Foundations of Logic-Based Trust Management

Moritz Y. Becker
Microsoft Research
Cambridge, UK

Alessandra Russo
Department of Computing
Imperial College, London, UK

Nik Sultana
Computer Laboratory
University of Cambridge, UK

***Abstract*—Over the last 15 years, many policy languages have been developed for specifying policies and credentials under the trust management paradigm. What has been missing is a formal semantics – in particular, one that would capture the inherently dynamic nature of trust management, where access decisions are based on the local policy in conjunction with varying sets of dynamically submitted credentials. The goal of this paper is to rest trust management on a solid formal foundation. To this end, we present a model theory that is based on Kripke structures for counterfactual logic. The semantics enjoys compositionality and full abstraction with respect to a natural notion of observational equivalence between trust management policies. Furthermore, we present a corresponding Hilbert-style axiomatization that is expressive enough for reasoning about a system's observables on the object level. We describe an implementation of a mechanization of the proof theory, which can be used to prove non-trivial meta-theorems about trust management systems, as well as analyze probing attacks on such systems. Our benchmark results show that this logic-based approach performs significantly better than the only previously available, ad-hoc analysis method for probing attacks.**

I. Introduction

Trust management [13] is an access control paradigm for decentralized systems that has attracted a lot of attention over the last 15 years. Research so far has focussed on concrete architectures and policy languages for trust management, and on policy analysis. This paper attempts to shed light on some of the more foundational aspects of trust management.

A. Trust Management

Trust management can be succinctly characterized by two distinctive features:

1) The access policy of the relying party is specified in a high-level *policy language* (e.g. [12], [46], [27], [38], [35], [40], [39], [24], [11], [10], [32], [6]).
2) Access decisions do not depend solely on the local policy, but also on digitally signed *credentials* that are submitted to the relying party together with the access request. Access is granted only if a *proof of compliance* can be constructed, showing that the requested permission Q is provable from the policy P combined with the set of credentials C.

The first feature effectively decouples the policy from the implementation of the enforcement mechanism, improving maintainability and flexibility in a context of quickly evolving access control requirements.

The second feature is necessitated by the fact that, in large decentralized systems, the relying party generally does not know the identity of the users requesting access in advance. Therefore, authorization has to be based on attributes rather than identity. Authority over these attributes may be delegated to trusted third parties, who may then issue credentials that assert these attributes or re-delegate authority to yet another party. The credentials that are used in trust management may thus be quite expressive, containing attributes, constraints and conditions, and delegation assertions. For this reason, the language for specifying credential assertions is typically the same as the one for specifying the local policy.

B. Trust Management Semantics

Given a derivability relation $\Vdash$ between sets of assertions and permissions, the basic mechanics of a trust management system can be specified as follows: a user's request Q is granted iff $P \cup C \Vdash Q$, where P is the relying party's local policy and C is the set of supporting credentials submitted by the user. All policy languages mentioned above can be specified in terms of such a derivability relation $\Vdash$; in the common case of Datalog-based policy languages, the relation $\Vdash$ is simply the standard Datalog entailment relation [21].

Hence we arrive at a natural notion of observational equivalence on policies that captures the essential aspects of trust management: two policies P and P' are equivalent iff for all sets C of credentials and all requests Q,

$$P \cup C \Vdash Q \iff P' \cup C \Vdash Q.$$

The fundamental question we are concerned with in this paper is whether an adequate model-theoretic semantics of trust management exists, i.e., one that matches this notion of observational equivalence. Neither the standard model-theoretic Datalog semantics based on minimal Herbrand models (for Datalog-based languages) nor the Kripke semantics for authorization logics related to ABLP [2] are adequate in this sense. While these semantics are sufficient for determining which permissions are granted by a *fixed* policy P and a *fixed* set C of supporting credentials, they do not provide any insight into questions that are particular to trust management, such as:

(a) Given the semantics of a policy P, which permissions Q are granted when P is combined with credential set C?
(b) Given the semantics of two policies P_1 and P_2, what is the semantics of their composition $P_1 \cup P_2$?
(c) What can an external user infer about an unknown policy merely by successively submitting requests together with varying sets of credentials and observing the relying party's responses?

C. Technical Contributions

We present the first formal trust management semantics that accurately captures the action of dynamically submitting

varying sets of credentials. It is compositional with respect to policy union and provides full abstraction [44] with respect to observational equivalence. These two properties together enable it to answer the questions (a) and (b) above.

Furthermore, we develop an axiomatization that is sound and complete with respect to the model-theoretic semantics, and provides inferentially complete object-level reasoning about a trust management system's observables. For example, judgements such as "*if a policy grants access to Q_1 when combined with set C_1, and denies access to Q_2 when combined with set C_2, then it must grant access to Q_3 when combined with C_3*" can be expressed as a formula in the logic, and be proved (or disproved) within it. It is this expressive power that enables the logic to directly answer questions such as (c) above, and thus to analyze *probing attacks*, a recently identified class of attacks in which the attacker infers confidential information by submitting credentials and observing the trust management system's reactions [32], [4], [8]. Perhaps even more strikingly, it is expressive enough to prove general *meta-theorems* about trust management systems, e.g. "if a policy satisfies some negation-free property, then this property will still hold when the policy is combined with an arbitrary credential set".

A language-independent semantics would be too abstract to provide any interesting insights. Our trust management semantics is specific to Datalog, and thus applicable to the wide range of Datalog-based policy languages. Datalog has arguably been the most popular logical basis for languages in this context; examples include Delegation Logic [38], SD3 [35], RT [40], [39], Binder [24], Cassandra [11], [10], and SecPAL [6].

The remainder of the paper is structured as follows. We introduce in Section II a simple language for reasoning about Datalog-based trust management policies, defined by a relation $\Vdash$, that captures the intuitive operational meaning of policies and credential submissions. This relation itself is straightforward, but, as we argue in Section III, universal truths (that hold for all policies) are both useful and highly non-trivial. This justifies the need for a logic with a formal semantics with a notion of validity that coincides with the intuitive notion of universal truths in trust management systems (Section IV). The corresponding axiomatization is presented in Section V. Section VI describes our implementation of a theorem prover for the logic. Applications and performance results are discussed in Section VII. We review related work in Section VIII and conclude with Section IX. The proofs of our theorems are lengthy; we relegate them to a technical report [9]. Our implementation is available at http://research.microsoft.com/counterdog.

II. A Simple Trust Management Language

We fix a countable set **At** of propositional variables called *atoms*.[1] A Datalog *clause* is either an atom p or of the form $p :- p_1, ..., p_n$, where $p, p_1, ..., p_n \in \mathbf{At}$. A *policy* γ is a finite set of clauses. We write Γ to denote the set of all policies.

Atoms correspond to atomic facts that are relevant to access control, e.g. "Alice can execute run.exe" or "Bob is a part-time student" or "the system is in state Red". From the point of view of the Datalog engine, the atoms have no inherent meaning beyond the logical dependencies specified within the policy (and the submitted credentials). It is the responsibility of the *reference monitor*, which acts as an interface between requesters and resources, to query the policy in a meaningful way. For instance, if Alice attempts to execute run.exe, the reference monitor would check if the corresponding atom `CanExec(Alice, run.exe)` is derivable from the policy in union with Alice's submitted credentials.

To specify when an atomic query $p \in \mathbf{At}$ is derivable from a policy γ, we introduce the relation symbol $\Vdash$:

$$\gamma \Vdash p \text{ iff } p \in \gamma \text{ or } \exists \vec{p} \subseteq_{\text{fin}} \mathbf{At} : (p :- \vec{p}) \in \gamma \ \wedge \ \forall p' \in \vec{p}.\ \gamma \Vdash p'. \tag{1}$$

We can straightforwardly extend $\Vdash$ to Boolean compound formulas φ, and the trivially true query:

$$\begin{aligned} &\gamma \Vdash \top. \\ &\gamma \Vdash \neg\varphi \text{ iff } \gamma \not\Vdash \varphi. \\ &\gamma \Vdash \varphi \wedge \varphi' \text{ iff } \gamma \Vdash \varphi \text{ and } \gamma \Vdash \varphi'. \end{aligned} \tag{2}$$

The relation $\gamma \Vdash \varphi$ may be read as "φ *holds in* γ".

It is the negated case where Datalog differs from classical logic: in the latter, $\neg p$ is entailed by a set of formulas γ only if p is false in *all* models of γ. In Datalog, on the other hand, only the *minimal* model of γ is considered. This fits in well with the decentralized security model, where knowledge is generally incomplete, and thus the absence of information should lead to fewer permissions.

The purpose of our language is not just to specify concrete policies, but to speak and reason about policy behaviors in a trust management context. In particular, recall that the outcome of queries is not just dependent on the service's policy alone, but also on the submitted *credentials*, which are also Datalog clauses. To express statements about such interactions, we introduce the notation $\Box_\gamma \varphi$, which informally means "if the set of credentials γ were submitted to the policy, then φ would be true". The policy is evaluated in union with the credentials, so we define

$$\gamma \Vdash \Box_{\gamma'} \varphi \text{ iff } \gamma \cup \gamma' \Vdash \varphi. \tag{3}$$

The full syntax of *formulas* in our trust management reasoning language is thus summarized by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_\gamma \varphi$$

We write Φ to denote the set of all formulas.

As usual, we define $\varphi \vee \varphi'$ as $\neg(\neg\varphi \wedge \neg\varphi')$, $\varphi \to \varphi'$ as $\neg\varphi \vee \varphi'$, and $\varphi \longleftrightarrow \varphi'$ as $(\varphi \to \varphi') \wedge (\varphi' \to \varphi)$. The unary operators $\Box$ and $\neg$ bind more tightly than the binary ones, and $\wedge$ and $\vee$ more tightly than $\to$ and $\longleftrightarrow$. Implication ($\to$) is right-associative, so we write $\varphi_1 \to \varphi_2 \to \varphi_3$ for $\varphi_1 \to (\varphi_2 \to \varphi_3)$.

[1] In practice, first-order predicates are used as atoms instead of propositional letters, but if the domain is finite, as is usually the case, the first-order case reduces to the propositional one. We choose the latter presentation for simplicity.

Example II.1. Let γ_0 be the policy $\{p :- q, r;\ \ p :- s;\ \ q :- p, t;\ \ q :- u\}$ (we use the semicolon as separator in clause sets, to avoid the ambiguity with the comma).

1) Without supporting credentials, no atom holds in γ_0:
$$\gamma_0 \Vdash \neg v, \text{ for all } v \in \mathbf{At}.$$
2) If u and r were submitted as supporting credentials, then p would hold in γ_0:
$$\gamma_0 \Vdash \Box_{\{u;\ r\}} p.$$
3) If credential s were submitted, and then t were submitted, then q would hold in γ_0:
$$\gamma_0 \Vdash \Box_{\{s\}}\Box_{\{t\}} q.$$
This is, of course, equivalent to submitting both at the same time: $\gamma_0 \Vdash \Box_{\{s;\ t\}} q$.
4) Submitted credentials may include non-atomic clauses:
$$\gamma_0 \Vdash \Box_{\{s :- q;\ u\}} p.$$

When are two policies (observationally) equivalent? Intuitively, they are equivalent if they both make the same set of statements true, under every set of submitted credentials. This notion can be formalized using the standard Datalog containment relation $\preceq$, as follows:

Definition II.2 (Containment, equivalence)**.** Let $\gamma_1, \gamma_2 \in \Gamma$. Then γ_1 *is contained in* γ_2 ($\gamma_1 \preceq \gamma_2$) iff for all finite $\vec{p} \subseteq \mathbf{At}$ and $p \in \mathbf{At}$:
$$\gamma_1 \cup \vec{p} \Vdash p \Rightarrow \gamma_2 \cup \vec{p} \Vdash p.$$

Two policies γ_1 and γ_2 are *equivalent* ($x \equiv y$) iff $\gamma_1 \preceq \gamma_2$ and $\gamma_2 \preceq \gamma_1$.

