Security Protocol Analysis in Context: Computing Minimal Executions Using SMT and CPSA [⋆]

Daniel J. Dougherty[1], Joshua D. Guttman[1,2], and John D. Ramsdell[2]

[1] Worcester Polytechnic Institute
[2] The MITRE Corporation

Abstract. Cryptographic protocols are used in different environments, but existing methods for protocol analysis focus only on the protocols, without being sensitive to assumptions about their environments.
LPA is a tool which analyzes protocols in context. LPA uses two programs, cooperating with each other: CPSA, a well-known system for protocol analysis, and Razor, a model-finder based on SMT technology. Our analysis follows the enrich-by-need paradigm, in which models of protocol execution are generated and examined.
The choice of which models to generate is important, and we motivate and evaluate LPA's strategy of building *minimal* models. "Minimality" can be defined with respect to either of two preorders, namely the homomorphism preorder and the embedding preorder (i.e. the preorder of injective homomorphisms); we discuss the merits of each. Our main technical contributions are algorithms for building homomorphism-minimal models and for generating a set-of-support for the models of a theory, in each case by scripting interactions with an SMT solver.

1 Introduction

Cryptographic protocol analysis is well-developed. Many tools and rigorous techniques can determine what confidentiality, authentication (e.g. [35,16,5,11]), and indistinguishability properties (e.g. [6,8,9]) protocols satisfy.

However, what goals a protocol *needs* to achieve depends on the applications that use it. The applications require certain security functionality; a protocol is acceptable if it achieves at least what that functionality relies on. Often, an attack shows that a protocol ensures less than an application needed. For instance, the TLS resumption attacks [37], cf. [4,41] show that the protocol did not allow the server application to distinguish unauthenticated input at the beginning of a data stream from subsequent authenticated input. This may lead to erroneous authorization decisions.

Conversely, a protocol may be good enough for an application because of *environmental assumptions* the application ensures. For instance, some protocols fail if the same long-term key is ever used by a principal when playing the server

⋆ This work was partially support by the US NSF under Grant No. 1714431

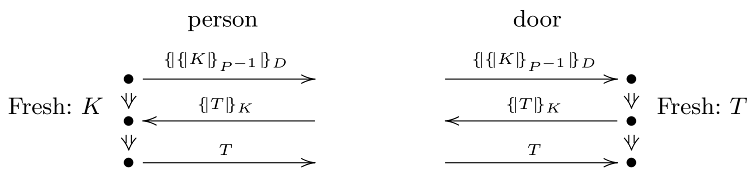

Fig. 1. DoorSEP Protocol

role and also when playing a client role. However, some applications ensure that no server ever executes the protocol in the client role at all. This policy would ensure that an otherwise weak protocol reliably supports the application's needs.

Logical Protocol Analysis is our term for combining a protocol analyzer with these additional concerns, which we analyze via model finding. Our goal is to analyze cryptographic protocols that include trust axioms that cannot be stated using the typical input to a protocol analyzer. We will carry this idea out using the model finder Razor [42] and CPSA, a specialized protocol analysis tool [36,20].

Flawed protocols are often deployed before the flaws are understood, and embedded in widely used devices. Such protocols can still achieve desired security goals when used in a restricted context. If the context can be modeled using environmental assumptions and other trust axioms, Logical Protocol Analysis can be used to discover whether the goals are met in the actual context of use.

An Example: DoorSEP As a motivating scenario consider the Door Simple Example Protocol (DoorSEP), derived from an expository protocol [7] that was designed to have a weakness. Despite this, the protocol achieves the needs of an application, given a trust assumption. Section 4.1 has more detail.

Imagine a door D which is equipped with a badge reader, and a person P equipped with a badge. When the person swipes the badge, the protocol executes. Principals such as doors or persons are identified by the public parts of their key pairs, with D^{-1} and P^{-1} being the corresponding private keys. We write $\{\!|M|\!\}_K$ for the encryption of message M with key K. We represent digital signatures $\{\!|M|\!\}_{P^{-1}}$ as if they were the result of encrypting with P's private key.

P initiates the exchange by creating a fresh symmetric key K, signing it, and sending it to the door D encrypted with the door's public key. D extracts the symmetric key after checking the signature, freshly generates a token T, and sends it—encrypted with the symmetric key—back to P. P demonstrates they are authorized to enter by decrypting the token and sending it as plaintext to the door. The two roles of DoorSEP are shown in Fig. 1, where each vertical column displays the behavior of one of the roles.

CPSA finds an undesirable execution of DoorSEP. Assume the person's private key P^{-1} is uncompromised and the door has received the token it sent out. Then CPSA finds that P freshly created the symmetric key K. However, nothing ensures that the person meant to open door D. If P ever initiates a run with a compromised door C, the adversary can perform a man-in-the-middle attack, decrypting using the compromised key C^{-1} and re-encrypting with D's public

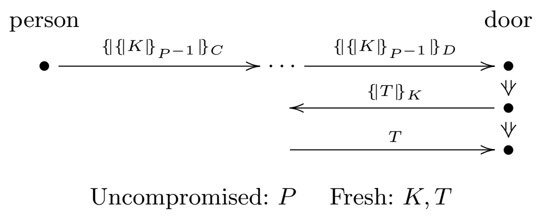

Fig. 2. DoorSEP Weakness

key, as elided in the $\cdots$ in Fig. 2. Thus, without additional assumptions, the door cannot authenticate the person requesting entry.

But possibly we can trust the person to swipe her badge only in front of doors our organization controls. And we can ensure that our doors have uncompromised private keys. If so, then the adversary cannot exercise the flaw. We regard this as a *trust assumption*, and we can express it as an axiom:

Trust Assumption 1. If an uncompromised signing key P^{-1} is used to prepare an instance of the first DoorSEP message, then its owning principal has ensured that the selected door D has an uncompromised private key.

The responsibility for ensuring the truth of this axiom may be split between P and the organization controlling D. P makes sure to swipe her badge only at legitimate doors of the organization's buildings. The organization maintains a security posture that protects the corresponding private keys.

Is DoorSEP good enough, assuming the trust axiom? To analyze DoorSEP under trust assumption 1, we use a model finder, namely *Razor* [42]. We provide it a theory leading to a model containing the man-in-the-middle attack. We then add the trust axiom above. The axiom entails that the adversary cannot decrypt the message sent by the P.

The generated model is then given to CPSA, which infers that the door can decrypt the person's message only if $C = D$, i.e. if P intended it D. Thus, the protocol does its job; namely, ensuring that the door opens only when an authorized person requests it to open.

1.1 Protocols and theories

Security conclusions require protocol analysis combined with other properties, which we will assume are given axiomatically by a theory $\mathcal{G}$. We also regard a protocol Π as determining an axiomatic theory $Th(\Pi)$, namely the theory of Π's executions, as Π runs possibly in the presence of a malicious adversary. Thus, we would like to understand the joint models of $\mathcal{G} \cup Th(\Pi)$, where of course these theories may share vocabulary. In the DoorSEP case, $\mathcal{G}$ is the trust axiom. The models of $\mathcal{G} \cup Th(\Pi)$ are runs of DoorSEP in which the doors and people act as assumed in $\mathcal{G}$.

