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.

Keywords: protocol analysis tools, stateful protocols, TPM, PKCS #11.

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. Some protocols involve multiple state histories; for instance, an online bank transfer manipulates the state of the destination account as well as the state of the source account.

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.

Contributions of this paper. We make four main contributions:

- We identify two central axioms of state that formalize the semantics of state-respecting behaviors (Def. 6). Each time a state is produced,
  1. it can be consumed by at most one subsequent transition.
  2. it cannot be observed after a subsequent transition consumes it.

  The first axiom is the essence of how the state-respecting analysis differs from standard message-based analysis. By contrast, once a message has been transmitted, it can be delivered (or otherwise consumed) repeatedly in the future.

  The second axiom, like the reader/writer principle in concurrency, allows observations to occur without any intrinsic order among them, so long as they all occur while that state is still available. It preserves the advantages of a partial order model, as enriched with state.
- We prove that our model of state-respecting behaviors exactly matches an alternative model in which state is maintained by a family of traditional state machines, whose transitions are triggered by synchronization events in a state-respecting behavior.
- We incorporated these two axioms into the tool CPSA [24], obtaining a tool that can perform state-respecting enrich-by-need protocol analysis.
- 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 giving some background, we describe 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 addresses a logical interpretation of enrich-by-need analysis and observes that this framework may be used, unmodified, for stateful protocols as we model them. We end with a brief comment on conclusions and future work.

Background: Strand spaces. We work within the strand space framework. A *strand* is a (usually short) finite sequence of events, where the events are

message transmission nodes;
message reception nodes; and
state synchronization nodes.

Each message transmission and reception node is associated with a message that is sent or received. State synchronization nodes will be related to states via two different models in Section 3.

The behavior of a principal in a single, local run of one role of a protocol forms a strand. We call these *regular strands*. We also represent basic actions of an adversary as strands, which we call *adversary strands*. Adversary strands never need state synchronization nodes, since our model of the adversary allows it to use the network as a form of storage that never forgets old messages.

A protocol Π is represented by a finite set of strands, called the *roles* of the protocol, together with some auxiliary information about freshness and non-compromise assumptions about the roles. We write $\rho \in \Pi$ to mean that ρ is one of the roles of the protocol Π. The *regular strands of* Π are then all strands that result from any roles $\rho \in \Pi$ by applying a substitution that plugs in values in place of the parameters occurring in ρ.

For more information on strand spaces, see e.g. [14, 27]. For the version containing state synchronization events as well as transmissions and receptions, see [15, 23].

Background: Enrich-by-need analysis. In our form of protocol 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 our [16]), 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 (a strand); 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 the one provided by strand spaces. In partially ordered models, 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.

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, 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. Initially, 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):

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

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 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 state in this protocol is the collision-free nature of hashing, ensuring that for every x

$$\#(\mathsf{obt}, \#(n, s)) \quad \neq \quad \#(\mathsf{ref}, x) \tag{1}$$

Formal protocol model. We formalize the TPM-based version of the Envelope Protocol using strand spaces [14]. Messages and states are represented as elements of a crypto term algebra, which is an order-sorted quotient term algebra. Sort $\top$ is the top sort of messages. Messages of sort A (asymmetric keys), sort S (symmetric keys), and sort D (data) 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.

We represent each TPM command with a separate role that receives a request, consults and/or changes the state and optionally provides a response. As shown in Fig. 1, we use $m{\rightarrow}\bullet$ and $\bullet{\rightarrow}m$ 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. We write $m \Rightarrow n$ to indicate that m precedes n immediately on the same strand.

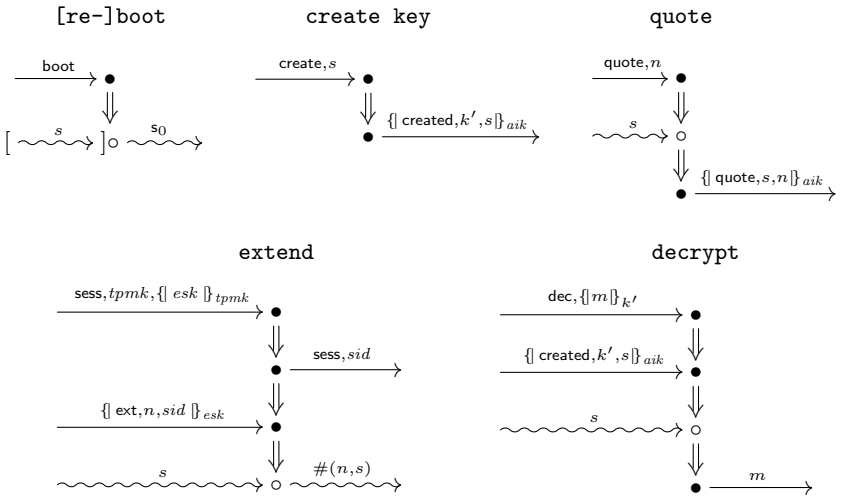

Fig. 1. TPM roles

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 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.

Since the `quote` role puts the state s into a message, and the `extend` role puts a message into the state, in our formalization states are the same kind of entity as messages.