This definition may seem a bit narrow at first, but the following proposition shows that it actually coincides with the intuitive notion that exactly the same set of formulas (including $\Box$-formulas!) holds in two equivalent policies.

Proposition II.3. Let $\gamma_1, \gamma_2 \in \Gamma$.
$$\gamma_1 \equiv \gamma_2 \text{ iff } \forall \varphi \in \Phi.\ \gamma_1 \Vdash \varphi \Leftrightarrow \gamma_2 \Vdash \varphi$$

Example II.4.
1) $\emptyset \preceq \gamma$, for all $\gamma \in \Gamma$.
2) $\{a\} \preceq \{a;\ b\} \preceq \{a;\ b;\ c\}$
3) $\{a :- b, c\} \preceq \{a :- b\} \preceq \{a\}$
4) $\{a :- d;\ \ d :- b\} \equiv \{a :- b, c;\ \ a :- d;\ \ d :- b\}$.

III. Universal Truths

The relation $\Vdash$ from Section II is a straightforward specification of what a policy engine in a trust management system does. It is merely the standard Datalog evaluation relation extended with the $\Box$-operator for expressing the action of submitting supporting credentials. The relation is easy to evaluate (for a given γ and φ), and it directly reflects the intuition of the operational workings of a trust management system. So why should we bother developing a formal semantics that, as we shall see, is much more complex?

- A model-theoretic semantics lets us interpret and manipulate policies as mathematical objects in a syntax-independent way. It also provides additional insights into, and intuitions about, trust management systems.
- To prove that a formula is not a theorem, it is often easier to construct a counter-model (or in our case, a counter-world) than to work directly in the proof theory.
- The relation $\Vdash$ actually does not even provide a proof theory for formulas φ: it is of no help in answering the more interesting (but much harder) question if φ is *valid*, i.e., if holds in *all* policies γ. A formal semantics is the first step towards a corresponding proof theory.

The first two answers also apply to the question on the benefits of having a model-theoretic semantics for any logic in general. The third point is perhaps the most important from a practical perspective: in policy analysis, we are not mainly interested in the consequences of concrete policies and concrete sets of submitted credentials, but in *universal truths* φ that hold in *all* policies (or all policies that satisfy some properties).

Definition III.1. We write $\Vdash \varphi$ iff φ holds in all policies, i.e., $\forall \gamma \in \Gamma.\ \gamma \Vdash \varphi$.

The following examples illustrate that the reasoning techniques required in proving universal truths φ are beyond those directly provided by the definition of $\Vdash$.

Example III.2. If p is true in some policy when credential $q :- r$ is submitted, then p would also be true in the same policy if credential q were submitted:
$$\Vdash \Box_{\{q :- r\}} p \to \Box_{\{q\}} p$$
Intuitively, q is "more informative" than $q :- r$ (more formally, $\{q :- r\} \preceq \{q\}$), and providing more information can only lead to more (positive) truths, as Datalog is monotonic.

Example III.3. If submitting a and b individually is not sufficient for making c hold in some policy, but submitting both of them together *is* sufficient, then a cannot possibly hold in the policy:
$$\Vdash \neg\Box_{\{a\}} c \wedge \neg\Box_{\{b\}} c \wedge \Box_{\{a;\ b\}} c \to \neg a$$
For suppose a were true in the policy. Then submitting both a and b would be equivalent to submitting just b, but this contradicts the observation that submitting solely b does not make c true.

Example III.4. If a does not hold in some policy, and submitting d is not sufficient for making e hold, but submitting both credentials $b :- a$ and $d :- c$ *is* sufficient, then c must hold in that policy, and furthermore, a would hold if credential d were submitted:
$$\Vdash \neg a \wedge \Box_{\{d\}} \neg e \wedge \Box_{\{b :- a;\ d :- c\}} e \to c \wedge \Box_{\{d\}} a.$$
This small example is already too complex to explain succinctly by informal arguments, but it illustrates that reasoning about universal truths is far from trivial. We later present a formal proof of this statement in Example V.4.

A. Probing Attacks

There is a class of attacks on trust management systems called *probing attacks* [32], [4], [8], in which the attacker gains knowledge of secrets about the policy by submitting a series of access requests together with sets of supporting credentials, and by observing the system's reactions. Checking if a probing attack allows the attacker to infer a secret can be very complex, but it turns out that we can express probing attacks succinctly and directly as universal truths in our language.

Here is a simple (and naïve) example of a probing attack. A service S has a policy γ that includes the publicly readable rule

$$S.canRegister(x) :- x.hasConsented(S). \tag{4}$$

Informally, this should mean "S says that x can register with the service if x says (or has issued a credential saying) that he or she consents with S's terms and conditions". The service also exposes the query $S.canRegister(x)$ to any user x.

Suppose the user (and attacker) A self-issues a conditional credential

$$A.hasConsented(S) :- A.isRegistered(B), \tag{5}$$

which informally means "A says that A consents to S's terms and conditions, if A says that B is registered". A then submits this credential together with the query $S.canRegister(A)$, and observes that the answer is 'no'. From this single observation, she learns that neither $A.hasConsented$ nor $A.isRegistered(B)$ hold in γ – or else the query would have yielded the answer "yes". This is not very interesting so far, as she has only learnt about the falsity of statements made by herself.

But suppose she can also issue *delegation* credentials of the form $A.p :- D.p$. Such credentials are usually used to express delegation of authority; for example, to delegate authority over who is a student to university U, A would issue the credential $A.isStudent(x) :- U.isStudent(x)$. But here A abuses this mechanism by issuing the delegation credential

$$A.isRegistered(B) :- S.isRegistered(B). \tag{6}$$

Now she submits this credential together with the first conditional credential, and evaluates the same query. By observing the service's reaction to this second probe, and combining this with her previous observation, she then learns whether B is registered (according to S!) or not: the service's answer is "yes" iff $\gamma \Vdash S.isRegistered(B)$. She has thus *detected* a fact in γ that had nothing to do with the original query, and may well be confidential. Moreover, it is generally not possible to protect against probing attacks without crippling the intended policy using simple syntactic input sanitization or by enforcing strict non-interference (see [4] for details).

We now show how this attack can be expressed as a universal truth. Let c_1 and c_2 be the credentials 5 and 6, respectively. A's knowledge about the public clause (4) in the policy translates into

$$\varphi_1 = \Box_{\{A.hasConsented(S)\}} S.canRegister(A).$$

Her first observation is translated into

$$\varphi_2 = \Box_{\{c_1\}} \neg S.canRegister(A),$$

and the second observation into

$$\varphi_3 = \Box_{\{c_1,c_2\}} S.canRegister(A) \text{ or}$$
$$\varphi_3' = \Box_{\{c_1,c_2\}} \neg S.canRegister(A),$$

depending on the service's reaction. Then the following holds:

$$\Vdash \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \rightarrow S.isRegistered(B)$$
$$\Vdash \varphi_1 \wedge \varphi_2 \wedge \varphi_3' \rightarrow \neg S.isRegistered(B)$$

We will later present a logic that can prove such statements, and thus can also be used to reason about probing attacks (see Example V.5).

Note that Examples III.3 and III.4 can also be interpreted as probing attacks. For instance, in Example III.4, let us assume that e is the only query publicly exposed by the service, and the attacker initially only knows that a does not hold in the service's policy. The attacker possesses three authenticated credentials: d and $b :- a$ and $d :- c$. By submitting first d together with the query e, and after that $\{b :- a;\ d :- c\}$ together with the same query, and by observing the service's reactions to these two *probes*, the attacker detects (provided she is sufficiently clever) that $c \wedge \Box_{\{d\}} a$ holds in the policy. Depending on the circumstances, this may constitute a breach of secrecy.

We can succinctly define the notions of probes, probing attack, detectability and opacity from [4], [8] in our language.

Definition III.5. A *probe* π is a formula of the form $\Box_\gamma \psi$, where $\gamma \in \Gamma$ is called the *probe credential set* and ψ is a $\Box$-free formula from Φ called the *probe query*.

An *observation* of a probe π under a policy γ_0 is either π if $\gamma_0 \Vdash \pi$, and otherwise $\neg\pi$.

A *probing attack* on γ_0 consisting of probes $\{\pi_1, ..., \pi_n\}$ is the conjunction of the observations of $\pi_i \in \{\pi_1, ..., \pi_n\}$ under γ_0.

Clearly, by the above definition, if φ is a probing attack on γ_0, then $\gamma_0 \Vdash \varphi$. But there may be other policies γ that also have the property that φ holds in them. In the absence of other additional knowledge, the attacker cannot distinguish between γ_0 and any such γ. To put it positively, the attacker learns from the probing attack φ precisely that γ_0 is in the equivalence class of policies in which φ holds. We denote this equivalence class induced by probing attack φ by $|\varphi| = \{\gamma \mid \gamma \Vdash \varphi\}$.

Now if in all these policies, some property φ' holds, then the attacker *knows* with absolute certainty that φ' holds in γ_0 in particular, in which case we say that φ' is detectable. Conversely, if there exists some policy within $|\varphi|$ in which φ' does *not* hold, the attacker cannot be certain that φ' holds in γ_0, in which case we say that φ' is *opaque*.

Definition III.6 (Detectability, opacity)**.** A formula $\varphi' \in \Phi$ is *detectable* in a probing attack φ on a policy γ_0 iff

$$\forall \gamma \in |\varphi|.\ \gamma \Vdash \varphi'.$$

A formula φ' is *opaque* in a probing attack φ iff it is not detectable in φ, or equivalently,

$$\exists\gamma \in |\varphi|.\ \gamma \not\Vdash \varphi'.$$

Theorem III.7 (Probing attacks)**.** A formula φ' is detectable in a probing attack φ iff $\Vdash \varphi \rightarrow \varphi'$.

This theorem again underlines the importance of being able to reason about universal truths.

IV. Semantics

The model-theoretic semantics we are looking for has to satisfy four requirements:

1) Capturing trust management: given φ and the semantics of γ, it is possible to check if $\gamma \Vdash \varphi$.
2) Supporting a notion of validity: φ is valid (in the model theory) iff $\Vdash \varphi$.
3) Full abstraction [44]: two policies are equivalent ($\equiv$) iff their respective semantics are equal.
4) Compositionality: the semantics of $\gamma_1 \cup \gamma_2$ can be computed from the individual semantics of γ_1 and γ_2.

A. Naïve Approaches

We first consider some simple approaches to developing a formal semantics that may immediately come to mind, and show why they fail.