Enrich-by-need Indeed, our approach is to construct *minimal models* in a *homomorphism order*. We refer to these minimal models as *shapes* [20]. The shapes show all of the minimal, essentially different things that can happen subject to $\mathcal{G} \cup Th(\Pi)$: every execution contains instances—meaning homomorphic images—of the shapes. This is useful to the security analyst who can inspect the minimal models and appraise whether they are compatible with his needs. The analyst can do this even without being able to explicitly state the key security goals. In the case in which $\mathcal{G} = \emptyset$, so that only $Th(\Pi)$ matters, generating these shapes is the central functionality of CPSA [36].

We call this approach to security analysis *enrich-by-need*, since we build homomorphism-minimal models by rising stepwise in the homomorphism order, gradually generating them all. CPSA does so using a "authentication test" method, which yields a compact, uniform way to generate the set of minimal models of the protocol theory [20,27].

Indeed, when the set of shapes is finite, we can summarize them in a formula, the disjunction of the *diagrams* of each. We regard this as the conclusion of an implication; the diagram of the starting scenario is the hypothesis. This *shape analysis sentence* is a strongest security goal achieved by the protocol that has the hypothesis chosen [21,34]. Lemma 9 justifies this idea.

We extend CPSA here to cooperate with another tool to provide models of the whole theory $\mathcal{G} \cup Th(\Pi)$. We effectively split $Th(\Pi)$ into two parts, a hard part T_h and an easy part T_e. Only CPSA will handle the hard part.

We use a general-purpose model-finder, *Razor* [42] to look for minimal models of $\mathcal{G} \cup T_e$ that extend a fragment of a model. When the resulting model $\mathbb{A}$ contains additional behavior of Π, we return to CPSA to handle the hard part T_h, enriching $\mathbb{A}$ with some possible executions. We then return these extensions to Razor. If this process terminates, we have a minimal joint model. By iterating our search, we obtain a covering set of minimal joint models. Razor, in turn, is built as a wrapper around a Satisfiability Modulo Theories (SMT) solver, specifically Z3 [12].

Contributions. We have two goals. First, we define and justify the methods that the new Razor uses to drive Z3 to generate homomorphism-minimal models of a given theory. These homomorphisms are not necessarily embeddings; that is, a homomorphism to construct may map distinct values in its source model to the same value in its target model. To begin with, we need a method to construct, from a model $\mathbb{A}$, a set of sentences $\mathit{homFrom}_{\mathbb{A}}$, true in precisely those models $\mathbb{B}$ such that there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$. We also need a method to construct, from a model $\mathbb{A}$, a set of sentences $\mathit{homTo}_{\mathbb{A}}$, true in precisely those models $\mathbb{B}$ such that there is a homomorphism from $\mathbb{B}$ to $\mathbb{A}$. We show how to use these two resources to compute a set of minimal models that covers all of the models; this method is codified in Razor.

Second, we develop a particular architecture for coordinating Razor and CPSA. In this architecture, Razor handles all aspects of $\mathcal{G} \cup Th(\Pi)$ *except* that it does not enrich a fragmentary execution of Π to obtain its shapes, i.e. the minimal executions that are its images. Instead, we generate an input to CPSA that contains the substructure $\mathbb{A}_0$ containing only protocol behavior. CPSA computes the shapes and extracts the strongest security goal that applies to $\mathbb{A}_0$. It returns this additional information to Razor, which then iterates. We call this cooperative architecture LPA for *Logical Protocol Analysis*.

Structure of the paper In Section 2 we fix some preliminary definitions and notation; we introduce the two existing tools which coordinate to make LPA in Section 3. In Section 4 we describe LPA itself and how it is used to analyze the DoorSEP protocol. Section 5 is a development of some of the underlying theory of using SMT solving to compute and present models, with an emphasis on the question: *which models should be presented to the user?* We end with conclusions and a discussion of future work. Some proofs have been omitted, and some discussion condensed, for lack of space; see [14] for a fuller treatment.

Related Work Model-finding is an active area of investigation [29,10,43,3,38]. But existing model-finders compute an essentially random set of models. Close in spirit to our goals and techniques are lightweight formal methods tools such as Alloy [26] and Margrave [31]. Aluminum [32] supports exploration by returning minimal models: it instruments the model-finding engine of Alloy.

Logic programming languages produce single, *least* models as a consequence of their semantics; this is not a notion of minimality based on homomorphisms, and is traditionally tied to Horn-clause theories. Generalizations of minimality for non-Horn theories have already been used in specifying the semantics of disjunctive logic programming [28] and in non-monotonic reasoning, especially circumscription [39].

Our previous work on a Cryptographic Protocol Programming Language [24,22] led to a programming language that would allow protocol actions to be controlled by a trust management policy.

The Tamarin prover [30] can limit the context in which a protocol is to be analyzed by restricting its analysis to a user-specified subset of all protocol traces. In contrast, our primary interests lie in *enriching* the context in which analysis is done and in generating principled output instances. There was also related work in the applied π-calculus [19,18]. Protocol analysis sometimes builds in environmental assumptions in a security goal hypothesis, by assuming that some keys are uncompromised, or that some principal names are unequal. However, the focus of research has been on the pure problem of determining the security properties of protocols in isolation.

2 Foundations

2.1 Models and Homomorphisms

In this chapter we present some of the foundations of model-finding, focusing on the use of an SMT solver. In broadest terms, model-finding is the following task: given a logical theory $\mathcal{T}$, produce one or more (finite) models of $\mathcal{T}$.

Of course a typical satisfiable theory will have many models. Special emphasis is given in this paper to the question of *which models should be presented to* *the user?* One answer—embodied in the LPA tool—is based on the fundamental notion of *homomorphism* between models, with a focus on models that are *minimal* (see Section 5) in the pre-order determined by homomorphism.

Fix a signature Σ. A *model* $\mathbb{A}$ for signature Σ is defined as usual: a collection of sets interpreting the sorts of Σ, and a collection of functions and relations interpreting the function and relation symbols of Σ. In this paper we work with finite models exclusively.

Definition 2. *Let $\mathbb{A}$ and $\mathbb{B}$ be Σ-models. A* homomorphism *from $\mathbb{A}$ to $\mathbb{B}$ is a sort-indexed family of maps such that*

1. $\mathbb{A} \models f[a_1, \dots, a_n] = a$ *implies* $\mathbb{B} \models f[h(a_1), \dots, h(a_n)] = h(a)$ *and*
2. $\mathbb{A} \models R[a_1, \dots, a_n]$ *implies* $\mathbb{B} \models R[h(a_1), \dots, h(a_n)]$.

Write $\mathbb{A} \precsim \mathbb{B}$ if there is a homomorphism $h : \mathbb{A} \to \mathbb{B}$, and write $\mathbb{A} \approx \mathbb{B}$ if $\mathbb{A} \precsim \mathbb{B}$ and $\mathbb{B} \precsim \mathbb{A}$. Write $\mathbb{A} \precsim^i \mathbb{B}$ if there is an injective homomorphism $h : \mathbb{A} \to \mathbb{B}$, and write $\mathbb{A} \approx^i \mathbb{B}$ if $\mathbb{A} \precsim^i \mathbb{B}$ and $\mathbb{B} \precsim^i \mathbb{A}$. We will sometimes use the phrase "hom-cone of $\mathbb{A}$" to refer to the set of models $\mathbb{B}$ for which there is a homomorphism $h : \mathbb{A} \to \mathbb{B}$.

