Formal Support for Standardizing Protocols with State

Joshua D. Guttman, Moses D. Liskov,
John D. Ramsdell, and Paul D. Rowe

The MITRE Corporation

Abstract. Many cryptographic protocols are designed to achieve their goals using only messages passed over an open network. Numerous tools, based on well-understood foundations, exist for the design and analysis of protocols that rely purely on message passing. However, these tools encounter difficulties when faced with protocols that rely on non-local, mutable state to coordinate several local sessions.
We adapt one of these tools, CPSA, to provide automated support for reasoning about state. We use Ryan's Envelope Protocol as an example to demonstrate how the message-passing reasoning can be integrated with state reasoning to yield interesting and powerful results.

1 Introduction

Many protocols involve only message transmission and reception, controlled by rules that are purely local to a session of the protocol. Typical protocols for authentication and key establishment are of this kind; each participant maintains only the state required to remember what messages must still be transmitted, and what values are expected in messages to be received from the peer.

Other protocols interact with long-term state, meaning state that persists across different sessions and may control behavior in other sessions. A bank account is a kind of long term state, and it helps to control the outcome of protocol sessions in the ATM network. Specifically, the session fails when we try to withdraw money from an empty account. Of course, one session has an effect on others through the state: When we withdraw money today, there will be less remaining to withdraw tomorrow.

Hardware devices frequently participate in protocols, and maintain state that helps control those protocols. For example, PKCS#11 devices store and use keys, and are constrained by key attributes that control e.g. which keys may be used to wrap and export other keys. Trusted Platform Modules (TPMs) maintain Platform Configuration Registers (PCRs) some of which are modified only by certain special instructions. Thus, digitally signing the values in these registers attests to the history of the platform.

State-based protocols are more challenging to analyze than protocols in which all state is session-local. Among the executions that are possible given the message flow patterns, one must identify those for which a compatible sequence

of states exists. Thus, to justify standardizing protocols involving PKCS#11 devices or TPMs, one must do a deeper analysis than for stateless protocols. Indeed, since these devices are themselves standardized, it is natural to want to define and justify protocols that depend only on their required properties, rather than any implementation specific peculiarities.

The goal of this paper is to explain formal ideas that can automate this analysis, and to describe a support tool that assists with it.

Enrich-by-need analysis. In our analysis, the input is a fragment of protocol behavior. The output gives zero or more executions that contain this fragment. We call this approach "enrich-by-need" analysis (borrowed from Guttman [15]), because it is a search process that gradually adds information as needed to explain the events that are already under consideration.

An analysis begins with an execution fragment $\mathbb{A}$, which may, for instance, reflect the assumption that one participant has engaged in a completed local session; that certain nonces were freshly chosen; and that certain keys were uncompromised.

The result of the analysis is a set S of executions enriching the starting fragment $\mathbb{A}$. An algorithm implementing this approach is sound if—for every possible execution $\mathbb{C}$ that enriches $\mathbb{A}$–there is a member $\mathbb{B} \in S$ such that $\mathbb{C}$ enriches $\mathbb{B}$.

We do not require S to contain all possible executions because there are infinitely many of them if any. For instance, executions may always be extended by including additional sessions by other protocol participants. Thus, we want the set S to contain representatives that cover all of the *essentially different* possibilities. We call these representatives S the *shapes* for $\mathbb{A}$.

In practice, the set S of shapes for $\mathbb{A}$ is frequently finite and small.

When we start with a fragment $\mathbb{A}$ and find that it has the empty set $S = \emptyset$ of shapes, that means that no execution contains all of the structure in $\mathbb{A}$. To use this technique to show confidentiality assertions, we include a disclosure event in $\mathbb{A}$. If $\mathbb{A}$ extends to no possible executions at all, we can conclude that this secret cannot be revealed. If S is non-empty, the shapes are attacks that show how the confidentiality claim could fail.

The set S of shapes, when finite, also allows us to ascertain whether authentication properties are satisfied. If each shape $\mathbb{B} \in S$ satisfies an authentication property, then every possible execution $\mathbb{C}$ enriching $\mathbb{A}$ must satisfy the property too: They all contain at least the behavior exhibited in some shape, which already contained the events that the authentication property required.

This style of analysis is particularly useful in a partially ordered execution model, such as *strand spaces* provide [25]. In this model, when events e_1, e_2 are causally unrelated, neither precedes the other. In linearly ordered execution models, both interleavings $e_1 \prec e_2$ and $e_2 \prec e_1$ are possible, and must be considered. When there are many such pairs, this leads to exponentially many interleavings. None of the differences between them are significant.