The standard model-theoretic interpretation of a set of Datalog clauses is its minimal Herbrand model, i.e., the set of atoms that hold in it. But in this approach, the policy γ_0 from Example II.1 would have the same semantics as the empty policy $\emptyset$, namely the empty model, even though the two policies are clearly not equivalent (Def. II.2). Hence such a semantics would not be fully abstract. This semantics is not compositional either: from the semantics of $\{p :- q\}$ (which is again empty) and of $\{q\}$, we cannot construct the semantics of their union. Therefore, this semantics is clearly unsuitable in a trust management context, where it is common to temporarily extend the clause set with a set of credentials. In fact, this semantics fails on all four accounts regarding our requirements.

We could also attempt to interpret a Datalog clause $p :- p_1, ..., p_n$ as an implication $p_1 \wedge ... \wedge p_n \rightarrow p$ in classical (or intuitionistic) logic, and a policy γ as the conjunction of its clauses: $[\![\gamma]\!] = \bigwedge_{c\in\gamma}[\![c]\!]$. As shown by Gaifman and Shapiro ([28]), this semantics would indeed be both compositional and fully abstract. However, this interpretation does not correctly capture the trust management relation $\Vdash$, as we show now. First of all, we would need to translate $\Box$-formulas into logic. The obvious way of doing this would be to interpret $\Box_\gamma\varphi$ as the implication $[\![\gamma]\!] \rightarrow [\![\varphi]\!]$. Then, for instance, we have $\{p :- q\} \Vdash \Box_q p$, and correspondingly also $[\![\{p :- q\}]\!] \models [\![\Box_q p]\!]$, since $[\![\{p :- q\}]\!] = [\![\Box_q p]\!] = q \rightarrow p$. Thus we might be led to conjecture

$$\gamma \Vdash \varphi \overset{?}{\Longleftrightarrow} [\![\gamma]\!] \models [\![\varphi]\!].$$

Unfortunately, this correspondence does not hold in general. Consider the formula $\varphi = \neg q \wedge \Box_q p$. Then $[\![\varphi]\!] = \neg q \wedge (q \rightarrow p)$. But $\{p :- q\} \Vdash \varphi$, whereas $[\![\{p :- q\}]\!] \not\models [\![\varphi]\!]$. We could try to fix this by only considering the minimal model of the semantics, since $\mathbf{minMod}([\![\{p :- q\}]\!]) \models \neg q$. But we can break this again: $\emptyset \not\Vdash \varphi$, whereas $\mathbf{minMod}([\![\emptyset]\!]) \models [\![\varphi]\!]$.

B. A counterfactual Kripke semantics

The crucial observation that leads to an adequate semantics is that both Datalog clauses and the trust management specific $\Box$-actions are *counterfactual*, rather than implicational, in nature. For instance, $p :- \vec{p}$ can be interpreted as the counterfactual "if $\vec{p}$ were added to the policy, then p would hold". Similarly, $\Box_\gamma\varphi$ can be read as "if γ were added to the policy, then φ would hold". (Note that the counterfactual conditional "if A were true then B would hold" is strictly stronger than the material implication "A $\rightarrow$ B", which vacuously holds whenever A is not true.)

Therefore, we can unify the notations and write $\Box_{\vec{p}}p$ instead of $p :- \vec{p}$. Moreover, instead of writing a policy γ as a set, we can just as well write it as a conjunction of clauses. We can thus rewrite the syntax for policies and formulas from Section II in the following, equivalent, form:

$$\begin{array}{ll} \text{Policies} & \gamma ::= \top \mid p \mid \Box_{\bigwedge \vec{p}}p \mid \gamma \wedge \gamma \\ \text{Formulas} & \varphi ::= \gamma \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_\gamma\varphi \end{array}$$

As before, we write Γ and Φ to denote the set of all policies and formulas, respectively. The relation $\Vdash$ is also defined as before, with the obvious adaptations to the new syntax.

Notation IV.1. Henceforth, we treat $\vec{\varphi}$ as syntactic sugar for $\bigwedge \vec{\varphi}$, and $p :- \vec{p}$ for $\Box_{\vec{p}}p$.

Interpreting $\Box$-formulas as counterfactuals, we can now give it a multi-modal Kripke semantics in the spirit of Lewis and Stalnaker [37], [50]: the counterfactual $\Box_\gamma\varphi$ holds in a possible world w if in those γ-satisfying worlds w' that are *closest* to w, φ holds. We will express the closeness relation using a ternary accessibility relation R, and later apply rather strong conditions on R in order to make it match the intended trust management context.

Definition IV.2 (Model, entailment)**.** A *model* M is a triple $\langle W, R, V\rangle$, where W is a set, $R \subseteq \wp(W) \times W \times W$, and $V : \mathbf{At} \rightarrow \wp(W)$.

Given a model M, we inductively define the model-theoretic entailment relation $\Vdash_M \subseteq W \times \Phi$ as follows. For all $w \in W$:

$$\begin{array}{l} w \Vdash_M \top \\ w \Vdash_M p \text{ iff } w \in V(p) \\ w \Vdash_M \neg\varphi \text{ iff } w \not\Vdash_M \varphi \\ w \Vdash_M \varphi_1 \wedge \varphi_2 \text{ iff } w \Vdash_M \varphi_1 \text{ and } w \Vdash_M \varphi_2 \\ w \Vdash_M \Box_\gamma\varphi \text{ iff } \forall w'.\ R_{|\gamma|_M}(w, w') \Rightarrow w' \Vdash_M \varphi, \end{array}$$

where $|\gamma|_M = \{w \in W \mid w \Vdash_M \gamma\}$. Similarly, we write $|w|_M$ to denote the set $\{\gamma \in \Gamma \mid w \Vdash_M \gamma\}$.

Intuitively, a world $w \in W$ corresponds to a policy; more precisely, to the $\preceq$-maximal policy in $|w|_M$. Vice versa, a policy γ corresponds to a world, namely the $\preceq_M$-minimal

world in $|\gamma|_M$, where $\preceq_M$ is an ordering on worlds that reflects the containment relation $\preceq$ on policies (Def. IV.3). (Actually, in Def. IV.4, we associate γ simply with the entire cone $|\gamma|_M$.)

Definition IV.3 (World containment)**.** Given a model $M = \langle W, R, V\rangle$ and $x, y \in W$,

$$x \preceq_M y \text{ iff } \forall \gamma \in \Gamma : x \Vdash_M \gamma \text{ implies } y \Vdash_M \gamma.$$

Definition IV.4 (Semantics)**.** The *semantics* of γ (with respect to M) is $|\gamma|_M$.

As it is, this definition keeps the meaning of R completely abstract, but we can already prove that the semantics is compositional, irrespective of R:

Theorem IV.5 (Compositionality)**.** For all models M, and $\gamma_1, \gamma_2 \in \Gamma$:

$$|\gamma_1 \wedge \gamma_2|_M = |\gamma_1|_M \cap |\gamma_2|_M.$$

In order to satisfy the remaining three requirements from Section II, we have to put some restrictions on the models, and in particular on the accessibility relation R. We call models that satisfy these constraints *TM models* (Def. IV.7). Intuitively, $R_{|\gamma|_M}(w, w')$ should hold if w' is a world that is closest to w of those worlds in which γ holds. But what do we mean by 'closest'? If we interpret worlds as policies, then w' is the policy that results from *adding* γ, and *nothing more but* γ, to w. So we have to consider all worlds that are larger than w (since we are adding to w) and also satisfy γ, and of these worlds we take the $\preceq_M$-minimal ones (since we are adding nothing more but γ) (Def. IV.7(1)).

The other two constraints (Def. IV.7(2) and IV.7(3)) ensure that there is a one-to-one correspondence between policies and worlds.

Definition IV.6. If $(X, \leq)$ is a pre-ordered set ($\leq$ is a reflexive transitive relation on X) and Y a finite subset of X, then $\mathbf{min}_{\leq}(Y) = \{y \in Y \mid \forall y' \in Y : y' \not< y\}$, and $\mathbf{max}_{\leq}(Y) = \{y \in Y \mid \forall y' \in Y : y' \not> y\}$.

Definition IV.7 (Trust management model)**.** A model $M = \langle W, R, V\rangle$ is a *TM model* iff

1) $\forall \gamma \in \Gamma,\ x, y \in W.$
   $R_{|\gamma|_M}(x, y)$ iff $y \in \mathbf{min}_{\preceq_M}\{w \mid w \succeq_M x \wedge w \in |\gamma|_M\}$,
2) $\forall \gamma \in \Gamma,\ \exists w \in W.\ \gamma \in \mathbf{max}_{\preceq}|w|_M$, and
3) $\forall w \in W,\ \exists \gamma \in \Gamma.\ \gamma \in \mathbf{max}_{\preceq}|w|_M$.

To gain a better intuition for TM models, it is useful to consider the following, particular TM model: imagine a labeled directed graph with a vertex for each $\gamma \in \Gamma$ (these are the worlds W). There is an edge from γ_1 to γ_2, labeled with γ, whenever $\gamma_2 = \gamma_1 \cup \gamma$ (corresponding to the accessibility relation $R_{|\gamma|}$).

So a TM model models all possible policies and all possible trust management interactions (submitting a set of credentials γ for the duration of a query) with these policies. The following theorem shows that TM models indeed precisely capture the trust management relation $\Vdash$, and Theorem IV.9 states that the semantics is fully abstract.

Theorem IV.8 (Capturing trust management)**.** Let $M = \langle W, R, V\rangle$ be a TM model, $\gamma \in \Gamma$ and $\varphi \in \Phi$.

$$\gamma \Vdash \varphi \text{ iff } \forall w \in \mathbf{min}_{\preceq_M}|\gamma|_M.\ w \Vdash_M \varphi$$

Theorem IV.9 (Full abstraction)**.** For all TM models M, and $\gamma_1, \gamma_2 \in \Gamma$:

$$\gamma_1 \equiv \gamma_2 \text{ iff } |\gamma_1|_M = |\gamma_2|_M.$$

The property that is hardest to satisfy (and to prove) is the requirement that the model theory should support a notion of validity that coincides with judgements of the form $\Vdash \varphi$, i.e., universal truths about trust management policies. This is formalized in Theorem IV.11.

Definition IV.10 (Trust management validity)**.** φ is *TM-valid* (we write $\Vdash_{\text{TM}} \varphi$) iff for all TM models $M = \langle W, R, V\rangle$ and $w \in W$: $w \Vdash_M \varphi$.

Theorem IV.11 (Supporting validity)**.**

$$\Vdash_{\text{TM}} \varphi \text{ iff } \Vdash \varphi.$$

Example IV.12. Consider the following (false) statement: "in all policies in which $p \to q$ holds, $\Box_p q$ also holds." By the contrapositive of Theorem IV.11, we can prove that this is not true, i.e., $\not\Vdash (p \to q) \to \Box_p q$, by identifying a counter-world w in a TM model M such that $w \Vdash_M (p \to q) \wedge \neg\Box_p q$. By Def. IV.2, this is equivalent to

$$w \Vdash_M \neg p \wedge \neg\Box_p q \text{ or } w \Vdash_M q \wedge \neg\Box_p q.$$

Let w be a $\preceq_M$-minimal world in all of W. By minimality, $w \Vdash_M \gamma$ only if γ is universally true. Neither p nor $\Box_p q$ (assuming $p \neq q$) are universally true, hence $w \Vdash_M \neg p$ and $w \Vdash_M \neg\Box_p q$, as required.