Definition 3. *Let $\mathcal{M}$ be a class of models. A model $\mathbb{M} \in \mathcal{M}$ is a-*minimal *for $\mathcal{M}$ if whenever $\mathbb{A} \in \mathcal{M}$ and $\mathbb{A} \precsim \mathbb{M}$, we have $\mathbb{A} \approx \mathbb{M}$. The definition of i-minimal is similar, using injective homomorphisms. (The modifier "a−" is to suggest "arbitrary".)*

The notion of the *core* of a model is standard [25,17]; it is important for us because cores will give canonical representatives of $\approx$ equivalence classes.

Core are defined in terms of *retractions,* as follows.

Definition 4. *A* retraction *$r : \mathbb{A} \to \mathbb{B}$ is a homomorphism such that there is a homomorphism $e : \mathbb{B} \to \mathbb{A}$ with $r \circ e = \mathsf{id}_{\mathbb{B}}$.*

A submodel $\mathbb{C}$ of $\mathbb{A}$ is a core *of $\mathbb{A}$ if there is a retraction $r : \mathbb{A} \to \mathbb{C}$ but no retract $r' : \mathbb{A} \to \mathbb{C}'$ for any proper submodel $\mathbb{C}'$ of $\mathbb{C}$.*

A model $\mathbb{C}$ is a core *if it is a core of itself.*

Definition 5 (PE formula, Geometric theory).

A formula is positive-existential, *or* PE*, if it is built from atomic formulas (including* true *and* false*) using $\wedge$, $\vee$ and $\exists$. A* geometric *sentence is one of the form*

$$\forall \vec{x}. \quad \alpha(\vec{x}) \to \beta(\vec{x})$$

where α and β are positive-existential.

Theorem 6. *The following are equivalent, for a formula $\alpha(\vec{x})$:*

1. *α is preserved by homomorphism: if $h : \mathbb{A} \to \mathbb{B}$ is a homomorphism, and $\vec{a}$ is a vector of elements from $\mathbb{A}$ such that $\mathbb{A} \models \alpha[\vec{a}]$, then $\mathbb{B} \models \alpha[\vec{ha}]$.*
2. *α is logically equivalent to a PE formula.*

3. α is equivalent, in the category $\mathcal{M}_\Sigma$ of finite models, to a PE formula.

Proof. The equivalence of (1) and (2) is a classical result in model theory when considering arbitrary models. The equivalence of (1) and (3) is a deep result of Rossman [40].

The case for geometric logic as a logic of observable properties was made clearly by Abramsky [1]. As detailed in [21], typical security goals for protocols are naturally expressed as geometric sentences. (As is well-known, *any* theory is equisatisfiable with one in conjunctive normal form, by introducing Skolem functions. Such an enrichment of the theory signature is not innocent, however, since it has consequences for the existence of homomorphisms between models.)

It is straightforward to see that when T is geometric, if $\mathbb{A}$ is a model of T then a retraction of $\mathbb{A}$ is a model of T.

Lemma 7. *Let T be a geometric theory, $\mathbb{A} \models T$, and $r : \mathbb{A} \to \mathbb{B}$ a retraction. Then $\mathbb{B} \models T$.*

Definition 8. *If $\mathcal{M}$ is a class of Σ-models and $\mathcal{M}_0 \subseteq \mathcal{M}$ say that $\mathcal{M}_0$ is an* a-set of support for *$\mathcal{M}$ if for all $\mathbb{B} \in \mathcal{M}$, there exists $\mathbb{A} \in \mathcal{M}_0$ with $\mathbb{A} \precsim \mathbb{B}$. Similarly for i-set of support.*

A set of support for a class of models provides a complete "testbed" for entailment of geometric sentences:

Lemma 9. *Let $\sigma \equiv \forall \vec{x}.\ \alpha(\vec{x}) \to \beta(\vec{x})$ be geometric and let $\mathcal{M}$ be a class of models. Let $\mathcal{M}_0$ be an a-set of support for $\{\mathbb{A} \in \mathcal{M} \mid \mathbb{A} \models \exists \vec{x}.\ \alpha(\vec{x})\}$. If every model in $\mathcal{M}_0$ satisfies σ then every model in $\mathcal{M}$ satisfies σ.*

Proof. Let $\mathbb{P} \in \mathcal{M}$ with $\mathbb{P} \models \alpha[\vec{a}]$; we want to show that $\mathbb{P} \models \beta[\vec{a}]$. Let $\mathbb{M} \in \mathcal{M}_0$ with $\mathbb{M} \precsim \mathbb{P}$. Since $\mathbb{M} \models \sigma$, $\mathbb{M} \models \beta[\vec{a}]$. Since β is PE and $\mathbb{M} \precsim \mathbb{P}$, $\mathbb{P} \models \beta[\vec{a}]$.

2.2 Strand Spaces

We can formalize protocol executions as models, as follows. A run of a protocol is viewed as an exchange of messages by a finite set of local sessions of the protocol. Each local session is called a *strand:* a strand is a sequence of nodes n, each of which is a *transmission* or a *reception* of the *message* $msg(n)$ at that node.

A *strand space* Θ is a finite sequence of strands. A message that originates in exactly one strand of Θ is *uniquely originating*, and represents a freshly chosen value. A message is *mentioned* in Θ if it occurs in a strand of Θ, or if it is an asymmetric key, its inverse occurs in a strand of Θ. A message that is mentioned but originates nowhere in Θ is *non-originating*, and often represents an uncompromised key.

A *protocol* Π is a finite set of strands, which are the *roles* of the protocol. A strand s is an *instance* of a role $\rho \in \Pi$, if $s = \alpha(\rho)$, i.e. if s results from ρ by applying a substitution α to parameters in ρ.

Skeletons are fragmentary executions of the regular participants, which factor out adversary behavior. A *skeleton* $\mathbb{K} = (\mathsf{nodes}, \preceq, \mathsf{non}, \mathsf{unique})$ consists of a finite set of regular nodes, a partial ordering on them, a set of values assumed non-originating, and a set of values assumed uniquely originating. These components are designed to code in the aspects of executions that we care about, namely the ordering, and what values are uncompromised ("non") or freshly chosen ("unique").

A skeleton $\mathbb{K}$ is an *execution*, or *realized*, iff for every message received in $\mathbb{K}$, the Dolev-Yao adversary [13] can derive that message with the help of earlier transmissions in $\mathbb{K}$.

Associated with each CPSA protocol Π is a first-order language $\mathcal{L}(\Pi)$ used to specify security goals [21]. The language can be used to exchange information between CPSA and an SMT solver. These mechanisms are described in Section 4.

3 Constituent Tools

CPSA The Cryptographic Protocol Shapes Analyzer [35] (CPSA) can be used to determine if a protocol achieves authentication and secrecy goals. CPSA will—given a protocol Π and a skeleton of interest $\mathbb{K}$—generate all of the minimal, essentially different realized skeletons that are homomorphic images of $\mathbb{K}$. We call these minimal, essentially different skeletons *shapes*, and, although in general there could be infinitely many of them, frequently there are very few of them.

