A Hybrid Analysis for Security Protocols with State

John D. Ramsdell Daniel J. Dougherty Joshua D. Guttman Paul D. Rowe

***Abstract*—**

Cryptographic protocols rely on message-passing to coordinate activity among principals. Each principal maintains local state in individual local sessions only as needed to complete that session. However, in some protocols a principal also uses state to coordinate its different local sessions. Sometimes the non-local, mutable state is used as a means, for example with smart cards or Trusted Platform Modules. Sometimes it is the purpose of running the protocol, for example in commercial transactions.

Many richly developed tools and techniques, based on well-understood foundations, are available for design and analysis of pure message-passing protocols. But the presence of cross-session state poses difficulties for these techniques.

In this paper we provide a framework for modeling stateful protocols. We define a hybrid analysis method. It leverages theorem-proving—in this instance, the PVS prover—for reasoning about computations over state. It combines that with an "enrich-by-need" approach—embodied by CPSA—that focuses on the message-passing part. As a case study we give a full analysis of the Envelope Protocol, due to Mark Ryan.

I. Introduction

Protocol analysis is largely about message-passing in a model in which every message transmitted is made available to the adversary. The adversary can deliver the messages transmitted by the regular (i.e. compliant) principals, if desired, or not. The adversary can also retain them indefinitely, so that in the future he can deliver them, or messages built from them, repeatedly.

However, some protocols also interact with long-term state. For instance, the Automated Teller Machine network executes protocols that interact with the long-term state stored in banks' account databases. Protocol actions are constrained by that long-term state; for instance, an ATM machine may be told not to dispense cash to a customer, because his account has insufficient funds. Protocol actions also cause updates to long-term state; for instance, when a successful withdrawal occurs, the bank reduces the funds still available in the customer's account. In addition to electronic finance and commerce, protocols that are controlled by long-term state, and can update the state in controlled ways, are also important in trusted computing, i.e. in systems using Trusted Platform Modules for attestation and secrecy. Indeed, in a world where software interacts with real-world resources in interoperable ways, cryptographic protocols that manipulate long-term, extra-protocol state will be increasingly central.

This work partially supported by the US National Security Agency, and partially supported by the National Science Foundation under grant CNS-1116557. Authors' email addresses: {guttman,prowe,ramsdell}@mitre.org, {dd,guttman}@wpi.edu.

Long-term state is fundamentally different from message passing. The adversary can always choose to redeliver an old message. But he cannot choose to redeliver an old state; for instance, the adversary in an ATM network cannot choose to replay a withdrawal, applying it to a state in which he has sufficient funds, in case he no longer does. Regular principals maintain long-term state across protocol executions in order to constrain subsequent executions, and ensure that future runs will behave differently from past runs.

The Cryptographic Protocol Shapes Analyzer [23] (CPSA) is our program for automatically characterizing the possible executions of a protocol compatible with a specified partial execution. It is grounded in strand space theory. There exists a mathematically rigorous theory [18] that backs up the implementation of CPSA in Haskell, and proves the algorithm produces characterizations that are complete, and that the algorithm enumerates these characterizations.

It is natural to encode state manipulation in terms of message-passing. Reading from the state corresponds to receiving a message bearing the state, and writing the state corresponds to sending an appropriate message. Under these conventions CPSA can be used to analyze protocols with state: by adding state-bearing receive/transmit event pairs to the roles of a protocol that interact with the state, CPSA attempts to find paths through state space as part of the executions it generates.

Unfortunately, CPSA may construct some executions which are in fact not possible, specifically, executions in which a state-bearing message is transmitted from one node and then received by two different state-receiving nodes. Thus CPSA can only provide an approximate analysis; a more refined approach is called for.

Our contribution in this paper is a semantically sound method for applying CPSA to systems that include a state component. In this method, CPSA is coupled with the Prototype Verification System [20] (PVS) proof assistant.

In PVS, a version of strand space theory is specified. On top of this theory, the results of a CPSA analysis can be encoded as a PVS axiom, an axiom justified by the CPSA completeness result [22]. Subsequent PVS derivations might imply the existence of additional message transmission/receptions, leading to an enriched CPSA analysis. In this way the theorem-proving and execution-finding analysis activities cooperate, over the common semantic foundation of strand space theory.

Outline of the Analysis. The core paradigm remains CPSA's enrich-by-need approach [16]. That is, we consider the question, what kinds of executions are possible, assuming that a particular pattern of events has occurred? Authentication properties may be verified by observing that all executions contain certain events as required. Confidentiality properties may be

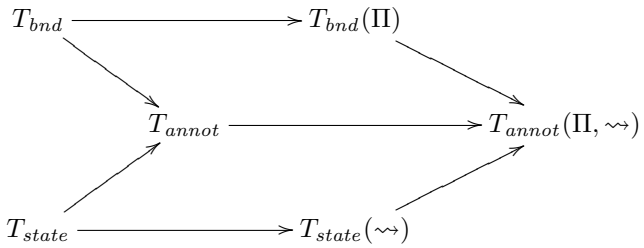

Fig. 1. Theory Inclusions

verified by considering patterns that include a disclosure, and observing that no kinds of executions are possible.

Our method involves a conversation (so to speak) between CPSA and PVS. The main steps are:

1. Within PVS we define (i) an encompassing theory T_{bnd} of strand spaces and protocol executions ("bundles") and (ii) a theory T_{state} of transition relations and the state histories they permit. We combined these into a theory T_{annot} of protocol executions where some steps in a protocol role are annotated with a state transition. Furthermore, T_{bnd} can be augmented with information about a particular protocol Π, producing $T_{bnd}(\Pi)$, and T_{state} may be augmented with information about a particular transition relation $\rightsquigarrow$, producing $T_{state}(\rightsquigarrow)$. We arrive at an enrichment of T_{annot} , the theory $T_{annot}(\Pi, \rightsquigarrow)$.
2. CPSA carries out enrich-by-need protocol analysis on the protocols, with state-manipulation modeled as message-passing, but without any special knowledge about state transition histories. Its results may be summarized in a sentence, called a *shape analysis sentence* [21], [16], in the language of $T_{bnd}(\Pi)$. The shape analysis sentence may be used as an axiom in proofs within PVS.
3. Within the state transition theory $T_{state}(\rightsquigarrow)$ in PVS, we prove useful lemmas. Using these, we infer consequences in the annotated protocol theory $T_{annot}(\Pi, \rightsquigarrow)$. Some of these consequences are in the more limited vocabulary of $T_{bnd}(\Pi)$. We call them *bridge lemmas*, because they bridge between the state world and the protocol world.
4. We use the bridge lemmas in combination with the state analysis sentences to prove conclusions about protocol runs in $T_{bnd}(\Pi)$. If we prove a contradiction, that implies that the situation given to CPSA cannot in fact occur. Otherwise, we may prove that additional message transmissions and receptions occurred.
5. We incorporate these additional nodes into a new CPSA starting point, and allow CPSA to draw conclusions. Additional round trips are possible.