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. 2

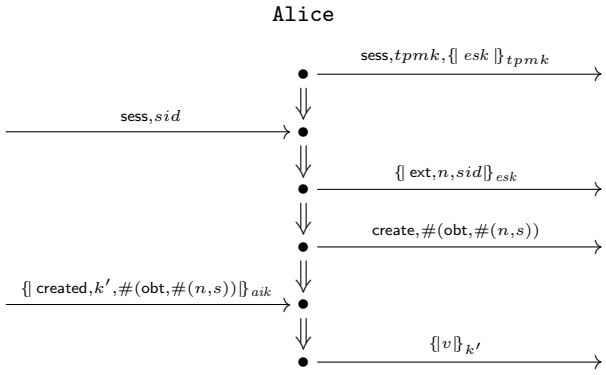

Fig. 2. Alice's role

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

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. 3 constrain what the parents can do.

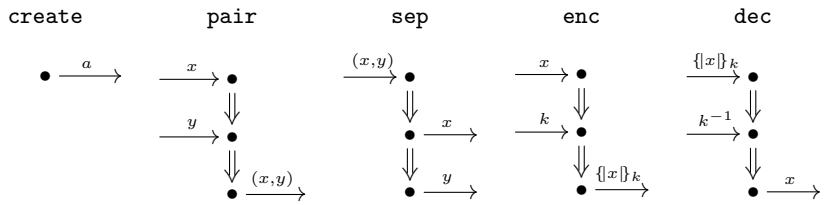

Fig. 3. Adversary roles, where a in the `create` role must be an atomic message.

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 State-respecting bundles

In this section, we introduce a model of protocol behavior in the presence of global state; it is new in this paper. It enriches the notion of a bundle, which is the longstanding strand space formalization of global behaviors [27, 14].

Definition 1 (Bundle). *Suppose that Σ is a finite set of strands. Let $\Rightarrow$ be the strand succession relation on $\mathsf{nodes}(\Sigma)$. Let $\rightarrow \subseteq \mathsf{nodes}(\Sigma) \times \mathsf{nodes}(\Sigma)$ be any relation on nodes of Σ such that $n_1 \rightarrow n_2$ implies that n_1 is a transmission event, n_2 is a reception event, and $\mathsf{msg}(n_1) = \mathsf{msg}(n_2)$.*

$\mathcal{B} = (\mathcal{N}, \rightarrow)$ is a bundle *over Σ iff $\mathcal{N} \subseteq \mathsf{nodes}(\Sigma)$, and*

1. *If $n_2 \in \mathcal{N}$ and n_1 precedes it on the same strand in Σ, then $n_1 \in \mathcal{N}$;*
2. *If n_2 is a reception node, there is exactly one $n_1 \in \mathcal{N}$ such that $n_1 \rightarrow n_2$; and*
3. *The transitive closure $(\Rightarrow \cup \rightarrow)^+$ of the two arrow relations is acyclic.*

$\mathcal{B}$ is a bundle of protocol *Π iff every strand with nodes in $\mathcal{B}$ is either an instance of a role of Π, or else an instance of one of the adversary roles in Fig. 3.*

Any finite behavior should have these properties, since otherwise some participant starts a role of the protocol in the middle, or receives a message no one sent, or else the (looping) pattern of events is causally impossible. By acyclicity, every bundle determines a partial ordering $\preceq_{\mathcal{B}}$ on its nodes, where $n_1 \preceq_{\mathcal{B}} n_2$ means that some path of one or more arrows $\rightarrow, \Rightarrow$ leads from n_1 to n_2 in $\mathcal{B}$.

We incorporate state transition histories directly into the bundles. To do this, we enrich the bundles with a new relation $\rightsquigarrow$ that propagates the current state from one event to another. We do this so that our analysis method can work with a single object that has both message dependencies and state dependencies within it. We also distinguish between *state transitions* and *state observations*. Transitions need to be linearly ordered if they pertain to a single device, but many state observations may occur between a single pair of state transitions. They are like *read* events in parallel computation: There is no need for concurrency control to sequentialize their access to the state, as long as they are properly nested between the right transition events.

This is an advantage of the strand space approach, which focuses on partially ordered execution models. It is important for enrich-by-need analysis, where the exponential number of interleavings must be avoided.

Later in this section, we will introduce a model containing a number of traditional state machines, where we correlate the synchronization nodes with transitions in their state histories. We make this model more rigorous in Section 3.2, where we prove an exact match between the state respecting behaviors we use here and the more traditional model of state machine histories.

3.1 Enriching Bundles with State

We now enrich the bundles to incorporate states, and to propagate them from node to node, just as transmissions and receptions propagate messages.

The diagrams in Section 2 suggest a way to incorporate state into bundles: We enrich them so that each state synchronization event is associated with messages representing states. A transition event is associated with a pair, representing the pre-state before the transition together with the post-state after it. The pre-state must be obtained from an earlier synchronization event. The post-state is produced by the transition, and may thus be passed to later events. We also now distinguish state observation events; these are associated with a single state, which is like a pre-state since it is received from an earlier event that produced it. We also identify initiation events, which initialize a devices state and serve as the beginning of a state computation history.

