Methods and Limitations of Security Policy Reconciliation[*]

Patrick McDaniel
AT&T Research
pdmcdan@research.att.com

Atul Prakash
University of Michigan
aprakash@eecs.umich.edu

Abstract

A security policy is a means by which participant session requirements are specified. However, existing frameworks provide limited facilities for the automated reconciliation of participant policies. This paper considers the limits and methods of reconciliation in a general-purpose policy model. We identify an algorithm for efficient two-policy reconciliation, and show that, in the worst-case, reconciliation of three or more policies is intractable. Further, we suggest efficient heuristics for the detection and resolution of intractable reconciliation. Based upon the policy model, we describe the design and implementation of the Ismene policy language. The expressiveness of Ismene, and indirectly of our model, is demonstrated through the representation and exposition of policies supported by existing policy languages. We conclude with brief notes on the integration and enforcement of Ismene policy within the Antigone communication system.

1. Introduction

Policy is frequently the means by which the requirements of communication participants are identified and addressed. Session policies are stated by the different participants and organizations for the services supporting the communication. At present, facilities for the reconciliation of participant policies in existing policy frameworks are limited in scope and semantics. Hence, policies must be reconciled manually, a frequently complex process. Alternatively, governing authorities must dictate policy. In that case, session participants accepting dictated policy have limited ability to affect how session security is defined.

[*]This work is supported in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0508. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

Automated reconciliation is a means by which the possibly divergent requirements of session participants can be met. Participants specify their requirements through policy. These policies are reconciled at run-time, resulting in a session-defining specification. In this case, session security is the result of all requirements, rather than dictated by a single authority.

A session security policy defines security-relevant properties, parameters, and facilities used to support a session. Thus, a session policy states how security directs behavior, the entities allowed to participate, and the mechanisms used to achieve security objectives. This broad definition extends much of existing policy; dependencies between authorization, access control, data protection, key management, and other facets of a communication can be represented within a unifying policy. Moreover, requirements frequently differ from session to session, depending on the nature of the session and the environment in which it is conducted. Hence, the conditional requirements of all parties are defined in a policy specification.

This paper considers the definition, efficiency, and methodologies of security policy reconciliation within a general-purpose policy model. This model defines policy as the collection of interdependent statements of provisioning and authorization. Each statement identifies context-sensitive session requirements. A reconciliation algorithm attempts to identify a *policy instance* compliant with the stated requirements. Our investigation shows that in the worst case, reconciliation of two policies is tractable, but reconciliation of three or more is not. We identify several heuristics for detecting and combating intractable policy reconciliation.

We further consider the related problems of policy compliance and analysis. A compliance algorithm determines whether an instance is consistent with the requirements stated in a policy. The analysis algorithm determines whether the provisioning of a session adheres to a set of assertions that express correctness constraints on a policy instance. We identify efficient algorithms for both compliance and analysis. We demonstrate that a more general form of analysis is intractable (coNP).

The Ismene policy language and supporting infrastructure is built upon the model and algorithms defined throughout. The expressiveness of Ismene, and indirectly the applicability of our policy model, is demonstrated through the representation and exposition of policies defined in several popular policy languages. We describe the integration and enforcement of Ismene policy within the Antigone communication system.

Policy has been used in different contexts as a vehicle for representing authorization and access control [31, 5, 9, 32, 29], peer session security [33], quality of service guarantees [7], and network configuration [3, 2]. These approaches define a policy language or schema appropriate for their target problem domain. This paper expands on this work by defining a general approach in which policy is used to both provision and to regulate access to communication services.

The problem of reconciling policies in an automated manner is only beginning to be addressed. In the two-party case, the emerging Security Policy System (SPS) [33] defines a framework for the specification and reconciliation of security policies for the IPSec protocol suite [23]. Reconciliation is largely limited to intersection of specified data structures. In the multi-party case, the DCCM system [13] provides a negotiation protocol for provisioning. DCCM defines the session policy from the intersection of policy proposals presented by each potential member. Each proposal defines a range of acceptable values along a multi-dimensional policy structure. Hence, reconciliation in these systems is largely based on the intersection of policy schema. In contrast, this work attempts to define a general framework upon which more flexible expression-oriented policies are defined and reconciled.