Structure. The body of this paper describes an application of our method to the Envelope Protocol, a protocol that interacts with a Trusted Platform Module (TPM) to achieve an important security goal. Section III describes the protocol Π. Section IV describes our TPM model, $T_{state}(\rightsquigarrow)$. Section V presents the theory of bundles T_{bnd} encoded within PVS, and specializes this to $T_{bnd}(\Pi)$, demonstrating our main trick of including state-bearing receive-transmit pairs to encode the state transitions. Section VI describes CPSA, our protocol analysis tool and what results CPSA infers in $T_{bnd}(\Pi)$. Section VII links the state world and the protocol world $T_{annot}(\Pi, \rightsquigarrow)$. The relevant bridge lemma is stated and applied to prove the Envelope Protocol security goal.

II. Related Work

The problem of reasoning about protocols and state has been an increasing focus over the past several years. Protocols using TPMs and other hardware security modules (HSMs) have provided one of the main motivations for this line of work.

A line of work was motivated by HSMs used in the banking industry [17], [25]. This work identified the effects of persistent storage as complicating the security analysis of the devices. Much work explored the significance of this problem in the case of PKCS #11 style devices for key management [6], [7], [12]. These papers, while very informative, exploited specific characteristics of the HSM problem; in particular, the most important mutable state concerns the *attributes* that determine the usage permitted for keys. These attributes should usually be handled in a monotonic way, so that once an attribute has been set, it will not be removed. This justifies using abstractions that are much more typical of standard protocol analysis.

In the TPM-oriented line of work, an early example using an automata-based model was by Gürgens et al. [14]. It identified some protocol failures due to the weak binding between a TPM-resident key and an individual person. Datta et al.'s "A Logic of Secure Systems" [9] presents a dynamic logic in the style of PCL [8] that can be used to reason about programs that both manipulate memory and also transmit and receive cryptographically constructed messages. Because it has a very detailed model of execution, it appears to require a level of effort similar to (multithreaded) program verification, unlike the less demanding forms of protocol analysis.

Mödersheim's set-membership abstraction [19] works by identifying all data values (e.g. keys) that have the same properties; a change in properties for a given key K is represented by translating all facts true for K's old abstraction into new facts true of K's new abstraction. The reasoning is still based on monotonic methods (namely Horn clauses). Apparently this abstraction will not allow us to prove that one event will never happen, after a change in properties. Thus, it seems not to be a strategy for reasoning about TPM usage, for instance in the envelope protocol.

The paper [15] by one of us developed a theory for protocols (within strand spaces) as constrained by state transitions, and applied that theory to a fair exchange protocol. It introduced the key notion of *compatibility* between a protocol execution ("bundle") and a state history. In the current paper we will also rely on the same notion of compatibility, which was somewhat hidden in [15]. However, the current paper does not separate the protocol behavior from state history as sharply as did [15].

A group of papers by Ryan with Delaune, Kremer, and Steel [10], [11], and with Arapinis and Ritter [3] aim broadly to adapt ProVerif for protocols that interact with long-term state.

ProVerif [4], [1] is a Horn-clause based protocol analyzer with a monotonic method: in its normal mode of usage, it tracks the messages that the adversary can obtain, and assumes that these will always remain available. Ryan et al. modify the form of the facts with which ProVerif works. Instead of a one-place predicate $\texttt{att}(m)$ meaning that the attacker may possess message m, they use a two-place predicate $\texttt{att}(u, m)$ meaning that the adversary may possess m at some time when the long-term state is u. In [3], the authors provide a compiler from a process algebra with state-manipulating operators to sets of Horn clauses using this primitive. Care is needed, however, to ensure that the resulting Horn clauses lead to terminating runs of the ProVerif resolution engine. In [11], the authors analyze protocols with specific syntactic properties that help ensure termination. In particular, they bound the state values that may be stored in the TPMs. In this way, the authors verify two protocols using the TPM, including the envelope protocol that we also study below.

Our current approach has strengths and weaknesses relative to the ProVerif approach. An advantage is that it works within a single comprehensive framework, namely that of strand spaces. Proofs about state within PVS succeeded only when definitions and lemmas were properly refined, and all essential details represented. As a result, our confidence is high that our proofs about protocols have their intended meaning.

Weaknesses of our approach are twofold. PVS is labor intensive. In the current form, it is justified only for very high assurance applications. While the theory of strand spaces is reusable unchanged, there were numerous specific proofs for the Envelope Protocol that would be hard to reuse directly. Some of these proofs could serve as a high-level template that can be followed when analyzing other protocols. This weakness will be addressed in future work.

A second problem concerns termination. CPSA, like ProVerif, does not terminate in some cases. One must be very careful when adding state-bearing messages to a protocol for analysis by CPSA. CPSA was updated during this work to allow us to control termination for the Envelope Protocol; however, there are some simple protocols, which others can analyze, that we currently cannot, such as the Wrap-Decrypt attack on PKCS#11 HSMs [5].

III. The Envelope Protocol

The proof of an important security goal of the Envelope Protocol [2] was the focus of most of our effort. The protocol allows someone to package a secret such that another party can either reveal the secret or prove the secret never was and never will be revealed.

a) Protocol motivation.: The plight of a teenager motivates the protocol. The teenager is going out for the night, and her parents want to know her destination in case of emergency. Chafing at the loss of privacy, she agrees to the following protocol. Before leaving for the night, she writes her destination on a piece of paper and seals the note in an envelope. Upon her return, the parents can prove the secret was never revealed by returning the envelope unopened. Alternatively, they can open the envelope to learn her destination.

The parents would like to learn their daughter's destination while still pretending that they have respected her privacy. The parents are thus the adversary. The goal of the protocol is to prevent this deception.

b) Necessity of long-term state.: The long-term state is the envelope. Once the envelope is torn open, the adversary no longer has access to a state in which the envelope is intact. A protocol based only on message passing is insufficient, because the ability of the adversary monotonically increases. At the beginning of the protocol the adversary can either return the envelope or tear it. In a purely message-based protocol the adversary will never lose these abilities.

c) Cryptographic version.: The cryptographic version of this protocol uses a TPM to achieve the security goal. For this protocol, we restrict our attention to a subset of the TPM's functionality. In particular we model the TPM as having a state consisting of a single Platform Configuration Register (PCR) and only responding to five commands. With a `boot` command, the PCR is set to a known value. The `extend` command takes a piece of data, d, and replaces the current value val of the PCR with the hash of d and val, i.e. $\#(d, val)$. In fact, the form of `extend` that we model, which is an `extend` within an encrypted session, also protects against replay. These are the only commands that alter the value in a PCR.