Initiation nodes $\circ \rightsquigarrow s$ record the event of creating a new state. We use $\mathsf{init}\, s$ to indicate an initiation of state to s.

Observation nodes $s \rightsquigarrow \circ$ record the current state without changing it. We use $\mathsf{obsv}\, s$ to indicate an observation of state s.

Transition nodes $s_0 \rightsquigarrow \circ \rightsquigarrow s_1$ represent 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.

In specifying protocols and their state manipulations, we can use the style illustrated in Fig. 1. There, an observation such as the synchronization node in the `quote` role, acquires a message on the incoming $\rightsquigarrow$ arrow. In this case, it is a variable s, which is itself a parameter to the role which contributes to the subsequent transmitted message. The `decrypt` role also has an incoming $\rightsquigarrow$ arrow labeled with s; in this case, the role can proceed to engage in this event only if the value s equals a previously available parameter acquired in the previous reception node. The `extend` role has a transition node, in which any pre-state s will be updated to a new post-state by hashing in the parameter n.

These pre- and post-state annotations, using parameters that appear elsewhere in the roles, determine subrelations of the transition relation associated with each instance of a role. An instance of the `extend` role with a particular value n_0 for the parameter n will engage only in state transformations that hash in that value n_0.

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 require observation events be ordered in a specific sequence. This violates the principled choice that our execution model not include unnecessary ordering.

In the Introduction, we defined a protocol to be a finite set of strands called the *roles* of the protocol. An *enriched protocol* Π^+ will be a protocol Π enriched with a classification of its state synchronization events into $\mathsf{init}, \mathsf{tran}$, and obsv nodes, with each of those annotated with messages defining their pre- and post-states. The *regular strands* of Π^+ are all of the substitution instances of the roles of Π^+, including the instances of the pre- and post-states on the synchronization nodes.

An enriched bundle uses $\rightsquigarrow$ arrows to track the propagation of the state of each device involved in the behavior. This is not a sufficient model for reasoning about state, which requires also the two axioms of Defn. 6, but it provides the objects from which we will winnow the state-respecting bundles.

Definition 2 (Enriched bundles). $\mathcal{B}^+ = (\mathcal{N}, \rightarrow, \rightsquigarrow)$ *is an* enriched bundle *iff* $(\mathcal{N}, \rightarrow)$ *is a bundle, and moreover:*

1. $n_1 \rightsquigarrow n_2$ *implies that* n_1 *is an* init *or* tran *event and* n_2 *is an* obsv *or* tran *event, and the post-state of* n_1 *equals the pre-state of* n_2*;*
2. *For each* obsv *or* tran *event* n_2*, there exists a unique* n_1 *such that* $n_1 \rightsquigarrow n_2$*;*
3. *The transitive closure* $(\Rightarrow \cup \rightarrow \cup \rightsquigarrow)^+$ *of the three arrow relations is acyclic. We refer to the partial order it determines as* $\prec_{\mathcal{B}^+}$ *or* $\prec$ *when* $\mathcal{B}^+$ *is clear.*

Enriched bundles are not a sufficient execution model, however, because they do not capture what is essentially different about state as compared to messages: the way that the next transition event consumes a state value, such that it cannot be available again unless a new transition creates it again. We can see this by connecting our current set-up to a state-machine model.

3.2 Bundles with Explicit Computations

we introduce a formal model of executions, where protocol behavior drives state machine executions, as briefly introduced in Section 3. Message transmissions and receptions occur alongside the state transition histories of zero or more stateful devices. The message behavior here satisfies the usual bundle properties for protocol behavior (see Def. 1). Some events do not send or receive messages, but synchronize with the state of one or more devices. Thus, a bundle together with a family of state transition histories counts as a possible execution if the steps of the state transition histories match with the state synchronization events in the bundle. This model is adapted from our earlier work [15].

When a protocol executes in coordination with devices that maintain state, the execution must have the structure of a bundle, as far as the message-passing behavior is concerned, and must *also* meet the constraints that the devices impose. Each device must undergo a possible state transition history, and each transition should be caused by something, namely by some state synchronization event in the bundle.

Definition 3 (Computations). *Let $\{D_i\}_{i\in I}$ be a family of devices, indexed by some set I. Assume that each device D_i has a set of states St_i, with initial state $s_0^i \in \mathsf{St}_i$ and transition relation $\rhd_i \subseteq \mathsf{St}_i \times \mathsf{St}_i$.*

1. *A* state transition history *or* computation *for D_i is a finite sequence of states $\mathcal{C} = \langle s_0, s_1, \ldots, s_\ell \rangle$ starting with the initial state $s_0 = s_0^i$ and, for every j, if $0 \leq j < \ell$, then $s_j \rhd_i s_{j+1}$.*
2. *A $\{D_i\}_{i\in I}$-*family *of computations is a family $\{\mathcal{C}_i\}_{i\in I}$ indexed by the same set I such that each $\mathcal{C}_i$ is a computation for D_i.*

A *correlation* is a function that offers a synchronization node in a bundle to match each step in a computation in some family.