A logical interpretation. This type of analysis has a natural logical interpretation. Suppose we explore the shapes of a fragment $\mathbb{A}$. We can summarize

all of the facts holding in $\mathbb{A}$ by a conjunction of atomic assertions; we call this conjunction the *characteristic formula* $\mathsf{cf}(\mathbb{A})$. The protocol analysis will tell us what must occur in all executions that satisfy this formula. In particular, suppose that the analysis delivers a finite set of shapes $S = \{\mathbb{B}_1, \ldots \mathbb{B}_k\}$. Each of the shapes $\mathbb{B}_i$ has a characteristic formula $\mathsf{cf}(\mathbb{B}_i)$. If $\boldsymbol{y_i}$ are the variables that occur in $\mathsf{cf}(\mathbb{B}_i)$ but not in $\mathsf{cf}(\mathbb{A})$, then we have learnt that in this branch of the analysis, $\exists \boldsymbol{y_i} \,.\, \mathsf{cf}(\mathbb{B}_i)$ holds. Hence, the logical content of the analysis is that the protocol ensures:

$$\mathsf{cf}(\mathbb{A}) \Longrightarrow \bigvee_{i \leq k} \exists \boldsymbol{y_i} \,.\, \mathsf{cf}(\mathbb{B}_i). \tag{1}$$

We call this the *shape analysis formula* for $\mathbb{A}$. In fact, it is the strongest security goal with hypothesis $\mathsf{cf}(\mathbb{A})$ that the protocol achieves.

In case the analysis tells us that $\mathbb{A}$ is impossible, i.e. that $S = \emptyset$, then the conclusion of the shape analysis formula is the empty disjunction. The empty disjunction is the constantly false formula. Thus, the shape analysis formula is equivalent to $\neg(\mathsf{cf}(\mathbb{A}))$.

For a fuller discussion of these ideas, as they apply to security protocol comparison and standardization, see work by Rowe et al. [24]. In particular, they argue that the natural partial ordering of implication among formulas leads to a useful way to measure and compare the security properties of alternative versions of protocols under standardization.

Contributions of this paper. In this paper, we make three main contributions:

1. We identify two central axioms of state that formalize the semantics of state-respecting executions (Def. 2).
   (a) One axiom characterizes how state transitions control protocol behavior. It says that the state produced by a transition can be consumed by at most one immediately subsequent transition.
   This is the essence of how the state-respecting analysis differs from standard message-based analysis.
   (b) A second axiom constrains state observations. An event in which state s is observed must occur after a transition that produces s, but before s is consumed by the next transition.
   Like the reader/writer principle in concurrency, this allows many observations to occur with no intrinsic order among them, so long as they all occur while that state is available. It preserves the advantages of our partial order model, as enriched with state.
2. We incorporated these two axioms into the tool CPSA [22], obtaining a tool that can perform state-respecting enrich-by-need protocol analysis.
3. We applied the resulting version of CPSA to an interesting TPM-based protocol, the Envelope Protocol [2], verifying that it meets its security goal. We have also analyzed some incorrect variants, obtaining attacks.

Roadmap. After describing the Envelope Protocol and the TPM behaviors it relies on (Section 2), we introduce our protocol model (Section 3) in both its plain form, and the form enriched by the axioms in Contribution 1. Section 4 describes the CPSA analysis in the original model where state propagation is not distinguished from message-passing, and in the enriched model. We turn to related work in Section 5. Section 6 briefly describes the logical interpretation mentioned above. The bottom line here is that no aspect of the logical language or its interpretation needs to change when it is transported to the enriched model. This adds evidence that our method is a clean and non-disruptive extension to the work of Rowe et al [24]. We end with a brief comment on conclusions and future work.

2 The Envelope Protocol

We use Mark Ryan's Envelope Protocol [3] as a concrete example throughout the paper. The protocol leverages cryptographic mechanisms supported by a TPM to allow one party to package a secret such that another party can either reveal the secret or prove the secret never was and never will be revealed, but not both.

It is a particularly useful example to consider because it is carefully designed to use state in an essential way. In particular, it creates the opportunity to take either of two branches in a state sequence, but not both. In taking one branch, one loses the option to take the other. In this sense, it utilizes the non-monotonic nature of state that distinguishes it from the monotonic nature of messages. Additionally, although the Envelope Protocol is not standardized, it demonstrates advanced and useful ways to use the TPM. Standardization of such protocols is under the purview of the Trusted Computing Group (TCG). It will be very useful to understand the fundamental nature of state and to provide methods and tools to support the future standardization of protocols involving devices such as the TPM.

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.

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 has the ability to either return the envelope or

tear it. In a purely message-based protocol the adversary will never lose these abilities.

Cryptographic version. The cryptographic version of this protocol uses a TPM to achieve the security goal. Here 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 PCR and only responding to five commands.

A `boot` command (re)sets the PCR to a known value. The `extend` command takes a piece of data, d, and replaces the current value s of the PCR state with the hash of d and s, denoted $\#(d, s)$. 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 to 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 a new key k' to be used only when a specific value is present in the PCR. In particular, the constraining value cv she chooses is the following:

$$cv = \#(\mathsf{obt}, \#(n, s))$$

where obt is a string constant and s represents an arbitrary PCR value prior the extend command. She then encrypts her secret v with k', denoted $\{|v|\}_{k'}$.