The TPM provides other services that do not alter the PCR. The `quote` command reports the value contained in the PCR and is signed in a way as to ensure its authenticity. The `create key` command causes the TPM can create an asymmetric key pair where the private part remains shielded within the TPM. However, it can only be used for decryption when the PCR has a specific value. The `decrypt` command causes the TPM to decrypt a message using this shielded private key, but only if the value in the PCR matches the constraint of the decryption key.

In what follows, Alice plays the role of the teenaged daughter packaging the secret. Alice calls the `extend` command with a fresh nonce n in an encrypted session. She uses the `create key` command constraining that new key to be used only when a specific value is present in the PCR. In particular, the constraining value cv she chooses is the following:

$$cv = \#(\text{“}\texttt{obtain}\text{”}, \#(n, val))$$

where val was the PCR value prior the extend command. She then encrypts her secret v with this newly created key.

Using typical message passing notation, Alice's part of the protocol might be represented as follows (where k' denotes the key created in the second line, and where we still ignore the replay protection):

$$\begin{array}{rcll} \text{A} & \rightarrow & \text{TPM} & : \{\!|\text{“}\texttt{extend}\text{”}, n|\!\}_k \\ \text{A} & \rightarrow & \text{TPM} & : \text{“}\texttt{create key}\text{”}, \#(\text{“}\texttt{obtain}\text{”}, \#(n, val)) \\ \text{TPM} & \rightarrow & \text{A} & : k' \\ \text{A} & \rightarrow & \text{Parent} & : \{\!|v|\!\}_{k'} \end{array}$$

The parent acts as the adversary in this protocol. We assume he can perform all the normal Dolev-Yao operations such as encrypting and decrypting messages when he has the relevant key, and interacting with honest protocol participants. Most importantly, the parent can use the TPM commands available in any order with any inputs he likes. Thus he can extend

the PCR with the string `obtain` and use the key to decrypt the secret. Alternatively, he can extend the PCR with the string `refuse` and then generate a TPM quote as evidence the secret will never be exposed. The goal of the Envelope Protocol is to ensure that once Alice has prepared the TPM and encrypted her secret, the parent should not be able to both decrypt the secret and also generate a refusal quote.

A crucial fact about the PCR role in this protocol is the injective nature of the hashing, ensuring that for every x

$$\#(\text{“}\texttt{obtain}\text{”}, \#(n, val)) \quad \neq \quad \#(\text{“}\texttt{refuse}\text{”}, x) \qquad (1)$$

IV. The TPM Model

In this section we introduce our TPM state theory $T_{state}(\rightsquigarrow)$ focusing on representing the value of the PCR and how the TPM commands may change it over time.

Messages and states form an order-sorted algebra [13]. Order-sorted algebras generalize many-sorted algebra, allowing a sort to be partially ordered below another. Any carrier set interpreting the former is then a subset of the carrier set interpreting the latter.

Figure 2 shows the signature of the algebra used in this paper. Sort M is the sort of TPM machine states and sort $\top$ is the top sort of messages. Messages of sort A (asymmetric keys), sort S (symmetric keys), sort D (data), and sort E (text) are called *atoms*. Messages are atoms, tag constants, or constructed using encryption $\{\!|\cdot|\!\}_{(\cdot)}$, hashing $\#(\cdot)$, and pairing $(\cdot,\cdot)$, where the comma operation is right associative and parentheses are omitted when the context permits. It is easy to show that each term t of the algebra is equal to a unique term t' with a minimal number of occurrences of the inverse operation $(\cdot)^{-1}$; we choose this t' to be the canonical representative of t.

We use the function pcr to coerce TPM states, which are of sort M, to messages, specifically to symmetric keys of sort S:

$$\begin{aligned} pcr &: \mathsf{M} \to \mathsf{S} \\ pcr(\mathsf{bt}) &= \mathsf{s}_0 \\ pcr(\mathsf{ex}(t, m)) &= \#(t, pcr(m)) \end{aligned}$$

where constant s_0 is known to all. Modeling the injectivity of the hash function (cf. Equation 1) we postulate that the function pcr is injective.

The definition of the TPM transition relation $\rightsquigarrow$ is

$$\begin{array}{lll} m_0 \rightsquigarrow m_1 & \text{iff} & m_1 = \mathsf{bt} \qquad (\texttt{boot}) \\ & \text{or} & \exists t : \top.\, m_1 = \mathsf{ex}(t, m_0) \qquad (\texttt{extend}) \\ & \text{or} & m_0 = m_1 \qquad (\texttt{quote}, \texttt{decrypt}) \end{array}$$

The `create key` command does not interact with the state.

In this framework we prove a crucial property of all executions which we express in terms of the notion of a state *having* a message. A state *has* a message if an extend operation with it is part of the state.

$$\begin{aligned} \mathsf{bt} \text{ has } t &= \text{false} \\ \mathsf{ex}(t_0, s) \text{ has } t_1 &= (t_0 = t_1) \text{ or } s \text{ has } t_1 \end{aligned}$$

For example, $\mathsf{ex}(\text{“}\texttt{obtain}\text{”}, \mathsf{ex}(v, \mathsf{bt}))$ has “obtain” and v, but it does not have “refuse”.

An infinite sequence of states π is a *path* if $\pi(0) = \mathsf{bt}$ and $\forall i \in \mathbb{N}.\, (\pi(i), \pi(i+1)) \in \rightsquigarrow$. Paths in this TPM model have several useful properties. For example, if a previous state is not a subterm of a state, there must have been a intervening boot. Also, if a state has a message, and a previous state is a boot state, there must have been an intervening transition that extends with the message. These two properties can be combined into the property used by the proof of the Envelope Protocol security goal: if a previous state is not a subterm of a state that has a message, there must have been an intervening transition that extends with the message. Lemma 1 formalizes this property in our state theory $T_{state}(\rightsquigarrow)$, and we proved it using PVS.

Lemma 1 (Prefix Boot Extend)**.**

$$\begin{array}{l} \forall \pi \in path, t : \top, i, k \in \mathbb{N}. \\ \quad i \le k \wedge \pi(k) \text{ has } t \\ \qquad \supset \pi(i) \text{ is a subterm of } \pi(k) \\ \qquad\quad \vee \exists j \in \mathbb{N}.\, i \le j < k \wedge \pi(j+1) = \mathsf{ex}(t, \pi(j)) \end{array}$$

V. Strand Spaces

This section introduces our strand space theory of the envelope protocol, $T_{bnd}(\Pi)$. In strand space theory [24], the *trace* of a strand is a linearly ordered sequence of events $e_0 \Rightarrow \cdots \Rightarrow e_{n-1}$, and an *event* is either a message transmission $+t$ or a reception $-t$, where t has sort $\top$. A *strand space* Θ is a map from a set of strands to a set of traces. In the PVS theory of strand spaces, the set of strands is a prefix of the natural numbers, so a strand space is a finite sequence of traces.