Language-based approaches for specifying authorization and access control have long been studied [31, 9, 32, 29], but they generally lack support for reconciliation. These systems typically identify a rigorous semantics for the evaluation of authorization statements. The PolicyMaker [5] and KeyNote [6] trust management systems provide a powerful framework for the evaluation of credentials. Trust management approaches focus on the establishment of chains of conditional delegation defined in authenticated policy assertions. Hence, policy is dictated by entities to which session authority is delegated, rather than through the reconciliation of participant requirements.

The following section considers the requirements of a general-purpose policy language. Section 3 considers the limits and methods of reconciliation in our general policy model. Section 4 presents the Ismene language. Section 5 illustrates the use of Ismene by representing policies supported by existing languages. Section 6 briefly discusses our experiences with the implementation and use of Ismene. Section 7 concludes.

2. Requirements

To illustrate the policy reconciliation needs, we present very simplified security requirements for an example conferencing application, `tc`. The `tc` application is to be deployed within a company, $widget.com$. $widget.com$'s organizational policy for `tc` requires the following:

- the confidentiality of all session content must be protected by encryption using $tripleDES$ or AES (provisioning requirement)
- the session is restricted to $widget.com$ employees (authorization requirement)

Now suppose $Alice$ wishes to sponsor a session of `tc` under the following policy:

- Alice wishes to use only AES cryptographic algorithm only (provisioning requirement); and
- she wishes to restrict the session to the $BlueWidgets$ team (access control requirement)

A basic requirement on a policy approach for this scenario is that it must reconcile the provisioning and access control requirements (policies) stated by any number of interested parties. It is through this process of reconciliation that a concrete, enforceable policy is developed. In the above example, Alice's and the widget.com policies are reconciled to arrive at a policy that restricts the participants to members of $widget.com$'s $BlueWidgets$ team (access control requirement), and `tc` must be configured so that all content is encrypted using AES (provisioning requirement).

In general, security requirements can be more complex. For example, Alice may wish to restrict access to certain hours of the day, require that the session be rekeyed periodically, etc. (environment-dependence). In some cases, the session must be able to make access control decisions based on the use and configuration of security mechanisms; for example, admit a member only if AES is being used for ensuring confidentiality. Our language permits such dependencies between authorization and provisioning policy. This represents a divergence from many existing works that treat authorization and provisioning independently.

3. Policy

This section presents the Ismene approach to policy management. Depicted in Figure 1, a session is established between two or more entities. Each participant in the session submits a set of relevant domain policies to the *initiator*. The initiator may be a participant or external entity (e.g., policy decision point [14]). Stated by a *policy issuer*, a session policy is a template describing legal session provisioning and the set of rules used to govern access.

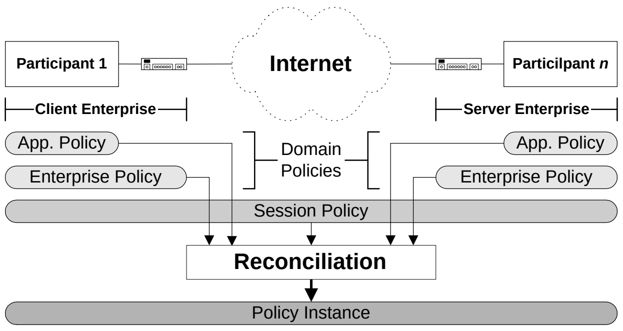


Figure 1. Policy construction - A session-specific policy instance for two or more participants is created by an initiator. Each participant submits a set of domain policies identifying the requirements relevant to the session. The initiator constructs the policy instance compliant with each domain and the session policy through reconciliation.

Domain policies state conditional requirements and restrictions placed on the session. In the scenario described in the previous section, Alice's domain policy states that AES must be used and the session restricted members of the BlueWidget team. The set of policies appropriate for a particular session is dependent on the environment in which it is to occur. The scenario described in Figure 1 depicts an environment in which the two participants state policies for the supported application, as well as their local enterprise environments. The instance is the result of the reconciliation of the session, application, and enterprise policies.

