Language based Policy Analysis in a SPKI Trust Management System

Arun K. Eamani and A. Prasad Sistla
University of Illinois at Chicago
aeamani,sistla@cs.uic.edu

January 7, 2005

Abstract

SPKI/SDSI is a standard for issuing authorization and name certificates. SPKI/SDSI can be used to implement a Trust Management System, where the policy for resource access is distributedly specified by multiple trusted entities. Agents in the system need a formal mechanism for understanding the current state of policy. We present a first order temporal logic, called FTPL for specifying properties of a given SPKI/SDSI policy state. We also present algorithms to check if a SPKI/SDSI policy state satisfies a property specified in FTPL.

1 Introduction

1.1 Motivation and Use of the Language

SPKI/SDSI (simple public key infrastructure/simple distributed security infrastructure) is a mechanism for specifying policy in a distributed access control system [EFL+99], [Ell99]. With standardized semantics it can also be viewed as a Trust Management Language [BFK99]. There has been much work done on analyzing fixed a priori defined properties of such an authorization system [JR01], [CEE+01]. In this paper we introduce a language based on general purpose temporal logic for specifying properties of such a system. This language, called FTPL: First order Temporal Policy analysis Logic, could be used by the agents to reason about the state of the policy. Such analysis is useful for understanding the current state of policy, auditing the past policy statements, to point out any violations in trust between agents and for aiding policy refinement/management. We present efficient methods for automatically checking if a given SPKI/SDSI certificate set satisfies a policy question specified in the logic. The logic that we use is an extension of the standard real-time temporal logic extended with quantifiers over the principals in the system. Many important properties such as the following can be expressed in our logic - *can two principals K_1 and K_2 access the resource R simultaneously at some point in future?* We can also specify queries in this logic, for example *retrieve all the principals who were able to access a resource R at some point in a time interval.*

There have been a string rewriting system [CEE+01] and a push down system(PDS) [JR01] to model certificate analysis problems in a SPKI/SDSI framework. We propose that temporal logic be used as a language to specify the issues of authorization, naming and validity. The policy analysis problem would be specified as a temporal logic formula f and we evaluate the formula on a given set of certificates $\mathcal{C}$ and a particular time instance t. The formulas of the logic could be interpreted not only to evaluate the truthness of a statement but also to reason about the state of the system by finding all instantiations of free variables which would make the formula true in the given state of system at a particular instant of time.

In access control models the usual analysis consists of safety analysis [HRU75]: does there exist a reachable state where a untrusted principal has access to the resource? A state in the system corresponds to a set of signed policy statements i.e certificates. We change the policy state by adding or deleting a certificate. Li et al [LWM03] propose security analysis where the state can change in a restricted manner. Security analysis is a study of security properties such as safety, availability, containment etc. Availability requires that in every reachable state a particular principal will be able to access a resource. Containment requires that in every reachable state if a particular principal has a property, say being able to access a resource, then that principal also satisfies a condition, like being member of a particular role. The above type of analysis is useful to understand how the current policy state can evolve.

We propose *policy analysis* in distributed access control models, where we analyze properties of a particular policy state. We consider a policy state as consisting of a set of policy statements each labeled with a validity interval. Each policy statement is valid during the time interval associated with it. At any point of time only a subset of the given set of certificates are valid. Policy analysis can also be viewed as a special case of security analysis where all the possible state transitions are know in advance, given that certificate issuers specify the time period during which the certificate is valid. We can reformulate the problems of security analysis to reason over time instead of reachable states. For example the availability property could be restated as: *at all times in future does a principal K have access to a resource R.* While, for practical reasons, we may want to reason about bounded availability: *at all times during a time interval, say a school semester is a student K able to access the school ftp server R.*

In a SPKI/SDSI system, given certificates containing validity intervals, authorization and name relationships vary with time. In a delegation based system where multiple agents make policy statements regarding access of a particular resource, no agent is aware of the overall state of the policy. There being a large number of policy statements, manual analysis is not an option. Many policy specification languages including SPKI/SDSI cannot specify constraints such as negative credentials, mutual exclusion, etc. necessitating a mechanism to make sure that agents didn't make policy statements which violate the unspecifiable constraints. In addition, analysis of current policy state is useful to formally reason whether the current policy state satisfies user defined properties and to gain knowledge for making decisions about changing the current policy state. Based on current policy state, we can also reason about issues of authorization and naming in future time.

We also propose *policy audit*, where we check for policy violations over a set of all the policy statements which were valid during the time of audit. In access control audit, the problem is of type: *did principal K access a resource R?*, where as in policy audit we check *whether a principal K had permissions to access a resource R?* or *whether principal K_1 gave authorization regarding a resource R to principal K_2*. Policy audit is a means to ascertain whether trusted agents were really trustworthy with regards to policy specification. By suitably shifting the timeline, we can use FTPL for purpose of reasoning about current policy state as well as for policy audit. For purposes of policy audit, we collect the set of all policy statements corresponding to the time of audit and evaluate the formulas of the logic over the static collection of policy statements labeled with validity periods.

The logic we propose would provide the resource administrator with a formal and a high level mechanism for specifying policy properties. The language could also be useful for the other types of users, like a client to reason about properties such as *"is there an authorization chain leading to the client from a particular principal?"* etc.

The paper will consist of five other sections, section [2] presents short description of related work. Section [3] consists of introduction to SPKI/SDSI and previous models for certificate analysis problems. Section [4] will consist of our proposed logic and examples of policy problems specifiable by the language and section [5] will contain algorithms to evaluate the formulas of the logic. Section [6] consists of conclusion and future work.

2 Related Work

There are other logics for SPKI/SDSI which model the name resolution, authorization and other features of SPKI/SDSI: Martin Abadi's logic to model local name spaces of SDSI [Aba97], Halpern and van der Meyden's Logic of Local Name Containment to characterize SDSI name resolution which was extended to deal with other SPKI issues like revocation, expiry dates, and tuple reduction [HvdM01]. Ninghui Li [Li00] proposes a logic program to describe the name resolution algorithm which also handles authorization certificates and threshold subjects. The purpose of our logic is neither to model the features of SPKI/SDSI nor to provide for its semantics but for policy analysis. Jha and Reps [JR01] propose using temporal logic to reason about certificate chain derivations in a given SPKI system, while we propose using temporal logic as a language for specifying certificate set analysis problems involving authorization, naming and validity.

There have been many languages, logic and semantic frameworks to express authorization policy in a distributed system. Datalog and its variants seem to be the language of choice to specify authorization policy [LM03]. A specific class of distributed access control systems attracting much attention are the Trust Management Systems [BFK99], [BFIK99]. The concept of Trust Management is that there exists a standard mechanism for expressing the access control policy and also a standard method to verify that an access request complies with the policy. This is called "Proof of Compliance". SPKI/SDSI was not intended by the authors to be a Trust Management System in that Proof of Compliance can be application dependent, but it can be viewed as a Trust Management System with standardized certificate chain reduction rules [EFL+99]. Though lot of work has been done in specifying policy and checking whether an access request is compliant with the policy in a distributed access control system, not much literature exists regarding a language based mechanism for detailed analysis of policy beyond simple authorization problems. To our knowledge FTPL is the first such language for high level policy analysis in not just a SPKI/SDSI system, but in the distributed access control framework. While we give a logic to formulate policy analysis problems in the context of SPKI/SDSI we feel that full fledged languages for policy analysis in other distributed access control systems and trust management systems are of much use to the users of the system.

We give a list of specifiable problems that could be of interest in SPKI/SDSI policy analysis including some given by Jha and Reps [JR01]. The problems they list can be qualified by time. For example Authorized Access 1:"Given resource R and principal K, is K authorized to access R?", can be more specifically written in the context of time as:"Given Resource R and principal K is K authorized to access R at a particular instant of time, or at some point in a time interval?" While Jha and Reps provide a lattice based time structure which can be used to model such problems, we give a logic based formalism to specify such problems.

3 SPKI/SDSI

3.1 Introduction

SPKI and SDSI are certificate based mechanisms for authorization and naming in distributed systems respectively. SPKI 1.0 : simple public key infrastructure, was a mechanism for expressing authorizations and delegations, where it was proposed that permissions be given directly to the public keys of entities. The most important contribution of SDSI: simple distributed security infrastructure, was to create local namespaces distinguished by the unique public key of the entity defining the names, instead of trying to create a globalized namespace. Features of SPKI 1.0 and SDSI 1.0 were integrated into SPKI/SDSI 2.0 [EFL+99]. In the future if we say SPKI we mean SPKI/SDSI 2.0 unless mentioned otherwise. One of the features of the SPKI is that every principal is free to issue certificates signed by himself unlike in X.509 PKI framework [ADT02] where there are separate set of principals called certificate authorities(CA) who are trusted to issue certificates bearing their signature.