In a strand space, a node identifies an event. The *nodes* of strand space Θ are $\{(s,i) \mid s \in Dom(\Theta), 0 \le i < |\Theta(s)|\}$, and the event at a node is $evt_\Theta(s,i) = \Theta(s)(i)$.

A message t_0 is *carried by* t_1, written $t_0 \sqsubseteq t_1$ if t_0 can be extracted from a reception of t_1, assuming the necessary keys are available. In other words, $\sqsubseteq$ is the smallest reflexive, transitive relation such that $t_0 \sqsubseteq t_0$, $t_0 \sqsubseteq (t_0, t_1)$, $t_1 \sqsubseteq (t_0, t_1)$, and $t_0 \sqsubseteq \{\!|t_0|\!\}_{t_1}$.

A message *originates* in trace c at index i if it is carried by $c(i)$, $c(i)$ is a transmission, and it is not carried by any event earlier in the trace. A message t is *non-originating* in a strand space Θ, written $non(\Theta, t)$, if it originates on no strand. A message t *uniquely originates* in a strand space Θ at node n, written $uniq(\Theta, t, n)$, if it originates in the trace of exactly one strand s at index i, and $n = (s, i)$.

The model of execution is a bundle. The pair $\Upsilon = (\Theta, \rightarrow)$ is a *bundle* if it defines a finite directed acyclic graph, where the vertices are the nodes of Θ, and an edge represents communication ($\rightarrow$) or strand succession ($\Rightarrow$) in Θ. For communication, if $n_0 \rightarrow n_1$, then there is a message t such that $evt_\Theta(n_0) = +t$ and $evt_\Theta(n_1) = -t$. For each reception node n_1, there is a unique transmission node n_0 with $n_0 \rightarrow n_1$. For a bundle Υ, its associated strand space will be denoted Θ_Υ unless the association is clear from the context.

Each acyclic graph has a transitive irreflexive relation $\prec$ on its vertices. The relation specifies the causal ordering of nodes in a bundle. A transitive irreflexive binary relation is also called a strict order.

Sorts:	M, $\top$, A, S, D, E		
Subsorts:	$\mathsf{A} < \top$, $\mathsf{S} < \top$, $\mathsf{D} < \top$, $\mathsf{E} < \top$		
Operations:	bt	$: \mathsf{M}$	TPM boot
	ex	$: \top \times \mathsf{M} \to \mathsf{M}$	TPM extend
	$(\cdot,\cdot)$	$: \top \times \top \to \top$	Pairing
	$\{\!\lvert \cdot \rvert\!\}_{(\cdot)}$	$: \top \times \mathsf{A} \to \top$	Asymmetric encryption
	$\{\!\lvert \cdot \rvert\!\}_{(\cdot)}$	$: \top \times \mathsf{S} \to \top$	Symmetric encryption
	$(\cdot)^{-1}$	$: \mathsf{A} \to \mathsf{A}$	Asymmetric key inverse
	$(\cdot)^{-1}$	$: \mathsf{S} \to \mathsf{S}$	Symmetric key inverse
	$\#$	$: \top \to \mathsf{S}$	Hashing
	$\mathsf{a}_i, \mathsf{b}_i$	$: \mathsf{A}$	Asymmetric key constants
	s_i	$: \mathsf{S}$	Symmetric key constants
	d_i	$: \mathsf{D}$	Data constants
	e_i	$: \mathsf{E}$	Text constants
	g_i	$: \top$	Tag constants
Equations:	${\mathsf{a}_i}^{-1} = \mathsf{b}_i \quad {\mathsf{b}_i}^{-1} = \mathsf{a}_i$		$(i \in \mathbb{N})$
	$\forall k : \mathsf{A}.\, (k^{-1})^{-1} = k$		$\forall k : \mathsf{S}.\, k^{-1} = k$

Fig. 2. Crypto Algebra with State Signature

$$
\begin{aligned}
\mathit{create}(t : \mathsf{A}|\mathsf{S}|\mathsf{D}|\mathsf{E}) &= +t \qquad \mathit{tag}_i = +\mathsf{g}_i \\
\mathit{pair}(t_0 : \top, t_1 : \top) &= -t_0 \Rightarrow -t_1 \Rightarrow +(t_0, t_1) \\
\mathit{sep}(t_0 : \top, t_1 : \top) &= -(t_0, t_1) \Rightarrow +t_0 \Rightarrow +t_1 \\
\mathit{enc}(t : \top, k : \mathsf{A}|\mathsf{S}) &= -t \Rightarrow -k \Rightarrow +\{\!|t|\!\}_k \\
\mathit{dec}(t : \top, k : \mathsf{A}|\mathsf{S}) &= -\{\!|t|\!\}_k \Rightarrow -k^{-1} \Rightarrow +t \\
\mathit{hash}(t : \top) &= -t \Rightarrow +\#t
\end{aligned}
$$

Fig. 3. Adversary Traces

The section so far has described the theory T_{bnd}. We now describe how this specializes to $T_{bnd}(\Pi)$. This description has been simplified somewhat from what is used in PVS. In the full theory, origination assumptions can be inherited from roles. See [22] for all the gory details.

When a bundle is a run of a protocol, the behavior of each strand is constrained by a role. Adversarial strands are constrained by roles as are non-adversarial strands. A *protocol* is a set of roles, and a *role* is a set of traces. A trace c is an *instance* of role r if c is a prefix of some member of r. More precisely, for protocol P, we say that bundle $\Upsilon = (\Theta, \rightarrow)$ is a *run of protocol* P if there exists a role assignment $ra \in \mathit{Dom}(\Theta) \to P$ such that for all $s \in \mathit{Dom}(\Theta)$, $\Theta(s)$ is an instance of $ra(s)$. In what follows, we fix the protocol P and only consider bundles that are runs of P.

The roles that constrain adversarial behavior are defined by the functions in Figure 3. The role defined by the function is all the traces that it generates. For example, the role associated with the function *pair* is $\{\mathit{pair}(t_0, t_1) \mid t_0, t_1 : \mathsf{M}\}$. For the encryption related roles, $k : \mathsf{A}|\mathsf{S}$ asserts that k is either a symmetric or asymmetric key. For the create role, $t : \mathsf{A}|\mathsf{S}|\mathsf{D}|\mathsf{E}$ asserts that t is an atom. The defining characteristic of an atom is it is that which the adversary can create out of thin air modulo origination assumptions.