In this section, we developed an adequate model-theoretic semantics for trust management. We started by interpreting both Datalog clauses and trust management interactions as counterfactuals, and taking a generic counterfactual model theory as the basis. We then customized the theory by adding constraints on the models of interest to arrive at TM models. The resulting semantics satisfies all four requirements from Section II, and it provides an intuition of a trust management service as a vertex in a labeled directed graph, where the reachable vertices represent the clause sets resulting from combining the service's policy with the submitted credential set (the edge label) to the service.

However, this semantics still does not give us much insight into proving judgements of the form $\Vdash \varphi$ (or, equivalently, $\Vdash_{\text{TM}} \varphi$). For this purpose, we equip the model theory with a corresponding proof theory in the following section.

V. Axiomatization

In standard modal logic, it is usually straightforward to derive an axiom in the proof theory from each frame condition in the model theory, i.e., a restriction on the accessibility relation R. (For example, reflexivity of R corresponds to the

axiom $\Box\varphi \to \varphi$.) This constructive method can also be applied to counterfactual multi-modal logic, if the frame conditions are relatively simple [49]. In our case, however, the restriction on R (Def. IV.7(1)) is too complex to be simply 'translated' into an axiom. The axiomatization presented below was actually conceived by guessing the axioms and rules, and adjusting them until the system was provably sound and complete with respect to the model theory.

Definition V.1. In the proof system below, let $\varphi, \varphi', \varphi'' \in \Phi$, $\gamma, \gamma' \in \Gamma$, $p \in \mathbf{At}$ and $\vec{p} \subseteq \mathbf{At}$. The proof system consists of the following axiom schemas:

$$\vdash \varphi \to \varphi' \to \varphi \tag{Cl1}$$
$$\vdash (\varphi \to \varphi' \to \varphi'') \to (\varphi \to \varphi') \to \varphi \to \varphi'' \tag{Cl2}$$
$$\vdash (\neg\varphi \to \neg\varphi') \to \varphi' \to \varphi \tag{Cl3}$$
$$\vdash \Box_\gamma(\varphi \to \varphi') \to \Box_\gamma\varphi \to \Box_\gamma\varphi' \tag{K}$$
$$\vdash \Box_\gamma\gamma \tag{C1}$$
$$\vdash \Box_\gamma\varphi \to \gamma \to \varphi \tag{C2}$$
$$\vdash \Box_{(p\,:\!-\,\vec{p})}\ \varphi \to (\vec{p} \to p) \to \varphi \quad \text{provided } \varphi \text{ is } \Box\text{-free} \tag{Dlog}$$
$$\vdash \Box_\gamma\neg\varphi \longleftrightarrow \neg\Box_\gamma\varphi \tag{Fun}$$
$$\vdash \Box_{\gamma\wedge\gamma'}\varphi \longleftrightarrow \Box_\gamma\Box_{\gamma'}\varphi \tag{Perm}$$

Additionally, there are three proof rules:

$$\text{If } \vdash \varphi \text{ and } \vdash \varphi \to \varphi' \text{ then } \vdash \varphi'. \tag{MP}$$
$$\text{If } \vdash \varphi \text{ then } \vdash \Box_\gamma\varphi. \tag{N}$$
$$\text{If } \vdash \gamma \to \gamma' \text{ and } \varphi \text{ is } \neg\text{-free then } \vdash \Box_{\gamma'}\varphi \to \Box_\gamma\varphi \tag{Mon}$$

Axioms (Cl1)–(Cl3) and Modus Ponens (MP) are from the Hilbert-style axiomatization of classical propositional logic [48]. It is easy to see that they are sound, irrespective of R, since the Boolean operators $\top$, $\wedge$ and $\neg$ are defined classically for $\Vdash_M$. Axiom (K) is the multi-modal version of the basic Distribution Axiom that is part of every modal logic ($\Box(\varphi \to \varphi') \to \Box\varphi \to \Box\varphi'$). Similarly, Rule (N) is the multi-modal version of the basic Necessitation Rule (if $\vdash \varphi$ then $\vdash \Box\varphi$).

Axioms (C1) and (C2) are also standard in counterfactual logic [49]. The former is the trivial statement that if γ were the case, then γ would hold. The latter states that the counterfactual conditional is stronger than material implication.

At first sight, Axiom (Dlog) may look similar to Axiom (C2), but the two are actually mutually independent. In fact, while the latter is standard, Axiom (Dlog) is deeply linked with the intuition that the possible worlds correspond to Datalog policies. Recall that, intuitively, the left hand side means "φ would hold in the policy if the credential $p\,:\!-\,\vec{p}$ were submitted". Now we expand the right hand side of the implication to

$$(\vec{p} \wedge \neg p) \vee \varphi.$$

So the axiom tells us that the left hand side holds only if it is the case that

- either φ holds in the policy anyway, even without submitting $p\,:\!-\,\vec{p}$,
- or the action of submitting the credential must be crucial for making φ true, but this is only possible if the conditions $\vec{p}$ of the credential are all satisfied in the policy, and furthermore p does not already hold in the policy (or else the credential could not possibly be crucial).

But the axiom only holds for $\Box$-free φ. To see why, consider the following instance of Axiom (Dlog), ignoring the side condition: $\Box_{q\,:\!-\,p}\Box_p q \to (p \to q) \to \Box_p q$. The left hand side is an instance of Axiom (C1), since $q\,:\!-\,p$ is just syntactic sugar for $\Box_p q$, so the formula simplifies to $(p \to q) \to \Box_p q$, which is not TM valid, as shown in Example IV.12.

Ax. (Dlog) can be made bidirectional by strengthening the right hand side:

Lemma V.2.

$$\vdash \Box_{(p\,:\!-\,\vec{p})}\varphi \longleftrightarrow \varphi \vee (\neg p \wedge \vec{p} \wedge \Box_p\varphi),$$

provided that φ is $\Box$-free.

Axiom (Fun) is also remarkable in that it is rather non-standard in modal logic. It is also the reason it is not useful to define a dual $\Diamond$-operator (i.e., $\Diamond_\gamma\varphi = \neg\Box_\gamma\neg\varphi$) in our logic, since $\Box$ and $\Diamond$ would be equivalent. The axiom is equivalent to the property that the accessibility relation R in a TM model $M = \langle W, R, V\rangle$ is essentially functional, i.e., for all $w \in W$, and $\gamma \in \Gamma$:

- $\exists w'.\ R_{|\gamma|_M}(w, w')$, and
- $\forall w_1, w_2.\ R_{|\gamma|_M}(w, w_1) \wedge R_{|\gamma|_M}(w, w_2) \Rightarrow w_1 \preceq_M w_2 \wedge w_2 \preceq_M w_1$.

On the intuitive Datalog level, Axiom (Fun) can easily be seen to be sound, since the statement "φ would not hold if γ were submitted" is equivalent to "it is not the case that φ would hold if γ were submitted".

Axiom (Perm) also corresponds to a property of R, namely that it is transitive (that's the 'if' direction) and dense (the 'only if' direction). It captures the intuition that submitting two credential sets in sequence is equivalent to just submitting their union.

Rule (Mon) expresses a monotonicity property on the subscripts of $\Box$, and can be reduced to a monotonicity property of TM models and $\neg$-free φ:

$$\forall w, w' \in W.\ w \Vdash_M \varphi \wedge w' \succeq_M w \Rightarrow w' \Vdash_M \varphi.$$

The intuition here is that submitting more or stronger credentials can only make more (positive) facts true. It is easy to see that this does not hold in general for negated statements: suppose p does not hold in a policy (with no submitted credentials); then the negated fact $\neg p$ holds. But $\neg p$ may cease to hold when credentials are submitted, in particular, when p is submitted. In other words, even though $p \to \top$ is valid, $\Box_\top\neg p \to \Box_p\neg p$ is not.

The main result of this section is that the axiomatization is sound and complete with respect to the model theory (Theorem V.3).

Theorem V.3 (Soundness and Completeness)**.**

$$\Vdash_{\text{TM}} \varphi \text{ iff } \vdash \varphi$$

The proof of soundness ($\vdash \varphi$ implies $\Vdash_{\text{TM}} \varphi$) formalizes the intuitions given above and proceeds, as usual, by structural induction on φ. The proof of completeness ($\Vdash_{\text{TM}} \varphi$ implies $\vdash \varphi$) is less standard, and can be roughly outlined thus:

1) We will prove the equivalent statement that if φ consistent (with respect to $\vdash$), then there exists a TM model $M = \langle W, R, V \rangle$ and $w \in W$ such that $w \Vdash_M \varphi$.
2) From Lemma V.2, it can be shown that every φ is equivalent to a formula φ' that only consists of conjunctions and negations of policies in Γ (i.e., one that does not contain vertically nested boxes).
3) Based on the property of TM models that every world corresponds to some policy in Γ, it is then possible to identify $w \in W$ such that $w \Vdash_M \varphi'$, whenever M is a TM model.
4) By soundness, this implies that $w \Vdash_M \varphi$. Furthermore, we can show that at least one TM model exists, and hence we arrive at the required existential conclusion.

Together with Theorem IV.11, we have the result

$$\Vdash \varphi \iff \Vdash_{\text{TM}} \varphi \iff \vdash \varphi.$$

We can thus use the axiomatization to prove universal truths about trust management systems.

Example V.4. We sketch a formal proof of the formula from Example III.4.

$$\vdash \neg a \wedge \Box_d \neg e \wedge \Box_{b\,:-\,a \wedge d\,:-\,c} e \rightarrow c \wedge \Box_d a$$

Proof: We first show that d is equivalent to $\Box_\top d$. The direction $\vdash \Box_\top d \rightarrow d$ follows directly from Axiom (C2). The same axiom also yields $\vdash \Box_\top \neg d \rightarrow \neg d$, the contrapositive of which is $\vdash d \rightarrow \Box_\top d$, together with Axiom (Fun). Therefore $\vdash d \longleftrightarrow \Box_\top d$.

Since $\vdash c \rightarrow \top$, we have $\vdash \Box_\top d \rightarrow \Box_c d$, according to Rule (Mon), and hence equivalently $\vdash d \rightarrow d\,:-\,c$. Taking this as the premise of Rule (Mon), we get $\vdash \Box_{d\,:-\,c} e \rightarrow \Box_d e$, the contrapositive of which is $\vdash \Box_d \neg e \rightarrow \Box_{d\,:-\,c} \neg e$, by Axiom (Fun).

Therefore, the assumption $\Box_d \neg e$ from the antecedent of the formula implies $\Box_{d\,:-\,c} \neg e$. Conjoining this with the assumption $\Box_{b\,:-\,a \wedge d\,:-\,c} e$, which is equivalent to $\Box_{d\,:-\,c} \Box_{b\,:-\,a} e$, by Axiom (Perm), we get

$$\Box_{d\,:-\,c} (\neg e \wedge \Box_{b\,:-\,a} e) \tag{7}$$

(as it can be easily shown that $\Box_{d\,:-\,c}$ distributes over $\wedge$).