In SPKI/SDSI resource owners can delegate access rights to trusted entities and they can issue certificates authorizing others, leading to a chain of certificates from the resource owner to the end user. In a SPKI/SDSI system principals and resources are identified by their corresponding public keys. We can still associate names with the principals. Every principal has a local name space and the principal is free to define local names in his domain independent of others. For example for a UIC student John, university may be defined as K_{UIC}, while for a UIUC student Jim, university may be defined as K_{UIUC}. K_{UIC}, K_{UIUC} are the public keys of the universities UIC and UIUC respectively. Public keys being unique throughout the system, a name qualified by the public key of the principal defining the name, is also unique.

Definition 1: An identifier is a word over a given alphabet. The set of identifiers is denoted by $\mathcal{I}$. The set of keys is denoted by $\mathcal{K}$.

Definition 2: A local name is a sequence consisting of a key followed by a single identifier. A local name K *friend* may be defined as $K_{bob}, K_{jim}, ...$ etc

There is another kind of name in SPKI/SDSI called extended name which increases the level of indirection in the naming scheme.

Definition 3 : An extended name is a sequence consisting of a key followed by two or more identifiers. An example of an extended name is K *brother friend*, where the meaning of *friend* would be evaluated in the context of $K's$ *brother*.

Definition 3: A Name is either a local name or an extended name. We denote the set of all names as $\mathcal{N}$.

Definition 4: A Term is either a key or a name. We use $\mathcal{T} = \mathcal{K} + \mathcal{N}$ to denote set of all terms.

Definition 5: We define a fully qualified name also to mean a public key or a local name or an extended name.

3.2 Certificate Structure

There are two types of certificates or certs in SPKI/SDSI: name certs and authorization(auth) certs. The function of a name cert is to define a local name as another term. Only the principal, in whose namespace the local name is defined, may issue the corresponding name certificate. In an auth cert a principal can grant or delegate permissions regarding accessing a resource to another term. We now describe the logical structure of the SPKI/SDSI certificates.

There are four fields in a name certificate (K, A, S, V) which is signed with the private key of the issuer: K^{-1}. K is the public key whose namespace is being considered, A is an identifier in $\mathcal{I}$ and S is an element of $\mathcal{T}$, i.e. it can be another public key, or a local name or an extended name. S is the term implied by the local name KA . V is a validity specification which states the validity condition for the certificate. The basic validity specification is of form $[t_1, t_2]$ where t_1, t_2 are absolute time constants. The certificate is said to be valid from time t_1 *to* t_2. If either t_1 or t_2 are not given we assume $-\infty$ or $+\infty$ respectively. Validity specification could also be a certificate revocation list or an online check [CEE+01]. In our model of SPKI we consider a validity specification to be a certain time period in the interval 0 to ∞.

The authorization certificates consist of five fields $(K, S, D, \mathcal{E}, V)$ signed by the private key of the issuer: K^{-1}. The main purpose of an SPKI authorization certificate is to grant or delegate a set of authorization actions as specified by $\mathcal{E}$ to the subject $S \in \mathcal{T}$.

K is the public key of the certificate issuer.

S is the Subject which can be either a key or a name.

D is the boolean delegation bit which controls the delegation rights.

$\mathcal{E}$ is the set of authorization rights which are granted or delegated to the subject by the issuer. Note that authorization actions have address of the resource encoded in them. In our logic when we say *read* we mean *read* of a particular resource R defined in the context.

V is the Validity specification and the observations made above for name certs are applicable here also.

Delegation bit D implies that when the bit is set to one then the subject can access and also delegate the rights specified by set $\mathcal{E}$ to other users in the system, if the bit is zero then the subject only gets the right to perform actions specified by $\mathcal{E}$, but cannot further delegate these rights to any other user.

3.3 Tuple Reduction

In a delegation based distributed access control system the resource owner permits others to specify policy regarding the access of the resource. Entities entrusted with delegation rights further propagate the access rights. This leads to a chain of certificates for accessing a resource. The principal requesting access would present the resource administrator with a requested authorization action and a chain of certificates which should prove that the requesting principal has the right to perform the requested action. Given a chain of certificates, one must deduce authorization and validity information from the chain. The rules which specify the inference conditions are called the tuple reduction rules. When an entity receives a set of certificates it verifies them for integrity and stores them in an unsigned form called the tuples. We use the terms certificates and tuples interchangeably as far as there is no confusion.

Consider a certificate issued by K_1 to a key K_2 with authorization actions $\mathcal{A}_1$ and and validity specification V_1, suppose delegation bit D_1 is also true. Then the certificate is logically represented as 5-tuple

$$C_1 : (K_1, K_2, D_1, \mathcal{A}_1, V_1)$$

Given D_1 to be true, let K_2 delegate authorization actions $\mathcal{A}_2$ to subject S_2, with validity condition V_2 and delegation bit D_2.

$$C_2 : (K_2, S_2, D_2, \mathcal{A}_2, V_2).$$

According to the tuple reduction rules the result of composition of the certificates C_1 and C_2 in that order is another certificate C' equivalent to the chain

$$C_1 \circ C_2 \equiv (K_1, S_2, D_2, AIntersect(\mathcal{A}_1, \mathcal{A}_2),$$

$$VIntersect(V_2, V_2)).$$

These equivalent certificates can also be used as a regular certificates in the SPKI system and are called Certificate Result Certificates (CRC).

$AIntersect()$ is the intuitive intersection of authorization action sets. Howell and Kotz [HK00] note that the intersection may not always be well defined. For the purpose of our logic we consider authorization actions to be simple constants. Date range intersection $VIntersect()$ is the intersection of corresponding validity intervals.

3.4 Models for SPKI Certificate Analysis.

There are two primary models for tuple reduction problems, one is the string rewriting model of Clarke et al [CEE+01] and the other is the push down system(PDS) based semantics of Jha and Reps [JR02]. We present the PDS based model of [JR02]

3.4.1 Push Down System based Semantics

Jha and Reps [JR01] model tuple reduction problems as configuration reachability problems in a push-down system. A push down system is similar to a push down automata but without the input alphabet. They view the authorization problem: *can a principal K_p access resource a R* as a configuration reachability problem in the pushdown automata and use the PDS model checking algorithms [EHRS00], [BEO97] to answer the problem. The configuration of a PDS contains information about the control state and stack state of the PDS at a particular point in the computation. A configuration of PDS corresponds to a term in a SPKI system. If a configuration G_T reaches another configuration G_S using the state transition rules, than the corresponding term T can resolve to term S, defining the "term reachability semantics" for the SPKI/SDSI system. PDS State transition rules correspond to the SPKI/SDSI certificates. The PDS formalization is also more expressible than the rewriting system of Clarke et al [CEE+01] to formulate various certificate analysis problems. We now describe the method to build a push down system from a given set of certificates and then model the certificate analysis problems using configuration reachability relation of the PDS. The primary issues of certificate analysis are that of name resolution and authorization derivability. For purposes of the algorithm we convert name certs of form (K, A, S, V) into the rewriting rule $KA \rightarrow S$, auth cert $(K, S, d, \mathcal{E}, V)$ is written as rule $K\square \rightarrow S\square$ if $d = 1$ else as $K\square \rightarrow S\blacksquare$ if $d = 0$. V and $\mathcal{E}$ are ignored for now, we consider that each resource has only one type of access right.

A pushdown system is formally defined as a triple $\mathcal{P} = (Q, \Gamma, \Delta)$, where Q is a finite set of state variables, Γ is a finite set of stack symbols, and $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma^*$ is the finite set of state transition rules. If $(q, \gamma, q', w) \in \Delta$ then we write $(q, \gamma) \hookrightarrow (q', w)$. The PDS $\mathcal{P}$ when in state q and γ on top of stack, uses the above transition rule to goto state q', popping γ and writing string w onto the stack. The $\hookrightarrow$ corresponds to the rewriting relation between various elements of a SPKI certificate. A configuration of $\mathcal{P}$ is a pair $\langle q, w\rangle$ where $q \in Q$ and $w \in \Gamma^*$. Here q denotes the control state and string w the stack state. The set of all configurations is denoted by $\mathcal{G}$. Corresponding to the transition relation of the PDS states we have the (immediate) reachability relation of the configurations. If $(q, \gamma) \hookrightarrow (q', w)$ then for all $v \in \Gamma^*, \langle q, \gamma v\rangle \Rightarrow \langle q', wv\rangle$, i.e. configuration $\langle q, \gamma v\rangle$ is said to be the immediate predecessor of $\langle q', wv\rangle$. The reflexive transitive closure and the transitive closure of the immediate reachability relationship($\Rightarrow$) are denoted by $\Rightarrow^*$ and $\Rightarrow^+$ respectively. A *run* of a *PDS* is a sequence of configurations $c_0, c_1, ...c_n$ such that c_i is an immediate predecessor of c_{i+1}. The run of a PDS corresponds to a SPKI certificate chain reduction.

