OPERATIONAL SEMANTICS FOR DKAL: APPLICATION AND ANALYSIS

YURI GUREVICH AND ARNAB ROY

Abstract. DKAL is a new authorization language based on existential fixed-point logic and more expressive than existing authorization languages in the literature. We present some lessons learned during the first practical application of DKAL and some improvements that we made to DKAL as a result. We develop operational semantics for DKAL and present some complexity results related to the operational semantics.

1. Introduction

Preamble. DKAL is a new declarative authorization language for distributed systems [6]; it is based on existential fixed-point logic [3] and is considerably more expressive than existing authorization languages in the literature. The first practical application of DKAL was to the problem of automating source asset management (SAM) at Microsoft. We partook an active role in that endeavor. The automation problem involves more than DKAL and is still a work in progress. While DKAL proved itself sufficiently expressive as well as convenient and elegant, the application necessitated some improvements of DKAL that became a part of a larger revision of DKAL [8]. In this paper we present some lessons learned during the first practical application of DKAL and the improvements that we made to DKAL. In addition, we present some related theoretical results. We presume that the reader is familiar with [6]; otherwise this paper is self-contained.

Source asset management (SAM). SAM is an example of asset management. Large software companies have many partners, contractors and subcontractors who need to access the sources of the various software products produced by the company. The ever-growing number of such requests necessitates clear access control policies regulating who is entitled to use what sources, where, for how long, and so on. The company also needs processes in place to efficiently implement those policies. DKAL enables formulating rich and sophisticated policies that result in simpler audit and feasible automation.

Lessons. The most important lesson that we learned working on SAM was this: separate the access control policy from the workflow. The workflow includes the procedural aspects of access decisions. The workflow can be quite complex to ensure the flexibility of the the way access decisions are made. But the policies should be succinct and easy to understand. Furthermore, the policies should guarantee important safety properties of the access decision process for any workflow. Various compliance requirement and regulations typically can be formulated as safety

The work was done in the summer of 2008 when the second author was an intern at Microsoft Research.

properties. It is desirable that the policies alone guarantee the compliance with those requirements and regulations thus eliminating the need to understand large amounts of procedural code.

One other lesson was that in the case of automated access control regulated by clear access control policies it is crucial for the auditing purposes to properly log all human judgments. No set of policies and processes makes industrial access control deterministic. The inherent non-determinism is resolved by human judgments in making and delegating access decision. Logging these judgments is necessary for auditing purposes as it allows the auditor to understand who made what decisions and whether they had the right to do so (either originally or by delegation).

Operational semantics and related complexity questions. As a result of the lessons learned, we made some theoretical advances to DKAL. We separated policy and workflow and defined clear operational semantics. Logging the right information is a natural byproduct of the way operational semantics is defined. In addition we obtained some theoretical results characterizing the complexity of analyzing natural properties of policies. We explored two communication models for defining workflows and proved that deciding reachability questions is possible in polynomial time in certain cases whereas deciding invariants is coNP-complete.

2. Related Work

Several languages have been developed in recent years to express authorization policies. The late genealogy of DKAL consists primarily of Binder, Delegation Logic and SecPAL. Binder [4] builds directly on Datalog, extending it with an import/export construct says that connects Datalog policies of different principals and makes the issuer of an imported assertion explicit. Delegation Logic [9, 10] does not have explicit issuers for all assertions but it has constructs specific to authorization including ones for delegation, representation, and thresholds.

SecPAL [1, 2, 5] has both explicit issuers and specific authorization constructs designed with distributed systems in mind. The number of constructs is deliberately kept low, but the language is expressive and captures numerous standard authorization scenarios. One important construct is *can say*: if A says B can say foo and if B says foo then A says foo. The can say construct can be nested. The semantics is defined by means of a few deduction rules but, for execution purposes, SecPAL reduces to safe Constraint Datalog. Nesting of the *can say* construct gives rise to relations whose arity depends on the nesting depth.

DKAL [6, 7] extends SecPAL in many directions as far as expressivity is concerned. But it is quite different and builds on Existential Fixed-Point Logic (EFPL) which is much richer than Constraint Datalog; EFPL is cast as Liberal Datalog in [3]. In particular DKAL allows one to freely use functions; as a result, principals can quote other principals. Contrary to SecPAL, communication in DKAL is targeted which is beneficial for security and liability protection. The additional expressivity is achieved within the same bounds of computational complexity as that of SecPAL.