Using typical message passing notation, Alice's part of the protocol might be represented as follows (where we temporarily ignore the replay protection for the `extend` command):

$$\begin{array}{rcl:l} \text{A} & \rightarrow \text{TPM} & : & \{|\mathsf{ext}, n|\}_k \\ \text{A} & \rightarrow \text{TPM} & : & \mathsf{create}, \#(\mathsf{obt}, \#(n, s)) \\ \text{TPM} & \rightarrow \text{A} & : & k' \\ \text{A} & \rightarrow \text{Parent} & : & \{|v|\}_{k'} \end{array}$$

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 refuse to learn the secret and extend the PCR with the string ref and then generate a TPM quote as evidence the secret will never be exposed. The goal of the Envelope Protocol

Sorts:	$\top$, A, S, D, E		
Subsorts:	$\mathsf{A} < \top$, $\mathsf{S} < \top$, $\mathsf{D} < \top$		
Operations:	$(\cdot,\cdot)$	$: \top \times \top \to \top$	Pairing
	$\{\!\lvert \cdot \rvert\!\}_{(\cdot)}$	$: \top \times \top \to \top$	Encryption
	$(\cdot)^{-1}$	$: \mathsf{A} \to \mathsf{A}$	Asymmetric key inverse
	$(\cdot)^{-1}$	$: \mathsf{S} \to \mathsf{S}$	Symmetric key inverse
	$\#$	$: \top \to \top$	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
	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. 1. Crypto algebra with state signature

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, $\{\!|\, \mathsf{quote}, \#(\mathsf{ref}, \#(n,s)), \{\!|v|\!\}_{k'} |\!\}_{aik}$.

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

$$\#(\mathsf{obt}, \#(n,s)) \quad \neq \quad \#(\mathsf{ref}, x) \tag{2}$$

Formal protocol model. We formalize the TPM-based version of the Envelope Protocol using strand spaces [13]. Messages and states are represented as elements of a crypto term algebra whose signature is shown in Fig. 1. The algebra is the initial quotient term algebra over the signature. 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 $\{\!\lvert \cdot \rvert\!\}_{(\cdot)}$, hashing $\#(\cdot)$, and pairing $(\cdot,\cdot)$, where the comma operation is right associative and parentheses are omitted when the context permits.

We represent each TPM command with a separate role that receives a request, consults and/or changes the state and optionally provides a response. We use $m{\to}\bullet$ and $\bullet{\to}m$ to represent the reception and transmission of message m respectively. Similarly, we use $s{\rightsquigarrow}\circ$ and $\circ{\rightsquigarrow}s$ to represent the actions of reading and writing the value s to the state.

As noted above, the `boot` role and the `extend` role are the only two roles that alter the state. This is depicted with the single event ${\rightsquigarrow}\circ{\rightsquigarrow}$ that atomically reads and then alters the state. The `boot` role receives the command and resets any current state s to the known value s_0. An alternate version of `boot` is needed to ensure that our sequences of state are well-founded. This version has a single state write event $\circ{\rightsquigarrow}\,\mathsf{s}_0$.

The `extend` role first creates an encrypted channel by receiving an encrypted session key *esk* which is itself encrypted by some other secured TPM asymmetric

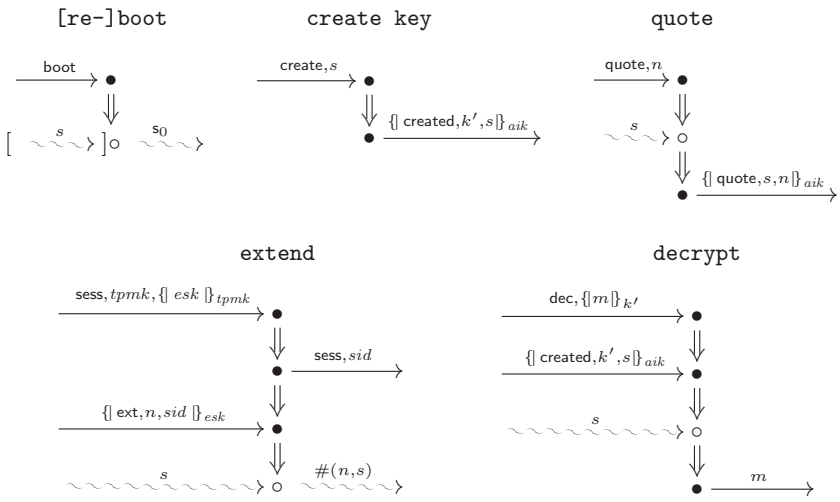

Fig. 2. TPM roles

key $tpmk$. The TPM replies with a random session id sid to protect against replay. It then receives the encrypted command to extend the value n into the PCR and updates the arbitrary state s to become $\#(n, s)$.

The `create key` role does not interact directly with the state. It receives the command with the argument s specifying a state. It then replies with a signed certificate for a freshly created public key k' that binds it to the state value s. The certificate asserts that the corresponding private key k'^{-1} will only be used in the TPM and only when the current value of the state is s. This constraint is leveraged in the `decrypt` role which receives a message m encrypted by k' and a certificate for k' that binds it to a state s. The TPM then consults the state (without changing it) to ensure it is in the correct state before performing the decryption and returning the message m.