Given a set of certs $\mathcal{C}$ with principals(keys) represented by $\mathcal{K}$ and identifiers by $\mathcal{I}$, we construct a PDS $\mathcal{P}_{\mathcal{C}}$ (Q, Γ, Δ) as follows.
The set of control states is the set of keys, $Q = \mathcal{K}$. Note that we only analyze authorization problems concerning a particular resource which is identified by a special key R in $\mathcal{K}$. The stack alphabet is set of identifiers and the delegation symbols, $\Gamma = \mathcal{I} \cup \{\square, \blacksquare\}$. The symbol $\square$ implies permission to delegate while $\blacksquare$ just grants access. The set Δ of state transitions contains a transition $(K\ \gamma \hookrightarrow K'\ \omega)$ for every cert rule $(K\ \gamma \rightarrow K'\ \omega)$. The term $K_1\ A\ B$ corresponds to the PDS configuration $\langle K_1, A\ \ B\rangle$, term K corresponds to $\langle K, \epsilon\rangle$. When the PDS is in configuration $\langle K_1,\ \ A\ \ B\rangle$ and there is a state transition rule(cert) $K_1\ \ A \hookrightarrow K_2$ then it goes to configuration $\langle K_2,\ \ B\rangle$. The reachability relation $\Rightarrow^*$ defines set of all terms which a given term can resolve to. For authorization derivability, the PDS configuration is of the form $\langle K, \square\rangle$ which means principal K can access and delegate permissions, or of the form $\langle K, \blacksquare\rangle$ which means K can just access. A principal K can access the resource R if there is a run of the PDS from configuration $\langle R, \square\rangle$ to either of the configurations $\langle K, \square\rangle$ or $\langle K, \blacksquare\rangle$. Whether a term can resolve into another term can be decided in the PDS system in time $O(n^2 L_{\mathcal{C}})$ where n is the total number of keys in the certificate system $\mathcal{C}$, and $L_{\mathcal{C}}$ is the sum of lengths of the right hand side(rhs) of all the certificates in the system. Length of the rhs of a cert is the number of non-key symbols on the rhs of the rule corresponding to the cert.

We use term reachability semantics of the PDS model to define the FTPL semantics. For that purpose we modify the PDS described above by adding an authorization variable v to the system to form a augmented PDS $\mathcal{P} = (Q, \Gamma, \delta, v)$. The authorization rights a principal grants to another principal are computed as the side-effects of the run of the PDS using the authorization variable v. At the start of a computation v is initialized to $\mathcal{A}$, the set of all authorization rights for the resource being considered. Each auth cert rule is labeled with the authorization rights granted to the subject of the cert. Accordingly, we also label the corresponding state transition rule in δ.

In the PDS $\mathcal{P}$ whenever we go from one configuration to another configuration using a state transition rule labeled with a set of authorization rights $\mathcal{E}$, we update the authorization variable v as $v = v \cap \mathcal{E}$. If the PDS goes from configuration $\langle K_1, \square\rangle$ to $\langle K_2, \square\rangle$ with the value of v as $\mathcal{A}_1$ at the end of the run, then K_1 directly/indirectly delegates to K_2 the authorization rights $\mathcal{A}_1$.

4 Logic for Policy Analysis

4.1 Definition of the Logic

We define a polyadic First order Temporal Policy analysis Logic (FTPL) as a language for specifying properties of a SPKI/SDSI certificate system. The formulas of the logic are to be evaluated on a semantic model of the SPKI/SDSI system. The policy analysis problem would be specified as a FTPL formula f which is interpreted on a given set of certificates at a particular instance of time. Users can choose application dependent semantics for the language, but we use the standard tuple reduction semantics [CEE+01] with the view of SPKI as a trust management system. The constructs of the logic deal with the issues of authorization, naming and validity. Though we model time as discrete in this paper, we can easily extend FTPL to interpret over continuous time.

The authorization reachability issue is modeled by the predicate $authorize(i, j, d, e)$, called the authorization reachability predicate, where i is a key or a variable and j is a fully qualified name or another variable. The components of *authorize* predicate d, e specify the delegation and authorization rights respectively. The boolean delegation control $d \in \{0, 1\}$, with 1 implying permission to access and delegate and 0 just granting access. The authorization control e is a subset of a set of constants $\mathcal{A}$, which specifies the set of authorization actions for a particular resource. The predicate $authorize(i, j, d, e)$ is true at a particular instance of time t with respect to a given set of certificates $\mathcal{C}$, if there is a chain of certificates, where the principal denoted by i directly/indirectly transfers the rights d, e to the principal or name denoted by j. Name certificates are not just mechanisms to resolve names to principals but can also be used to delegate permissions [Li00]. To reason exclusively about name resolution we define the predicate $resolve(p, q)$ where p is either a local or an extended name and q is either a fully qualified name or a variable. The predicate reasons whether the name denoted by p resolves to the term denoted by q using the name certificates given in $\mathcal{C}$. The type of a variable in case of both *authorize* and *resolve* is the set of all principals in the system. The only atomic formulas of the logic, are the authorization reachability predicate $authorize(i, j, d, e)$ and the name resolution predicate $resolve(p, q)$. The rest all formulas are combinations of the predicates with the associated FTPL operators. The operators of the logic consist of the usual boolean connectives $\neg$ *and* $\wedge$, the temporal nexttime operator $\mathbf{X}$, the bounded until operator $\mathbf{U}_{[t_1,t_2]}$ where t_1, t_2 are positive integers and $t_1 \leq t_2$, the principal quantifier $\exists$ where the universe of discourse is the set of principals(keys) present in the certificate system $\mathcal{C}$, over which the formulas of the logic are interpreted. These are the basic operators. We assume we have set $\mathcal{V}$ of variables. All the variables range over the set of principals present in the given certificate set $\mathcal{C}$. Let $\mathcal{K}$,$\mathcal{T}$ be the set of keys and terms present in certificate set $\mathcal{C}$ respectively and $\mathcal{A}$ be the set of authorization actions for a particular resource R.

The syntax of the logic is inductively defined as follows:

1)) If $i \in \mathcal{K} \cup \mathcal{V}$, $j \in \mathcal{T} \cup \mathcal{V}$, $d \in \{0, 1\}$ and $e \subseteq \mathcal{A}$ then $authorize(i, j, d, e)$ is a FTPL formula.

2) If $p \in \mathcal{N}$ and $q \in \mathcal{T} \cup \mathcal{V}$ then $resolve(p, q)$ is a FTPL formula.

3) If f and g are formulas of the logic, then $\neg f$ and $f \wedge g$ are also FTPL formulas.

4) If f and g are formulas of the logic $\mathbf{X}f, f\ \mathbf{U}_{[t_1,t_2]}\ g$, are also FTPL formulas.

5) If f is a formula of the logic then $(\exists i f(i))$is also a FTPL formula.

A variable i is bound in a formula f, if every occurrence of i in f is in the scope of a quantifier over i. If i occurs in f and is not bound then we call i a free variable of f. We use free variables in the formula to formulate queries over a SPKI system.

The formula will return a set of evaluations which when substituted for the free variables would make the logical formula true in the context of the given certificate system $\mathcal{C}$ at a particular instant of time t. For a formula f, let $\mathit{free\text{-}var}(f)$ denote the set of free variables of f. An evaluation say ρ for the formula f is a mapping from $\mathit{free\text{-}var}(f)$ to the set of principals, $\rho : \mathit{free\text{-}var}(f) \to \mathcal{K}$. We extend the domain of ρ to $(\mathit{free\text{-}var}(f) \cup \mathcal{T})$ so that for every $T \in \mathcal{T}, \rho(T) = T$. An interpretation for a formula f is a triple $(\mathcal{C}, t, \rho)$ where $\mathcal{C}$ is a set of certificates, $t \geq 0$ is a time instance and ρ is an evaluation.

We give the semantics of a formula f over an interpretation $(\mathcal{C}, t, \rho)$. For any certificate C let $C.val$ be the validity time interval of C. For a given set $\mathcal{C}$ of certificates and time t, let $\mathcal{C}_t = \{C \in \mathcal{C} | t \in C.val\}$. $\mathcal{C}_t$ denotes the set of certs valid at time t and $\mathcal{P}_{\mathcal{C}_t}(Q, \Gamma, \delta, v)$ denotes the PDS constructed from the set of certs $\mathcal{C}_t$ as described in previous section.