Definition 4 (Correlations). *Let $\mathcal{B}$ be a bundle, with synchronization nodes $\mathsf{sync}(\mathcal{B})$, and let $\{\mathcal{C}_i\}_{i\in I}$ be a $\{D_i\}_{i\in I}$-family of computations.*

A position *$p = i, j$ for $\{\mathcal{C}_i\}_{i\in I}$ is a pair such that $i \in I$ and $0 < j < \mathsf{length}(\mathcal{C}_i)$. Let Pos be the set of all positions for $\{\mathcal{C}_i\}_{i\in I}$.*

A correlation *$\phi\colon \mathsf{Pos} \to \mathsf{sync}(\mathcal{B})$ is a function from positions in the computation family to synchronization nodes of the bundle and such that:*

1. *$\mathsf{ran}(\phi) = \mathsf{sync}(\mathcal{B})$, i.e. ϕ is surjective onto the synchronization nodes; and*
2. *ϕ is consistent with the bundle ordering $\prec_{\mathcal{B}}$: i.e. let $R(n, n')$ mean that there exist i, j, k with $j < k$, $n = \phi(i, j)$, and $n' = \phi(i, k)$, and require:*

$$(\prec_{\mathcal{B}} \cup R)^+ \text{ is acyclic.}$$

A correlation ϕ is injective *iff $\phi(i,j) = \phi(i',j')$ implies $i = i'$ and $j = j'$.*

In general, the same node n may synchronize with positions in several different computations $\mathcal{C}_i$; an *injective* correlation does not exercise this possibility.

Typically, one would like to correlate nodes and state transitions more tightly, so that each synchronization node in a role causes a specific type of transition. In this context, a "type" of transition simply means a subset of the transition relation. The subset can also depend on the parameter values for the node in question. The set T of pairs

$$\{n, \sigma : n \in \mathsf{sync}(\Pi), \text{ substitution } \sigma \text{ assigns values to the parameters of } \rho\}$$

indexes a family of subrelations $R_{n,\sigma}$ of a transition relation $\rhd$:

$$R_{n,\sigma} \subseteq \rhd.$$

We also write the subrelations in the form $\rhd^{n,\sigma}$ in our model of execution:

Definition 5 (Execution). *Let $\mathcal{B}$ be a Π-bundle; let $\{\mathcal{C}_i\}_{i\in I}$ be a $\{D_i\}_{i\in I}$-family of computations; and let ϕ be a correlation between them. For each $i \in I$, let $\rhd_i^{n,\sigma}$ be a family of subrelations of $\rhd_i$.*

$(\mathcal{B}, \{\mathcal{C}_i\}_{i\in I}, \phi)$ *is a* $\Pi, \rhd_i$-execution *acting on the devices $\{D_i\}_{i\in I}$, subject to the subrelations $\rhd_i^{n,\sigma}$, iff for every $n' \in \mathsf{sync}(\mathcal{B})$, if:*

1. *$n' = \sigma(n)$ is an instance of role node n under substitution σ;*
2. *$n' = \phi(i,j)$;*
3. *$\mathcal{C}_i = \langle s_0, s_1, \ldots, s_\ell \rangle$;*

then $s_{j-1} \rhd_i^{n,\sigma} s_j$.

This gives a fairly general model of how the events in a protocol execution can drive the transitions of a family of devices. As discussed in our previous [15], it accounts both for events in which the protocol execution receives information out of the state and also for events in which the protocol execution deposits information into the state. There are two main changes here vis-a-vis [15]. First, we allow many devices to have separate state histories. Second, we omit the "labels" that were attached to synchronization nodes there, instead using the subrelations $\rhd_i^{n,\sigma}$ to correlate specific protocol events with types of state transition.

Each enriched protocol Π^+ determines a type of state machine. Its states (included in the set of messages) are all pre-states and post-states of the synchronization nodes of all instances of the roles of Π^+. A state machine has a set of initial states. In the state machine determined by Π^+, the initial states are the states $\sigma(s)$ such that some role $\rho \in \Pi^+$ has an initiation event $\mathsf{init}\, s$, and σ is a substitution determining an instance of ρ.

The state machine determined by Π^+ has the state transition relation $\rhd$ consisting of all pairs of states (s_1, s_2) where

$s_1 \rhd s_2$ iff there exists a state transition node of Π^+ with pre-state t_1 and post-state t_2 and a substitution σ, such that $s_1 = \sigma(t_1)$ and $s_2 = \sigma(t_2)$.

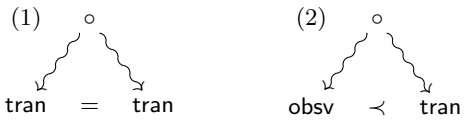

Fig. 4. State-respecting semantics. (1) State produced (either from a tran or init event) cannot be consumed by two distinct transitions. (2) Observation occurs after the state observed is produced but before that state is consumed by a subsequent transition.

A *state history* or *computation* is a finite or infinite sequence of states $s_0, s_1, \ldots$ that starts with an initial state s_0, and, for every i, if s_{i+1} is defined then $s_i \rhd s_{i+1}$.