The main novelty of this paper is that we take workflow into account and deal explicitly with the runtime evolution of policies. In this connection we extend DKAL with the *from* construct. In [6], the sender directs communication to particular receivers but the receivers are passive. The from construct allows the receivers to filter incoming information.

A limited policy evolution is possible in SecPAL: policy statements can be revoked but revocation is governed by static rules. There has been work on the analysis of dynamic evolution of authorization policies in the Trust Management framework [11]. The approach there has been to specify statically what kind of policy rules can be added or removed. They analyzed reachability and invariant checking. We do that too in our framework. The richness of DKAL compared to the simpler TM language makes these questions even more interesting.

3. Operational Semantics

We define operational semantics for communication in DKAL. This makes explicit the transaction of information that was implicit in [6]. Also we model situations where principals may listen only to information that they want to listen to by introducing a `from` construct. We deprecate the predicate $\texttt{knows}_0$ and the function $\texttt{said}_0$; we also rename $\texttt{tdon}_0$ to `tdon`. We modify the TrustApplication rule to

$$x \ \leq \ (p \ \texttt{said} \ x + p \ \texttt{tdon} \ x)$$

3.1. **Terminology.** Every principal has an immutable, or at least stable, core policy and a dynamic policy where it can assert communication statements. We use the following notation:

Π_p	Core policy of p.
Δ_p	Dynamic policy of p.
$\mathcal{K}_p$	$\{x : \Pi_p \cup \Delta_p \vdash x, \text{ no free variables in } x\}$

$\mathcal{K}_p$ is the *knowledge* of p.

3.2. **Communication Rules.** The main rule is COM. In simple form it says that if a principal A says an infon x to a principal B and if B is willing to accept x then x gets transmitted and B learns that A `said` x. In general, the rule is more complicated and involves a substitution. We restrict the infons being transmitted to ground infons.

$$\frac{\begin{cases}(A \ \texttt{to} \ \ q : x \leftarrow x_1, x_2, \cdots, x_m, \delta) \in \Pi_A \cup \Delta_A \\ (\text{B} \ \texttt{from} \ \ p : y) \in \Pi_B \cup \Delta_B \\ \{\eta x_1, \eta x_2, \cdots, \eta x_m\} \subseteq \mathcal{K}_A, \text{Substrate}_A \vDash \eta\delta \\ \eta p = A, \eta q = B, \eta x = \eta y, \text{ no free variables in } \eta x\end{cases}}{\Delta_B \ += \ A \ \texttt{said} \ \eta x} \ (\texttt{COM})$$

The last line means that Δ_B is augmented with infon A `said` ηx.

COM can be generalized to express common scenarios more succinctly. One such scenario is that if a principal B accepts an infon x, then it accepts infons of the form

$$z_1 \ \texttt{tdon} \ z_2 \ \texttt{tdon} \ \ldots \ z_k \ \texttt{tdon} \ x.$$

$$\frac{\begin{cases}(A \texttt{ to } q : z_1 \texttt{ tdon } z_2 \texttt{ tdon } \ldots z_k \texttt{ tdon } x \\ \qquad\qquad \leftarrow x_1, x_2, \cdots, x_m, \delta) \in \Pi_A \cup \Delta_A \\ (B \texttt{ from } p : y) \in \Pi_B \cup \Delta_B \\ \{\eta x_1, \eta x_2, \cdots, \eta x_m\} \subseteq \mathcal{K}_A, \text{Substrate}_A \vDash \eta\delta \\ \eta p = A, \eta q = B, \eta x = \eta y, \text{ no free variables in } \eta x \end{cases}}{\Delta_B \mathrel{+}= A \texttt{ said } \eta z_1 \texttt{ tdon } \eta z_2 \texttt{ tdon } \cdots \eta z_k \texttt{ tdon } \eta x} \ (\texttt{TCOM})$$

4. Examples

4.1. Alice wants to download an article from the Chux (an allusion to Chuck's) store. Best publishing house owns the copyright and delegates downloading permission to Chux. Chux has the policy that anybody meeting certain approval criteria can download the article. This gives rise to the following core policies.

Π_{best} :

best `to` p : chux `tdon` p *canDownload* article

Π_{chux} :

chux `to` $q : q$ *canDownload* article $\leftarrow$ *approve*$(q, \text{article})$

Π_{alice} :

alice : best `tdon`* alice *canDownload* article

Alice starts the process by issuing a dynamic `from` assertion.