By Axiom (Dlog), $\vdash \Box_{b\,:-\,a} e \rightarrow e \vee (a \wedge \neg b)$. Therefore, formula (7) implies

$$\Box_{d\,:-\,c} (a \wedge \neg b), \tag{8}$$

since Axiom (K) allows us to apply Modus Ponens under $\Box_{d\,:-\,c}$. We have thus shown that the antecedent of the original formula implies $\Box_{d\,:-\,c} a$. Furthermore, as we have shown, $\vdash d \rightarrow d\,:-\,c$, and hence by Rule (Mon), $\vdash \Box_{d\,:-\,c} a \rightarrow \Box_d a$. Modus Ponens yields one of the consequents of the original formula, $\Box_d a$.

For the other consequent, c, we apply Axiom (Dlog) to formula (8), which yields $(a \wedge \neg b) \vee (c \wedge \neg d)$. Combining this with the antecedent $\neg a$, we can then conclude c. ∎

Example V.5. We sketch a formal proof of the probing attack result from Section III-A. For brevity, we introduce abbreviated names for the atoms:

$$as = A.hasConsented(S)$$
$$sa = S.canRegister(A)$$
$$ab = A.isRegistered(B)$$
$$secret = S.isRegistered(B)$$

The statement that the attacker can detect $secret$ in the probing attack can then be expressed as

$$\vdash \Box_{as} sa \wedge \Box_{as\,:-\,ab} \neg sa \wedge \Box_{as\,:-\,ab \wedge ab\,:-\,secret} sa \rightarrow secret.$$

Proof: Assume the left hand side of the formula that we want to prove. From the previous proof, we have seen that sa is equivalent to $\Box_\top sa$. Since $as\,:-\,ab \rightarrow \top$, we thus have $sa \rightarrow \Box_{as\,:-\,ab} sa$, by Rule (Mon). Combining the contrapositive of this with the assumption, we get $\neg sa$. From the assumption and Ax. (C2), we get $as \rightarrow sa$, which together with $\neg sa$ gives $\neg as$.

Using Lemma (V.2), we can prove that $\Box_{as\,:-\,ab} sa$ is equivalent to $sa \vee (ab \wedge \neg as \wedge \Box_{as} sa)$.

Since the assumption $\Box_{as\,:-\,ab} \neg sa$ is equivalent to $\neg \Box_{as\,:-\,ab} sa$ (by Ax. (Fun)), it is therefore also equivalent to

$$\neg sa \wedge (\neg ab \vee as \vee \neg \Box_{as} sa).$$

We have already proved $\neg as$, and $\Box_{as} sa$ is in the antecedent. Therefore, we can conclude $\neg ab$.

Now consider $\Box_{as\,:-\,ab} \neg sa \wedge \Box_{as\,:-\,ab \wedge ab\,:-\,secret} sa$ in the assumption. By Ax. (Fun) and (Perm) and distributivity of $\Box$, this is equivalent to $\Box_{as\,:-\,ab} (\neg sa \wedge \Box_{ab\,:-\,secret} sa)$. By Ax. (K), we can apply Ax. (Dlog) on the inner box under the outer box to get

$$\Box_{as\,:-\,ab} (\neg sa \wedge (sa \vee (secret \wedge \neg ab))),$$

which implies $\Box_{as\,:-\,ab} secret$. Again applying Ax. (Dlog) yields $secret \vee (ab \wedge \neg as)$. But since we have proved $\neg ab$ above, we can conclude that $secret$ follows from the assumptions. ∎

VI. Mechanizing the Logic

Hilbert-style axiomatizations are notoriously difficult to use directly for building proofs, and they are also difficult to mechanize directly, because they are not goal-directed. In this section, we describe how a goal formula φ can be transformed into an equivalent formula in classical propositional logic that can be verified by a standard SAT solver. We have implemented a tool based on the contents of this section; some uses of the tool are described in Section VII.

Our axiomatization has certain characteristics that enables such a transformation. Firstly, Lemma V.2 shows that φ can be transformed into a formula in which all subscripts of boxes are $\Box$-free, and Ax. (Fun) and (Perm) allow us to distribute boxes through conjunctions, disjunctions and negations. This forms the basis of a *normalization* transform.

Secondly, for a given φ, it is sufficient to encode just a finite number of axiom instantiations in classical propositional logic in order to characterize the non-classical properties of $\Box$. This process is called *saturation*.

In this section, we use *literal* to mean a (possibly negated) atom, and $\Box$-*literal* to mean a (possibly negated) atom with some prefix of boxes, e.g. $\Box_{\Box_r p \wedge p} q$ and p are both $\Box$-literals (p is logically equivalent to $\Box_\top p$), whereas p is also a literal but $\Box_q p$ is not.

The reasoning process is described in more detail next.

Normalization and expansion. Following parsing, the goal formula is simplified through the elimination of subsumed subformulas; e.g., $\Box_{a :- b} c \wedge \Box_a c$ is simplified to $\Box_{a :- b} c$. The formula is then *normalized* by computing a negation normal form and distributing all boxes, such that boxes are only applied to literals, and negation is only applied to $\Box$-literals. We also use Ax. (Perm) to collect strings of boxes into a single box. Normalization takes care of Ax. (K), (Fun), and (Perm).

Next, the goal formula is *expanded* by applying Lemma V.2 exhaustively until all subscripts of $\Box$-literals are $\Box$-free. Expansion is a very productive process – it can cause the goal formula's size to increase exponentially. This step takes care of Ax. (Dlog) and Rule (N).

The resulting formula is negated and added to the *clause set*. The clause set collects formulas which will ultimately be passed to a SAT solver.

Saturation. Saturation generates propositional formulas that faithfully characterize the $\Box$-literals occurring in the clause set.

1) Let $\beta = \Box_{\bigwedge_{i=1}^n (q_i :- \vec{q_i})} p$ be a $\Box$-literal occurring in the clause set. If $\vdash \bigwedge_{i=1}^n (\vec{q_i} \to q_i) \to p$ holds (which is checked by the underlying SAT solver), we replace all occurrences of β by $\top$. This step is a generalization of Ax. (C1).
2) For each $\Box$-literal $\Box_\gamma p$ (where $\gamma \neq \top$) occurring in the clause set, we add the formulas $p \to \Box_\gamma p$ and $\Box_\gamma p \to \gamma \to p$.
3) For each pair of $\Box$-literals $\Box_{\gamma_1} p$, $\Box_{\gamma_2} p$ (where $\gamma_1 \neq \gamma_2$) occurring in the clause set, we add the formula

$$\Box_{\gamma_1} \gamma_2 \wedge \Box_{\gamma_2} p \longrightarrow \Box_{\gamma_1} p.$$

Intuitively, this formula encodes the transitivity of counterfactuals. Steps (2) and (3) together cover Ax. (C2) and Rule (Mon).

Since the second step may create new $\Box$-literals, the process is repeated until a fixed point is reached.

Propositionalization and SAT solving. After saturation completes, all $\Box$-literals in the clause set are uniformly substituted by fresh propositional literals. The resulting formulas are then checked by a standard SAT solver. Our implementation offers the choice between using the in-memory API of Z3[2] [23] and producing output in the DIMACS [25] format used by many SAT solvers such as MiniSAT [26].

The classical axioms (Cl1)–(Cl3) and Rule (MP) are covered by the SAT solver. We have therefore covered all axioms and rules, and thus the goal formula is valid iff the SAT solver reports unsatisfiability (since we negated the goal).

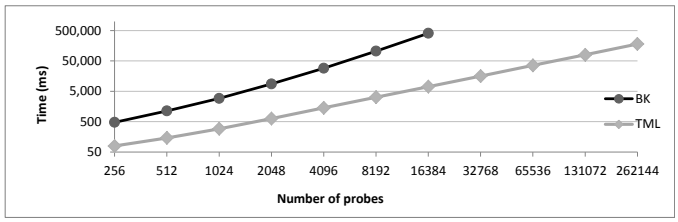

Figure 1. Comparison of timings for the TC3-based test series on a double logarithmic scale. BK is the tool from [7], TML is our tool.

VII. Applications and Performance

A. Probing attacks

As an example of how the axiomatization can be used for security analysis, and to compare the performance of our implementation, we conducted a small case study on analyzing probing attacks, based on the benchmark test cases described by Becker and Koleini [8], [7]. Their benchmark was set up to test the performance of their tool (henceforth referred to as BK) for verifying opacity and detectability in probing attacks. BK's algorithm attempts to construct a policy that is observationally equivalent to all probes but makes the fact to be detected false. The fact is opaque if BK manages to construct such a policy, and detectable otherwise. In contrast, our tool (henceforth referred to as TML, for Trust Management Logic) is a general theorem prover for our logic. By Theorems III.7, IV.11, and V.3, TML can be used to check opacity and detectability by constructing a formula corresponding to a probing attack and then proving it mechanically.

To keep this paper self-contained, we briefly describe the tested scenarios, and refer the reader to [7] for a more detailed explanation.

The compute cluster `Clstr` under attack has the following policy $\gamma_{\texttt{Clstr}}$:

$$\begin{aligned}
&\mathit{canExe}(\texttt{Clstr}, x, j) :- \\
&\quad \mathit{mem}(\texttt{Clstr}, x), \mathit{owns}(\texttt{Clstr}, x, j), \mathit{canRd}(\texttt{Data}, \texttt{Clstr}, j).\\
&\mathit{owns}(\texttt{Clstr}, x, j) :- \mathit{owns}(y, x, j), \mathit{isTTP}(\texttt{Clstr}, y).\\
&\mathit{mem}(\texttt{Clstr}, x, j) :- \mathit{mem}(y, x, j), \mathit{isTTP}(\texttt{Clstr}, y).\\
&\mathit{canRd}(\texttt{Data}, x, j) :- \mathit{canRd}(y, x, j), \mathit{owns}(\texttt{Data}, y, j).\\
&\mathit{owns}(\texttt{Data}, x, j) :- \mathit{owns}(y, x, j), \mathit{isTTP}(\texttt{Data}, y).\\
&\mathit{isTTP}(\texttt{Clstr}, \texttt{CA}).\\
&\mathit{isTTP}(\texttt{Data}, \texttt{CA}).
\end{aligned}$$

Here, x and y range over a set of users, and j ranges over a set of compute job identifiers. The first parameter

[2]Z3 is an SMT solver, but we only use its SAT solving capabilities.

of each predicate should be interpreted as the principal who says, or vouches for, the predicate. The policy stipulates that, according to Clstr, members who own a job can execute it, if Clstr can read the data associated with it according to data center Data. Clstr delegates authority over job ownership and membership to trusted third parties (TTP). Data delegates authority over read permissions to job data to data owners. Data also delegates authority over job data ownership to TTPs. Both Clstr and Data say that certificate authority CA is a TTP.