Finally, the `quote` role receives the command together with a nonce n. It consults the state and reports the result s in a signed structure that binds the state to the nonce to protect against replay. To ensure that our sequences of state are well-founded we also include another TPM role that creates the initial state. It has a single state write event $\circ \rightsquigarrow \mathsf{s}_0$, that writes the well-known boot value into the freshly created state.

We similarly formalize Alice's actions. Her access to the TPM state is entirely mediated via the message-based interface to the TPM, so her role has no state events. It is displayed in Fig. 3

Alice begins by establishing an encrypted session with the TPM in order to extend a fresh value n into the PCR. She then has the TPM create a fresh key that can only be used when the PCR contains the value $\#(\mathsf{obt}, \#(n, s))$, where s is whatever value was in the PCR immediately before Alice performed her

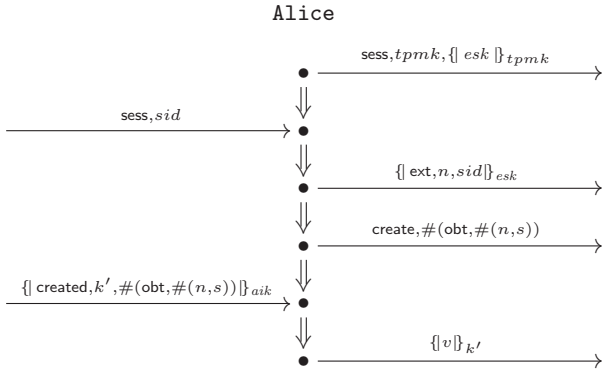

Fig. 3. Alice's role

extend command. Upon receiving the certificate for the freshly chosen key, she uses it to encrypt her secret v that gives her destination for the night.

The parents may then either choose to further extend the PCR with the value obt in order to enable the decryption of Alice's secret, or they can choose to extend the PCR with the value ref and get a quote of that new value to prove to Alice that they did not take the other option. The adversary roles displayed in Fig. 5 constrain what the parents can do. It is important to note that, like Alice's role, the adversary roles do not contain any state events. Thus the adversary can only interact with the state via the interface provided by the TPM commands.

We aim to validate a particular security goal of the Envelope Protocol using the enrich-by-need method. The parent should not be able to both learn the secret value v and generate a refusal token.

Security Goal 1 *Consider the following events:*

- *An instance of the Alice role runs to completion, with secret v and nonce n both freshly chosen;*
- *v is observed unencrypted;*
- *the refusal certificate* $\{|\mathsf{quote}, \#(\mathsf{ref}, \#(n,s)), \{|v|\}_{k'}|\}_{aik}$ *is observed unencrypted.*

These events, which we call jointly $\mathbb{A}_0$, are not all present in any execution.

3 An Execution Model for Protocols with State

The CPSA tool is based on strand space theory, in which an execution is described as a set of events partially ordered by two fundamental relations: strand succession ($\Rightarrow$) and transmission ($\rightarrow$). Strand succession represents the ordering, in sequence, of a viewpoint of a local participant, while transmission represents the causal ordering of a reception event following the transmission of the received

message. A *bundle* is a set of strands (sequences of events, each of which is a transmission or reception) in which for every reception event, there is a unique corresponding transmission that satisfies it.

Bundles are partially ordered (by the transitive closure of the two relations), rather than totally ordered. The non-comparison of certain events allows bundles to represent as large a class of real executions as possible. Two events not comparable in a bundle are events whose ordering is not observable or inferable by any protocol participant.

The diagrams in Section 2 imply a somewhat natural way to incorporate state into our execution model: we add a second type of event that deals with state rather than messages, along with an additional relation $\rightsquigarrow$. Our model contains three types of state events and two types of message events, each of which occur in our Envelope Protocol example.

- $(s \rightsquigarrow \circ)$ *Observation* records the current state without changing it. We use $\mathsf{obsv}\ s$ to indicate an observation of state s.
- $(s_0 \rightsquigarrow \circ \rightsquigarrow s_1)$ *Transition* represents the moment at which the state changes from a specific pre-state to a specific post-state. We use $\mathsf{tran}\ (s_0, s_1)$ to indicate a state transition with pre-state s_0 and post-state s_1.
- $(\circ \rightsquigarrow s)$ *Initiation* records the event of creating a new state. We use $\mathsf{init}\ s$ to indicate an initiation of state to s.
- $(\bullet \rightarrow m)$ *Transmission* is the sending of a message. We use $\mathsf{send}\ m$ to indicate the transmission of m.
- $(m \rightarrow \bullet)$ *Reception* is the receiving of a message. We use $\mathsf{recv}\ m$ to indicate the reception of a message m.