(1) alice asserts `from` r : alice *canDownload* article.

Assuming that the relation *approve*(*alice*, article) is true, we have the following evolution.

(2) $\Delta_{alice} \mathrel{+}=$ alice `from` r : alice *canDownload* article. By (1)

(3) $\Delta_{alice} \mathrel{+}=$ best `said` chux `tdon`
alice *canDownload* article. By (2), $\Pi_{best}, TCOM$

(4) $\Delta_{alice} \mathrel{+}=$ chux `said` alice *canDownload* article. By (2), COM

(5) $\Pi_{alice} \cup \Delta_{alice} \vdash$ alice `knows`
alice *canDownload* article. By (3, 4), Π_{alice}

4.2. We add more details to the previous example. Chux lets a customer download an article if she authorizes the right amount of money for payment and has a perfect payrate. Chux trusts accounts.Chux on customer's payrate.

Π_{chux} :

chux `to` a : a *canDownload* $s \leftarrow$

a *authorized* \$$k$ to chux for s, a *hasPayRate* Perfect,

price$(s) = k$.

chux : accounts.Chux `tdon` a *hasPayRate* e.

chux : a `tdon` a *authorized* \$$k$ to chux for s.

chux `from` a : a *authorized* \$$k$ to chux for s.

$\Pi_{accounts.Chux}$:

accounts.Chux `to` chux : a *hasPayRate* $e \leftarrow$ a *hasPayRate* e.

The last assertion is not a tautology. If accounts.chux knows that a has PayRate e then it says that to Chux.

Alice starts by authorizing payment to Chux for the article and then asks for the article. Chux then asks whether Alice has perfect payrate. The dynamic assertions are as follows:

alice asserts `to` chux : alice *authorized* \$40 to chux for article.

alice asserts `from` r : alice *canDownload* article.

chux asserts `from` a : alice *hasPayRate* e.

If the price of the article is \$40 and Alice has a perfect payrate, we can again infer using the operational semantics that Alice knows she can download the article.

4.3. In this example, Chux gives discounts to employees of Fabricam. Chux trusts Fabricam to state whether a customer is an employee. Chux also trusts Crypto, a certifying intermediary, to confirm that a company claims that someone is it's employee.

Π_{chux} :

chux : crypto `tdon` r `said` q *is an employee of* r.

chux : fabricam `tdon` q *is an employee of* fabricam.

chux : q *can take discount* 5X4302 $\leftarrow$ q *is an employee of* fabricam.

chux `from` a : r `said` q *is an employee of* r.

chux `from` a : q *is an employee of* fabricam.

Consider a scenario where

crypto asserts `to` chux : fabricam `said` chris *is an employee of* fabricam.

Using the operational semantics, we can infer that Chux knows Chris can take discount 5X4302.

5. Modeling Source Asset Management

In this section we explore the application of DKAL to a practical problem in large software companies, namely Source Asset Management (SAM). This encompasses formulating a policy and processes for making code access decisions.

A Project Manager in a large software company A wants a codeset to be accessed by staff at another company. In A, the codeset is owned by an Information Asset Owner (IAO) who usually delegates such access decisions to a Source Rep. The access decision takes into account the codeset, the company to access the codeset, the type of permission — read/write or read-only.

In our DKAL model, authorizations are managed by Authorization Managers (AMs) of the respective companies. AMs are automated tools modeled as principals. As far as code authorization is concerned, the AM of the company is (or represents) the company itself. The Project Manager, IAO and Source Rep are also modeled as principals. The policy of the authorization manager A-AM of A may be as follows:

$\Pi_{A\text{-AM}}$:

p `tdon` q $canGet(codeset, parameters) \leftarrow assetOwner(p, codeset)$.

p `tdon` q `tdon` r $canGet(codeset, parameters) \leftarrow$

p `tdon` r $canGet(codeset, parameters)$, p *is the manager of* q.

p `tdon` q $canAccess(code, parameters) \leftarrow$

p *is a* PM, r $canGet(codeset, parameters)$,

r *is the AM of* $q, code$ *is in* $codeset$.

p `tdon` q `tdon` r $canAccess(code, parameters) \leftarrow$

p `tdon` q $canAccess(code, parameters)$.