TC1. In the basic test case (TC1), the attacker Eve possesses four credentials $\gamma_{\texttt{Eve}}$:

$$owns(\texttt{CA}, \texttt{Eve}, \texttt{Job}).$$
$$mem(\texttt{CA}, \texttt{Eve}).$$
$$canRd(\texttt{Eve}, \texttt{Clstr}, \texttt{Job}).$$
$$canRd(\texttt{Eve}, \texttt{Clstr}, \texttt{Job}) :- mem(\texttt{Clstr}, \texttt{Bob}).$$

Clstr exports only one query to Eve:

$$\varphi_{\texttt{Eve}} = canExe(\texttt{Clstr}, \texttt{Eve}, \texttt{Job}).$$

With her four credentials and the query, Eve can form $2^4 = 16$ *probes* (*cf.* Def. III.5) of the form $\Box_\gamma \varphi_{\texttt{Eve}}$, for each $\gamma \subseteq \gamma_{\texttt{Eve}}$. These result in 16 *observations* under $\gamma_{\texttt{Clstr}}$: the observation corresponding to probe π is just π if $\gamma_{\texttt{Clstr}} \Vdash \pi$, and otherwise it is $\neg\pi$. The resulting *probing attack* φ_a under $\gamma_{\texttt{Clstr}}$ is then the conjunction of all 16 observations.

In TC1, Eve wishes to find out if Bob is not a member of Clstr – in other words, if $\neg mem(\texttt{Clstr}, \texttt{Bob})$ is *detectable*. By Theorems III.7, IV.11, and V.3, this is equivalent to checking

$$\vdash \varphi_a \to \neg mem(\texttt{Clstr}, \texttt{Bob}).$$

This is provable, and therefore Eve can detect that Bob is not a member.

TC2. The atomic clause $mem(\texttt{Clstr}, \texttt{Bob})$ is added to $\gamma_{\texttt{Clstr}}$, and the fact to be detected is changed to $mem(\texttt{Clstr}, \texttt{Bob})$. The corresponding formula is not provable, and hence $mem(\texttt{Clstr}, \texttt{Bob})$ is opaque.

TC3. Based on TC1, three irrelevant atomic clauses p_1, p_2, p_3 are added to $\gamma_{\texttt{Eve}}$, increasing the number of probes to $2^7 = 128$. The fact to be detected remains the same, and is indeed detectable.

TC4. This test case was omitted as it only tests a specific switch in Becker and Koleini's tool which is not relevant in our case.

TC5. Based on TC1, the probe query is changed to $\varphi_{\texttt{Eve}} = canExe(\texttt{Clstr}, \texttt{Eve}, \texttt{Job}) \wedge \neg isBanned(\texttt{Clstr}, \texttt{Eve})$. The fact remains detectable.

TC6. Based on TC5, the probe set is manually pruned to a minimal set that is sufficient to prove detectability. This reduces the number of probes from 16 down to only 3.

To get comparable performance numbers, we obtained the probing attack analyzing tool (henceforth referred to as BK) from Becker and Koleini and carried out the tests with both tools on an Intel Xeon E5630 2.53 GHz with 6 GB RAM. The table below summarizes the timings for all test cases, comparing BK with TML.

Test Case	# Probes	Timing (ms) BK	TML
1	16	28	10
2	16	28	10
3	128	218	44
5	16	3954	119
6	3	16	13

TML outperforms BK in all test cases. The performance gain is most notable in the more expensive test cases. To test if this is generally the case, we performed a test series, based on TC3, adding an extra irrelevant clause to the probe credential set $\gamma_{\texttt{Eve}}$ one by one. This doubles the number of probes (and thus the size of the formula to be proved) at each step.

Figure 1 compares the performance of both tools for this test series. TML's performance gain over BK increases exponentially with each added credential in $\gamma_{\texttt{Eve}}$. A probe credential of size 14 (resulting in 16,384 probes) was the maximum that BK could handle before running out of memory, taking 408 s (compared to 7 s with TML). We tested TML with up to 18 credentials (resulting in 262,144 probes), which took 179 s. A simple extrapolation suggests that TML can check a probing attack based on TC3 extended to 10^8 probes within less than three hours.

B. Proving Meta-Theorems

As we have seen, the axiomatization of the semantics together with our implementation enables us to mechanically prove universal truths about trust management systems – that is, statements that are implicitly quantified over all policies: a theorem $\vdash \varphi$ is equivalent to $\Vdash \varphi$, by Theorems IV.11 and V.3, which can be interpreted as "all policies γ satisfy the property φ".

But we want to go further than that. In this subsection, we show that we can use our implementation to automate proofs of *meta-theorems* about trust management. These are statements containing universally quantified meta-variables ranging over atoms, conjunctions of atoms, Γ or Φ. Our axiom schemas and Lemma V.2 are examples of such meta-theorems, with meta-variables p, $\vec{p}$, γ, φ etc.

In classical logic as well as all *normal* modal logics, proving such meta-theorems is trivial: if a propositional formula f is a theorem, then substituting any arbitrary formula f' for all occurrences of an atom p in f will also yield a theorem. In fact, the axiomatization of such logics often explicitly include a uniform substitution rule, and a finite number of axioms (rather than axiom schemas, as in our case).

Our logic breaks the uniform substitution property, as some of the axioms and rules have syntactic side conditions (e.g. Ax. (Dlog), Rule (Mon)). It is thus not a normal modal logic in the strict sense, but this does not pose any problems, and is perhaps even to be expected, as many belief-revision and other non-monotonic logics also break uniform substitution [42].

$$
\begin{array}{ll}
\Phi\text{-hole contexts} & \mathcal{C} ::= [\cdot] \mid \top \mid p \mid \neg\mathcal{C} \mid \Box_\gamma\mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \\
\Gamma\text{-hole contexts} & \mathcal{D} ::= [\cdot] \mid \top \mid p \mid \neg\mathcal{D} \mid \Box_\gamma\mathcal{D} \mid \Box_{[\cdot]}\mathcal{D} \mid \mathcal{D} \wedge \mathcal{D} \\
\mathbf{At}\text{-hole contexts} & \mathcal{E} ::= [\cdot] \mid \top \mid p \mid \neg\mathcal{E} \mid \Box_\gamma\mathcal{E} \mid \Box_{[\cdot]}\mathcal{E} \mid \Box_{p\,:-[\cdot]}\mathcal{E} \mid \mathcal{E} \wedge \mathcal{E}
\end{array}
$$

Figure 2. Φ-contexts with Φ-holes, Γ-holes and **At**-holes, respectively. A **At**-hole context takes as argument an atom or a conjunction of atoms.

The only downside is that proving meta-theorems is non-trivial, and manual proofs generally require structural induction over the quantified meta-variables. It is therefore not obvious if proving meta-theorems can be automated easily. After all, the range of the quantifiers is huge, and even infinite if **At** is infinite. We answer this question in the affirmative by presenting a number of proof-theoretical theorems on the provability of meta-theorems (meta-meta theorems, so to speak), that show that it is sufficient to just consider a small number of base case instantiations of meta-variables.

We will use *contexts* to formalize the notion of meta-theorem. A context is a Φ-formula with a 'hole' denoted by $[\cdot]$. We define three different kinds of contexts in Fig. 2, Φ-*hole*, Γ-*hole*, and **At**-*hole contexts*. Intuitively, the holes in a Φ-hole (Γ-hole, **At**-hole, respectively) context can be filled with any $\varphi \in \Phi$ ($\gamma \in \Gamma$, $\vec{p} \subseteq_{\text{fin}} \mathbf{At}$) to form a well-formed Φ-formula.

If $\mathcal{A}$ is a Φ-hole (Γ-hole, **At**-hole, respectively) context and $\alpha \in \Phi$ ($\alpha \in \Gamma$, $\alpha \subseteq_{\text{fin}} \mathbf{At}$), we write $\mathcal{A}[\alpha]$ to denote the Φ-formula resulting from replacing all holes in $\mathcal{A}$ by α.

It is easy to see that every Φ-hole context is also a Γ-hole context, and every Γ-hole context is also a **At**-hole context. Each of the three types of contexts completely cover all of Φ; in particular, the case $\Box_{\gamma\wedge[\cdot]}\mathcal{A}$ (for Γ-hole and **At**-hole contexts) is covered because $\Box_{\gamma\wedge[\cdot]}\mathcal{A}$ is equivalent to $\Box_\gamma\Box_{[\cdot]}\mathcal{A}$.

Theorem VII.1. Let $\mathcal{E}$ be a **At**-hole context, and let p be an atom that does not occur in $\mathcal{E}$.

$$\vdash \mathcal{E}[p] \;\text{ iff }\; \forall \vec{p} \subseteq_{\text{fin}} \mathbf{At}.\ \vdash \mathcal{E}[\vec{p}].$$

Theorem VII.2. Let $\mathcal{D}$ be a Γ-hole context, and let p and q be atoms that do not occur in $\mathcal{D}$.

$$\vdash \mathcal{D}[p] \wedge \mathcal{D}[\Box_q p] \;\text{ iff }\; \forall\gamma \in \Gamma.\vdash \mathcal{D}[\gamma]$$

Theorem VII.3. Let $\mathcal{C}$ be a Φ-hole context, and let p, q and r be atoms that do not occur in $\mathcal{C}$. Let $S = \{p;\ \Box_q p;\ \Box_{q\,:-\,r} p\}$.

$$(\forall\varphi_s \in S.\vdash \mathcal{C}[\varphi_s] \wedge \mathcal{C}[\neg\varphi_s]) \;\text{ iff }\; \forall\varphi \in \Phi.\vdash \mathcal{C}[\varphi]$$

These theorems enable us to mechanically prove meta-theorems about trust management. Essentially, they reduce a meta-level quantified validity judgement to a small number of concrete instances. There is only one case to consider for universally quantified atoms or conjunctions of atoms (Theorem VII.1), two cases for meta-variables ranging over policies (Theorem VII.2), and six cases (three of them negated) for meta-variables ranging over arbitrary formulas (Theorem VII.3).

Consider, for instance, the meta-statement

$$\forall\varphi \in \Phi.\ \vdash \varphi \longleftrightarrow \Box_\top\varphi.$$

More formally and equivalently, we could write

$$\forall\varphi \in \Phi.\ \vdash \mathcal{C}[\varphi],\ \text{where } \mathcal{C} = [\cdot] \longleftrightarrow \Box_\top[\cdot].$$

This can then be mechanically proved by proving just the six basic instances from Theorem VII.3.

It is easy to extend this method further. Meta-theorems with multiple meta-variables can also be mechanically proved with this approach by combining the theorems. For example,

$$\forall\varphi,\varphi' \in \Phi,\gamma \in \Gamma.\ \vdash \Box_\gamma(\varphi \wedge \varphi') \longleftrightarrow (\Box_\gamma\varphi \wedge \Box_\gamma\varphi')$$

reduces to $6 \times 6 \times 2 = 72$ propositional cases: six each for φ and φ', and two for γ.

We can also prove meta-theorems with side-conditions. If a meta-variable φ ranges over negation-free formulas from Φ, it is sufficient to prove the three positive instances from Theorem VII.3. Similarly, for meta-variables ranging over $\Box$-free φ (as in Lemma V.2), the number of cases reduces to two ($\varphi \mapsto p$ and $\varphi \mapsto \neg p$).