An initiator uses the reconciliation algorithm to create a *policy instance* compliant with the session and each domain policy. A policy is compliant if all stated requirements and restrictions are realized in the resulting instance. If an instance is found, it is used to govern the provisioning and authorization of the subsequent session. If an instance cannot be found, then the participants must revise the domain policies or abort the session. An instance concretely defines session provisioning and authorization. The initiator is trusted to reconcile the session and domain policies correctly[1].

A session policy in Ismene is authoritative; the instance must be directly derived from the session policy[2]. Domain policies are consulted only where flexibility is expressly granted by the issuer. Hence, the session policy acts as a template for operation, and domain policies are used to further refine the template toward a concrete instance. Conversely, domain policies represent the set of requirements that are deemed mandatory and relevant.

[1]Where deemed necessary, participants can efficiently validate an instance against the relevant domain policies prior to acceptance of the instance (see Section 3.4).

[2]Where no such authority is available, a default session policy that places no constraints on session security is used. In that case, participant domain policies are reconciled to derive the instance, and the default (session) policy where domain policies provide no guidance.

3.1. Policy Expressions

Session provisioning identifies the configuration of the security services used to implement the session. Ismene models provisioning as collections of security *mechanisms*. Associated with a mechanism is a set of *configuration* parameters used to direct its operation. Throughout, we use the term configuration to refer to a singular statement specifying a parameterized mechanism configuration. Each mechanism provides a distinct communication service that is configured to address session requirements. A provisioning expression explicitly states configuration through a set of mechanisms and parameters. To illustrate, consider the following (incomplete) Internet Key Exchange (IKE [19]) session policy:

```
cryptographic algorithm: 3DES (and)
hash algorithm: MD5 (and)
exchange algorithm: MODP, (Group 1 (or) Group 2)
```

This policy states that there are three mechanisms used to implement IKE; a cryptographic algorithm, a hash algorithm, and a Diffie-Hellman exchange. Moreover, the exchange must use either group 1 or 2 MODP values, but not both or neither. The policy requirements can be expressed more precisely as:

$$Crypto((3DES) \wedge Hash(MD5) \wedge \\ (Exchange(MODP, Group1) \oplus Exchange(MODP, Group2)$$

where each element of the expression specifies a mechanism (e.g., Crypto) and configuration (e.g., $3DES$).

Note that this policy must be further refined for it to be enforced; the session participants (IKE initiator and responder) must agree upon an exchange group (group 1 or group 2). *Provisioning reconciliation* resolves these ambiguities by attempting to find an *instance* that is consistent with each policy expression. Where multiple policies are considered, each must be satisfied.

In the remainder of this paper, policy statements identifying a range of acceptable, but mutually exclusive, behaviors (identified by the XOR operator $\oplus$) are called **pick** statements.

Policy expressions give an alternative and more general way of viewing the reconciliation problem than that provided in current policy languages. For example, In IKE, a requester (acting as the entity providing a domain policy) must provide a proposal that precisely mirrors that of the responder (whose policy represents a session policy). IKE reconciliation trivially finds an intersection of the fields of the policy proposal. In contrast, reconciliation in Ismene is formulated as a satisfaction problem; the initiator seeks an instance that satisfies the set of expressions. Hence, the provisioning expression in domain policies need only specify those aspects of policy that the issuer wishes to influence.

Authorization policy maps identities or credentials onto a set of access rights [31]. As in provisioning, authorization statements are modeled as logical expressions. Each authorization expression, called an *action clause*, is defined as a conjunction of positive conditionals[3]. For example:

$$read : ACL(/etc/hosts, bob, read) \wedge ID(bob) \wedge FILE(/etc/hosts)$$

states that "read operation should succeed if the user is Bob, the file being accessed is /etc/hosts, and the ACL for the file allows read access to Bob". As in other systems such as KeyNote [5], the interpretation of each conditional is left to the environment; the establishment of the identity, file, and the evaluation of the file's ACL is outside the scope of the policy specification.