Consider a scenario where A wants to allow a vendor B an access to a codeset, and where B-AM has the policy

$\Pi_{B\text{-AM}}$:

A-AM `tdon` B-AM $canGet(codeset, parameters)$.

A-AM `tdon` q $canAccess(code, parameters)$.

Suppose that Alan, Alfred, Andrew, Anthony and Alice work for A, and let Alan ▷ Alfred ▷ Andrew ▷ Anthony where p ▷ q means p is the manager of q. The role of Alice in the hierarchy will not matter. Suppose further that Alan is the IAO of *drivercodes*, a codeset that is to be given access to by B. A simplified workflow for the access decision may be as follows:

Access decision starts; Anthony goes to A-AM to request that B be given access to drivercodes.

A-AM *asserts* `from` p : B-AM $canGet$(drivercodes, params1).

Decision is escalated to Andrew, and then to Alfred.

Alfred *asserts* `to` A-AM : B-AM $canGet$(drivercodes, params1).

Alan *asserts* `to` A-AM : Alfred `tdon` B-AM $canGet$(drivercodes, params1).

A-AM communicates decision to B-AM.

A-AM *asserts* `to` B-AM : B-AM $canGet$(drivercodes, params1).

A developer Bruce in B needs a portion of drivercodes called gfx. Policy of Bruce is as follows:

Π_{bruce} :

B-AM `tdon` Bruce $canAccess(code, parameters)$.

A typical workflow to acquire this code would be as follows:

Bruce asks for access to the code.

Bruce $asserts$ `from` p : Bruce $canAccess$(gfx, params1).

In the workflow, this goes to B-AM. B-AM forwards the query along.

B-AM $asserts$ `from` p : Bruce $canAccess$(gfx, params1).

In the workflow, query goes to A-AM, which again forwards the query.

A-AM $asserts$ `from` p : Bruce $canAccess$(gfx, params1).

The workflow "chooses" who to ask about the query. It happens to be Alice. She decides to grant the code access to Bruce.

Alice $asserts$ `to` A-AM : Bruce $canAccess$(gfx, params1).

Anthony confirms that Alice has the authority to grant access.

Anthony $asserts$ `to` A-AM : Alice `tdon` Bruce $canAccess$(gfx, params1).

A-AM infers that Bruce can access the code and informs B-AM accordingly.

A-AM $asserts$ `to` B-AM : Bruce $canAccess$(gfx, params1).

B-AM trusts A-AM on such statements and concludes that Bruce can access the code. B-AM communicates the decision to Bruce.

B-AM $asserts$ `to` Bruce : Bruce $canAccess$(gfx, params1).

Bruce concludes that he can access the code.

As you can imagine, the workflow can proceed in myriad possible ways. Yet, the core policies of A-AM, B-AM and Bruce lets us state the following theorem, which holds independent of the workflow.

Theorem 1 (Safety)**.**

(1) *If* B*-AM* **`knows`** B*-AM* $canGet(codeset, parameters)$
then B*-AM* **`knows`** A*-AM* **`said`** B*-AM* $canGet(codeset, parameters)$
(2) *If Bruce* **`knows`** *Bruce* $canAccess(code, parameters)$
then Bruce **`knows`** B*-AM* **`said`** *Bruce* $canAccess(code, parameters)$

6. Complexity Analysis of Safety Properties

6.1. **Unrestricted Communication in Restricted DKAL.** In this subsection we consider a restricted, yet useful fragment $DKAL^{-}$ of DKAL which is DKAL without variables, without $+$, `canActAs, canSpeakAs, exists`, `tdon`* and without substrate constraints in the rules. Note that attributes allow us to deal with

roles even in the absence of `canActAs`, `canSpeakAs`. As opposed to `canActAs` and `canSpeakAs`, attributes take parameters. In the restricted fragment, we have three forms of infons. Firstly, there are *primitive infons* of the form Attribute$(a_1, \ldots, a_r)$ where elements a_i are regular. Secondly, we have *said infons* of the form p `said` x. Thirdly, we have *tdon infons* of the form p `tdon` x.

The communication is unrestricted in the following sense. We are interested in the consequences of the core policy of a given principal, say A, without restricting information that A can learn from other principals.