Definition 1 (Bundle). *Let tr be a set of disjoint sequences of events, where $\mathcal{N}$ is the set of all events occurring in some sequence. Let $\rightarrow, \rightsquigarrow \in \mathcal{N} \times \mathcal{N}$ where $\rightarrow$ is a relation from* send *events to* recv *events and $\rightsquigarrow$ is a relation from* init *or* tran *events to* tran *or* obsv *events. The triple $(tr, \rightarrow, \rightsquigarrow)$ is a* bundle *if:*

1. *For each message reception event $n_1 = \mathsf{recv}\ m$ there exists a unique $n_0 = \mathsf{send}\ m$ where $n_0 \rightarrow n_1$.*
2. *For each observation event $n_1 = \mathsf{obsv}\ s$ or transition event $n_1 = \mathsf{tran}\ (s, \cdot)$ there exists a unique n_0 such that either $n_0 = \mathit{init}\ s$ or $n_0 = \mathsf{tran}\ (\cdot, s)$.*
3. *$(\Rightarrow \cup \rightarrow \cup \rightsquigarrow)^+$–the transitive closure of the three arrow relations—is acyclic.*

A bundle is thus a set of strands in which for every event that expects an incoming arrow with a specific value, there is a unique corresponding event that satisfies it by producing an outgoing arrow of the same type with the same value.

Our axioms of state. Bundles are not a sufficient execution model, however, because they do not capture what is essentially different about state as compared to messages: its sequential nature.

The initiation and transition events are meant to describe the sequence of values a state goes through. The notion of bundle says nothing about the "out-degree" of an event. A message transmission event can satisfy more than one

message reception. However, a state event (initiation or transition) can satisfy *only* one state transition event.

Observation events are not strictly necessary; we could model the checking of a state value as a transition $s \rightsquigarrow \circ \rightsquigarrow s$. However, this would imply that several observation events be ordered in a sequence, even though they need not occur in any specific order, which violates the principled choice that our execution model not include unnecessary ordering. Thus, we include observation as a distinct type of state event.

Observations must occur in a well-defined place in the sequence of states. They require an incoming $\rightsquigarrow$ arrow that must come from either a transition or an initiation and represents the moment at which the state became the observed value. It can be inferred that such an observation occurs before any subsequent change in the state.

These two principles—that transitions occur in a non-forking sequence, and the observations occur before a subsequent change in the observed state—motivate our execution model.

Definition 2 (State-respecting bundle). *Let $\mathcal{B} = (tr, \rightarrow, \rightsquigarrow)$ be a stateful bundle. $\mathcal{B}$ is a* state-respecting bundle *if and only if:*

1. *if $n \rightsquigarrow n_0$ and the event at n_0 is a transition then if $n \rightsquigarrow n_1$ and the event at n_1 is also a transition, then $n_1 = n_0$;*
2. *The relation $\prec$ is acyclic, where $\prec$ is the smallest transitive relation such that (1) $(\Rightarrow \cup \rightarrow \cup \rightsquigarrow) \subset \prec$ and (2) whenever $n_0 \rightsquigarrow n_1$ and $n_0 \rightsquigarrow n_2$ where n_1 is a transition event and n_2 is an observation, then $n_2 \prec n_1$.*

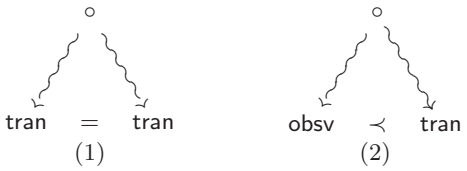

Fig. 4. State-respecting semantics. (1) State produced (either from a tran or init event) cannot be consumed by two distinct transitions. (2) Observation occurs after the state observed is produced but before that state is consumed by a subsequent transition.

3.1 Protocols and the adversary model

A protocol consists of a set of roles, each of which is a sequence of events. An *instance* of a role is an image of some prefix of the role's event sequence under an algebra map.

For most protocols, it is anticipated that there is a bundle in which *all* of the strands are instances of protocol roles. However, interesting attacks on protocols may require instances of another type of role, representing allowable attacker behavior. These *penetrator roles* are protocol-independent. For the basic crypto algebra, the five penetrator roles are pairing and decomposition (for assembling and dismantling pairs), encryption and decryption (for building and deconstructing encryptions), and creation (for production of *atomic* values not assumed to be unavailable to the attacker. See Fig. 5 for the adversary roles.

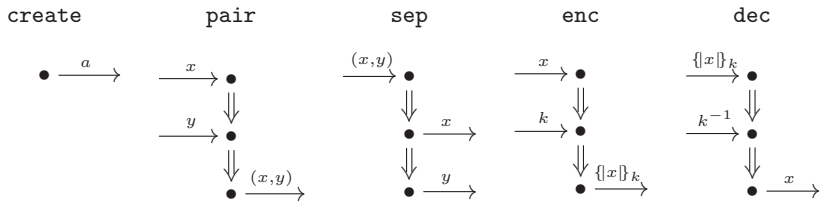

Fig. 5. Adversary roles

Analysis of a protocol is an exploration of the space of bundles consisting of a combination of protocol role instances and penetrator role instances.

Note that none of the penetrator roles involve any kind of state events. This is a deliberate choice to focus on an attack model in which the attacker can influence state only through the included protocol roles.