The enriched bundles are not a sufficient model for reasoning about state, because there are enriched bundles that do not correspond to any execution in this sense. We will illustrate this in Section 4.

3.3 Our Axioms of State

The initiation and transition events are meant to describe the sequence of states that a device passes 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 *at most one* state transition event.

Observations must occur in a constrained place in the sequence of states. They acquire an incoming $\rightsquigarrow$ arrow from a transition or an initiation. Any such observation occurs before a subsequent change in the state.

These two principles—that transitions do not fork, and observations must precede a transition that consumes their state—motivate our execution model. They are illustrated in Fig. 4.

Definition 6 (State-respecting bundle). *Let $\mathcal{B}^+ = (\mathcal{N}, \rightarrow, \rightsquigarrow)$ be an enriched bundle with precedence order $\prec$. $\mathcal{B}^+$ is* state-respecting *if and only if:*

1. *if $n \rightsquigarrow n_0$ and $n \rightsquigarrow n_1$, where n_0 and n_1 are* tran *events, then $n_1 = n_0$;*
2. *Let the relation $\prec^+$ be the smallest transitive relation including $\prec$ such that whenever n_0 is an* obsv *and n_1 is a* tran*, then*

$$n \rightsquigarrow n_0 \text{ and } n \rightsquigarrow n_1 \text{ implies } n_0 \prec^+ n_1. \tag{2}$$

Then $\prec^+$ is acyclic.

We call Clause 1 the *No State Split Principle*. Clause 2 is the *Observation Ordering Principle*.

These two axioms are adequate to provide a model of state. In particular, we now prove that the executions in the sense we formalize there correspond exactly to the state-respecting bundles of Def. 6.

3.4 Relating State-Enriched Bundles to Executions

The extended protocols Π^+ and state-respecting bundles relate easily to the $\mathcal{B}, \mathcal{C}, \phi$ model.

We now define a single transition system in terms of the state synchronization nodes of Π^+, i.e. a transition system that controls all of the devices D_i. If a node n on a role of Π^+ is an

Initiation node $n = \circ \rightsquigarrow t$, then $\rhd^{n,\sigma}$ is the singleton relation $\{\langle s_0, \sigma(t)\rangle\}$.
Observation node $n = t \rightsquigarrow \circ$, then $\rhd^{n,\sigma}$ is the singleton $\{\langle \sigma(t), \sigma(t)\rangle\}$, in which the post-state is unchanged.
Transition node $n = t_0 \rightsquigarrow \circ \rightsquigarrow t_1$, then $\rhd^{n,\sigma}$ is the singleton $\{\langle \sigma(t_0), \sigma(t_1)\rangle\}$.

Then the transition relation $\rhd(\Pi^+)$ for all devices D_i is defined to be the union

$$\bigcup_{n,\sigma} \rhd^{n,\sigma},$$

taking the union over all synchronization nodes $n \in \mathsf{sync}(\Pi^+)$, and all substitutions σ. We shall consider executions relative to the family of subrelations $\rhd^{n,\sigma}$.

If Π^+ is an extended protocol, let $\mathsf{fgt}(\Pi^+)$ result from it by forgetting the pre- and post-state annotations on the synchronization nodes. We will refer to a node on some role $\rho \in \mathsf{fgt}(\Pi^+)$ as an $\mathsf{init}, \mathsf{obsv}$, or tran node of $\mathsf{fgt}(\Pi^+)$ if it results from a node of the same kind in Π^+ by forgetting.

If $\mathcal{B}^+ = (\mathcal{N}, \rightarrow, \rightsquigarrow)$ is an enriched bundle for Π^+, then $\mathsf{fgt}(\mathcal{B}^+)$ is the Π-bundle $(\mathcal{N}', \rightarrow')$ where $\mathcal{N}'$ results from $\mathcal{N}$ by forgetting the pre- and post-state annotations on the synchronizations, and $\rightarrow'$ relates two nodes in $\mathcal{N}'$ iff $\rightarrow$ related their preimages in $\mathcal{N}$.

Lemma 1. *Given an extended protocol Π^+, let $\Pi = \mathsf{fgt}(\Pi^+)$ and $\rhd = \rhd(\Pi^+)$. Let $\mathcal{B}, \{\mathcal{C}_i\}_{i\in I}, \phi$ be a $\Pi, \rhd$-execution, with injective ϕ.*

There exists a state-respecting bundle $\mathcal{B}^+$ of Π^+ such that $\mathsf{fgt}(\mathcal{B}^+) = \mathcal{B}$.

Proof. Suppose that $\mathcal{B}, \{\mathcal{C}_i\}_{i\in I}, \phi$ is an execution. For each synchronization node in $\mathcal{B}$, we must decorate it with pre- and post-states (depending on its kind) from $\{\mathcal{C}_i\}_{i\in I}$ and $\rightsquigarrow$ arrows, obtaining a state-respecting bundle $\mathcal{B}^+$. It will then be immediate that $\mathsf{fgt}(\mathcal{B}^+) = \mathcal{B}$, since we will only add arrows and annotations that fgt discards.