CPSA begins a run with a protocol description and an initial scenario $\mathbb{K}_0$. The initial scenario is a partial description of executions of a protocol. If CPSA terminates, it characterizes all the executions of the protocol consistent with the initial scenario. For example, if it is assumed that one role of a protocol runs to completion, CPSA will determine what other roles must have executed.

Each skeleton $\mathbb{K}$ has a *characteristic sentence* $\sigma_{\mathbb{K}}$ such that, for all $\mathbb{K}'$, $h : \mathbb{K} \to \mathbb{K}'$ (for some homomorphism h) iff $\mathbb{K}' \models \sigma_{\mathbb{K}}$.

Homomorphisms play an essential role in CPSA. At each step in the algorithm, an unrealized skeleton $\mathbb{K}$ is replaced by a set of skeletons $\{\mathbb{K}_1, \ldots, \mathbb{K}_n\}$, called a cohort, by solving an authentication test [23]. The skeletons $\{\mathbb{K}_1, \ldots, \mathbb{K}_n\}$ form an *a-set of support* for the realized skeletons that are homomorphic images of $\mathbb{K}$. That is, if there is an execution (or "realized skeleton") $\mathbb{K}_r$ such that $h : \mathbb{K} \to \mathbb{K}_r$, then there exists some homomorphism $h' : \mathbb{K}_i \to \mathbb{K}_r$ such that $h = h' \circ h_i$.

For an initial scenario $\mathbb{K}_0$, CPSA produces a set of realized skeletons $\{\mathbb{K}_1, \ldots, \mathbb{K}_n\}$ and homomorphisms $h_i : \mathbb{K}_0 \to \mathbb{K}_i$. These are built up by a succession of cohort steps; thus, they remain an *a*-set of support for the realized skeletons that are homomorphic images of $\mathbb{K}_0$. The set $h_i : \mathbb{K}_0 \to \mathbb{K}_i$—called the *shapes* of this scenario—are a compact way of describing all of the executions compatible with the initial scenario. By Lemma 9, if a geometric sentence σ holds in each shape, then σ holds in every realized skeleton that is an image of $\mathbb{K}_0$.

There is a key geometric sentence that can be extracted from the results of a run of CPSA. A Shape Analysis Sentence (SAS) [34] encodes everything that has been learned about the protocol from a CPSA analysis starting with a given initial scenario. It holds in every realized skeleton of the protocol. A SAS is used to import the results of a CPSA analysis into the SMT solver.

The antecedent of a SAS is a conjunction of atomic formulas that specify the initial scenario $\mathbb{K}_0$. The universally quantified variables are the ones that occur in the antecedent. The conclusion is a disjunction of formulas, one for each shape. The i^{th} disjunct is an existentially quantified conjunction of atomic formulas that describes the mapping h_i and the additions to the antecedent required to specify shape $\mathbb{K}_i$.

Razor Razor is a general-purpose model-finder: it takes as input an arbitrary first-order theory T and attempts to find finite models of T (CPSA can be viewed as a domain-specific model-finder, working over various theories of strand spaces).

Razor finds models by (i) preprocessing the input theory as described below, (ii) using an off-the-shelf SMT solver, currently Z3, and (iii) postprocessing the results of the solver's output to fulfill certain goals: return *minimal* models by default, allowing the user to explore and augment models, and computing a set-of-support of models for T. Razor can be used in REPL mode or batch mode; only the latter is used as part of LPA (refer to [42] for a fuller description of Razor's REPL mode).

Once the SMT solver has determined that a theory T is satisfiable, and computed—internally—a model for T, the application must extract the model from the solver. But the API mandated by the SMT-Lib Standard (v.2.6) [2] for doing this is quite restricted. The model can be inspected only through certain commands returning the solver's internal representation of values of terms.

This is inconvenient for us, especially since the solver might create only a partial model internally.

To address this, we first ensure that the language we use to communicate with the solver has enough ground terms at each sort to name all elements of a model, by adding fresh constants. Then we can query the solver for the values of the functions and predicates, and build a "basic" model representation

$$\begin{array}{rll} \text{equations} & c_i = c_j & \text{and} \\ \text{equations} & f\vec{c} = c & \text{and} \\ \text{facts} & R\vec{c} & \end{array}$$

where the c_i range over the fresh constants. Using standard techniques we then construct from these equations a convergent (terminating and confluent) ground rewrite system, which facilitates working with the models.

4 LPA

This section shows how to use CPSA and Razor to analyze cryptographic protocols in context. Our architecture for LPA is displayed in Figure 3. An analysis begins with a CPSA protocol Π and an initial theory T_0. The initial theory contains a specification of the trust policy and a description of the initial scenario of the protocol as a collection of sentences in $\mathcal{L}^+(\Pi)$, an extension of $\mathcal{L}(\Pi)$.

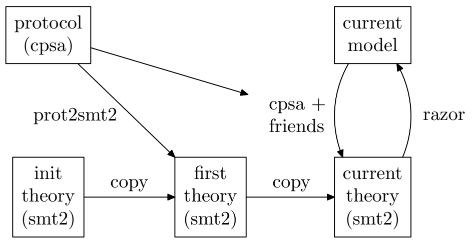

Fig. 3. LPA Architecture

The program `prot2smt2` uses protocol Π to generate a set of geometric axioms $Th(\Pi)$ [14, Section 2.1.5]. These axioms allow Razor to produce models from which skeletons can be extracted. For example, an axiom about the transitivity of node orderings allows Razor to compute the partial ordering of the nodes. Other axioms ensure that a uniquely originating value is received only after it is transmitted and that the double inverse of each asymmetric key is equal to itself.

The initial theory is appended to $Th(\Pi)$ to form the first theory T_1 to be analyzed by Razor. A skeleton is extracted from each model. If the skeleton is realized, the model describes the impact of the trust policy on complete executions of the protocol. If the skeleton is not realized, it is used as the initial scenario for CPSA. The results of CPSA is turned into a SAS (shape analysis sentence, cf Section 3) and added to the current theory for further analysis. The process is repeated until all of the extracted skeletons are realized.

4.1 Analyzing the Door Simple Example Protocol

We now expand on the analysis of the DoorSEP protocol introduced in Section 1. In this protocol, a person begins by generating a fresh symmetric key, signing it, and then encrypting the result using the door's public key. If the door accepts the first message, it responds by freshly generating a token and uses the symmetric key to encrypt it. If the door receives the token back unencrypted, the door concludes the person that generated the key is at the door and opens.

The initial theory specifies Trust Assumption 1 and the fact that the door is open. To assert the door is open, one asserts there is a strand that is a full length instance of the door role. We further assert that the person's private key is uncompromised.

Recall the diagram in Figure 3 to visualize the analysis process. After appending the initial theory to the protocol axioms $Th(\Pi)$, Razor finds model $\mathcal{M}_0$. As expected, model $\mathcal{M}_0$ specifies a full length door strand in which the person's private key is uncompromised and other facts such as the fact that double inverse of the model's asymmetric keys are equal to themselves.

At this stage, we have a model that characterizes an unrealized skeleton, and we would like to use CPSA to find out what else must have happened. The shape produced by CPSA is displayed in Figure 2. The shape shows the lack of mutual authentication built into this flawed protocol.