To simplify exposition, we let Σ_A be the set of said infons derived by the COM rule from Δ_A and communications to A by other principal. For example, if $(B$ `to` $A : x) \in \Delta_A$ and $(A$ `from` $B : x) \in \Delta_B$ then $(B$ `said` $x) \in \Sigma_A$. From the perspective of knowledge, it is sufficient to work with Σ_A, since the `from` and `to` statements by themselves do not add any knowledge for A; it is only when we combine corresponding `from` and `to` statements by the COM rule that additional knowledge is derived.

One interesting property is the reachability relation which is defined as follows: an infon I is reachable from a core policy Π_A if there exists a set Σ_A such that $\Pi_A \cup \Sigma_A \vdash A$ `knows` I.

Theorem 2 (Reachability)**.** *The reachability relation for $DKAL^-$ is polynomial time.*

Proof. Without loss of generality, we may assume that initially Σ_A is empty. We transform the total policy $\Pi_A \cup \Sigma_A$ of A in ways that preserve the reachability status of the given infon I.

Remove some said infons from core policy.

(1) Remove any rule $(p$ `said` $x) \leftarrow x_1, \ldots, x_k$ from Π_A and add the assertion $(p$ `said` $x)$ to Σ_A.
(2) If an infon p `said` x occurs in the body of a rule R in Π_A, remove the infon from the body of R and and add the assertion $(p$ `said` $x)$ to Σ_A.

The new policy $\Pi^1_A \cup \Sigma^1_A$ has the following property. In any core rule $y_0 \leftarrow y_1, \ldots, y_m$, every infon y_i is either primitive or a tdon infon. Obviously, $\Pi_A \cup \Sigma^1_A \vdash A$ `knows` I if and only if $\Pi^1_A \cup \Sigma^1_A \vdash A$ `knows` I.

Augment tdon rules. For any core rule p_1 `tdon` p_2 `tdon` $\ldots$`tdon` p_j `tdon` $x \leftarrow x_1, \ldots, x_k$ where x is primitive or a said infon, do the following. Add core rules

p_2 `tdon` p_3 `tdon` $\ldots$ p_j `tdon` $x \leftarrow p_1$ `tdon` p_2 `tdon` $\ldots$ p_j `tdon` x
p_3 `tdon` $\ldots$ p_j `tdon` $x \leftarrow p_2$ `tdon` p_3 `tdon` $\ldots$ p_j `tdon` x
$\ldots$
p_j `tdon` $x \leftarrow p_{j-1}$ `tdon` p_j `tdon` x
$x \leftarrow p_j$ `tdon` x

Also add the following assertions to Σ_A:

$$p_1 \texttt{ said } p_2 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$p_2 \texttt{ said } p_3 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$\ldots$$
$$p_{j-1} \texttt{ said } p_j \texttt{ tdon } x$$
$$p_j \texttt{ said } x$$

This way we obtain $\Pi_A^2 \cup \Sigma_A^2$. Observe that $\Pi_A \cup \Sigma_A^2 \vdash A \texttt{ knows } I$ if and only if $\Pi_A^2 \cup \Sigma_A^2 \vdash A \texttt{ knows } I$.

Any said infon $p \texttt{ said } x$ is reachable from any Π_A with a witness Σ_A asserting $(A \texttt{ from } p : x)$ and Σ_p asserting $(p \texttt{ to } A : x)$. So we may assume that I is primitive or a tdon infon. Recall that the DKAL derivability relation is polynomial time. Thus it suffice to prove that I is reachable from Π_A if and only if $\Pi_A^2 \cup \Sigma_A^2 \vdash A \texttt{ knows } I$. The if part is obvious. It remains to prove the only if part.

We claim that A cannot derive a new primitive or tdon infon as a result of learning any additional information, that is as a result of extending Σ_A^2 with any additional assertion R of said infon $p \texttt{ said } x$ (after using the COM rule with corresponding `from` and `to` assertions). Indeed, there are no said infons in the bodies of the rules in $\Pi_A^2 \cup \Sigma_A^2$. So adding R triggers no rule. Further, if $\Pi_A^2 \cup \Sigma_A^2 \vdash p \texttt{ tdon } x$ (x can be an infon of any type) then, by the second transformation above, $\Pi_A^2 \cup \Sigma_A^2 \vdash x$. So adding R does not help either. But these are the only ways that a said assertion can be helpful. □

We say that $(A \texttt{ knows } I_1) \longrightarrow (A \texttt{ knows } I_2)$ is an invariant for a core policy Π_A if $\Pi_A \cup \Sigma_A \vdash A \texttt{ knows } I_1$ implies $\Pi_A \cup \Sigma_A \vdash A \texttt{ knows } I_2$ for any set Σ_A. This gives rise to a binary invariant relation $\mathrm{Inv}(I_1, I_2)$ for $DKAL^-$.

