions that can be marshalled into an attack by I.

Going in reverse, we can also identify these interpretation rule errors and faulty assumptions by modeling the trace from Figure 4 as a protocol and attempting to satisfy the secrecy goals. Of particular interest is the secrecy of the message which I receives from S revealing Rb. To achieve the goal $S \mid\equiv P$ **maysee** $\{Rb\}_{Ri}$, we must satisfy the sub-goals $S \mid\equiv I$ **maysee** Rb and $S \mid\equiv S \stackrel{Ri}{\longleftrightarrow} I$. The first can only be satisfied by assumption, which reveals our first broken assumption – in the original protocol this translates to $S \mid\equiv A$ **maysee** Rb, when S has no proof that the key is actually intended for A. The second sub-goal can only be satisfied by forming interpretation rules that do not follow the patterns specified in [5], since S must believe an unauthenticated message from I. It follows that in the original protocol S must believe unauthenticated messages from A and B, which highlights the second error in the protocol, that is, the lack of proper authentication.

Thus, in both cases we are able to apply our heuristics to the TMN protocol, producing a more complete analysis through the combination of the tools than we were able to produce with either alone; in addition, the combination of the tools is able to catch more errors.

5.4 Related Work

Athena [11] is another tool that combines theorem proving and model checking, but in a different manner to achieve somewhat different benefits. Athena applies a state-space exploration approach to the process of proof-generation, along with the use of a compact representation. Unreachability theorems (possibly proved by theorem provers) can be used to prune the state space. The main benefit of this procedure is increased efficiency. However, if the property of interest is false for the protocol under consideration, the use of Athena does not yield any deeper insight into why the proof failed beyond presenting the user with a counterexample. In contrast, our combination of Brutus with RV gives the user an idea of which assumptions might have caused the protocol to fail. Using Athena with RV could possibly overcome the shortcoming of Athena mentioned above.

Another advantage of our method is that RV helps point out the configurations to model in Brutus that might lead to generation of counterexamples. This can be useful with Athena in cases where the proof search does not terminate.

6 Concluding Remarks

In this paper, we have compared the approaches and capabilities of two cryptographic protocol analysis tools – Brutus and RVChecker – by analyzing two security protocols with known flaws. Neither tool isolated all of the flaws in these protocols, but each tool missed different flaws; the complementary nature of their approaches suggested that it might be possible to combine these tools resulting in a more powerful yet still fully automated approach to formal protocol analysis. We do not claim that the methods suggested here for combining these two approaches are rigorous or complete, however, these methods of combination represent a beginning. Future work could see how they can be formalized and extended beyond the two protocols presented.

References

[1] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[3] D. Dolev and A. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(1):198–208, 1983.

[4] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[5] D. Kindred. *Theory Generation for Security Protocols*. PhD thesis, Carnegie Mellon University, 1999.

[6] D. Kindred and J.M. Wing. Fast, automatic checking of security protocols. In *Second USENIX Workshop on Electronic Commerce*, pages 41–52, Oakland, California, November 1996. USENIX.

[7] G. Lowe. Breaking and Fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96, LNCS 1055*, pages 147–166, 1996.

[8] W. Marrero, E. Clarke, and S. Jha. Verifying Security Protocols with Brutus. *to be submitted to ACM Transactions on Software Engg. and Methodology.*

[9] J.C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of the IEEE Symp. Security and Privacy, Oakland*, pages 141–153, 1997.

[10] L. Paulson. The Inductive Approach to verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

[11] D. Song. Athena: An automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 1999.

[12] Makoto Tatebayashi, Ntsume Matsuzaki, and David B. Newman, Jr. Key distribution protocol for digital mobile communication systems. In *Proceedings of CRYPTO'89*, pages 324–334.

[13] T. Woo and S. Lam. A semantic model for authentication protocols. In *IEEE Symposium on Research in Security and Privacy*, Oakland, CA, USA, May 1993.