The next step in the analysis makes use of the trust axiom. The result of the CPSA analysis is transformed into a SAS. The antecedent specifies the initial scenario described by the first model. The consequence specifies what else must be added to make the initial scenario into the complete execution shown in Figure 2.

When the SAS is added to the current theory, Razor finds one model $\mathcal{M}_1$. The skeleton extracted from this model is very similar to the shape in Figure 2 with one crucial difference: the key D' is uncompromised. Razor applied the trust axiom. The skeleton extracted from $\mathcal{M}_1$ is unrealized, so CPSA can make a contribution. It finds a SAS that extends the length of the person strand to full length and equates D and D'. The addition of this SAS produces model $\mathcal{M}_2$ that characterizes a realized skeleton with full agreement between the door and person strands. Because the skeleton is realized, CPSA has nothing more to contribute and the analysis terminates.

5 Minimality, Cores, and Set-of-Support

In this section we explore the question *which models should we compute and show to the user of a model-finding tool?* Our proposal, motivated by Lemma 9 and implemented by LPA, is: *compute a set-of-support for the input theory comprised of minimal models.* As we have observed there are two natural notions of minimality; we point out some theoretical differences between them. Most importantly, we present algorithms for computing minimal models and sets-of-support: these involve programming against the functionality of SMT solvers.

5.1 Comparing *i*-minimal and *a*-minimal

One way to think about a-minimality of a model $\mathbb{M}$ is that if any atomic fact of $\mathbb{M}$ is removed, the resulting model would no longer be a model of the theory at hand. In particular, since equality is an atomic predicate, if two terms denote—unnecessarily—the same model-element, this is a failure of a-minimality.

Neither of i-minimality or a-minimality implies the other.

Example 10.

- Let T be the single sentence $\exists x.P(x) \wedge \exists x.Q(x)$, and let $\mathbb{A}$ have one element a with $\mathbb{A} \models P[a] \wedge Q[a]$.
  Then $\mathbb{A}$ is i-minimal but not a-minimal: the model $\mathbb{B}$ with two elements a_1 and a_2 such that $\mathbb{B} \models P[a_1] \wedge Q[a_2]$ is strictly below $\mathbb{A}$ in the $\precsim$ preorder. ($\mathbb{B}$ is a-minimal for T.)
- Let T be $\exists x.P(x)$ and let $\mathbb{A}$ have two elements a_1 and a_2 with $\mathbb{A} \models P[a_1]$ and $\mathbb{A} \models P[a_2]$. Then $\mathbb{A}$ is a-minimal but is not i-minimal: the induced model determined by a_1 is a model of T.

However, an a-minimal model which is a core *will* be i-minimal.

Lemma 11. *If $\mathbb{A}$ is a-minimal for T and is a core, then $\mathbb{A}$ is i-minimal for T.*

Proof. Suppose $\mathbb{B}$ is a model of T and $j : \mathbb{B} \to \mathbb{A}$ is injective. Since $\mathbb{A}$ is *a*-minimal, there is a homomorphism $h : \mathbb{A} \to \mathbb{B}$. The composition $j \circ h$ is an endomorphism of $\mathbb{A}$. Since $\mathbb{A}$ is a core this map is injective, so h is injective, and $\mathbb{A} \approx^i \mathbb{B}$.

We should observe that for a given theory there might be no finite a-minimal models at all. An example is the theory with one unary function and no axioms. The initial (hence unique minimal) model of this theory is the natural numbers. Another way to put this is: the $\precnsim$ preorder is not well-founded in general.

On the other hand, we will typically add axioms to a theory to ensure that there is an upper bound on the size of its models. In such a case there will be only finitely many models of T, and the $\precnsim$ preorder will be well-founded. This is the key to the termination of many of the algorithms in this section. (There will always be *a*-minimal models for theories T that are bounded in this way.)

Lemma 12. *Let T be a theory with only finitely many models. Then the $\precnsim$ and $\precnsim^i$ preorders on models of T are well-founded.*

Proof. Suppose for the sake of contradiction that we have an infinite descending chain $\ldots \precnsim \mathbb{M}_2 \precnsim \mathbb{M}_1 \precnsim \mathbb{M}_0$ of strict homomorphisms. Then we have $\mathbb{M}_{i+k} \precnsim \mathbb{M}_i$ for any $k \geq 0$. Since T has finitely many models, we eventually get i and $k \geq 0$ with $\mathbb{M}_{i+k+1}$ isomorphic to $\mathbb{M}_i$. So $\mathbb{M}_{i+k+1} \precnsim \mathbb{M}_{i+1}$. But that implies $\mathbb{M}_i \precnsim \mathbb{M}_{i+1}$, a contradiction.

The same argument applies to $\precsim^i$ as well.

5.2 Minimal Models for protocol analysis.

When model-finding is used for protocol analysis, specifically when reasoning about an authentication goal, minimality with respect to arbitrary homomorphisms is of particular interest. Consider, for example, the analysis of the authentication properties of DoorSEP. The model $\mathbb{A}$ corresponding to the failure of authentication described in the Introduction is one in which there are keys for two *different* doors D and D' involved in the protocol run. The model $\mathbb{B}$ which would arise from identifying D and D' would still represent a protocol execution (indeed, the hoped-for behavior of the protocol). But $\mathbb{A}$ is strictly below this $\mathbb{B}$ in the $\precsim$ ordering, and it is $\mathbb{A}$ that gives insight in to the possibility of the man-in-the-middle attack (in the absence of the trust axiom, of course).

5.3 Computing Minimal Models and Set-of-Support

We present the following algorithms, each of which relies on the primitive operation of asking an SMT solver for a single finite model of a given theory. Recall that an SMTLib-compliant solver need not return any *particular* model for a satisfiable theory, and that repeated requests to a solver for the same theory will typically return the same model.

Fix a theory T.

- iMinimize: given model $\mathbb{A} \models T$, compute an i-minimal model $\mathbb{M} \models T$ with $\mathbb{M} \precsim^i \mathbb{A}$.
- aMinimize: given model $\mathbb{A} \models T$, compute an a-minimal model $\mathbb{M} \models T$ with $\mathbb{M} \precsim \mathbb{A}$.
- computeCore: given model $\mathbb{A}$, compute the core of $\mathbb{A}$.
- SetOfSupport (resp. iSetOfSupport): compute a stream of models comprising a (resp. injective) set of support for theory T.
- aHomTo (resp. iHomTo): given model $\mathbb{A} \models T$, compute a sentence $homTo_{\mathbb{A}}$ defining the models $\mathbb{P} \models T$ such that there is a (resp. injective) homomorphism $h : \mathbb{P} \to \mathbb{A}$.
- aHomFrom (resp. iHomFrom): given model $\mathbb{A} \models T$, compute a sentence $homFrom_{\mathbb{A}}$ defining the models $\mathbb{P} \models T$ such that there is a (resp. injective) homomorphism $h : \mathbb{A} \to \mathbb{P}$.