In the following, we discuss a number of meta-theorems that we verified using the tool, based on Theorems VII.1–VII.3 (in addition to proving them manually). These meta-theorems provide interesting general insights into Datalog-based trust management systems. Moreover, they have also been essential in our (manual) proofs of soundness and completeness (Theorem V.3).

$$\forall\varphi,\varphi' \in \Phi,\gamma \in \Gamma.\ \vdash \Box_\gamma(\varphi \wedge \varphi') \longleftrightarrow (\Box_\gamma\varphi \wedge \Box_\gamma\varphi')$$

As in standard modal logic, the $\Box$-operator distributes over conjunction. The trust management interpretation is equally obvious: submitting credential set γ to a policy results in a new policy that satisfies the property $\varphi \wedge \varphi'$ iff the new policy satisfies both φ and φ'. Proving this theorem took 574 ms.

$$\forall\varphi,\varphi' \in \Phi,\gamma \in \Gamma.\ \vdash \Box_\gamma(\varphi \vee \varphi') \longleftrightarrow (\Box_\gamma\varphi \vee \Box_\gamma\varphi')$$

In most modal logics, $\Box$ does *not* distribute over $\vee$. This theorem holds only because the accessibility relation is functional, or equivalently, because the result of combining a credential set with a policy is always uniquely defined. But again, the theorem is obviously true in the trust management interpretation: submitting credential set γ to a policy results in a new policy that satisfies the property $\varphi \vee \varphi'$ iff the new policy satisfies either φ or φ'. (577 ms)

$$\forall\varphi \in \Phi.\ \vdash \varphi \longleftrightarrow \Box_\top\varphi$$

Submitting an empty credential set is equivalent to not submitting anything at all. (3 ms)

$$\forall\varphi \in \Phi^+,\gamma \in \Gamma.\ \vdash \varphi \to \Box_\gamma\varphi,$$

where Φ^+ denotes the set of $\neg$-free formulas in Φ. This can be interpreted as a monotonicity property of the accessibility

relation, and also of credential submissions: positive properties are retained after credential submissions. (95 ms)

$$\forall \varphi \in \Phi, \gamma, \gamma' \in \Gamma.\ \vdash \Box_\gamma \Box_{\gamma'} \varphi \longleftrightarrow \Box_{\gamma'} \Box_\gamma \varphi$$

This property, corresponding to a commutative accessibility relation, is also unusual in multi-modal logics. A simple corollary is that all permutations of arbitrary strings of boxes are equivalent, or that the order in which credentials are submitted is irrelevant. (3059 ms)

$$\forall \varphi \in \Phi, p \in \mathbf{At}, \vec{p} \subseteq_{\text{fin}} \mathbf{At}.\ \vdash \vec{p} \to (\Box_p \varphi \longleftrightarrow \Box_{p :- \vec{p}} \varphi)$$

If $\vec{p}$ holds in a policy, then submitting the atomic p results in a policy that is indistinguishable from the policy resulting from submitting the conditional credential $p :- \vec{p}$. (30 ms)

$$\forall \varphi \in \Phi^+, \gamma_1, \gamma_2 \in \Gamma.\ \vdash \Box_{\gamma_1} \gamma_2 \wedge \Box_{\gamma_2} \varphi \to \Box_{\gamma_1} \varphi$$

This theorem asserts that credential-based derivations can be applied transitively. More precisely: if, after submitting credential set γ_1, the credential set γ_2 would be derivable from the combined policy, and if submitting γ_2 directly would be sufficient for making property φ true, then γ_1 alone would also be sufficient. This only holds for negation-free φ. A simple counter-example can be constructed from instantiating $\gamma_1 = p$, $\gamma_2 = \top$, and $\varphi = \neg p$. (1078 ms)

$$\forall \varphi \in \Phi, \gamma \in \Gamma.\ \vdash \gamma \to (\varphi \longleftrightarrow \Box_\gamma \varphi)$$

If a policy contains the clauses γ, then submitting γ as credential set is equivalent to not submitting anything at all. This holds even for properties φ containing negation. (157 ms)

C. Proving one's own completeness

In Section VI, we gave some informal justifications as to why the reduction to propositional logic, which our implementation is based on, is not only sound (which is relatively easy to prove manually) but also complete with respect to the axiomatization. We did not prove completeness completely manually, but instead used the implementation itself to assist in the proof, thereby letting the implementation effectively prove its own completeness!

The main reason why this is possible is the ability to prove meta-theorems mechanically (Theorems VII.1–VII.3). With this feature in place, we mechanically verified all axiom *schemas*. What this proves is that the reduction rules, as implemented, cover all axioms.

It remained to show that all rules are covered as well. Recall that there are three rules, Modus Ponens, (Mon), and (N). Modus Ponens is built into the underlying SAT solver. We manually proved that Rule (Mon) can be replaced by the axiom schema $\vdash \Box_{\gamma_1} \gamma_2 \wedge \Box_{\gamma_2} \varphi \to \Box_{\gamma_1} \varphi$, which we mechanically verified.

To prove coverage of Rule (N), we perform a rule induction over $\vdash \varphi$ in order to conclude $\vdash^* \Box_\gamma \varphi$ (where $\vdash^*$ denotes the proofs performed by the implementation). All the base cases, i.e., the cases where φ is an instance of an axiom, were proven mechanically, again as meta-theorems (for example, for Ax. (C1), we prove $\forall \gamma, \gamma' \in \Gamma.\ \vdash \Box'_\gamma (\Box_\gamma \gamma)$). The two remaining cases, where $\vdash \varphi$ is a rule application, were easy to prove manually.

Together, these results prove that $\vdash \varphi$ implies $\vdash^* \varphi$, in other words, that the implementation is complete. The correctness of the proof rests on a couple of assumptions: the soundness of the implementation itself, the correctness of the underlying SAT solver, and the correctness of our manual proofs. We are confident about the implementation's soundness, as the reduction rules it is based on are sound, and it has been extensively tested. To achieve an even higher level of confidence about the semi-mechanically proven completeness result, one could mechanically verify all computer-generated subproofs, since an automated proof *verifier* would be much smaller and simpler than our proof generator.

VIII. Related Work

Trust Management. Blaze et al. coined the term 'trust management' in their seminal paper [13], referring to a set of principles for managing security policies, security credentials and trust relationships in a decentralized system. In their proposed paradigm, decentralization is facilitated by making policies depend on submitted credentials and by enabling local control over trust relationships. Policies, credentials and trust relationships should be expressed in a common language, thereby separating policy from the application. Early examples of trust management languages include PolicyMaker [13], KeyNote [12], and SPKI/SDSI [46], [27].

Li et al. [41], [40] argue that authorization in decentralized systems should depend on delegatable attributes rather than identity, and call systems that support such policies and credentials attribute-based access control (ABAC) systems. In essence, their ABAC paradigm is a refinement of trust management that makes the requirements on the expressiveness of credentials and policies more explicit: principals may assert parameterized attributes about other principals; authority over attributes may be delegated to other principals (that possess some specified attribute) via trust relationship credentials; and attributes may be inferred from other attributes. Their proposed policy language, RT, satisfies all these requirements. Like its predecessor, Delegation Logic (DL) [38], RT can be translated into Datalog. (A more expressive variant of RT, RT^C [39], can be translated into Datalog with constraints [34].)

Datalog has also been chosen as the basis of many other trust management languages. Examples include a language by Bonatti and Samarati [14], [15], SD3 [35], Binder [24], Cassandra [11], [10], a language by Wang et al. [54], one by Giorgini et al. [30], [31] and SecPAL [5], [6].

Apart from their relation to Datalog, what most of these languages have in common is that attributes are qualified by a principal who "says" it, and is vouching for the attribute's truth. In a credential, this principal coincides with the credential's issuer. For example, in Binder, the fact (or condition) that principal A is a student, according to authority C, could be expressed as $C.isStudent(A)$; similarly, in SecPAL, one would write C **says** A $isStudent$. This qualifier does not

extend Datalog's expressiveness, as it is easy to translate a qualified atom $C.p(\vec{e})$ into a normal Datalog atom $p(C, \vec{e})$.

The **says** operator can be traced back to an authorization logic by Abadi et al. (ABLP) [2], [36]. Even though it predates the paper by Blaze et al., ABLP could be seen as a trust management language. It introduced the **says** operator – but in contrast to the simpler Datalog-based languages, ABLP and related languages such as ICL [29], CCD [1] and DKAL [32], [33] treat the **says** (or **said**, in the case of DKAL) construct as a proper unary operator in the logic, which cannot be simply translated into an extra predicate parameter. Our semantics therefore does not cover these languages.

Previous work on trust management semantics. The Datalog-based languages inherit their semantics from Datalog. The most common way to present Datalog's semantics is as the minimal fixed point of the immediate consequence operator $\mathbf{T}_\gamma$, parameterized on a Datalog program γ [21]. The result is the set of all atoms p that are true in γ. Our inductive definition of $\gamma \Vdash p$ coincides with this semantics: $\gamma \Vdash p$ iff $p \in \mathbf{T}_\gamma^\omega(\emptyset)$. A model-theoretic semantics can be given by taking the minimal Herbrand model (i.e., the intersection of all Herbrand models) of γ, and a proof-theoretic semantics can be defined using resolution strategies [3]. All three flavours of the standard semantics are equivalent, but, as we have shown in Section IV, they are not adequate for modeling Datalog-based trust management policies that are combined with varying sets of credentials.

Abadi et al. [2] define ABLP axiomatically and then give it a model-theoretic semantics based on Kripke structures. However, the axiomatization is not complete with respect to the semantics. Further work along these lines has been done by Garg and Abadi [29], who present sound and complete translations from a minimal logic with a **says** operator called ICL, and various extensions of it, into the classical modal logic S4. Similar, Gurevich and Neeman [33] provide a Kripke semantics for DKAL2, the successor of the DKAL [32]. These modal semantics are straightforward compared to the one presented here, but this is because they have a completely different focus, namely providing a modal interpretation of the **says** (or **said**) operator. As we have argued above, this operator is not very interesting in the context of the more practical, Datalog-based, languages (at least from a foundational point of view). The focus of our semantics is to give a modal interpretation of the turnstile operator $:-$ in Datalog policies and of credential submissions in a trust management context.

Related logics and logic programming. It has been noted before that the standard Datalog semantics does not enjoy compositionality and full abstraction relative to program union. Gaifman and Shapiro [28] propose a semantics for logic programs that is compositional, fully abstract and preserves congruence with respect to program union. These properties are achieved by interpreting logic program clauses as implicational formulas, as a result of which all dependencies between atoms are preserved. However, this semantics does not give us the desired behavior (see Section IV). The problem, in essence, stems from the fact that material implication is inadequate as an interpretation of conditional if-then statements [47], and thus also of Datalog clauses (in our context) and credential submissions: if $\neg p$ holds in a policy, it follows that $p \rightarrow q$ also holds, for every q. However, it should *not* follow that the clause $q :- p$ is contained in the policy; and similarly, it is *not* justified to infer that q would hold if credential p were submitted and combined with the policy.

One of the main claims of this paper is that clauses and credential submissions ought to be modeled as counterfactual statements. The complexity of our semantics, then, stems from the fact that simple, truth-functional Boolean operators cannot offer an adequate account of counterfactuals. Stalnaker [50] and Lewis [37] were the first to propose a Kripke semantics for counterfactuals, based on a similarity ordering on worlds: essentially, "if p were true, then q would be true" holds in a world w if of those worlds in which p is true, the ones that are most similar to w also make q true. Our semantics is based on the same basic framework. Our definition of what "most similar" means is novel, as is our counterfactual interpretation of Datalog. Therefore, our work could also be seen as a novel semantics for Datalog in general. However, the action of dynamically injecting varying sets of clauses into a Datalog program is a characteristic that is rather specific to trust management, hence it is more appropriate to frame our semantics specifically as a trust management semantics.