As mentioned in the introduction, strand space theory is not equipped to handle protocols with state. Our approach is to use an under constrained notion of state within the unmodified strand space theory to show many bundles are not runs of the protocol, and then add missing constraints using PVS to eliminate remaining bundles that are incompatible with the true notion of state.

Each TPM operation is encoded by a role. The transition associated with an instance of a TPM operation is encoded by a receive-transmit pair of state-bearing events. For transition $m_0 \rightsquigarrow m_1$, the role contains

$$\cdots \Rightarrow -\{\!|\, \mathsf{g}_0, \mathit{pcr}(m_0)|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_0, \mathit{pcr}(m_1)|\!\}_{\#k} \Rightarrow \cdots,$$

where k is a symmetric key that is uncompromised and known only to all TPM operations, and the key is hashed and tag g_0 is included to ensure that a state-bearing message is never confused with any other protocol message. This encoding of state transitions is less restrictive than the true notion of state because more than one TPM operation can consume and transform one state-bearing message.

Using these receive-transmit pairs of state-bearing messages the TPM roles are represented in Figure 4, where tag g_1 is obtain and tag g_2 is refuse. In the *extend* role, we now show the two initial messages that provide replay prevention; the TPM supplies a fresh nonce as a session ID that must appear with the value to be extended into the PCR. The *createkey* role does not interact with the state. It simply creates a key that will be constrained by the state in the boot role.

In this notation the Alice role is the following. We also show Alice's messages which implement replay-prevention.

$$
\begin{aligned}
&\mathit{alice}(\mathit{sid}, v : \mathsf{D}, \mathit{esk} : \mathsf{S}, k, \mathit{tpmk}, \mathit{aik} : \mathsf{A}, n : \mathsf{E}, p : \top) = \\
&\quad +(\mathsf{g}_4, \mathit{tpmk}, \{\!|\mathit{esk}|\!\}_{\mathit{tpmk}}) \Rightarrow -(\mathsf{g}_4, \mathit{sid}) \\
&\qquad \Rightarrow +\{\!|\, \mathsf{g}_5, n, \mathit{sid}|\!\}_{\mathit{esk}} \Rightarrow +(\mathsf{g}_9, \#(\mathsf{g}_1, \#(n, p))) \\
&\qquad \Rightarrow -\{\!|\, \mathsf{g}_8, \#(\mathsf{g}_1, \#(n, p))|\!\}_{\mathit{aik}} \Rightarrow +\{\!|v|\!\}_k
\end{aligned}
$$

VI. CPSA

This section discusses how we use our analysis tool CPSA to infer results in the theory $T_{bnd}(\Pi)$. CPSA carries out enrich-by-need analysis, and characterizes the set of bundles consistent with a partial description of a bundle. These partial descriptions are called *skeletons*. CPSA takes as input an initial skeleton, and when it terminates it outputs a set of

$$
\begin{aligned}
&boot(k:\mathsf{S}, p:\top) = && (\mathsf{g}_3 \text{ is boot})\\
&\quad -\mathsf{g}_3 \Rightarrow -\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_0, \mathsf{s}_0 \,|\!\}_{\#k}\\
&extend(sid:\mathsf{D}, tpmk:\mathsf{A}, esk, k:\mathsf{S}, p, t:\top) = && (\mathsf{g}_4 \text{ is session, } \mathsf{g}_5 \text{ is extend})\\
&\quad -(\mathsf{g}_4, tpmk, \{|esk|\}_{tpmk}) \Rightarrow +(\mathsf{g}_4, sid) \Rightarrow -\{\!|\, \mathsf{g}_5, t, sid \,|\!\}_{esk}\\
&\qquad \Rightarrow -\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_0, \#(t,p) \,|\!\}_{\#k}\\
&quote(k:\mathsf{S}, aik:\mathsf{A}, p, n:\top) = && (\mathsf{g}_6 \text{ is quote})\\
&\quad -(\mathsf{g}_6, n) \Rightarrow -\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_6, p, n \,|\!\}_{aik}\\
&decrypt(m, t:\top, k', aik:\mathsf{A}, k:\mathsf{S}) = && (\mathsf{g}_7 \text{ is decrypt, } \mathsf{g}_8 \text{ is created})\\
&\quad -(\mathsf{g}_7, \{|m|\}_{k'}) \Rightarrow -\{\!|\, \mathsf{g}_8, k', p \,|\!\}_{aik} \Rightarrow -\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +\{\!|\, \mathsf{g}_0, p \,|\!\}_{\#k} \Rightarrow +m\\
&createkey(k, aik:\mathsf{A}, t:\top) = && (\mathsf{g}_9 \text{ is create key})\\
&\quad -(\mathsf{g}_9, t) \Rightarrow +\{\!|\, \mathsf{g}_8, k, t \,|\!\}_{aik}
\end{aligned}
$$

Fig. 4. State-Bearing Traces

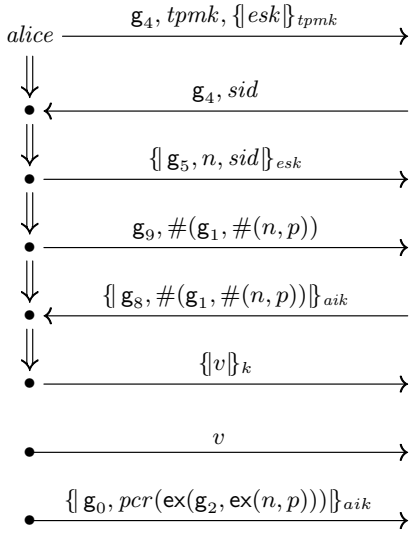

Fig. 5. Alice Point-Of-View

more descriptive skeletons with the property that any bundle containing the structure in the initial skeleton also contains all the structure in one of the output skeletons. In particular, it infers all of the non-adversarial behavior that must be present in any bundle satisfying the initial description. Of course for some initial skeletons there may be no bundles that are consistent with them. In this case, CPSA outputs the empty set.

Consider the security goal for the Envelope Protocol. It should be the case that there are no bundles in which an instance of the Alice role runs to completion and in which both the secret is leaked and a refusal token is generated. We can feed CPSA an input skeleton representing this undesirable situation. The input is vizualized in Figure 5.

We would hope CPSA could determine that no bundles are consistent with this input and return the empty set. However, our technique of using state-bearing messages to represent the TPM state transitions underconstrains the set of possible state paths. For this reason, CPSA actually produces one skeleton in its output. This skeleton represents some activity that must have occured within the TPM in any bundle conforming to the initial skeleton. It contains an instance of the decrypt role (to explain the secret leaking), an instance of the quote role (to explain the creation of the refusal token), and several instances of the extend role (to explain how the TPM state evolved in order to allow the other two operations).