The algorithms aMinimize and computeCore each rely on the sentences $homTo_{\mathbb{A}}$ and $homFrom_{\mathbb{A}}$. The latter of these is subtle, so **we first present the other algorithms in terms of these, then develop aHomTo and aHomFrom.**

5.4 *i*-Minimization

The following procedure was originally developed for use in the *Aluminum* tool [33] For this algorithm we use the notation $flip_{\mathbb{P}}$ to denote

$$\bigwedge\{\neg\alpha \mid \alpha \text{ is atomic, } \mathbb{P} \models \neg\alpha\} \wedge \bigvee\{\neg\beta \mid \beta \text{ is atomic, } \mathbb{P} \models \beta\}$$

Note in particular that if c and c' are constants naming distinct elements of a model $\mathbb{P}$, then $c \neq c'$ is one of the conjuncts of $flip_{\mathbb{P}}$.

Algorithm 13 (i-Minimize).

```
input: theory $T$ and model $\mathbb{A} \models T$
output: model $\mathbb{P} \models T$ such that $\mathbb{P}$ is $i$-minimal for $T$ and $\mathbb{P} \precsim^i \mathbb{A}$
initialize: set $\mathbb{P}$ to be $\mathbb{A}$
while $T' \overset{def}{=} T \cup \{flip_{\mathbb{P}}\}$ is satisfiable
    set $\mathbb{P}$ to be a model of $T'$
return $\mathbb{P}$
```

Lemma 14. *Algorithm 13 is correct: if $\mathbb{A}$ is a finite model of T then Algorithm 13 terminates on $\mathbb{A}$, and the output $\mathbb{P}$ is an i-minimal model of T with $\mathbb{P} \precsim^i \mathbb{A}$*

Proof. Each iteration goes down in the $\precnsim^i$ ordering, thus termination. To show that the result is i-minimal for T, it suffices to argue that the result is a minimal T-submodel of the input, under the submodel ordering. But this is clear from the definition of the sentences *flip*.

5.5 *a*-Minimization

Computing a-minimal models is harder. If we bound the size of the domain(s) of our models then a-minimal models exist: the $\precsim$ preorder is well-founded, so the set of minimal elements with respect to this order is non-empty. The question is, how to compute a-minimal models?

The idea is that, given a model $\mathbb{A}$, we can use the sentences $homTo_{\mathbb{A}}$ and $homFrom_{\mathbb{A}}$ to iterate the process of constructing a model that is strictly below $\mathbb{A}$ in the $\precsim$ ordering.

Algorithm 15 (a-Minimize).

```
**input:** theory $T$ and model $\mathbb{A} \models T$
**output:** model $\mathbb{P} \models T$ such that $\mathbb{P}$ is a-minimal for $T$ and $\mathbb{N} \precsim \mathbb{A}$
**initialize:** set $\mathbb{P}$ to be $\mathbb{A}$
**while** $T' \overset{def}{=} T \cup \{homTo_{\mathbb{P}}\} \cup \{\neg homFrom_{\mathbb{P}}\}$ is satisfiable
    set $\mathbb{P}$ to be a model of $T'$
**return** $\mathbb{P}$
```

Lemma 16. *Algorithm 15 is correct: if $\mathbb{A}$ is a finite model of T then Algorithm 13 terminates on $\mathbb{A}$, and the output $\mathbb{P}$ is an a-minimal model of T with $\mathbb{P} \precsim \mathbb{A}$*

Proof. Each iteration constructs a model lower in the $\precsim$ ordering; termination follows from well-foundedness of the $\precsim$ ordering.

5.6 Computing Cores

Cores are interesting for us because—when the input theory T is geometric—they give a way to build models that are both a-minimal and i-minimal.

Testing whether a model is a core is NP-complete [25]. So computing cores is presumably expensive, from a worst-case complexity perspective. But it is not difficult, using an SMT solver, to write a program that behaves well in practice. The key point is the well-known observation that a model $\mathbb{C}$ has no proper retracts if and only if it has no proper endomorphisms.

Definition 17. *If $\mathbb{A}$ is a finite model for signature Σ, the sentence $endo_{\mathbb{A}}$, over the signature Σ_h that extends Σ by adding a new function symbol $h_s : S \to S$ at each sort S, is the conjunction of*

- *the diagram of $\mathbb{A}$,*
- *the sentence expressing "h is a homomorphism", and*
- *the sentence expressing "h is not injective."*

Algorithm 18 (ComputeCore).

```
**input:** model $\mathbb{A}$ over signature $\Sigma$
**output:** a core $\mathbb{P}$ of $\mathbb{A}$
```

```
**initialize:** Set $\mathbb{P}$ to be $\mathbb{A}$
**while** $endo_{\mathbb{P}}$ is satisfiable
    let $\mathbb{P}'$ be a model of $endo_{\mathbb{P}}$;
    let $\mathbb{P}_0$ be the image of $endo_{\mathbb{P}}$ in $\mathbb{P}$';
    let $\mathbb{P}$ be the reduct of $\mathbb{P}_0$ to the original signature $\Sigma$
**return** $\mathbb{P}$
```

Lemma 19. *Algorithm 18 computes a core of its input.*

Proof. The algorithm terminates because the size of the model $\mathbb{P}$ decreases at each iteration. The resulting model is a core, since it has no proper endomorphisms.

5.7 Set of Support

We take the ability to generate a set-of-support for the class of all models of a theory T to be a natural notion of "completeness" in model-finding. Lemma 9 makes a precise claim of completness with respect to reasoning about geometric consequences of T.

It should be noted that if a class $\mathcal{C}$ is a set-of-support for a theory T with respect to i-homomorphisms then $\mathcal{C}$ is a set-of-support for T with respect to a-homomorphisms; this is immediate from the definitions.

There will be typically many more models comprising an i-set of support. However, it is true that if there is a *finite* $\mathcal{C}$ which is a set-of-support for a theory T with respect to a-homomorphisms then there is a finite $\mathcal{C}'$ set-of-support for T with respect to i-homomorphisms. To see this, suppose $\mathcal{C}$ is a set of support for a class of models. Each $\mathbb{A}$ in this set has a finite number of i-minimal models $B_1, \dots B_k$ below it. The collection of all these taken over the models in $\mathcal{C}$ makes a i-set of support.

Computing sets-of-support is another application of the $homFrom_{\mathbb{A}}$ technique. Given theory T and model $\mathbb{A}$, if we construct the theory $T' \overset{def}{=} T \cup \{\neg homFrom_{\mathbb{A}}\}$ then calls to the SMT solver on theory T' are guaranteed to return models of T outside the hom-cone of $\mathbb{A}$ if any exist. So a set-of-support for T can be generated by iterating this process.

Completeness of this strategy does not require that the models $\mathbb{A}$ we work with are minimal. But if we do work with minimal models there will be fewer iterations. We give SetOfSupport here, for iSetOfSupport simply use i-minimal models and the $iHomFrom_{\mathbb{A}}$ sentence.

*Algorithm 20 (*SetOfSupport*).*

```
**input:** theory $T$ and profile $prf$
**output:** a stream $\mathbb{M}_1, \mathbb{M}_2, \dots$ of minimal models of $T$ such that for any
    $prf$-model $\mathbb{P} \models T$, there is some $i$ such that $\mathbb{M}_i \precsim \mathbb{P}$.
**initialize:** set theory $T^*$ to be $T$
**while** $T^*$ is satisfiable
    let $\mathbb{M}$ be an $a$-minimal model of $T^*$
    **output** $\mathbb{M}$
    set $T^*$ to be $T^* \cup \neg homFrom_{\mathbb{M}}$
```

5.8 Hom-To

This is straightforward "solver programming". Given model $\mathbb{A}$, we want to characterize those $\mathbb{P}$ such that there is a hom $h : \mathbb{P} \to \mathbb{A}$, by constructing a sentence $homTo_{\mathbb{A}}$ axiomatizing such models.