3.2 Enrich-by-need for stateful protocols

The CPSA tool is driven by the bundle model of executions, but its actual operations are on a different sort of object called *skeletons*, which are partial descriptions of an execution that abstract away adversarial strategies for delivering messages. CPSA enrich-by-need analysis is based on refining a skeleton $\mathbb{A}$ with at least one undeliverable message, into a set of skeletons $\{\mathbb{B}_i\}$ each of which refines $\mathbb{A}$, such that any execution that includes the structure in $\mathbb{A}$ includes the structure of at least one of the $\mathbb{B}_i$.

In order to handle state appropriately within CPSA analysis, we expanded the notion of a skeleton to include initiation, transition, and observation events explicitly and also to explicitly represent the $\rightsquigarrow$ arrows. The enrich-by-need analysis needs a second case, to handle skeletons with at least one unexplained state event, that is, either a transition or observation without an incoming $\rightsquigarrow$. We solve this case by considering all possible state events that could possibly provide the required state—either a fresh state event derived from a protocol role, or a pre-existing one. In the latter case, if the unexplained reception is a transition, the pre-existing state event must not have already had its state consumed by a different transition.

We also maintain a skeleton ordering so that the ordering reflects the ordering described in Condition 2 of Def. 2, and we discard a skeleton if it contains a cycle.

4 Analysis of the Envelope Protocol

The two conditions of Def. 2 identify the crucial aspects of state that distinguish state events from message events. They axiomatize necessary properties of state that are not otherwise captured by the properties of bundles. In order to give the reader some intuition for these properties, we present several analyses of the Envelope Protocol in this section. We begin by contrasting two analyses; one is based on plain bundles that only satisfy Definition 1, while the other is based on state-respecting bundles that also satisfy Definition 2

Plain vs. state-respecting bundles. Recall that the Envelope Protocol was designed to satisfy Security Goal 1. That is, there should be no executions in which (1) Alice completes a run with fresh, randomly chosen values for v and n, (2) v is available unencrypted on the network, and (3) the refusal certificate Q is also available on the network. Whether we use plain bundles or state-respecting bundles as our model of execution, the analysis begins the same way. The relevant fragment of the point at which the two analyses diverges is depicted in Fig. 6. The reader may wish to refer to the figure during the following description of the enrich-by-need process. The first three steps describe how we infer the existence of the top row of strands from right to left. The last two steps explain how we infer the strands in the bottom row from left to right.

1. The presence of v in unencrypted form implies the existence of a `decrypt` strand to reveal it.
2. The `decrypt` strand requires the current state to be $\#(\mathsf{obt}, \#(n, s))$, so our new principle of enrichment for state implies the existence of an extend strand with input value obt.
3. This newly inferred `extend` strand, in turn must have its current state $\#(n, s)$ explained which is done by another `extend` strand that receives the value n from Alice.
4. The presence of the quoted refusal token Q implies the existence of a `quote` strand to produce it.
5. The `quote` strand requires the state to be $\#(\mathsf{ref}, \#(n, s))$, which allows us to infer the third `extend` strand.

At this point in the analysis, the underlying semantics of bundles begins to matter. Our analysis still must explain how the state became $\#(n, s)$ for this last `extend` strand. If we use plain bundles that do not satisfy Definition 2, then we may re-use the `extend` strand inferred in Step 3 as an explanation. This would cause us to add a $\rightsquigarrow$ arrow between these two state events (along the dotted arrow $*$ of Fig. 6) forcing us to "split" the state coming out of the earlist extend strand. Further enrichments allow us to discover a bundle compatible with our

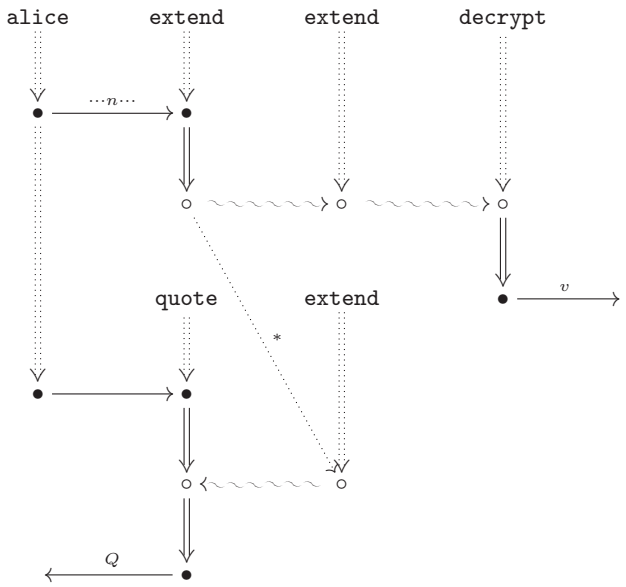

Fig. 6. A crucial moment in the CPSA analysis of the Envelope Protocol, demonstrating the importance of our first axiom of state.

starting point, contrary to Security Goal 1. It is important to note, however, that all bundles that enrich the fragment with the split state are not state-respecting.