Theorem 3 (Invariant Checking - Lower Bound)**.** *The invariant relation for $DKAL^-$ is coNP-hard.*

Proof. Reduction from `3-SAT`. Given a `3-SAT` instance $\phi \equiv (x_{11} \vee x_{12} \vee x_{13}) \wedge \cdots \wedge (x_{n1} \vee x_{n2} \vee x_{n3})$, where each x_{ij} is either one of the literals $x_1, \cdots, x_k$ or one of them negated, we construct a policy Π_A:

$$c_1 \leftarrow B \texttt{ said } x_{11} \qquad c_1 \leftarrow B \texttt{ said } x_{12} \qquad c_1 \leftarrow B \texttt{ said } x_{13}$$
$$\ldots$$
$$c_n \leftarrow B \texttt{ said } x_{n1} \qquad c_n \leftarrow B \texttt{ said } x_{n2} \qquad c_n \leftarrow B \texttt{ said } x_{n3}$$

$$F \leftarrow B \texttt{ said } x_1, B \texttt{ said } x_1'$$
$$\ldots$$
$$F \leftarrow B \texttt{ said } x_k, B \texttt{ said } x_k'$$

$$\phi \leftarrow c_1, \cdots, c_n$$

Here $c_1, \ldots, c_n, F, \phi$ are infons. Now we set $I_1 = \phi$ and $I_2 = F$. Observe that ϕ has a satisfying assignment iff there exists a Σ_A such that $\Pi_A \cup \Sigma_A \vdash A \texttt{ knows } \phi$, but $\Pi_A \cup \Sigma_A \nvdash A \texttt{ knows } F$. □

Theorem 4 (Invariant Checking - Upper Bound)**.** *The invariant relation for $DKAL^-$ is in* `coNP`*.*

Proof. **Case 1 :** I_1 is of the form p `said` x. In this case we set $\Sigma_A = \{p \texttt{ said } x\}$ and check whether $\Pi_A \cup \Sigma_A \vdash A$ `knows` I_2. If not then $\mathrm{Inv}(I_1, I_2)$ fails. If yes then $\mathrm{Inv}(I_1, I_2)$ holds; this is because both I_1 and I_2 will continue to hold if more assertions are added.

Case 2 : I_1 is not a said infon. We claim this: if $\mathrm{Inv}(I_1, I_2)$ fails then there is a subset Σ'_A of Σ^2_A, where Σ^2_A is the dynamic policy constructed in the proof of theorem 2, such that $\Pi_A \cup \Sigma'_A \vdash A$ `knows` I_1 but $\Pi_A \cup \Sigma'_A \nvdash A$ `knows` I_2. Recall that no additional (to Σ^2_A) communication allows A to derive any new primitive or tdon infon. Thus if $\Pi_A \cup \Sigma^2_A \nvdash A$ `knows` I_1, the invariant trivially holds. Suppose therefore, $\Pi_A \cup \Sigma^2_A \vdash A$ `knows` I_1. Now if $\Pi_A \cup \Sigma^2_A \nvdash A$ `knows` I_2, then Σ^2_A is itself a certificate of non-membership. So we additionally suppose $\Pi_A \cup \Sigma^2_A \vdash A$ `knows` I_2.

Let Σ'' be the set of minimum cardinality such that $\Pi_A \cup \Sigma'' \vdash A$ `knows` I_1 and $\Pi_A \cup \Sigma'' \nvdash A$ `knows` I_2. Let $\Sigma' = \Sigma'' \cap \Sigma^2_A$ and let $\Sigma_\delta = \Sigma'' - \Sigma'$. It must be that $\Pi_A \cup \Sigma' \nvdash A$ `knows` I_1 and $\Pi_A \cup \Sigma_A \nvdash A$ `knows` I_2, since otherwise Σ' will be of cardinality at most equal to Σ'' and it will be a subset of Σ^2_A. Now, if p `said` $x \in \Sigma_\Sigma$, then p `said` x cannot be in the body of some rule, nor can any rule having a head of the form q_1 `tdon` q_2 `tdon` $\cdots$`tdon` p `tdon` x be in Π_A, since these are already in Σ^A_2. Therefore, starting from Σ' we cannot add any *useful* assertions of the form p `said` x to make A `knows` I_1 true as I_1 does not begin with a `said` . Hence Σ_δ must be empty, making Σ'' a subset of Σ^2_A. Since Σ^2_A is of poly-size in the input, we have a poly-sized certificate of non-membership. □

Corollary 1 (Invariant Checking)**.** *The invariant relation for $DKAL^-$ is* `coNP`*-complete.*

Remarks.