Figure 6 displays the relevant portion of CPSA's output displaying only the state-bearing nodes of the extend strands inferred by CPSA. Notice that two of the extend strands branch off from the third strand. This is a state split in which a single state evolves in two distinct ways. The technique of using state-bearing messages is not sufficient to preclude this possibility.

CPSA's enrich-by-need approach is closer to model finding than to theorem proving. In order to use CPSA's results to our advantage we need to express its conclusions in the logical theory $T_{bnd}(\Pi)$. For that purpose we transform our skeletons into formulas in order-sorted logic and define what it means for a bundle to satisfy these formulas. The sorts are the message algebra sorts augmented with a sort Z for strands and sort N for nodes. The atomic formula $\mathsf{htin}(z,h,c)$ asserts that strand z has a length of at least h, and its trace is a prefix of trace c. The formula $n_0 \ll n_1$ asserts node n_0 precedes node n_1. The formula $\mathsf{non}(t)$ asserts that message t is non-originating, and $\mathsf{uniq}(t,n)$ asserts that message t uniquely originates at node n. Finally, the formula $\mathsf{sends}(n,t)$ asserts that the event at node n is a transmission of message t. The roles of the protocol serve as function symbols. A skeleton is represented by the conjunction of all facts true in the skeleton.

We encode an entire CPSA analysis by first encoding the input skeleton and the output skeletons. The analysis is then encoded as an implication with the formula for the input on the left, and the disjunction of the formulas for the outputs on the right. When CPSA discovers that there are no bundles compatible with the initial skeleton, the conclusion is encoded as the empty disjunction, $\bot$.

The satisfaction relation is defined using the clauses in Figure 7. It relates a bundle, a variable assignment, and a formula: $\Upsilon, \alpha \models \Phi$. A bundle Υ is described by a skeleton iff the skeleton's sentence Φ is modeled by Υ, written $\Upsilon \models \Phi$.

The formula $\exists X.\Phi$ that specifies the initial skeleton rele-

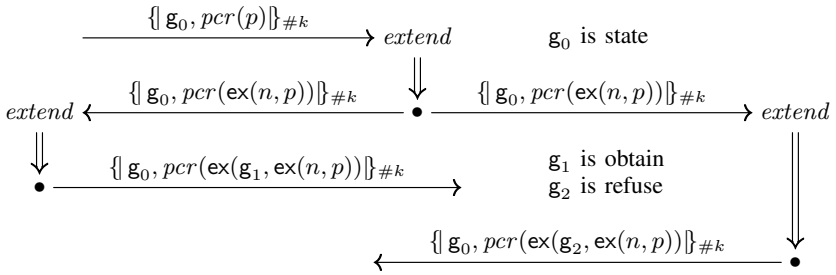

Fig. 6. State Splitting

$$
\begin{array}{ll}
\Upsilon, \alpha \models x = y & \text{iff } \alpha(x) = \alpha(y); \\
\Upsilon, \alpha \models \mathsf{htin}(z, h, c) & \text{iff } |\Theta_\Upsilon(\alpha(z))| \geq \alpha(h) \text{ and} \\
 & \quad \Theta_\Upsilon(\alpha(z)) \text{ is a prefix of } \alpha(c); \\
\Upsilon, \alpha \models n_0 \ll n_1 & \text{iff } \alpha(n_0) \prec_\Upsilon \alpha(n_1); \\
\Upsilon, \alpha \models \mathsf{non}(t) & \text{iff } non(\Theta_\Upsilon, \alpha(t)); \\
\Upsilon, \alpha \models \mathsf{uniq}(t, n) & \text{iff } uniq(\Theta_\Upsilon, \alpha(t), \alpha(n)); \\
\Upsilon, \alpha \models \mathsf{sends}(n, t) & \text{iff } evt_{\Theta_\Upsilon}(\alpha(n)) = +\alpha(t).
\end{array}
$$

Fig. 7. Satisfaction

vant to the Envelope Protocol security goal (Figure 5) is

$$
\begin{array}{l}
\exists v : \mathsf{D}, esk : \mathsf{S}, k, aik : \mathsf{A}, n : \mathsf{E}, p : \top, z : \mathsf{Z}, n_1, n_2 : \mathsf{N}. \\
\quad \mathsf{htin}(z, 4, alice(v, esk, k, aik, n, p)) \wedge \mathsf{sends}(n_1, v) \\
\qquad \wedge \mathsf{sends}(n_2, \{\!|\, \mathsf{g}_0, pcr(\mathsf{ex}(\mathsf{g}_2, \mathsf{ex}(n, p)))\, |\!\}_{aik}) \\
\qquad \wedge \mathsf{non}(aik) \wedge \mathsf{non}(esk) \\
\qquad \wedge \mathsf{uniq}(n, (z, 1)) \wedge \mathsf{uniq}(v, (z, 4))
\end{array}
$$

The output skeleton is much larger and its formula $\exists X.\Psi$ is correspondingly large. The relevant part of this formula representing the fragment in Figure 6 is

$$
\begin{array}{l}
\exists n : \mathsf{E}, esk, k : \mathsf{S}, p : \top, z_1, z_2, z_3 : \mathsf{Z}. \\
\quad \mathsf{htin}(z_1, 3, extend(esk, k, pcr(p), n)) \\
\qquad \wedge \mathsf{htin}(z_2, 3, extend(esk, k, pcr(\mathsf{ex}(n, p)), \mathsf{g}_1)) \\
\qquad \wedge \mathsf{htin}(z_3, 3, extend(esk, k, pcr(\mathsf{ex}(n, p)), \mathsf{g}_2))
\end{array}
$$

The full formula for Ψ has many more conjuncts.

The results of CPSA's analysis for the Envelope Protocol can thus be represented in $T_{bnd}(\Pi)$ as the following.

Lemma 2. $\forall X.(\Phi \supset \exists Y.\Psi)$

Unlike Lemma 1, this lemma is not derived within PVS. Rather, it is established by CPSA in the course of its analysis and imported into PVS as an axiom.

VII. Reasoning About Messages and State

This section presents some details of the theory $T_{annot}(\Pi, \rightsquigarrow)$. We then show how the previous lemmas combine allowing us to conclude that the security goal of the Envelope Protcol is achieved.

In $T_{annot}(\Pi, \rightsquigarrow)$, the state transitions associated with a protocol are specified by annotating some events in a role of Π with a subset of the transition relation $\rightsquigarrow$. The reason for annotating events with a subset of the transition relation, rather than an element, will be explained at the end of this section.