We define the satisfiability relation $\models$ between an interpretation and a FTPL formula as follows:

$(\mathcal{C}, t, \rho) \models \mathit{authorize}(i, j, d, e)$ if the PDS constructed from set of certificates $\mathcal{C}_t$, $\mathcal{P}_{\mathcal{C}_t}$ can go from configuration $\langle \rho(i), \Box \rangle$ to the configuration $\langle \rho(j), \Box \rangle$, with authorization variable $v \supseteq e$, at the end of the computation, when $d = 1$ i.e. $\langle \rho(i), \Box \rangle \Rightarrow^* \langle \rho(j), \Box \rangle$. If $d = 0$ the PDS $\mathcal{P}_{\mathcal{C}_t}$ can reach either the above configuration or the configuration $\langle \rho(j), \blacksquare \rangle$.
Note that if the key $\rho(i)$ directly/indirectly issues term $\rho(j)$ with greater delegation and authorization rights than specified by d and e respectively, the *authorize* predicate still holds true.

$(\mathcal{C}, t, \rho) \models \mathit{resolve}(p, q)$ if the PDS $\mathcal{P}_{\mathcal{C}_t}$ can go from configuration corresponding the name p, to the configuration corresponding the term $\rho(q)$.

$(\mathcal{C}, t, \rho) \models \neg f$ iff $(\mathcal{C}, t, \rho) \not\models f$

$(\mathcal{C}, t, \rho) \models f \wedge g$ iff $(\mathcal{C}, t, \rho) \models f$ and $(\mathcal{C}, t, \rho) \models g$

$(\mathcal{C}, t, \rho) \models \mathbf{X} f$ iff $(\mathcal{C}, t+1, \rho) \models f$.
$\mathbf{X} f$ is true at an instance of time t iff f is true in the next instance of time $t+1$.

$(\mathcal{C}, t, \rho) \models f \ \mathbf{U}_{[t_1, t_2]} \ g$ iff $\exists t' \in [t + t_1, t + t_2]$ such that $(\mathcal{C}, t', \rho) \models g$ and $\forall t'' : t \leq t'' < t' \ (\mathcal{C}, t'', \rho) \models f$. $f \ \mathbf{U}_{[t_1, t_2]} \ g$ is true at time t, iff formula f is true from time t to t', where at t', g will be true and $t' - t$ should be within bounds of $[t_1, t_2]$.

$(\mathcal{C}, t, \rho) \models \exists i f(i)$ iff there exists an $K_1 \in \mathcal{K}$ such that $(\mathcal{C}, t, \rho') \models f(K_1)$ where ρ' is a restriction of the domain of ρ to $(\mathit{free\text{-}var}(f(K_1)) \cup \mathcal{T})$.

For convenience of expression we introduce derived operators $\vee, \Diamond, \Box, \forall, \triangleleft$ which are extensions of the basic FTPL operators.

$$f \vee g \equiv \neg(\neg f \wedge \neg g)$$

$$\Diamond_{[t_1, t_2]} f \equiv \mathit{True} \ \mathbf{U}_{[t_1, t_2]} \ f$$

$$\Box_{[t_1, t_2]} f \equiv \neg \Diamond_{[t_1, t_2]} \neg f$$

$$\forall i f(i) \equiv \neg \exists i \neg f(i)$$

$$\exists i \triangleleft G \ f(i) \equiv \exists i (\mathit{resolve}(G, i) \wedge f(i))$$

$$\forall i \triangleleft G \ f(i) \equiv \forall i (\mathit{resolve}(G, i) \supset f(i))$$

$\vee$ is the standard propositional operator. $\Diamond, \Box$ are the standard temporal logic operators. $\forall$ is the universal quantifier. $\triangleleft$ is the name resolution operator. $\Diamond_{[t_1, t_2]} f$ states whether f is true during any time instance from t_1 to t_2. $\Box_{[t_1, t_2]} f$ states whether formula f is true throughout time interval t_1, t_2. Intuitively the name resolution operator $\triangleleft$ resolves a name (local or extended) to its principals according to the SDSI name resolution principles and then evaluates the formula over the restricted domain.

Given a formula f with free variables and a set of certificates $\mathcal{C}$ and a time instance t, we interpret the formula f as a query returning a set of tuples, which are evaluations satisfying the formula f. For example $\mathit{authorize}(R, x, 1, \{r\})$, where x is a free variable, denotes a query which returns all the principals x who have the permission to delegate right r of the resource R at the current instance of time.

4.2 Using the Logic to analyze the state of a given SPKI system

All the formulas are evaluated in the context of a given certificate set $\mathcal{C}$ and with respect to a particular resource R unless otherwise specified. Constructing the required set of certificates $\mathcal{C}$ for evaluating a formula in a distributed environment is non-trivial. Like others [CEE$^+$01], [JR02] we assume that we have the required set of certificates to evaluate the formulas of the logic. Distributed database lookup and goal-based certificate discovery methods seem to be promising approaches in retrieving the required set of certificates dynamically, especially for the SDSI part of the system [LWM01].

From the perspective of a resource administrator a practical guideline seems to be to cache all the certificate chains which are presented as a proof of compliance by the access requesting clients. The resource administrator may cache these certificates in a secure storage and use the FTPL logic as a means of offline Policy Audit. Though the view that certificates don't exist till they are used seems to be suited for policy audit, it can have unanticipated results as we show with an example later in the section. The language FTPL can be used in two analysis modes with respect to time. By defining the origin of time "0" to be current time instance, we can reason about the current state of the policy. For auditing a set of policy statements made from a particular time t in the past we can time shift the origin of time to t. We now illustrate the use of language to specify policy analysis problems.

This formula specifies that at every instance during the period $[t_1, t_2]$ atleast one principal in the set defined by fully qualified name G has 'w' access to resource R (Group availability property):

$$\Box_{[t_1,t_2]} \exists i \triangleleft G \; authorize(R, i, 0, \{w\})$$

This formula specifies whether a blacklisted principal K_B ever had 'read' access to a resource R during the time of audit $[0, t_a]$. Here "0" refers to the start time of the audit, and "t_a" the end time of the audit:

$$\Diamond_{[0,t_a]} authorize(R, K_B, 0, \{read\})$$

This formula specifies that principal K will be able to delegate 'read' right for the file R in future:

$$\Diamond_{[0,\infty]} authorize(R, K, 1, \{read\})$$

This formula specifies that given resource R and name G, there is an authorization chain granting G permissions to perform action 'w' on R at current time:

$$authorize(R, G, 0, \{w\})$$

This formula specifies that given resource R and name G, all the principals denoted by G are authorized to perform action 'w' on R at current time:

$$\forall i \triangleleft G(authorize(R, i, 0, \{w\}))$$

Note that the later formula implies the former, but not vice versa.

This formula specifies that both the principals K_1 *and* K_2 have simultaneous 'write' access to a resource R at some time in the future(mutual exclusion):

$$\Diamond_{[0,\infty]}[(authorize(R, K_1, 0, \{write\}) \wedge authorize(R, K_2, 0, \{write\}))]$$

This formula specifies that both the principals K_1 *and* K_2 have 'write' access to a resource R at some time in the future:

$$\Diamond_{[0,\infty]}(authorize(R, K_1, 0, \{write\}) \wedge \Diamond_{[0,\infty]} authorize(R, K_2, 0, \{write\}))$$

This formula specifies that a principal belonging to local name $K_{foo}\, Accountant$ is directly/indirectly granting write permissions for a resource at some point of time to a principal of local name $K_{foo}\, Purchaser$ (conflict of interest).

$$\Diamond_{[0,\infty]} \exists i, j(authorize(i, j, 0, \{write\}) \wedge resolve(K_{foo}\, Accountant, i) \wedge resolve(K_{foo}\, Purchaser, j))$$

Given resource R , this query returns all the principals authorized to perform actions 'r' and 'w' on R at time t_a:

$$\Diamond_{(t_a,t_a)} authorize(R, x, 0, \{r, w\}).$$

Where x is the free variable which returns all valuations satisfying the above formula.

This formula specifies whether principal K_1 can access resource R before K_2:
$\neg authorize(R, K_2, 0, \{r\})\ \mathbf{U}_{[0,\infty]}$

$$authorize(R, K_1, 0, \{r\})$$

All the previous properties and queries assumed that we evaluate the formula against a single certificate system $\mathcal{C}$. In the following query we need to consider different set of certificates for different subcomponents of the formula.