- If we constrain the principals in theorem 2 to assert at most one of more than one possible choices, then reachability is `NP`-hard. The proof is simple: just get rid of the 'F' rules in a reduction proof similar to theorem 3.

6.2. **Restricted Communication in Unrestricted DKAL.** In this subsection we analyze the complexity of deciding classes of safety properties for unrestricted DKAL. The policy does not have any `from` rules. The workflow interacts with the DKAL inference machine through `from` assertions only. The content of `from` assertions is unrestricted within the bounds of syntactic well-formedness.

Since the `to` assertions can only be derived from the core policy, we need to look at the union of the policies of all the principals $\bigcup_\alpha \Pi_\alpha$ in order to analyze reachability and invariant checking. We assume that the number of variables in each rule in each policy is fixed a-priori and that α ranges over all principals in the system. Also since the set of dynamic assertions Δ_A for a principal A is only going to contain `from` assertions, Σ_A as described in the previous section would be irrelevant here. Hence we work with Δ's in this section.

Theorem 5 (Reachability Checking)**.** *The reachability relation for DKAL is in polynomial time.*

Proof. We issue `from` statements for all the `to` rules in the policies. Let us call the union of these assertions Δ'. We then check whether $\bigcup_\alpha \Pi_\alpha \cup \Delta' \vdash A$ `knows` I.

This is doable in polytime since Δ' is polysized in the policies. It is easy to see that Δ' is the required Δ, since if A `knows` I is not derivable using Δ' with the policies then no further `from` assertion will help derive it. □

Theorem 6 (Invariant Checking - Lower Bound)**.** *The invariant relation for DKAL is coNP-hard.*

Proof. The proof is similar to theorem 3. We will reduce `3-SAT` to $\overline{\texttt{INVARIANT}}$. Given a `3-SAT` instance $\phi \equiv (x_{11} \vee x_{12} \vee x_{13}) \wedge \cdots \wedge (x_{n1} \vee x_{n2} \vee x_{n3})$, where each x_{ij} is either one of the literals $x_1, \cdots, x_k$ or one of them negated, we construct the following policy:

Π_A :

$c_1 \leftarrow B$ `said` x_{11} $\quad c_1 \leftarrow B$ `said` x_{12} $\quad c_1 \leftarrow B$ `said` x_{13}

$\cdots$

$c_n \leftarrow B$ `said` x_{n1} $\quad c_n \leftarrow B$ `said` x_{n2} $\quad c_n \leftarrow B$ `said` x_{n3}

$F \leftarrow B$ `said` x_1, B `said` x'_1

$\cdots$

$F \leftarrow B$ `said` x_k, B `said` x'_k

$\phi \leftarrow c_1, \cdots, c_n$

Π_B :

B `to` $A : x_1$

B `to` $A : x'_1$

$\cdots$

B `to` $A : x_n$

B `to` $A : x'_n$

Now we set $I_1 = \phi$ and $I_2 = F$. Observe that ϕ has a satisfying assignment iff there exists a Δ_A such that $\Pi_A \cup \Delta_A \vdash A$ `knows` ϕ, but $\Pi_A \cup \Delta_A \nvdash A$ `knows` F. Hence the proof. □

Theorem 7 (Invariant Checking - Upper Bound)**.** *The invariant relation for DKAL is in coNP.*

Proof. Let $\Delta = \{P$ `from` $Q : X(a, b, \cdots, k) \mid P, Q \in \alpha\}$, where

- Q `to` $P : X(A, B, \cdots, K) \leftarrow$ *body* is a rule in the policy of Q and $(a, b, \cdots, k)$ enumerate all possible instantiations of the variables $(A, B, \cdots, K)$.
- Q `to` $r : X(A, B, \cdots, K) \leftarrow$ *body* is a rule in the policy of Q and $(a, b, \cdots, k)$ enumerate all possible instantiations of the variables in $(A, B, \cdots, K)$.