We use $\perp$ for an event that is not annotated, and $\uparrow a$ for an event that is annotated with a. The events that are annotated are the transmissions associated with receive-transmit pairs of state-bearing messages.

$$
\begin{array}{ccccc}
\cdots \Rightarrow & -\{\!|\, \mathsf{g}_0, pcr(m_0)\, |\!\}_{\#k} & \Rightarrow & +\{\!|\, \mathsf{g}_0, pcr(m_1)\, |\!\}_{\#k} & \Rightarrow \cdots \\
\perp & \perp & & \uparrow\{(m_0, m_1)\} \cap \rightsquigarrow & \perp
\end{array}
$$

A node in a bundle is annotated if the event in the role from which it was instantiated is annotated. The set of nodes in Υ that are annotated is $anode(\Upsilon)$, and $anno(\Upsilon, n, a)$ asserts that node n in Υ is annotated with some $a \subseteq \rightsquigarrow$. In the Envelope Protocol, a node annotated by a TPM extend role cannot be an instance of any other role.

Our goal is to reason only with bundles that respect state semantics. A bundle Υ with a transition annotating role assignment is *compatible* [15, Def. 11] with transition relation $\rightsquigarrow$ if there exists $\ell \in \mathbb{N}$, $f \in anode(\Upsilon) \to \{0, 1, \ldots, \ell - 1\}$, and $\pi \in path$ such that

1. f is bijective;
2. $\forall n_0, n_1 \in anode(\Upsilon).\, n_0 \prec n_1 \iff f(n_0) < f(n_1)$;
3. $\forall n \in anode(\Upsilon), a \subseteq \rightsquigarrow.$
   $anno(\Upsilon, n, a) \supset (\pi(f(n)), \pi(f(n) + 1)) \in a.$

A bundle that satisfies $T_{annot}(\Pi, \rightsquigarrow)$ is a compatible bundle.

Because the function f is bijective, all annotated nodes in a compatible bundle are totally ordered. Looking back at Figure 6, either the nodes in the leftmost strand precede the nodes in the rightmost strand or succeed them.

The compatible bundle assumption allows one to infer the existence of nodes that are not revealed by CPSA. In the case of the Envelope Protocol this is done by importing the Prefix Boot Extend Lemma (Lemma 1) from $T_{state}(\rightsquigarrow)$ into the strand space world by proving the following lemma within $T_{annot}(\Pi, \rightsquigarrow)$ using PVS.

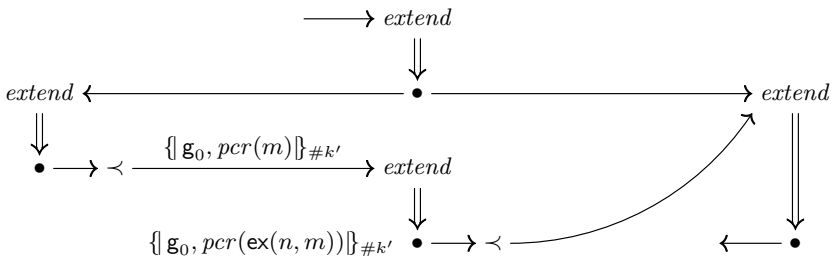

Fig. 8. Inferred Extend Strand

Lemma 3 (Bridge)**.**

$$\begin{array}{l}\forall \Upsilon.\ compatible(\Upsilon) \supset \\ \quad \forall n_0, n_1 \in anode(\Upsilon), t : \top, a_0, a_1 \subseteq \rightsquigarrow, m_0, m_1 : \mathsf{M}. \\ \qquad anno(\Upsilon, n_0, a_0) \wedge anno(\Upsilon, n_1, a_1) \\ \qquad\quad \wedge\, n_0 \prec n_1 \wedge m_1 \text{ has } t \\ \qquad\quad \wedge\, a_0 \subseteq \{(m_2, m_3) \mid m_3 = m_0\} \\ \qquad\quad \wedge\, a_1 \subseteq \{(m_2, m_3) \mid m_2 = m_1\} \\ \qquad\quad \supset m_0 \text{ is a subterm of } m_1 \\ \qquad\qquad \vee\, \exists n \in anode(\Upsilon), m : \mathsf{M}. \\ \qquad\qquad\quad anno(\Upsilon, n, \{(m_2, m_3) \mid m_2 = m \\ \qquad\qquad\qquad\qquad \wedge\, m_3 = \mathsf{ex}(t, m_2)\}) \\ \qquad\qquad\quad \vee\, n_0 \prec n \wedge n \prec n_1\end{array}$$

The proof of the Bridge Lemma makes use of every part of the definition of compatibility.

The implication in the Prefix Boot Extend Lemma corresponds to the second implication in the Bridge Lemma. The correspondence of the conclusions of each implication is straightforward, however, the hypothesis of the Bridge Lemma is much more complicated than the one in the Prefix Boot Extend Lemma. Yet all it is saying is that the beginning and ending states over the range of the path are m_0 and m_1, where as in the Prefix Boot Extend Lemma, those states are simply referred to by $\pi(i)$ and $\pi(k)$.

This Bridge Lemma allows us to infer the existence of another extend strand between the two strands that represent the state split. This theorem is also proved with PVS in $T_{annot}(\Pi, \rightsquigarrow)$; however, syntactically it is a sentence of the language of $T_{bnd}(\Pi)$. That is, $T_{annot}(\Pi, \rightsquigarrow)$ is not conservative over $T_{bnd}(\Pi)$, because $T_{annot}(\Pi, \rightsquigarrow)$'s models are only the compatible bundles. The theorem is the following.

Theorem 4 (Inferred Extend Strand)**.**

$$\begin{array}{l}\forall z_0, z_1 : \mathsf{Z}, t, t_0, t_1 : \top, m_0, m_1 : \mathsf{M}, esk_0, esk_1, k_0, k_1 : \mathsf{A}. \\ \quad \mathsf{htin}(z_0, 2, extend(esk_0, k_0, pcr(m_0), t_0)) \\ \qquad \wedge\, \mathsf{htin}(z_1, 2, extend(esk_1, k_1, pcr(m_1), t_1)) \\ \qquad \wedge\, (z_0, 1) \ll (z_1, 0) \wedge m_1 \text{ has } t \\ \qquad \supset \mathsf{ex}(t_0, m_0) \text{ is a subterm of } m_1 \\ \qquad\quad \vee\, \exists z : \mathsf{Z}, m : \mathsf{M}, esk, k : \mathsf{A}. \\ \qquad\qquad \mathsf{htin}(z, 2, extend(esk, k, pcr(m), t)) \\ \qquad\qquad\quad \wedge\, (z_0, 1) \ll (z, 0) \wedge (z, 1) \ll (z_1, 0)\end{array}$$

The consequence of Theorem 4 can be visualized as in Figure 8.