Algorithm 21 (HomTo).

```
**input:** model $\mathbb{A}$ over signature $\Sigma$.
**output:** sentence $homTo_{\mathbb{A}}$ in an expanded signature $\Sigma^+$, such that for any
model $\mathbb{P} \models \Sigma$, $\mathbb{P} \precsim \mathbb{A}$ iff there is an expansion $\mathbb{P}^+$ of $\mathbb{P}$ to $\Sigma^+$ with $\mathbb{P}^+ \models$
$homTo_{mM}$.

**define** $\Sigma^+$ to be the extension of $\Sigma$ obtained by
  - adding a set of fresh constants naming elements of the domain of $\mathbb{A}$
  - adding a function symbol $h_S : S \to S$ at each sort $S$
**return** $homTo_{\mathbb{A}}$ as the conjunction of the following sentences, one for each
function symbol $f$ and predicate $R$ in $\Sigma$. Here $\vec{e}$ and $e'$ range over the names
for elements of $\mathbb{A}$.
```

$$\forall \vec{x}, y.\ f\vec{x} = y \implies \bigvee \{ (\vec{hx} = \vec{e} \wedge y = e') \mid \mathbb{A} \models f\vec{e} = e' \}$$

$$\forall \vec{x}.\ R\vec{x} = true \implies \bigvee \{ (\vec{hx} = \vec{e}) \mid \mathbb{A} \models R\vec{e} = true \}$$

For $iHomTo_{,}$ simply add a sentence to say that h is injective.

Lemma 22. *There is a homomorphism from $\mathbb{B}$ to $\mathbb{A}$ iff there is a model $\mathbb{B}^+ \models homTo_{\mathbb{A}}$ such that $\mathbb{B}$ is the reduction to Σ of $\mathbb{B}^+$.*

5.9 Hom-From

Our eventual goal is: given a model $\mathbb{A}$, find a formula to capture **not** being in the hom-cone of $\mathbb{A}$. This is more interesting than the aHomTo problem, because we are going to *negate* the sentence we build, to express hom-cone-avoidance. Since universal quantifiers can be bottlenecks in SMT-solving, we want to minimize the number of existential quantifiers we use here.

The ideal outcome would be to construct an existential sentence capturing the complement of the hom cone of $\mathbb{A}$. Equivalently we might look for a structure $\mathbb{D}$ such that for any $\mathbb{X}$, $\mathbb{X} \precsim \mathbb{D}$ iff $\mathbb{A} \not\precsim \mathbb{X}$. This is called "homomorphism duality" in the literature [15]. Such a structure doesn't always exist, and even if it does, it can be exponentially large in the size of $\mathbb{A}$ [15]. So we turn to heuristic methods.

Our strategy is to construct a sentence (to be negated) which is guaranteed to characterize models in the hom-cone of $\mathbb{M}$, then refine this sentence to eliminate (some) quantifiers. We start with the equations of the standard model representation for $\mathbb{A}$ as described in Section 2. By replacing the Razor-defined constants by existentially-quantified variables we arrive at a sentence $rep_{\mathbb{A}}$, which is a positive-existenial sentence (without disjunctions).

By the fact that homomorphisms preserve positive existential formulas and the fact that the equations of $rep_{\mathbb{A}}$ completely describe the functions and predicates true of $\mathbb{A}$ we have:

Lemma 23. *Let $\mathbb{A}$ and $\mathbb{F}$ be Σ models. Then $\mathbb{A} \precsim \mathbb{F}$ iff $\mathbb{F} \models rep_{\mathbb{A}}$.*

The trouble with $rep_{\mathbb{A}}$ is that it has as many existential quantifiers in $rep_{\mathbb{A}}$ as there are domain elements. If we were to take $homFrom_{\mathbb{A}}$ to be $rep_{\mathbb{A}}$, simply negating this would lead to a sentence inconvenient for the SMT solver. We can compress the representation, though. This will lead to a nicer representation sentence, which we will take as our $homFrom_{\mathbb{M}}$.

Algorithm 24 (HomFrom).

```
input: model $\mathbb{A}$ over signature $\Sigma$
output: sentence $homFrom_{\mathbb{A}}$ over signature $\Sigma$, such that for any model
$\mathbb{P} \models \Sigma$, $\mathbb{A} \precsim \mathbb{P}$ iff $\mathbb{P} \models homFrom_{\mathbb{A}}$.
comment: sentence $homFrom_{\mathbb{A}}$ is designed to use as few existential quanti-
fiers as possible, in a "best-effort" sense.
initialize: Set sentence $homFrom_{\mathbb{A}}$ to be $rep_{\mathbb{A}}$, the standard model repre-
sentation sentence for $\mathbb{A}$.
while there is a conjunct in the body of $homFrom_{\mathbb{A}}$ of the form $f(t_1, \ldots, t_n) =$
$x$ such that $x$ does not occur in any of the $t_i$,
  - replace all occurrences of $x$ in $homFrom_{\mathbb{A}}$ by $f(t_1, \ldots, t_n)$. Erase the
    resulting trivial equation $f(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ and erase the $(\exists x)$
    quantifier in front.
```

For *iHomFrom*, first enrich $rep_{\mathbb{A}}$ to say that each of the fresh constants naming elements of $\mathbb{A}$ is distinct. The rest of the development goes through as described.

Lemma 25. *For any model $\mathbb{P} \models \Sigma$, $\mathbb{A} \precsim \mathbb{P}$ iff $\mathbb{P} \models homFrom_{\mathbb{A}}$. Similarly for $iHomFrom_{\mathbb{A}}$ and $\precsim^i$.*

The order in which we do these rules matters, in the sense that smaller formulas result if we process nodes as follows. Construct a graph in which the nodes are the variables occurring in the set of equations, and in which, if $fx_1 \ldots x_n = x$ is a rule, then there is an edge from each x_i to x. Then process the nodes according to the preorder given by this graph.

Example 26. Start with

$$\sigma \equiv \exists x_0 x_1 x_2 \;.\; fx_0 = x_2 \land fx_1 = x_0 \land fx_2 = x_1 \land c = x_2$$

Making the graph as defined above, we treat the variables in the order x_2, then x_1 then x_0. We then derive, in order:

$$\begin{aligned} \exists x_0 x_1 \;.\; & fx_0 = c \land fx_1 = x_0 \land fc = x_1 \\ \exists x_0 \;.\; & fx_0 = c \land ffc = x_0 \\ & fffc = c \end{aligned}$$

An SMT solver will work more happily with $fffc \neq c$ than with $\neg\sigma$.