The correlation ϕ tells us how to decorate the nodes. Since ϕ is surjective onto $\mathsf{sync}(\mathcal{B})$, every $n \in \mathsf{sync}(\mathcal{B})$ will be annotated. Since ϕ is injective, there is no risk of conflict between two different computation steps.

Construction: We consider each computation $\mathcal{C}_i$ and work recursively on steps j, where $0 < j < \mathsf{length}(\mathcal{C}_i)$, within $\mathcal{C}_i$.

In the base case, when $j = 1$, we know that the preceding value was the initial state s_0; by the decomposition of $\rhd$ into subrelations, we know that $\phi(i, 1)$ is an instance of an initiation node of Π. Thus, we decorate $\phi(i, 1)$ with $\mathsf{init}\,(\mathcal{C}_i(1))$.

Suppose, for the step case, that $j > 1$. We need now to decorate $\phi(i,j)$ with a pre-state and possibly post-state, and we need to provide it with an incoming $\rightsquigarrow$ arrow.

For the $\rightsquigarrow$ arrow, if $\phi(i,j-1)$ is an initiation or transition node, we add an arrow $\phi(i,j-1) \rightsquigarrow \phi(i,j)$. If $\phi(i,j-1)$ is an observation node, then it has an incoming arrow from some $n_1 \rightsquigarrow \phi(i,j-1)$, and we add an arrow from the same $n_1 \rightsquigarrow \phi(i,j)$.

If $\phi(i,j)$ is an instance of an observation node of Π, by the decomposition of $\rhd$ into subrelations, we know that $\mathcal{C}_i(j-1) = \mathcal{C}_i(j)$. We decorate $\phi(i,j)$ with $\mathsf{obsv}\,(\mathcal{C}_i(j))$. If instead $\phi(i,j)$ is not an instance of any observation nodes of Π, we decorate $\phi(i,j)$ with $\mathsf{tran}\,(\mathcal{C}_i(j-1), \mathcal{C}_i(j))$.

Invariants: Our construction maintains the following invariants:

1. Whenever $n_1 \rightsquigarrow n_2$, the post-state of n_1 is well-defined, the pre-state of n_2 is well defined, and the two states are equal.
2. Every $\rightsquigarrow$ arrow points from $\phi(i,j)$ to $\phi(i,k)$ where $j < k$ and the first arguments are equal.
3. If $\phi(i,j) \rightsquigarrow \phi(i,k)$, then j is the largest $j' < k$ such that $\phi(i,j')$ is an init or tran node and the post-state of $\phi(i,j')$ equals the pre-state of $\phi(i,k)$.
4. The set of indices $\{k\colon \phi(i,j) \rightsquigarrow \phi(i,k)\}$ of $\rightsquigarrow$ successors of $\phi(i,j)$ forms an interval $[j+1, \ell]$; if $j < k < \ell$, then $\phi(i,k)$ is an obsv node.

Invariant 3 helps us to infer that invariant 4 holds.

$\mathcal{B}^+$ is state-respecting: By invariant 1, Clause 1 of Def. 2 is satisfied. In the construction, if $n = \phi(i,j)$ and n is not an init node, then $j \neq 1$, so n obtains a single incoming $\rightsquigarrow$ arrow. So Clause 2 is satisfied. Moreover, by Clause 2 of Def. 4, $(\rightarrow \cup \Rightarrow \cup \rightsquigarrow)^+$ is acyclic.

Thus, the resulting $\mathcal{B}^+$ is an enriched bundle. We must now show that it satisfies the two axioms of state in Def. 6.

Suppose then that $n \rightsquigarrow n_1$ and $n \rightsquigarrow n_2$ forms a state-split, where n_1, n_2 are distinct tran-nodes. By surjectiveness and invariant 2, $n = \phi(i,j)$, $n_1 = \phi(i,k)$, and $n_2 = \phi(i,k')$, where $j < k, k'$. By symmetry, we may assume $j < k < k'$. But then by invariant 4, $\phi(i,k)$ is an obsv node contrary to assumption. Thus the No State Split Principle (Def. 6, Clause 1) is satisfied.

Turning to Clause 2, the Observation Ordering Principle, consider the set S of pairs n_0, n_1 such that n_0 is an obsv node, n_1 is a tran node, and for some n, $n \rightsquigarrow n_0$ and $n \rightsquigarrow n_1$. Then for each such pair, by invariant 4, we have $n = \phi(i,j)$, $n_0 = \phi(i,k)$, $n_1 = \phi(i,k')$ where $j < k < k'$. Therefore, we have $S \subseteq R$ for the R in Clause 2 for the correlation ϕ (Def. 4). Thus, acyclicity follows. □

Lemma 2. *Given an extended protocol Π^+, let $\Pi = \mathsf{fgt}(\Pi^+)$ and $\rhd = \rhd(\Pi^+)$. Let $\mathcal{B}^+$ be a state-respecting bundle of Π^+.*

There exists a $\Pi, \rhd$-execution $\mathsf{fgt}(\mathcal{B}^+), \{\mathcal{C}_i\}_{i\in I}, \phi$ with ϕ injective.