This query returns all the principals who will be excluded from performing action 'w' currently on the resource R if all the certificates issued by a compromised key K, $\mathcal{C}_K \subseteq \mathcal{C}$ are revoked now.

$$\{x| \, (\mathcal{C}, 0, \emptyset) \models authorize(R, x, 0, \{w\})\} -$$

$$\{x| \, (\mathcal{C} - \mathcal{C}_K, 0, \emptyset) \models authorize(R, x, 0, \{w\})\}$$

Certain problems like *"is there a authorization chain from the resource R to a principal K without involving the compromised key K'"*, are not directly expressible in the logic FTPL. By using the above mentioned method we can still answer such questions.

Reasoning about roles in a SPKI system.
Li [Li00]states that local names in the SPKI framework can be interpreted as "distributed roles". Distributed in the sense that they are not specified by a centralized authority in the traditional sense of the roles. Roles are thought of as an abstraction for a set of users and a set of permissions, they can include other roles too [San98]. Roles can be implemented in a SPKI system using local names by giving permissions directly to local names instead of principals. Principals inherit privileges by virtue of being members of a "local name". This approach is useful to simplify policy specification in many real-world scenarios. We can use FTPL to reason about the interpretation of local name as roles. The predicate *resolve* can be used to reason about role membership and role hierarchy specified in a distributed fashion. The following formula specifies that role corresponding local name R_2 dominates that of R_1 : $resolve(R_1, R_2)$

Static separation of duty: This formula specifies that two roles R_1 and R_2 have the same user as a member in both the roles at same time.

$$\Diamond_{[0,\infty]} \exists i((resolve(R_1, i) \wedge resolve(R_2, i))$$

Containment: This formula specifies that there is a principal not contained by role R having 'r' access to a resource P at any point of time:

$$\Diamond_{[0,\infty]} \exists i \, (authorize(P, i, 0, \{r\}) \wedge \neg \, resolve(R, i))$$

We can also use FTPL to reason about trust violations in a SPKI system. Consider the following scenario :
A resource K_{Univ} in a university, which the students in CS and ECE dept need to have read access, some teaching assistants and special students(say research assistants) in CS dept also need to have a write access. But no non-CS student is supposed to have a write access to the resource. Given that it is not possible to specify negative permissions in a SPKI system the resource administrator has to trust that the principals with delegation authority will not violate the university policy.

$(K_{Univ}, K_{CSDept} students, \{read\}, 0, \{t_1, t_2\})$
$(K_{Univ}, K_{ECEDept} students, \{read\}, 0, \{t_1, t_2\})$
$(K_{Univ}, K_{CSDept}, \{read, write\}, 1, \{t_1, t_2\})$: So that the CS dept admin can give write permissions to selected TA's and some special CS students.
$(K_{CSDept}, K_{TA1}, \{read, write\}, 0, \{t_1, t_2\})$
$(K_{CSDept}, K_{TA2}, \{read, write\}, 1, \{t_1, t_2\})$: TA2 can give write permission to some special CS students.
$(K_{CSDept}, students, K_{studenti}, \{t_1, t_2\})$: Name Certificate for CS student 'i' including TA's and special CS students.

In the above system the CS admin or TA2 can violate the "university policy" by giving permissions to a non-CS student.

The university resource admin wants to check whether there is a Non-CS student having write access to the resource.

$$\Diamond_{[0,\infty]} \exists i \, [(authorize(K_{Univ}, i, 0, \{write\}) \wedge$$

$$\neg resolve(K_{CSDept} students, i)]$$

Evaluating a FTPL formula on a partial set of certificates.
In the case where we evaluate the above formula over a set of certificates obtained by caching the previous access request chains we may get a false-positive. Consider the scenario where TA2 from start uses the following chain to get resource authorization,

$$[(K_{Univ}, K_{CSDept}, \{read, write\}, 1, \{t_1, t_2\})\circ$$

$$(K_{CSDept}, K_{TA2}, \{read, write\}, 1, \{t_1, t_2\})]$$

then the name cert validating TA2 to be a student of CSDept is not cached by the resource administrator leading to a false positive. When the resource administrator catches a possible policy violation he can use manual resolution or a goal-directed procedure to confirm the result. The other type of violations are caused by lack of authorization certs, for example when a principal grants permissions to a blacklisted principal, but the blacklisted principal doest not access the resource and no chain involving the blacklisted principal is sent as a proof of authorization, then the corresponding formula evaluated over the cached set of certs returns a false answer. But this is a benign violation in the sense that though a certificate has been issued in violation of the policy, that certificate has not been used. In this case the serious problems which arise because of evaluating a formula on a partial state of the system are due to lack of name certificates. Goal directed algorithms for retrieving the distributed set of SDSI name certificates already exist in the literature [LWM01].

5 Evaluating the formulas

In this section, we give an algorithm that takes a certificate system $\mathcal{C}$ and a FTPL formula f and outputs a set of pairs of the form (ρ, t) such that f is satisfied with respect to the evaluation ρ at time t, i.e., $(\mathcal{C}, t, \rho) \models f$. For ease of presentation of the algorithm, we assume that there is only one resource and one access right. It can be easily generalized to the case of multiple access rights. If f has k free variables, the algorithm actually computes a relation R_f which is a set of $(k+1)$-tuples such that the first k values in each tuple define an evaluation ρ and the last value in the tuple is a list of time intervals such that for all instances t in these time intervals $(\mathcal{C}, t, \rho)$ satisfies f. Actually, we compute the relations R_g for each subformula g of f. These relations are computed inductively in increasing lengths of g. In the base case, g is an atomic subformula which is of the form $authorize(i, j, d, e)$ or is of the form $resolve(p, q)$. Recall that for the atomic formula $authorize(i, j, d, e)$, i can be a variable or a key, and j can be a variable or a term. We assume that j is a variable or a key (if j is a term then by introducing new keys and new rules we can reduce it to a case where j is a key. For example in order to evaluate a predicate $authorize(K_1, K_2\ A_1\ , d, e)$ over certificate set $\mathcal{C}$, we add new key K_3 and a new rule $K_2 A_1 \rightarrow K_3$ and evaluate $authorize(K_1, K_3, d, e)$ in the augmented system. The total number of new symbols and rules we introduce is bounded by the sum of right hand sides of the certificate rules). Similarly we assume that in $resolve(p, q)$, p can be a name and q can be a key or a variable. In the first step our algorithm, we compute the relations R_g for the case when g is an atomic formula. In the second step, we compute the relations R_g for the case when g is not atomic.

In the first step, the algorithm operates as follows. Recall that each certificate in the set $\mathcal{C}$ is associated with a valid interval. We assume that t_{min} and t_{max}, respectively, are the minimum of the beginning points and the maximum of the end points of the validity intervals of certificates in $\mathcal{C}$. Let m be the number of certificates in $\mathcal{C}$. Using the validity intervals of the certificates, we compute a sequence of increasing times $T_0, T_1, ..., T_{l-1}$(These sequences can be easily computed from the sequence obtained by taking the begin and end times of all the validity intervals of certificates in $\mathcal{C}$ and sorting them in increasing order) and a sequence of sets of certificates $\mathcal{C}_0$,...,$\mathcal{C}_{l-2}$ such that $l \leq 2m$, $T_0 = t_{min}$, $T_{l-1} = t_{max} + 1$ and for each i, $1 \leq i < l$, the set of certificates in $\mathcal{C}$ that are valid at every time instance in the interval $[T_i, T_{i+1} - 1]$ is the same and is given by $\mathcal{C}_i$. We can see that the complexity of the above procedure is dominated by the complexity for sorting the time points and hence is $O(m \log m)$.

For each $i = 0, ..., l-2$ we do as follows. Using the set of certificates $\mathcal{C}_i$, for each atomic formula g of f we do as follows. Consider the case when g is $authorize(p, q, d, e)$. In this case, both p, q are either a variable or a key. We compute the set $Eval_{g,i}$ of evaluations ρ for g such that the following property is satisied: if $d = 1$ then the the PDS $\mathcal{P}_{\mathcal{C}_i}$ can go from the configuration $\langle \rho(p), \Box \rangle$ to the configuration $\langle \rho(q), \Box \rangle$; if $d = 0$ then the PDS can go to the above configuration or to the configuration $\langle \rho(q), \blacksquare \rangle$. Note that if p is not a variable then $\rho(p) = p$; similar condition holds for q. If neither of p, q is a variable then ρ is the empty evaluation; in this case either $Eval_{g,i}$ contains the empty evluation indicating the satisfaction of g at every time instance in the interval $[T_i, T_{i+1}-1]$ or it is the empty set indicating the non-satisfaction of g in the above interval. If g is $resolve(p, q)$ then we compute the set $Eval_{g,i}$ of evaluations ρ such that the above PDS can go from the configuration corresponding to the name p to the configuration corresponding to the term $\rho(q)$ (recall that p has to be a name and q can be a variable or a term; also, recall that the configuration corresponding to the name of the form $K\ A\ B$ is $\langle K, A\ B \rangle$).