Putting all the pieces together, the proof of the Envelope Protocol security goal can be summarized as follows. We derive Lemma 1 within $T_{state}(\rightsquigarrow)$. Lemma 3 is the counterpart within $T_{annot}(\Pi, \rightsquigarrow)$, which is the key result used to prove Theorem 4, which sits at the intersection of $T_{annot}(\Pi, \rightsquigarrow)$ and $T_{bnd}(\Pi)$. All of this is done within PVS. Meanwhile, CPSA is used to derive Lemma 2 in the theory $T_{bnd}(\Pi)$. This combines with Theorem 4 (under both possible orderings of the extend strands) to provide two more starting points to feed back into CPSA which, in turn, discovers that these partial executions cannot in fact occur. This cooperation between CPSA and PVS ultimately shows that the security goal for the Envelope Protocol is achieved. That is, if Alice runs to completion, then there can be no execution in which her parent both learns her encrypted secret and also generates a refusal quote.

But why annotate events with subsets of the transition relation rather than elements of it? The extend role does not guarantee its received state bearing message is of the form $\{\!|\, \mathsf{g}_0, pcr(m_0) |\!\}_{\#k}$. All it says is that it has the form $\{\!|\, \mathsf{g}_0, t_0 |\!\}_{\#k}$. Bundles in which t_0 is not in the range of the pcr function must be eliminated from consideration.

The formalism has been set up so that in a bundle, a node's annotation is either a singleton or the empty set. The actual trace and annotations are

$$\begin{array}{ccccccc}\cdots & \Rightarrow & -\{\!|\, \mathsf{g}_0, t_0 |\!\}_{\#k} & \Rightarrow & +\{\!|\, \mathsf{g}_0, t_1 |\!\}_{\#k} & \Rightarrow & \cdots \\ \bot & & \bot & & \uparrow \mathcal{A} & & \bot\end{array}$$

where $\mathcal{A} = \{(m_0, m_1 \mid t_0 = pcr(m_0) \wedge t_1 = pcr(m_1))\} \cap \rightsquigarrow$.

A bundle in which a received state encoding message is not in the range of the pcr function will have a node annotated with the empty set. This bundle does not respect state semantics and is eliminated from consideration by the definition of compatibility.

VIII. Conclusion

The proof of the Envelope Protocol security goal presented here shows a detailed example of our method for applying CPSA to systems that include a state component. CPSA was coupled with about 2400 lines of PVS specifications to produce a proof of a difficult security goal. The method is sound due to the use of the common foundation of strand space theory for all reasoning.

The approach could be improved in two main ways. First, the proofs within PVS are strenuous. We would like to develop a method in which—apart perhaps from a few key reusable lemmas in the state theory $T_{state}(\rightsquigarrow)$—the remainder of the reasoning concerning both state and protocol behavior occurs automatically in CPSA's automated, enrich-by-need manner. Second, there is some artificiality in the state-threading representation that we have used here. It requires the protocol

description to make explicit the details of the full state, and to express each state change in a syntactic, template-based form. Moreover, the state information is also redundantly encoded in the annotations that appear in $T_{annot}(\Pi, \rightsquigarrow)$. Our earlier work [15] instead encapsulated all of the state information in a labeled transition relation. The protocol definitions contain only a type of "neutral node" which are neither transmissions nor receptions. These nodes are associated with the same labels as appear in labeled transitions. This allows us to define "compatibility," and to work with protocol and state definitions as independent modules. We intend also to explore this style of definition.

References

[1] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM*, 52(1):102–146, January 2005.

[2] K. Ables and M. Ryan. Escrowed data and the digital envelope. *Trust and Trustworthy Computing*, pages 246–256, 2010.

[3] Myrto Arapinis, Eike Ritter, and Mark Dermot Ryan. Statverif: Verification of stateful processes. In *Computer Security Foundations Symposium (CSF)*, pages 33–47. IEEE, 2011.

[4] Bruno Blanchet. An efficient protocol verifier based on Prolog rules. In *14th Computer Security Foundations Workshop*, pages 82–96. IEEE CS Press, June 2001.

[5] Jolyon Clulow. On the security of PKCS#11. In *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2003.

[6] Véronique Cortier, Gavin Keighren, and Graham Steel. Automatic analysis of the security of xor-based key management schemes. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 538–552. Springer, 2007.

[7] Véronique Cortier and Graham Steel. A generic security api for symmetric key management on cryptographic devices. In *Computer Security–ESORICS 2009*, pages 605–620. Springer, 2009.

[8] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[9] Anupam Datta, Jason Franklin, Deepak Garg, and Dilsun Kaynar. A logic of secure systems and its application to trusted computing. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 221–236. IEEE, 2009.

[10] Stéphanie Delaune, Steve Kremer, Mark D Ryan, and Graham Steel. A formal analysis of authentication in the tpm. In *Formal Aspects of Security and Trust*, pages 111–125. Springer, 2011.

[11] Stéphanie Delaune, Steve Kremer, Mark D. Ryan, and Graham Steel. Formal analysis of protocols based on TPM state registers. In *IEEE Symposium on Computer Security Foundations*. IEEE CS Press, June 2011.

[12] Sibylle Fröschle and Nils Sommer. Reasoning with past to prove pkcs# 11 keys secure. In *Formal Aspects of Security and Trust*, pages 96–110. Springer, 2011.

[13] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[14] Sigrid Gürgens, Carsten Rudolph, Dirk Scheuermann, Marion Atts, and Rainer Plaga. Security evaluation of scenarios based on the tcg's tpm specification. In *Computer Security–ESORICS 2007*, pages 438–453. Springer, 2007.

[15] Joshua D. Guttman. State and progress in strand spaces: Proving fair exchange. *Journal of Automated Reasoning*, 48(2):159–195, 2012.

[16] Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 2014.

[17] Jonathan Herzog. Applying protocol analysis to security device interfaces. *IEEE Security & Privacy*, 4(4):84–87, 2006.

[18] Moses D. Liskov, Paul D. Rowe, and F. Javier Thayer. Completeness of CPSA. Technical Report MTR110479, The MITRE Corporation, March 2011. http://www.mitre.org/publications/technical-papers/completeness-of-cpsa.

[19] Sebastian Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. *ACM Conference on Computer and Communications Security*, pages 351–360, 2010.

[20] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag. http://pvs.csl.sri.com.

[21] John D. Ramsdell. Deducing security goals from shape analysis sentences. The MITRE Corporation, April 2012. http://arxiv.org/abs/1204.0480.

[22] John D. Ramsdell. Proving security goals with shape analysis sentences. Technical Report MTR130488, The MITRE Corporation, September 2013. http://arxiv.org/abs/1403.3563.

[23] John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. http://hackage.haskell.org/package/cpsa.

[24] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[25] Paul Youn, Ben Adida, Mike Bond, Jolyon Clulow, Jonathan Herzog, Amerson Lin, Ronald Rivest, and Ross Anderson. Robbing the bank with a theorem prover. In *Security Protocols Workshop*, 2007. Available at http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-644.pdf.