Proof. **Determining I and partitioning the nodes.** By well-founded induction, for every synchronization node n_1, there exists an initiation node n_0

such that $n_0 \rightsquigarrow^* n_1$. By Def. 2, Clause 1, if n_0 is an initiation node, then for all n, $n \not\rightsquigarrow n_0$. By induction and the uniqueness in Def. 2, Clause 2, there is exactly one initiation node n_0 such that $n_0 \rightsquigarrow^* n_1$. Define the index set $I = \{n_0 \in \mathsf{sync}(\mathcal{B}^+) \colon n_0 \text{ is an } \mathsf{init}\}$. Now, the I-indexed family of sets of nodes:

$$\mathcal{P} = \{\, \{n_1 \colon n_0 \rightsquigarrow^* n_1\} \,\}_{n_0 \in I}$$

is a partition of the synchronization nodes indexed by init nodes.

Consider now an init node n_0 and the partition element P_{n_0} of $\mathcal{P}$ where

$$P_{n_0} = \{n_1 \colon n_0 \rightsquigarrow^* n_1\}.$$

We will show how to construct a computation $\mathcal{C}_{n_0}$ and a piece of the correlation $\lambda j \,.\, \phi(n_0, j)$ that will cover P_{n_0}.

Ordering the nodes. First, consider the tran, init nodes in P_{n_0}. We claim that they are *linearly ordered* by $\rightsquigarrow^+$. For otherwise, let n_1, n_2 be distinct incomparable nodes under $\rightsquigarrow^+$. By the definition of P_{n_0}, $n_1 \neq n_0 \neq n_2$, since n_0 is related to every node in P_{n_0}. Thus, n_1, n_2 are tran nodes. By well-foundedness, we may assume that n_1, n_2 are each chosen to be a $\rightsquigarrow^+$-minimal pair of incomparable tran nodes in P_{n_0}. Since n_1 is an tran node, it has a predecessor n_1'. By minimality, either $n_1' \rightsquigarrow^+ n_2$ or else $n_2 \rightsquigarrow^+ n_1'$. But the latter implies $n_2 \rightsquigarrow^+ n_1$, so in fact $n_1' \rightsquigarrow^+ n_2$. By minimality of n_2, $n_1' \rightsquigarrow n_2$. So $n_1' \rightsquigarrow n_1$ and $n_1' \rightsquigarrow n_2$, contradicting the No State Split Principle.

So tran, init nodes in P_{n_0} are linearly ordered by $\rightsquigarrow^+$.

Consider any pair of adjacent tran, init nodes $n_1 \rightsquigarrow n_2$. Let $\mathcal{O}$ be the set of obsv nodes n_o such that $n_1 \rightsquigarrow n_o$. $\mathcal{O} \cup \{n_1, n_2\}$ are partially ordered by $\prec^+$ since $\mathcal{B}^+$ satisfies the Observation Ordering Principle. Let $n_1, o_1, \ldots, o_k, n_2$ be any linearization of this set compatible with $\prec^+$.

Applying this throughout P_{n_0}, we obtain a sequence $\langle n_0, \ldots, n_\ell \rangle$ containing all the nodes of P_{n_0}, where the sequence ordering extends the $\prec^+$ ordering.

Defining $\mathcal{C}_{n_0}$ and $\lambda j \,.\, \phi(n_0, j)$. We now define $\mathcal{C}_{n_0}$ and the n_0 slice of ϕ by stipulating:

1. $\mathcal{C}_{n_0}(0) = s_0$, the initial state;
2. $\mathcal{C}_{n_0}(j+1)$ is the post-state of n_j if it has a post-state, and the pre-state of n_j if it is an obsv node;
3. $\phi(n_0, j+1) = n_j$ for all j where $0 \leq j < \mathsf{length}(\mathcal{C}_{n_0})$.

Now by the definition of $\langle n_0, \ldots, n_\ell \rangle$, ϕ satisfies the order constraint (Clause 2), and the other clauses for correlations are immediate; moreover, ϕ is injective because the $\lambda j \,.\, \phi(n_0, j)$ are disjoint for different partition classes P_{n_0}. The triple $\mathsf{fgt}(\mathcal{B}^+), \{\mathcal{C}_i\}_{i \in I}, \phi$ is an execution of $\Pi, \triangleright(\Pi^+)$, by the definition of $\triangleright(\Pi^+)$. □

3.5 Enrich-by-need for stateful protocols

In order to analyze stateful protocols with respect to state-respecting bundles (Def. 6), we adapted the Cryptographic Protocol Shapes Analyzer (CPSA)

which performs automated protocol analysis with respect to (traditional) bundles (Def. 1). CPSA uses the enrich-by-need method as described in the Introduction. That is, it progressively extends an execution fragment $\mathbb{A}$ into a set of execution fragments $\{\mathbb{B}_i\}$. The extending occurs only as needed, namely, when the execution fragment does not contain enough information to fully describe a bundle. For message-only protocols, extending is necessary exactly when a message received at node n cannot be derived by the adversary using previously sent messages as inputs to a web of adversary strands.