Now it should be easy to see how R_g can be calculated from the sets $Eval_{g,i}$ for $i = 0, ..., (l-2)$. Recall that, if g has k free variables then R_g is a set of $(k+1)$ tuples (here $k \leq 2$). For each evaluation ρ that appears in at least some $Eval_{g,i}$, R_g has a single tuple whose first k components give the evaluation ρ and whose $(k+1)^{st}$ component is the list of all time intervals $[T_i, T_{i+1}-1]$ such that ρ is in $Eval_{g,i}$.

In the above procedure, we defined the sets $Eval_{g,i}$ using a PDS $\mathcal{P}_{\mathcal{C}_i}$. However, we plan to employ a And-or graph from which the sets $Eval_{g,i}$ can be computed for all g using a single fix point computation.

Now we describe the And-or graph construction for a set $\mathcal{D}$ of certificates. First using the certificate set $\mathcal{D}$, we first define a nondeterministic normalized pushdown system $\mathcal{P}$, from which we define an equivalent context free grammar $\mathcal{G}$, which is converted in to an And-Or graph $\mathcal{H}$. The advantages of using the And-Or graph over the PDS model of Jha and Reps [JR02] is that we have the reachability relationships between various principals in one structure from which the required relations can be computed efficiently. The And-or graph $\mathcal{H}$ can be computed directly from $\mathcal{C}$ without constructing either of $\mathcal{P}$ or $\mathcal{G}$. However, their definition gives a better intuition leading to an easy proof of correctness of the algorithm.

The given certificate system $\mathcal{D}$ is expressed as a set of rewriting rules, the name certs correspond to the rules of form $K_1\ A \rightarrow K_2\ A_1\ A_2\ ...A_n$. The auth certs correspond to rules of the form $K\ \Box \rightarrow K_1\ A_1\ A_2\ ..A_n\Box$. Corresponding to each key K we introduce two symbols $K^{\Box}, K^{\blacksquare}$. $\mathcal{K}^{\#}$ is the augmented set of keys consisting of the original keys and their associated symbols.

Given the set $\mathcal{D}$ of cert rules, all the rules will be converted to a normalized transition function δ of a pushdown system. The pushdown system $\mathcal{P}$ we consider is a three tuple (Q, Γ, δ). Q is the set of states as given below, Γ is the stack alphabet containing identifiers and delegation symbols of a SPKI system, $\delta \subseteq Q \times \{\Gamma \cup \{\epsilon\}\} \times Q \times \{\Gamma \cup \{\epsilon\}\}$ is the transition function. δ can have transitions either of type $(K_1, A_1, K_2, \epsilon)$ or $(K_1, \epsilon, K_2, A_1)$, which means that PDS can go from state K_1 to state K_2 either by popping or pushing a symbol $A_1 \in \Gamma$ on top of the stack. ϵ is the empty string character. Thus in each transition a symbol is pushed or is popped from the stack and both do not occur in the same transition. Let l_i be the length of right hand side of rule i, $L_{\mathcal{D}}$ the sum of the lengths of the right hand side of the cert rules.

$Q = \mathcal{K}^{\#} \cup \{(i,j) |\ 1 \leq i \leq |\mathcal{D}|, 1 \leq j \leq \ell_i\}$, $\Gamma = \mathcal{I} \cup \{\Box, \blacksquare\}$. Each rewriting rule is converted into a set of four tuples in δ. If we have the i'th cert(name or auth) with length ℓ_i as $K_i\ A_i \rightarrow K'_i\ m_{(i,1)}...m_{(i,\ell_i)}$; $A_i, m_{(i,1)}...m_{(i,\ell_i)} \in \Gamma$, then it will be converted to normalized tuples of set δ as follows:

For each K, the PDS when in state $K^{\Box}$ pushes $\Box$ onto the stack and goes to state K.

$$(K^{\Box}, \epsilon, K, \Box) \in \delta.$$

In a state K, the PDS non-deterministically chooses a rule i whose left hand side key is K and and the left hand side symbol(identifier or delegation bit) matches the symbol on top of the stack, it pops the symbol currently on top of the stack and goes to state (i, l_i) if $l_i > 0$ or else if $l_i = 0$ to state K'_i given by the right hand side key of rule i.

$$(\ K, A_i, (i, \ell_i), \epsilon\ ) \in \delta.$$

$$(\ K, A_i, K'_i, \epsilon) \in \delta$$

In a state of the form (i, j) it pushes the j'th identifier of the right hand side of the i'th cert and goes to state $(i, j-1)$ if $j > 1$, else if $j = 1$ it goes to the state K'_i given by the right hand side key of rule i.

$$(\ (i,j), \epsilon, (i, j-1), m_{(i,j)}\ ) \in \delta \ \textit{for all}\ 2 \leq j \leq \ell_i.$$

$$(\ (i,j), \epsilon, K'_i, m_{(i,j)}\ ) \in \delta \ \textit{when}\ j = 1.$$

When the PDS is in state of type K and the top of the stack symbol is either $\square$ *or* $\blacksquare$ then it goes to state $K^{\square}$ or $K^{\blacksquare}$ respectively.

$$(K, \square, K^{\square}, \epsilon) \in \delta,\ (K, \blacksquare, K^{\blacksquare}, \epsilon) \in \delta$$

We can see that in a SPKI system $\mathcal{D}$ a key K_1 gives authorization to another key K_2 if the PDS $\mathcal{P}$ when started in state $K_1^{\square}$ on an empty stack reaches either of the states $K_2^{\square}$ or $K_2^{\blacksquare}$ with the stack empty at the end.

It is easy to see that size of δ is $O(L_{\mathcal{D}} + |\mathcal{D}|)$ and size of Q is $O(|\mathcal{K}| + L_{\mathcal{D}})$.

From the normalized PDS $\mathcal{P}$ we construct an equivalent context free grammar $\mathcal{G}$ according to the rules given in [Sip01]. The variables (i.e., non-terminals) of $\mathcal{G}$ are $\{A_{pq} | p, q, \in Q\}$. The production rules of $\mathcal{G}$ are described below.

For each $p, q, r, s \in Q$ and $A \in \Gamma$, such that δ contains the transitions (p, ϵ, r, A), (s, ϵ, A, q) (i.e., the first transition pushes A onto the stack while the second pops the same symbol from the stack) we have the rule $A_{pq} \to A_{rs}$ in $\mathcal{G}$.

For each $p, q, r \in Q$ we have the rule $A_{pq} \to A_{pr} A_{rq}$ in $\mathcal{G}$.

For each $p \in Q$, we have the rule $A_{pp} \to \epsilon$ in $\mathcal{G}$.

The only terminal symbol of the grammar is the null string ϵ. Let $Q' \subset Q$ be the set $\mathcal{K} \cup \{(i, l_i) | 1 \leq i \leq |\mathcal{D}|\}$. Due to the semantics of SPKI, it is easy to see that among the rules of type $A_{pq} \to A_{pr} A_{rq}$, the ones that are useful are those in which $q, r \in Q'$. It is easy to see that the number of rules of this type is bounded by $O((|\mathcal{K}| + L_{\mathcal{D}})(|\mathcal{K}| + |\mathcal{D}|)|\mathcal{D}|)$. The total number of rules is $O((|\mathcal{K}| + L_{\mathcal{D}})(|\mathcal{K}||\mathcal{D}| + |\mathcal{D}|^2))$. The CFG $\mathcal{G}$ has the following property. The PDS $\mathcal{P}$ can go from state p to q starting and ending in a empty stack iff we can generate the empty string ϵ from CFG symbol A_{pq}. The term reachability problem of the PDS has been converted to a word problem in the CFG.

From the above grammar $\mathcal{G}$ we construct a directed And-Or graph $\mathcal{H} = (V, E)$. The vertex set V is a disjoint union of two sets V_1, V_2. V_1 is the set of all pairs of the form $[p, q]$ where $p \in Q$ and $q \in Q'$. V_2 is the set of all triples of the form $[p, q, r]$ where $p \in Q$ and $q, r \in Q'$. Each vertex in V_1 is an "or" vertex while each vertex in V_2 is an "and" vertex. The set E of edges is defined as follows. From each node of the form $[p, r, q]$ there are exactly two edges going to the vertices $[p, r]$ and $[r, q]$. For each rule of the form $A_{pq} \to A_{rs}$ in $\mathcal{G}$, we have an edge from ($[p, q]$ to $[r, s]$). For each rule of the form $A_{pq} \to A_{pr} A_{rq}$ in $\mathcal{G}$, we have an edge from $[p, q]$ to the vertex $[p, r, q]$. Note that there may be cycles in the graph.