If, on the other hand, we only allow state-respecting bundles, Condition 1 of Definition 2 does not allow us to re-use the `extend` strand inferred in Step 3 to explain the state found on the strand of Step 5. Instead, we are forced to infer yet another `extend` strand that receives Alice's nonce n. However, since Alice uses an encrypted session that provides replay protection, the adversary has no way to return the TPM state to $\#(n, s)$. Thus, although there are plain bundles that violate Security Goal 1, there are no state-respecting bundles that do so.

A flawed version. We also performed an analysis of the Envelope Protocol, removing the assumption that Alice's nonce n is fresh, to demonstrate our state-respecting variant's ability to automatically detect attacks. The analysis proceeds similarly; as in the previous analysis we decline to add a $\rightsquigarrow$ arrow along $*$ thanks to our stateful semantics. However, the alternative possibility that a fresh `extend` strand provides the necessary state proves to work out. Because n is not freshly chosen, the parent can engage in a distinct `extend` session with the same n.

Note that our analysis does not specify that the $s = \mathsf{s}_0$, where s is the state of the PCR when first extended. For the case where $s = \mathsf{s}_0$, the attack is to reboot the TPM after obtaining one value (either the refuse token or Alice's secret), re-extend the boot state with n, and then obtain the other. In fact, so long as

s represents a state of the PCR that the parent can induce, a similar attack is possible.

4.1 The Importance of Observer Ordering

The Envelope Protocol example demonstrates the crucial importance of capturing our first axiom of state correctly. The second axiom, involving the relative order of observations and state transition, is no less crucial to correct understanding of stateful protocols.

Another example protocol illustrates the principle more clearly. Suppose a hardware device is capable of producing keys that are meant to be managed by the device and not learnable externally. If the device has limited memory, it may be necessary to export such a key in an encrypted form so the device can utilize external storage.

Thus, device keys can be used for two distinct purposes: for encryption / decryption of values on request, or for encrypting internal keys for external storage. It is important that the purpose of a given key be carefully tracked, so that the device is not induced to decrypt one of its own encrypted keys.

Suppose that for each key, the device maintains a piece of state, namely, one of three settings:

- A wrap key is used only to encrypt internal keys
- A decrypt key may be used to encrypt or decrypt arbitrary values specified by a user
- An initial key has not yet been assigned to either the "wrap" or "decrypt" state.

If a key in the wrap state can later be put in the decrypt state, a relatively obvious attack becomes possible: while in the wrap state, the device encrypts some internal key, and later, when the key is in the decrypt state, the device decrypts the encrypted internal key.

However, if keys can never exit the wrap state once they enter it, this attack should not be possible. If we were to represent this protocol within CPSA, we would include the following roles:

- A `create key` role that generates a fresh key and initializes its state to initial
- A `set wrap` role that transitions a key from initial or decrypt to wrap.
- A `set decrypt` role that transitions a key from initial to decrypt.
- A `wrap` role in which a user specifies two keys (by reference), and the device checks (with an observer) that the first is in the wrap state and if so, then encrypts the second key with the first and transmits the result.
- A `decrypt` role in which a user specifies a key (by reference) and a ciphertext encrypted under that key, and the device checks (with an observer) that the key is in the decrypt state and if so, then decrypts the ciphertext and transmits the resulting plaintext.

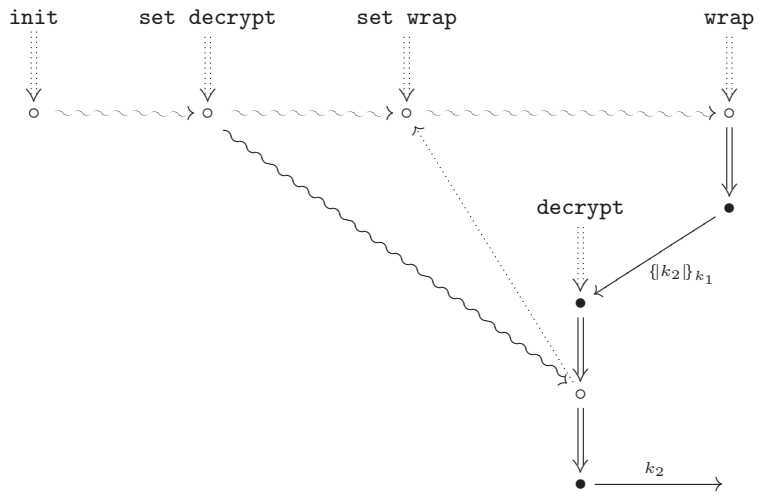

Fig. 7. Observer ordering example

Note that the attack should not be possible. However, the bundle described in Fig. 7 is a valid bundle, and fails to be state-respecting only because of our axiom about observers. Our second axiom induces an ordering so that the observer in the `decrypt` strand occurs before the following transition event in the `set wrap` strand. The induced ordering is shown in the figure with a single dotted arrow; note the cycle among state events present with that ordering that is not present without it.

5 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 [16, 26]. This work identified the effects of persistent storage as complicating the security analysis of the devices. There was also a strong focus on the case of PKCS #11 style devices for key management [5, 6, 11]. 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 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. [12]. 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" [8] presents a dynamic logic in the style

of PCL [7] 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). Thus, it seems not to be a strategy for reasoning about TPM usage, for instance in the Envelope Protocol.