We adapted CPSA in several ways to account for the properties of state synchronization nodes in state-respecting bundles. First, we added state synchronization nodes to the internal data structures of the tool. We then augmented the tool to recognize that extending is necessary when a state synchronization node n has pre-state s, but there is no node n_0 with post-state s such that $n_0 \rightsquigarrow n$. Finally, we implemented the corresponding rules for extending execution fragments by adding state synchronization nodes that supply the necessary state. In doing so, we experimented with two versions, one works for enriched bundles that need not satisfy the two axioms from Def. 6, and one which enforces these axioms. This former version allow us to perform analyses that lead to bundles satisfying Def. 2 which do not correspond to any executions of the state-machine model. The latter eliminates these ersatz results.

One advantage to the use of state-respecting bundles is that it allowed us to integrate an analysis of the stateful part of the protocol in a modular fashion. Our implementation did not require us to alter any of the code that extends the message passing portion of protocols. We thus provide a clean separation of the two distinct aspects of stateful protocols in an integrated whole.

The next section explores several examples that demonstrate the results of these two versions and hopefully provide some intuition about why the two axioms of state are necessary.

4 Analysis of the Envelope Protocol

The two conditions of Def. 6 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 enriched 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 enriched bundles that only satisfy Definition 2, while the other is based on state-respecting bundles that also satisfy Definition 6

Enriched 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 enriched 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. 5. 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.

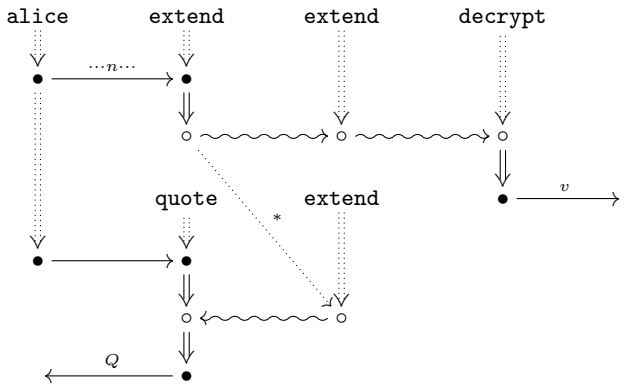

Fig. 5. A crucial moment in the CPSA analysis of the Envelope Protocol, demonstrating the importance of our first axiom of state.

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 state explanation 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 enriched bundles that do not satisfy Definition 6, 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. 5) forcing us to "split" the state coming out of the earlist extend strand. Further steps allow us to discover an enriched bundle compatible with our starting point, contrary to Security Goal 1. Importantly, however, all

enriched bundles that extend the fragment with the split state are non-state-respecting.

If, on the other hand, we only allow state-respecting bundles, Condition 1 of Definition 6 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 enriched 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 $s = \mathsf{s}_0$, 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. More generally, as long as s is a state 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, motivated by a well-known issue with PKCS #11 (see, e.g. [9]), 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.
- An initial key has not yet been assigned to either use.

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.

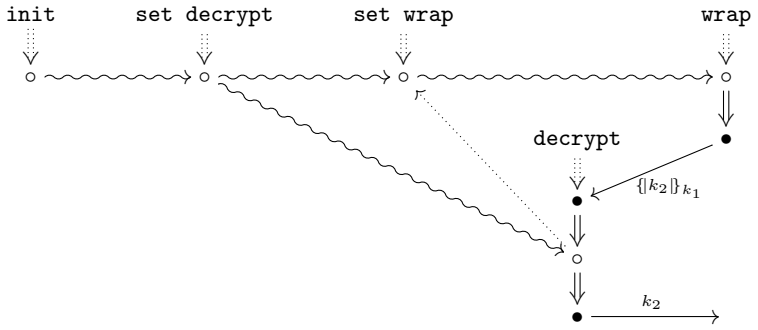

Fig. 6. Observer ordering example

Note that the attack should not be possible. However, the bundle described in Fig. 6 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 [18, 28]. 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, 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 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. [13]. 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 [21] 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 [15] 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. [23] that used CPSA to draw conclusions in the states-as-messages model. Additional consequences could then be proved using the theorem prover PVS [22], working within a theory of both messages and state organized around compatibility.

A group of papers by Ryan with Delaune, Kremer, and Steel [10, 11], 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 [11], 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 [20] 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 [19] 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 [19, 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. 6. 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 [26]. 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, and predicates that are protocol-independent and describe important properties of bundles. The latter includes 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.

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. 6, 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, and Mark D. Ryan. Composition of password-based protocols. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 239–251. IEEE Computer Society Press, June 2008.
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. 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.
14. 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.
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*, 22(2):201–267, 2014.
17. Joshua D. Guttman, Moses D. Liskov, John D. Ramsdell, and Paul D. Rowe. Formal support for standardizing protocols with state (extended version). Arxiv, sep 2015. Available at.

18. Jonathan Herzog. Applying protocol analysis to security device interfaces. *IEEE Security & Privacy*, 4(4):84–87, 2006.
19. Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *IEEE Symposium on Security and Privacy*, pages 163–178, 2014.
20. 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.
21. 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.
22. 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, June 1992. Springer-Verlag. `http://pvs.csl.sri.com`.
23. 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.
24. John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. `http://hackage.haskell.org/package/cpsa`.
25. 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`.
26. 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.
27. 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.
28. 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`.