Now, we compute a function $F\ : V \to \{True,$ $False\}$ which is the least fix point that satisfies the following conditions: for each vertex u of the form $[p, p]$, $F(u)\ = True$; for each vertex u of the form $[p, q]$ $(p \neq q)$, $F(u)$ is the disjunction of all $F(v)$ such that $(u, v) \in E$; for each vertex $u \in V_2$, $F(u)$ is the conjunction of all $F(v)$ such that $(u, v) \in E$. It is well known that this fix point can be computed in time linear in the size of $\mathcal{H}$ using a simple iterative approach as follows. With each vertex u, we maintain a variable $label(u)$ which is initialized to $True$ for vertices of the form $[p, p]$ and is initialized to $False$ for all other vertices. We also maintain a counter $c(u)$ with each vertex u which is initialized to zero for all vertices. The algorithm also maintains a set X of vertices. Initially, X is the set of nodes of the form $[p, p]$. After this we iterate the following procedure as long as X is non-empty. We delete a node v from X, examine all nodes u from which there is an edge to v; if $u \in V_1$ (i.e.,is an or-node) and $c(u)$ is zero then we increment $c(u)$, set $label(u)$ to $True$ and add u to the set X; if $u \in V_2$ (i.e., is an "and" node) and $c(u) < 2$ then we increment $c(u)$, further if $c(u) = 2$ after this increment, we set $label(u)$ to $True$ and add u to X.

It is easily shown that at the end of the above algorithm, for any vertex u of the form $[p, q]$, $label(u)$ is $True$ iff the contex free grammar $\mathcal{G}$ can generate the empty string ϵ from the non-terminal A_{pq}. Using this and the results of [Sip01], the following theorem is easily proved.

Theorem: At the end of the above algorithm, for any vertex u of the form $[p, q]$, $label(u) = True$ iff the PDS when started in state p with empty stack reaches state q with empty stack.

The size of $\mathcal{H}$ which is $(|V| + |E|)$ can be shown to be $O((|\mathcal{K}| + L_{\mathcal{D}})(|\mathcal{K}||\mathcal{D}| + |\mathcal{D}|^2))$. Thus the complexity of the above fix point computation is $O((|\mathcal{K}| + L_{\mathcal{D}})(|\mathcal{K}||\mathcal{D}| + |\mathcal{D}|^2))$.

Recall that, at the beginning of the section, we described a method for computing relation R_g for each atomic formula g. This procedure used a sequence of sets of certificates $\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_{l-2}$ where $l \leq 2$,.

R_g is constructed by computing $Eval_{g,i}$ for $i = 0...l-2$. It should be easy to see that, for any fixed i, $Eval_{g,i}$ for all g can be computed by constructing the And-or graph corresponding to the set of certs $\mathcal{C}_i$.

$Eval_{g,i}$ can be computed from the And-or graph $\mathcal{H}_i$ corresponding to the set of certificates C_i as follows.

Assume g is $authorize(p, q, d, e)$. If both p, q are variables and $d = 1$ then $Eval_{g,i}$ is the set of all pairs (K_1, K_2) such that the vertex $(K_1^{\square}, K_2^{\square})$ is labeled $True$ in graph $\mathcal{H}_i$. If $d = 0$ then we check if either the above vertex or the vertex $(K_1^{\square}, K_2^{\blacksquare})$ are labeled $True$ in $\mathcal{H}_i$. If both p, q are keys say K_1, K_2 and $d = 1$, ρ is the set containing empty evaluations of the vertex $(K_1^{\square}, K_2^{\square})$ is labeled $True$. Other cases are similarly handled.

Assume g is $resolve(p, q)$. In this case p is a name and q can be a key or a variable. We can assume that p is the prefix of the right hand side of some rule j in $\mathcal{C}_i$(If this is not the case then we can add a dummy rule whose right hand side is p). Assume that the length of p(number of identifiers) is k.
If q is a key then $Eval_{g,i}$ is the empty evaluation if the node $[(j, k), q]$ in $\mathcal{H}_i$ is labeled $True$. If q is a variable then $Eval_{g,i}$ is the set of all K_1 such that the node $[(j, k), K_1]$ in $\mathcal{H}_i$ is labeled $True$.

In the above computation if $\mathcal{C}_{i+1} \supseteq \mathcal{C}_i$, then the graph $\mathcal{H}_{i+1}$ is obtained by adding new edges to $\mathcal{H}_i$, corresponding to additional certs in $\mathcal{C}_{i+1}$. The fixpoint computation on $\mathcal{H}_{i+1}$ can start with the labels computed for $\mathcal{H}_i$ and continue with the additional edges till a new fixpoint is reached. Thus we can take advantage of the incremental computation nature of the And-or Structure. It is to be noted that if $\mathcal{C}_{i+1} \subseteq \mathcal{C}_i$, then $\mathcal{H}_{i+1}$ is obtained from $\mathcal{H}_i$ by deleting some edges. However we cannot apply incremental computation while deleting edges in the graph $\mathcal{H}_i$, as the fixpoint computation is not monotonic with deletion of edges in the And-or structure.

Since $|\mathcal{C}_i| \leq m$, from the above analysis we can see that R_g, for any atomic g, can be computed in time $O(m(|\mathcal{K}| + |L_{\mathcal{C}}|)(\mathcal{K}m + m^2))$.

In our algorithm presented below for computing R_f, we need to maintain lists of intervals of time. A list of intervals is *maximal* if every pair of intervals in the list is non-overlapping and non-adjacent (two time intervals $[s, s'], [t, t']$ are non-overlapping and non-adjacent if $t > s' + 1$ or $s > t' + 1$). We maintain all lists in such a way that they are maximal and the intervals are in sorted order. We call such lists as normalized. We define size of such a list as the number of intervals in it. All our lists are in normalized form and all operations preserve this property. The union of two normalized lists L_1 and L_2 is another normalized list covering exactly the union of the time points covered by the two lists. The intersection of two normalized lists is a normalized list that covers exactly the time points common to both of the lists. The union and intersection of two normalized lists can be computed in time proportional to the sum of the sizes of the two lists. The complement of a normalized list L_1 is a normalized list that covers exactly all time points in the interval $[0, \infty]$ that are not covered by L_1. The complement of a normalized list can also be computed in time linear in the size of the list.

A set of m certificates each labeled with a time interval, defines a set of atmost $2m$ intervals, where corresponding to each interval in the set we have a set of certificates valid throughout that interval. Any atomic formula g is either valid throughout the interval or not valid at all. In any relation R_g corresponding to an atomic formula g, the normalized list associated with a tuple of R_g is of length atmost $2m$. When $g = \neg h$, corresponding to an evaluation ρ_l of h there is a normalized list l of time intervals during which the evaluation satisfies h. The time during which g is valid is given by normalized list v obtained from complement of l as defined above. We can see that length of v is of order $|l|$. Same is the case when g is obtained from other unary operations on h. Consider a formula $g = h \wedge h'$, where evaluations $\rho_l, \rho_{l'}$ corresponding to h and h' respectively, are valid during the time intervals given by normalized lists l, l'. Then g is valid during time given by list v obtained from intersection of l and l'. Any interval in l can intersect atmost with two intervals of l', so the length of list v can be atmost $(|l| + |l'|)$. Similarly we can show that for a relation R_g obtained from $\mathbf{U}$ composition of two relations $R_h, R_{h'}$, the lenght of normalized lists corresponding to relation R_g is of order of sum of the lengths of the lists corresponding to R_h and $R_{h'}$. For a formula g the length of the normalized list of time intervals corresponding to an evaluation is of $O(m|g|)$ denoted by V_g, where $|g|$ is the size of the formula g.

Now we give the second part of the algorithm based on structural induction for computing relation R_g of a subformula g when g is not atomic. The algorithm computes relation R_g for each subformula g of f in the increasing lengths of the subformula g. On termination of the algorithm we have the required relation R_f.

Let $|R_g|$ denotes the number of tuples in the relation R_g corresponding to a formula g and as shown above the length of any normalized list appearing in a tuple of R_g corresponding to the formula g, is of $O(m|g|)$ where m is the number of certificates, then the size of relation R_g is given by $O(|R_g|m|g|)$.

We denote list corresponding to a tuple t as L_t, which is a normalized list. We call it as the validity list of the tuple t.

If the subformula is of the type $g = \neg h$, for every tuple t of form $(K_1, K_2, ..., K_n, L_t) \in R_h$ we include a tuple $t' = (K_1, K_2, ..., K_n, \overline{L_t})$ in R_g. $\overline{L_t}$ is the complement of the validity list L_t as defined earlier. As shown previously we can compute $\overline{L_t}$ in time linear in the size of the list. Hence we can compute R_g in time $O(|R_h|m|h|)$.