5.10 Section Summary

1. i-minimal models for a theory T always exist; there may be no finite a-minimal models for a given theory.
2. a-minimal models are better suited to protocol analysis since they do not make unnecessary identifications between terms.
3. i-minimal models are easier to compute than a-minimal models.
4. If T is a geometric theory, and $\mathbb{M}$ is an a-minimal model and a core, then $\mathbb{M}$ is i-minimal (Lemma 11).
5. If a class $\mathcal{C}$ is a set-of-support for a theory T with respect to i-homomorphisms then $\mathcal{C}$ is a set-of-support for T with respect to a-homomorphisms.
6. If there is a finite $\mathcal{C}$ which is a set-of-support for a theory T with respect to a-homomorphisms then there is a finite $\mathcal{C}'$ set-of-support for T with respect to i-homomorphisms.

6 Conclusions and Future Work

In this paper, we have developed a method for analyzing systems with cryptographic protocols in the context of first-order theories such as trust assumptions, and presented a detailed analysis of a specific example, the DoorSEP protocol.

We have described an implementation of these methods as the Logical Protocol Analysis (LPA) system. LPA is a coordination between a general-purpose model-finder, Razor, and a cryptographic protocol-specific tool, CPSA. We have shown how to share labor between Razor and CPSA so that the latter can apply its authentication test solving methods, while Razor is handling the remainder of the axiomatic theory of the protocol together with some non-protocol axioms.

We explored the comparative virtues of minimality with respect to injective homomorphisms versus arbitrary homomorphisms, and developed algorithms for finding minimal models and computing a set-of-support of models for a theory.

Unfortunately, as the size of a protocol grows, so does the size of its theory, and SMT solvers struggle with performance in the presence of a significant number of universal quantifiers. In future work, we plan to reorganize the software architecture to be one in which only subtheories are delivered to the solver, preferably governing smaller parts of the domain.

References

1. S. Abramsky. Domain Theory in Logical Form. *Ann. Pure Applied Logic*, 1991.
2. C. Barrett, A. Stump, C. Tinelli, et al. The SMT-LIB standard: Version 2.0. In *Proc. 8th International Workshop on Satisfiability Modulo Theories*, volume 13, page 14, 2010.
3. P. Baumgartner, A. Fuchs, H. De Nivelle, and C. Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. *Journal of Applied Logic*, 2009.
4. K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *IEEE Symposium on Security and Privacy*, 2014.

5. B. Blanchet. From secrecy to authenticity in security protocols. In *9th Static Analysis Symposium*, number 2477 in LNCS, pages 342–359. Springer Verlag, September 2002.
6. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 86–100. IEEE CS Press, May 2004.
7. B. Blanchet. *Vérification automatique de protocoles cryptographiques: modèle formel et modèle calculatoire.* Habilitation thesis, Université Paris-Dauphine, Nov. 2008.
8. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.
9. R. Chadha, V. Cheval, Ştefan Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23:1–23:32, 2016.
10. K. Claessen and N. Sörensson. New Techniques that Improve MACE-Style Finite Model Finding. In *CADE Workshop on Model Computation-Principles, Algorithms, Applications*, 2003.
11. C. Cremers and S. Mauw. *Operational semantics and verification of security protocols.* Springer, 2012.
12. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
13. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
14. D. J. Dougherty, J. D. Guttman, and J. D. Ramsdell. Homomorphisms and minimality for enrich-by-need security analysis. arXiv.org, Apr. 2018. `http://arxiv.org/abs/1804.07158`.
15. P. L. Erdös, D. Pálvölgyi, C. Tardif, and G. Tardos. Regular families of forests, antichains and duality pairs of relational structures. *Combinatorica*, 37(4):651–672, 2017.
16. S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V, FOSAD 2007–2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2009.
17. R. Fagin, P. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems (TODS)*, 30(1):174–210, 2005.
18. C. Fournet, A. Gordon, and S. Maffeis. A type discipline for authorization policies. In M. Sagiv, editor, *European Symposium on Programming*, volume LNCS No of *LNCS*. Springer Verlag, 2005.
19. A. D. Gordon and R. Pucella. Validating a web service security abstraction by typing. *Formal Asp. Comput.*, 17(3):277–318, 2005.
20. J. D. Guttman. Shapes: Surveying crypto protocol runs. In V. Cortier and S. Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, Cryptology and Information Security Series. IOS Press, 2011.
21. J. D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.
22. J. D. Guttman, J. C. Herzog, J. D. Ramsdell, and B. T. Sniffen. Programming cryptographic protocols. In R. D. Nicola and D. Sangiorgi, editors, *Trust in Global Computing*, number 3705 in LNCS, pages 116–145. Springer, 2005.

23. J. D. Guttman and F. J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, June 2002.
24. J. D. Guttman, F. J. Thayer, J. A. Carlson, J. C. Herzog, J. D. Ramsdell, and B. T. Sniffen. Trust management in strand spaces: A rely-guarantee method. In D. Schmidt, editor, *Programming Languages and Systems: 13th European Symposium on Programming*, number 2986 in LNCS, pages 325–339. Springer, 2004.
25. P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.
26. D. Jackson. *Software Abstractions.* MIT Press, 2 edition, 2012.
27. M. D. Liskov, P. D. Rowe, and F. J. Thayer. Completeness of CPSA. Technical Report MTR110479, The MITRE Corporation, Mar. 2011. `http://www.mitre.org/publications/technical-papers/completeness-of-cpsa`.
28. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.
29. W. McCune. MACE 2.0 Reference Manual and Guide. *CoRR*, 2001.
30. S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The Tamarin prover for the symbolic analysis of security protocols. In *Computer Aided Verification (CAV)*, pages 696–701, 2013.
31. T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. The Margrave Tool for Firewall Analysis. In *USENIX Large Installation System Administration Conference*, 2010.
32. T. Nelson, S. Saghafi, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Aluminum: Principled Scenario Exploration Through Minimality. In *Int. Conf. Soft. Eng.*, 2013.
33. T. Nelson, S. Saghafi, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Aluminum: Principled scenario exploration through minimality. In *35th International Conference on Software Engineering (ICSE)*, pages 232–241, 2013.
34. J. D. Ramsdell. Deducing security goals from shape analysis sentences. The MITRE Corporation, Apr. 2012. `http://arxiv.org/abs/1204.0480`.
35. J. D. Ramsdell and J. D. Guttman. CPSA4: A cryptographic protocol shapes analyzer, 2017. `https://github.com/ramsdell/cpsa`.
36. J. D. Ramsdell, J. D. Guttman, and M. Liskov. CPSA: A cryptographic protocol shapes analyzer, 2016. `http://hackage.haskell.org/package/cpsa`.
37. E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), Feb. 2010.
38. A. Reynolds, C. Tinelli, A. Goel, and S. Krstic. Finite Model Finding in SMT. In *Int. Conf. Computer Aided Verification*, 2013.
39. A. Robinson. *Handbook of Automated Reasoning*, volume 2. Elsevier, 2001.
40. B. Rossman. Homomorphism preservation theorems. *Journal of the ACM (JACM)*, 55(3):15, 2008.
41. P. D. Rowe, J. D. Guttman, and M. D. Liskov. Measuring protocol strength with security goals. *International Journal of Information Security*, February 2016. DOI 10.1007/s10207-016-0319-z, `http://web.cs.wpi.edu/~guttman/pubs/ijis_measuring-security.pdf`.
42. S. Saghafi, R. Danas, and D. J. Dougherty. Exploring theories with a model-finding assistant. In *25th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2015.
43. E. Torlak and D. Jackson. Kodkod: A Relational Model Finder. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2007.