Guttman [14] 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. This led to work by Ramsdell et al. [21] that used CPSA to draw conclusions in the states-as-messages model. Additional consequences could then be proved using the theorem prover PVS [20], working within a theory of both messages and state organized around compatibility.

A group of papers by Ryan with Delaune, Kremer, and Steel [9, 10], and with Arapinis and Ritter [2] 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. address the inherent non-monotonicity of adversary's capabilities by using a two-place predicate $\mathrm{att}(u, m)$ meaning that the adversary may possess m at some time when the long-term state is u. In [2], the authors provide a compiler from a process algebra with state-manipulating operators to sets of Horn clauses using this primitive. In [10], the authors analyze protocols with specific syntactic properties that help ensure termination of the analysis. 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.

Meier, Schmidt, Cremers, and Basin's tamarin prover [18] uses multiset rewriting (MSR) as a semantics in which to prove properties of protocols. Since MSR suffices to represent state, it provides a way to prove results about protocols with state. Künnemann studied state-based protocol analysis [17] in a process algebra akin to StatVerif, which he translated into the input language of tamarin to use it as a proof method. Curiously, the main constructs for mutable state and concurrency control (locking) are axiomatized as properties of traces rather than encoded within MSR (see [17, Fig. 10]).

Our work. One distinguishing feature of this work is our extremely simple modification to the plain message passing semantics to obtain a state-respecting model. These are the two Axioms 1–2 in Def. 2. We think it is an attractive characteristic of the strand space framework that state reflects such a clean foundational idea. Moreover, this foundational idea motivated a simple set of alterations to the enrich-by-need tool CPSA.

6 Protocol Security Goals

The enrich-by-need analysis performed in our enhanced version of CPSA is fully compatible with the language of goals found in previous work such as [24]. The goal language is based on two classes of predicates:

- **Role-related predicates** that relate an event or parameter value to its use within a specific protocol role.
- **Bundle-related predicates** that are protocol-independent and describe important properties of bundles. These include the ordering of events as well as assumptions about freshly chosen values and uncompromised keys.

Both classes of predicates apply within state-respecting bundles in a natural way. The role-related predicates are sensitive only to the position of an event in the sequence of events of a role, and to the choice of parameter values in that instance of the role. Indeed, nodes that represent state transitions or observations are handled in exactly the same way, since they have positions in the role and parameter values in just the same way as the message transmission and reception events.

Freshness and uncompromised keys are of course handled in the same way in the state-respecting semantics. For instance, a value first arising in the post-state of a transition event can originate freshly there, just as it could originate freshly when first transmitted.

Thus, the state-respecting version of CPSA can verify formulas expressing security goals in exactly the same way as the previous version, and with the same semantic definitions.

Conclusion. In this paper, we have argued that CPSA—and possibly other formalized protocol analysis methods—can provide reliable analysis when protocols are standardized, even when those protocols are manipulating devices with long-term state. A core idea of the formalization are the two axioms of Def. 2, which encapsulate the difference between a message-based semantics and the state-respecting semantics.

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. Myrto Arapinis, Eike Ritter, and Mark Dermot Ryan. Statverif: Verification of stateful processes. In *Computer Security Foundations Symposium (CSF)*, pages 33–47. IEEE, 2011.
3. Myrto Arapinis, Mark Ryan, and Eike Ritter. StatVerif: Verification of stateful processes. In *IEEE Symposium on Computer Security Foundations*. IEEE CS Press, June 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. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. Joshua D. Guttman. Shapes: Surveying crypto protocol runs. In Veronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, Cryptology and Information Security Series. IOS Press, 2011.
14. Joshua D. Guttman. State and progress in strand spaces: Proving fair exchange. *Journal of Automated Reasoning*, 48(2):159–195, 2012.
15. Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.
16. Jonathan Herzog. Applying protocol analysis to security device interfaces. *IEEE Security & Privacy*, 4(4):84–87, 2006.
17. Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *IEEE Symposium on Security and Privacy*, pages 163–178, 2014.
18. Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The tamarin prover for the symbolic analysis of security protocols. In *Computer Aided Verification (CAV)*, pages 696–701, 2013.
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, Daniel J. Dougherty, Joshua D. Guttman, and Paul D. Rowe. A hybrid analysis for security protocols with state. In *Integrated Formal Methods*, pages 272–287, 2014.
22. John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. `http://hackage.haskell.org/package/cpsa`.

23. John D. Ramsdell, Joshua D. Guttman, Jonathan K. Millen, and Brian O'Hanlon. An analysis of the CAVES attestation protocol using CPSA. MITRE Technical Report MTR090213, The MITRE Corporation, December 2009. `http://arxiv.org/abs/1207.0418`.
24. Paul D. Rowe, Joshua D. Guttman, and Moses D. Liskov. Measuring protocol strength with security goals. Submitted to *IJIS* in the SSR 2014 special issue. Available at `http://web.cs.wpi.edu/~guttman/pubs/ijis_measuring-security.pdf`, April 2015.
25. 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.
26. 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`.