We note that since the number of variables in a rule is fixed and all possible instantiations is upper bounded by the policy size, Δ is poly-sized in the input.

We claim that a certificate of non-membership is a subset of Δ. We prove this by arguing that given any set of `from` statements F, there is a set $F' \in \Delta$ such that $\bigcup_\alpha \Pi_\alpha \cup F \vdash I$ *iff* $\bigcup_\alpha \Pi_\alpha \cup F' \vdash I$. This is because

- Any `from` statement, that does not unify with any of the `to` rule heads, will never lead to additional knowledge.
- A `from` statement, that unifies with the head of a rule Q `to` $P : X(A, B, \cdots, K)$, say P `from` $Q : X(A', B', \cdots, K')$, leads to the same set of additional knowledge as does a set of statements in Δ: P `from` $Q : X(a, b \cdots, k)$ which enumerates all possible instantiations of the variables in $(A', B', \cdots, K')$.
- A `from` statement, that unifies with the head of a rule Q `to` $r : X(A, B, \cdots, K)$, say P `from` $Q : X(A', B', \cdots, K')$, leads to the same set of additional knowledge as does a set of statements in Δ: P `from` $Q : X(a, b \cdots, k)$ which enumerates all possible instantiations of the variables in $(A', B', \cdots, K')$.

Finally, the certificate of non-membership is a set $\Delta' \in \Delta$, such that $\bigcup_\alpha \Pi_\alpha \cup \Delta' \vdash A$ `knows` I_1, but $\bigcup_\alpha \Pi_\alpha \cup \Delta' \nvdash A$ `knows` I_2. □

Corollary 2 (Invariant Checking)**.** *The invariant relation for DKAL is* `coNP`*-complete.*

References

[1] Moritz Y. Becker, Cédric Fournet and Andrew D. Gordon, "SecPAL: Design and Semantics of a Decentralized Authorization Language", Technical Report MSR-TR-2006-120, Microsoft Research, September 2006.

[2] Moritz Y. Becker, Cédric Fournet and Andrew D. Gordon, "SecPAL: Design and Semantics of a Decetralized Authorization Language", 20th IEEE Computer Security Foundations Symposium (CSF), 3–15, 2007.

[3] Andreas Blass and Yuri Gurevich, "Two Forms of One Useful Logic: Existential Fixed Point Logic and Liberal Datalog", Bulletin of the European Association for Theoretical Computer Science, Number 95 (June 2008), 164–182.

[4] John DeTreville, "Binder, a Logic-Based Security Language", in IEEE Symposium on Security and Privacy, 105-113, 2002.

[5] Blair Dillaway, "A Unified Approach to Trust, Delegation, and Authorization in Large-Scale Grids", Technical Paper, Microsoft Corporation, September 2006, http://research.microsoft.com/en-us/projects/SecPAL/, seen on Dec 24, 2008.

[6] Yuri Gurevich and Itay Neeman, "DKAL: Distributed-Knowledge Authorization Language," 21st IEEE Computer Security Foundations Symposium (CSF 2008), 149–162.

[7] Yuri Gurevich and Itay Neeman, "DKAL: Distributed-Knowledge Authorization Language," Microsoft Research Tech Report MSR-TR-2008-09, January 2008.

[8] Yuri Gurevich and Itay Neeman, "DKAL 2.0," (tentative title)

[9] Ninghui Li, "Delegation Logic: A Logic-Based Approach to Distributed Authorization", Ph.D. thesis, New York University, September 2000.

[10] Ninghui Li, Benjamin N. Grosof and Joan Feigenbaum, "Delegation Logic: A Logic-Based Approach to Distributed Authorization", ACM Trans. on Information and System Security (TISSEC) 6:1 (February 2003), 128–171.

[11] Ninghui Li, William H. Winsborough, and John C. Mitchell, "Beyond Proof-of-Compliance: Safety and Availability Analysis in Trust Management", in Proceedings of 2003 IEEE Symposium on Security and Privacy, 123–139, May 2003.

Microsoft Research Redmond
E-mail address: gurevich@microsoft.com

Department of Computer Science, Stanford University
E-mail address: arnab@cs.stanford.edu