Consider a subformula $g = h \wedge h'$, where $R_h, R_{h'}$ are the relations computed for formulas h and h', having $(i+1)$ and $(j+1)$ attributes respectively. If h, h' have v number of common variables, then R_g has $(i+j-v+1)$ attributes. For a given instantiation ρ if h is satisfied during an interval I (in the validity list) and h' during I', g is satisfied during time $I \cap I'$. For a tuple t_1 in R_h and a tuple t_2 in $R_{h'}$, we include a tuple t_{12} in R_g, if the values of tuples corresponding to the common variables are equal and the intersection of corresponding validity lists is not empty. We include in the tuple t_{12} all the values related to the variables(without repeating the common variable values) and the validity list in the tuple t_{12} is given by intersection of the validity lists in t_1 and t_2. Given that we can compute the intersection of two validity lists in time linear in the sum of their sizes, We can compute R_g in time $O(|R_h||R_{h'}|m(|h| + |h'|))$.

If the subformula is of the type $g = \mathbf{X}h$, then for every tuple in R_h of the form $t_h = (K_1, ...K_m, L_{t_h})$ we include a tuple $t_g = (K_1, ...K_m, L_{t_g})$ in R_g where for every $[t_i, t_j] \in L_{t_h}$ we include $[max(t_i - 1, 0), t_j - 1] \in L_{t_g}$. We can compute L_{t_g} in time linear in the size of L_{t_h}. Hence we can compute R_g in time $O(|R_h|m|h|)$.

If the subformula is $g = \exists i h(i)$, then for all tuples in R_h of the form $(K_1, ..., K_{i-1}, K_i, K_{i+1}, ...K_m, L_{t_K})$, $(K_1, ..., K_{i-1}, L_i, K_{i+1}, ...K_m, L_{t_L})$,..., where $K_i, L_i, ...$ correspond to instantiations of variable i, we include the tuple $t_g = (K_1, ..., K_{i-1}, K_{i+1} ...K_m, \{L_{t_K} \cup L_{t_L} \cup ...\})$ in R_g. The union of a set of validity lists can be computed in time linear in the sum of their sizes. Hence we can calculate R_g in time $O(|R_h|m|h|)$.

Consider a subformula $g = h\mathbf{U}_{[t_1,t_2]}h'$, where $R_h, R_{h'}$ are the relations computed for formulas h and h', having $(i+1)$ and $(j+1)$ attributes respectively. If h, h' have v number of common variables, then R_g has $(i+j-v+1)$ attributes. For a given instantiation ρ of values if h is satisfied during the times given by validity list L_1, h' during L_2, we give a method to calculate L_{12} the list of time intervals during which g is satisfied. We define two intervals $[p_1, p_2] \in L_1, [q_1, q_2] \in L_2$ as compatible if they overlap or if $[p_1, p_2], [q_1, q_2]$are adjacent i.e. $q_1 = p_2 + 1$. To compute L_{12} take two compatible intervals $I_1 = [p_1, p_2] \in L_1, I_2 = [q_1, q_2] \in L_2$, then by the definition of the bounded until operator we can show that the corresponding interval $I_{12} \in L_{12}$ during which the formula g holds good is $[max(T - t_2, p_1), max(T - t_1, p_1)]$ where $T = min(p_2 + 1, q_2)$. Thus we compute all the intervals in L_{12} from the compatible intervals of L_1 and L_2. Given that we maintain the validity lists in a normalized condition we can calculate L_{12} in time linear in the sum of sizes of validity lists L_1 and L_2. For a tuple t_1 in R_h and a tuple t_2 in $R_{h'}$, we include a tuple t_{12} in R_g, if the values of tuples corresponding to the common variables are equal and the composition of corresponding validity lists as defined above is not empty. We include in the tuple t_{12} all the values related to the variables(without repeating the common variable values) and the validity list in the tuple t_{12} is given by composition of corresponding validity lists in t_1 and t_2 as defined above. We can compute R_g in time $O(|R_h||R_{h'}|m(|h| + |h'|))$.

Thus we calculate the relation R_f for the formula f inductively from the components of the formula. Let n be the total number of principals in the system, L the sum of lengths of right hand side of all certificates and k the number of variables in the formula f. For any formula $g(i_1, ..., i_{l_i})$ with l_i number of variables the size of relation R_g is of order $O(n^{l_i}m|g|)$ because each of the variables can be any of the n principals in the system. The normalized validity list of a tuple in R_g can be maintained in the size of $O(m|g|)$. The relation R_g obtained by composition of relations R_h and $R_{h'}$ can be computed in time $O(|R_h||R_{h'}|m(|h| + |h'|))$. If h contains l_i variables and h' contains l_j variables, R_g can be computed in time $O(n^{l_i}n^{l_j}m(|h| + |h'|)) = O(n^{l_i+l_j}m(|h| + |h'|))$. Thus the overall formula can be computed in time $O(n^{l_i+l_j+\cdots}m(|h| + |h'| + ...)) = O(n^k m|f|)$. Since the size of formula $|f| \approx k$ the complexity of evaluating the formula f given the relations for the atomic formulas $authorize$ and $resolve$ is $O(n^{|f|}m|f|)$. The overall complexity of evaluating the formulas of the logic FTPL is $O(n^{|f|}m|f|) + O(m(n+L)(nm+m^2))$.

6 Conclusion and Future Work

In this paper, we have proposed a language for analyzing a set of policy statements labeled with validity intervals and gave a list of problems that could be specifiable by the language. We also gave algorithms for computing the formulas of the logic in a incremental fashion. In future we propose to modify the semantics of the modal operators of FTPL to consider a state transition model for the SPKI access control system.

References

[Aba97] Martin Abadi. On SDSI's linked local name spaces. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[ADT02] A. Arsenault, Diversinet, and S. Turner. *Internet X.509 Public Key Infrastructure: Roadmap*. PKIX working group-Internet Draft, 2002.

[BEO97] A. Bouajjani, J. Esparza, and O.Maler. Reachability analysis of pushdown automata: application to model-checking. *Lecture Notes in Computer Science*, 1243, 1997.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. pages 185–210, 1999.

[BFK99] Matt Blaze, Joan Feigenbaum, and Keromytis Keromytis. KeyNote: Trust management for public-key infrastructures. In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols—6th International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–66, Cambridge, United Kingdom, 1999. Springer-Verlag, Berlin Germany.

[CEE+01] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4), 2001.

[EFL+99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, September 1999. See `ftp://ftp.isi.edu/in-notes/rfc2693.txt`.

[EHRS00] Esparza, Hansel, Rossmanith, and Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV: International Conference on Computer Aided Verification*, 2000.

[Ell99] Carl M. Ellison. *RFC 2692: SPKI requirements*. The Internet Society, September 1999. See `ftp://ftp.isi.edu/in-notes/rfc2692.txt`.

[HK00] Jon Howell and David Kotz. A formal semantics for SPKI. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes in Computer Science*, pages 140–158. Springer-Verlag, October 2000.

[HRU75] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. In *Proceedings of the fifth ACM symposium on Operating systems principles*, pages 14–24. ACM Press, 1975.

[HvdM01] Joseph Y. Halpern and Ron van der Meyden. A logic for sdsi's linked local name spaces. *Journal of Computer Security*, 9:105–142, 2001.

[JR01] Somesh Jha and Thomas Reps. Analysis of spki/sdsi certificates using model checking. In *15th IEEE Computer Security Foundations Workshop*, pages 129–144, Cape Breton, Canada, 24–26 June 2001. IEEE Computer Society Press.

[JR02] S. Jha and T. Reps. Analysis of spki/sdsi certificates using model checking. In *Computer Security Foundations Workshop (CSFW)*, June 2002.

[Li00] Ninghui Li. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 2–15. IEEE Computer Society Press, jul 2000.

[LM03] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, January 2003. To appear.

[LWM01] Li, Winsborough, and Mitchell. Distributed credential chain discovery in trust management. In *SIGSAC: 8th ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2001.

[LWM03] Ninghui Li, William H. Winsborough, and Mitchell Mitchell. Beyond Proof-of-Compliance: Safety and availability analysis in trust management. In *Proceedings of the 2003 Symposium on Security and Privacy*, pages 123–139, Los Alamitos, CA, May 11–14 2003. IEEE Computer Society.

[San98] R. S. Sandhu. Role-based access control. In M. Zerkowitz, editor, *Advances in Computers*, volume 48. Academic Press, 1998.

[Sip01] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 20 Park Plaza, Boston, MA 02116, 2001.