Much work has been done on axiomatizations of multi-modal counterfactual logic. A good overview can be found in a paper by Ryan and Schobbens [49]. Their paper also contains a comprehensive listing of axioms proposed in the literature together with the corresponding frame conditions.

At first sight, hypothetical Datalog [16], [17] bears some resemblance to our logic. Hypothetical Datalog allows clauses such as $p :- (q \ : \ \texttt{add}\ r)$, meaning "$p$ is derivable provided that, if r were added to the rule base, q would hold". In our logic, this would correspond to $(q :- r) \rightarrow p$. But our logic is significantly more expressive: hypothetical Datalog cannot express statements that are hypothetical at the top-level and/or hypothetically add non-atomic clauses, for instance "p would hold if the conditional (credential) 'if r were true then q would hold' were added". In our logic, this statement can be expressed as $\Box_{q :- r} p$. Moreover, the work on hypothetical Datalog (and similar works on hypothetical reasoning) is only concerned with query evaluation against concrete rule bases, and not with the harder problem of universal validity.

Probing attacks. We identified the security analysis of probing attacks as one practical area on which the present work is likely to have an impact. The problem of probing attacks has gained attention only rather recently. Gurevich and Neeman were the first to identify this general vulnerability of logic-based trust management systems [32]. In [4], probing attacks are framed in terms of the information flow properties opacity [43], [19] and its negation, detectability. Becker and Koleini developed a tool for checking detectability of confidential facts in Datalog policies, based on constructing a counter-

policy, i.e., one that conforms to the given probes but makes the confidential fact false. The fact is detectable if and only if no such policy can be found. We use and extend their benchmark to compare the performance of our logic-based approach in Section VII-A.

IX. Concluding Discussion

Logics and semantics have long played an important, and successful, role in security research, especially in the area of cryptographic protocols [51] . (A prominent example has been the struggle to find an adequate semantics for BAN logic [20], see e.g. [22], [18], [52], [53]). The area of trust management, however, has hitherto not been investigated from a foundational, semantics-based point of view.

Evaluating a Datalog policy is a straightforward task, and so is taking the union of two sets of Datalog clauses. At first sight, then, it may come as an unwelcome surprise that our semantics, and the axiomatization, of Datalog-based trust management is so complex. Indeed, if one were only interested in the results of evaluating access queries against a concrete policy under a concrete set of submitted credentials, then a formal semantics would be unnecessary. But if one is interested in *reasoning about* the behavior of trust management systems, it is necessary to formulate universal truths that are quantified over all policies. Proving such statements is remarkably hard, even though the base language is so simple. As we have seen, neither the standard Datalog semantics nor the Kripke semantics for ABLP and related languages properly captures Datalog-based trust management. The situation has actually been worse than that of BAN logic,[3] since, prior to the present work, not even a sound and complete proof system existed, let alone a formal semantics.

Our formal semantics is defined by the notion of TM models and the TM validity judgement $\Vdash_{\text{TM}}$, and the axiomatization of TM validity is given by the proof system $\vdash$. Theorem V.3 shows that the proof system is sound and complete with respect to the semantics. So what role does the relation $\Vdash\!\vdash$, which we introduced in Section II, play? We need it, because a *semantics*, despite the term's etymology, does not really convey the *meaning* of the logic. As Read [45] puts it,

> [f]ormal semantics cannot itself be a theory of meaning. It cannot explain the meaning of the terms in the logic, for it merely provides a mapping of the syntax into another formalism, which itself stands in need of interpretation.

Of course, the relation $\Vdash\!\vdash$ is also just "another formalism", but it is one that is much closer to the natural language description of what a trust management system does, and can therefore more easily be accepted as "obviously" correct. Without it (and Theorem IV.11 providing the glue), there would be a big gap between the intuitive meaning of the language and its formalization.

What the formal semantics does provide is a number of alternative, less obvious, interpretations of a trust management system. TM models are abstract, purely mathematical objects that are independent of the language's syntax. They capture precisely (and only) the essential aspects of a trust management system. The easiest interpretation of a TM model is a graph in which two policies are connected when one is the result of submitting a set of credentials to the other.

A deeper alternative interpretation is that trust management logic is a counterfactual logic – a logic that avoids the paradoxes of material implication. Both policy clauses as well as statements about credential submissions are counterfactual, rather than implicational, statements. They state what *would* be the case if something else *were* the case.

As Ryan and Schobbens have noted, counterfactual statements can also be interpreted as hypothetical minimal updates to a knowledge base [49]. Under this interpretation, a credential submission $\Box_\gamma\varphi$ would be equivalent to saying that φ holds in a policy after it has been minimally updated with credential set γ. The restrictions on the accessibility relation (Def. IV.7) can then be seen as a precise specification of what constitutes a minimal update to a policy.

Hence, from a foundational point of view, our semantics provides new insights into the nature of trust management. From a more practical point of view, it led us to an axiomatization that can be mechanized. We showed how our implementation could be put to good use by applying it to the analysis of probing attacks. It is the first automated tool that can feasibly check real-world probing attacks of realistic size, comprising millions of probes. But our implementation is a general automated theorem prover for our language, the expressiveness of which goes far beyond that needed for probing attacks. In particular, we used the implementation to prove general meta-theorems about trust management – some of which are intuitively obvious (but not necessarily easy to prove), and some of which are decidedly non-trivial (such as Lemma V.2 or lemmas that help prove the implementation's own completeness).

Our logic is decidable, since every formula is equivalent to a (potentially much larger) propositional formula. However, the complexity of the logic remains an open question. We also leave the development of a first-order version of the logic to future work: in this version, atoms would be predicates with constant and variable parameters, and clauses would be implicitly closed under universal quantification.

Acknowledgements We thank Mark Ryan and Stephen Muggleton for fruitful discussions, and Christoph Wintersteiger for his support with Z3. We are also grateful for the valuable comments from the anonymous reviewers.

[3] Cohen and Dam succinctly described the BAN situation thus [22]: "While a number of semantics have been proposed for BAN and BAN-like logics, none of them capture accurately the intended meaning of the epistemic modality in BAN [...]. This situation is unsatisfactory. Without a semantics, it is unclear what is established by a derivation in the proof system of BAN: A proof system is merely a definition, and as such it needs further justification."

References

[1] M. Abadi. Access control in a core calculus of dependency. *Electronic Notes in Theoretical Computer Science*, 172:5–31, 2007.

[2] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] M. Y. Becker. Information flow in credential systems. In *IEEE Computer Security Foundations Symposium*, pages 171–185, 2010.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *IEEE Computer Security Foundations Symposium*, pages 3–15, 2007.

[6] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[7] M. Y. Becker and M. Koleini. Information leakage in datalog-based trust management systems. Technical Report MSR-TR-2011-11, Microsoft Research, 2011.

[8] M. Y. Becker and M. Koleini. Opacity analysis in trust management systems. *Proceedings of the 14th Information Security Conference (ISC2011)*, pages 229–245, 2011.

[9] M. Y. Becker, A. Russo, and N. Sultana. Foundations of logic-based trust management. Technical Report MSR-TR-2012-10, Microsoft Research, 2012.

[10] M. Y. Becker and P. Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 159–168, 2004.

[11] M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *IEEE Computer Security Foundations*, pages 139–154, 2004.

[12] M. Blaze, J. Feigenbaum, and A. D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.

[13] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[14] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 134–143. ACM, 2000.

[15] P. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

[16] A. Bonner. A logic for hypothetical reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 480–484, 1988.

[17] A. Bonner. Hypothetical Datalog: complexity and expressibility. *Theoretical Computer Science*, 76(1):3–51, 1990.

[18] C. Boyd and W. Mao. On a limitation of BAN logic. In *Advances in Cryptology (EUROCRYPT'93)*, pages 240–247. Springer, 1994.

[19] J. Bryans, M. Koutny, L. Mazaré, and P. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008.

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

[21] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[22] M. Cohen and M. Dam. A completeness result for BAN logic. *Methods for Modalities*, 4, 2005.

[23] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin / Heidelberg, 2008.

[24] J. Detreville. Binder, a logic-based security language. In *IEEE Symposium on Security and Privacy*, pages 105–113, 2002.

[25] DIMACS Challenge – Satisfiability: Suggested Format. ftp://dimacs.rutgers.edu/pub/challenge/satisfiability, 1993.

[26] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 333–336. Springer Berlin / Heidelberg, 2004.

[27] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, and Thomasand. SPKI certificate theory, RFC 2693, September 1999.

[28] H. Gaifman and E. Shapiro. Fully abstract compositional semantics for logic programs. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 134–142. ACM, 1989.

[29] D. Garg and M. Abadi. A modal deconstruction of access control logics. In *Foundations of Software Science and Computation Structures (FOSSACS'08)*, pages 216–230, 2008.

[30] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering meets trust management. *International Conference on Trust Management*, pages 176–190, 2004.

[31] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *International Journal of Information Security*, 5(4):257–274, 2006.

[32] Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 149–162, 2008.

[33] Y. Gurevich and I. Neeman. DKAL 2 – a simplified and improved authorization language. Technical Report MSR-TR-2009-11, Microsoft Research, 2009.

[34] J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[35] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[36] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[37] D. Lewis. *Counterfactuals*. Harvard University Press, 1979.

[38] N. Li, B. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In *IEEE Symposium on Security and Privacy*, pages 27–42, 2000.

[39] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Practical Aspects of Declarative Languages*, pages 58–73, 2003.

[40] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Symposium on Security and Privacy*, pages 114–130, 2002.

[41] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 156–165. ACM, 2001.

[42] D. Makinson. Ways of doing logic: What was different about AGM 1985? *Journal of Logic and Computation*, 13(1):3–14, 2003.

[43] L. Mazaré. Using unification for opacity properties. In *In Proceedings of the Workshop on Issues in the Theory of Security (WITS'04*, pages 165–176, 2004.

[44] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[45] S. Read. *Relevant logic*. Blackwell Oxford, 1988.

[46] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure, August 1996.

[47] B. Russell. *Principles of Mathematics*. London: G. Allen & Unwin, 1903.

[48] M. Ryan and M. Sadler. Valuation systems and consequence relations. In D. G. S. Abramsky and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 2–74. 1992.

[49] M. Ryan and P. Schobbens. Counterfactuals and updates as inverse modalities. *Journal of Logic, Language and Information*, 6(2):123–146, 1997.

[50] R. Stalnaker. A theory of conditionals. *Studies in logical theory*, 2:98–112, 1968.

[51] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *IEEE Symposium on Security and Privacy*, pages 156–170. IEEE, 1991.

[52] P. Syverson and P. Van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Symposium on Security and Privacy*, pages 14–28. IEEE, 1994.

[53] W. Teepe. BAN logic is not 'sound', constructing epistemic logics for security is difficult. *Workshop on Formal Approaches to Multi-Agent Systems*, 6:79–91, 2006.

[54] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55. ACM, 2004.