3.2. Provisioning Reconciliation

Provisioning reconciliation searches for a set of mechanism configurations that satisfy the policy expressions. We show in Appendix A that in its most general form, reconciliation of even one expression is intractable; any instance of positive, one-in-k satisfiability [30, 15], a known intractable problem, can be reduced to the problem of finding a solution that satisfies a policy expression with pick statements. This result is in stark contrast to needs of policy management; the algorithms used to manage policy must be efficient. In response, we place the following restriction of the construction of policy:

> *Policy Restriction*: A mechanism configuration can only be stated in at most one pick statement in a policy.

For example, if a, b, and c are mechanism configurations, the following policy expression is not allowed by the above restriction in a single policy because a occurs twice in the policy expression:

$$(a \oplus b) \wedge (a \oplus c)$$

On the other hand, the policy expression presented in Section 3.1 is legal because $Exchange(MODP, Group1)$ and $Exchange(MODP, Group2)$ are considered different mechanism configurations, though they refer to the same mechanism.

Based on this restriction, the following algorithm reconciles a session policy and one domain policy. Figure 2 presents an example of the algorithm being applied on a sample session and domain policy. A thorough treatment of this and all algorithms identified in this paper is presented in [26].

Two-Policy Reconciliation Algorithm

1. Collapse *equivalent configurations* (*described below*), if necessary. This step is not necessary on the example policy in Figure 2.
2. Reduce the session and domain policies.
   (a) Remove each configuration in the session or domain policy that is not in the other policy.
      - If any pick statement in the session policy becomes empty, then it is removed (the domain policy does not provide any guidance of that pick).
      - If any pick statement in the domain policy becomes empty, then the policy cannot be reconciled (the session policy does not allow any configuration in the pick statement).
3. Remove any pick statement containing a single configuration in one policy, and remove the corresponding pick statement containing that configuration in the other. Place the single configuration in the instance.
4. Represent each pick statement as a node in a graph. Add an edge between two nodes, labeled with the configuration, if the pick statements share the configuration. The problem of finding an acceptable configuration is equivalent to finding an edge cover of size $N/2$ on the resulting graph, where N is the number of nodes in the graph. If N is odd, or no such cover can be found, then the policies cannot be reconciled. The edge cover problem on graphs is known to be in P and an efficient algorithm can be found in [18].

An *equivalent configuration* is a set of two or more configurations contained within the same pick statement in both policies. Consider the sub-clauses $(d \oplus e \oplus f)$ and $(d \oplus e \oplus b)$ of the example session and domain policies. With respect to the reconciliation algorithm, d and e can be considered to be equivalent configurations; any instance including d can replace d with e and still satisfy both policies. Equivalent configurations must be replaced with a single meta-configuration in step 1, and restored in the instance after reconciliation is completed. For two policies, equivalent configurations can be easily found in polynomial time by simply looking for overlap between pick statements of the two policies. With equivalent configurations, the output of 2-policy reconciliation can be a policy expression, rather than an instance.

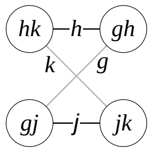
[3]Because of the complexity imposed by the negative conditions, we only consider positive conditions in this paper [5]. As many systems adopt this approach, this does not significantly affect our ability to represent existing policies (see Section 5)

(Session Policy) $S = (a \oplus b \oplus c) \wedge (d \oplus e \oplus f) \wedge (g \oplus h \oplus i) \wedge (j \oplus k \oplus l)$
(Domain Policy) $D = (f) \wedge (d \oplus e \oplus b) \wedge (h \oplus k) \wedge (g \oplus j)$

1) $S = (abc) \wedge (def) \wedge (ghi) \wedge (jkl)$
$D = (f) \wedge (deb) \wedge (hk) \wedge (gj)$
$I = \emptyset$ $\rightarrow$

2) $S = (b) \wedge (def) \wedge (gh) \wedge (jk)$
$D = (f) \wedge (deb) \wedge (hk) \wedge (gj)$
$I = \emptyset$ $\rightarrow$

3) $S = (gh) \wedge (jk)$
$D = (hk) \wedge (gj)$
$I = b, f$ $\rightarrow$

4) 

$\rightarrow$ $I = b, f, h, j$

Figure 2. Reconciliation - the Ismene reconciliation algorithm iteratively reduces the intersection of the session (S) and domain (D) policies. Any reconcilable policy will converge on configurations (denoted by single letter variables – e.g. a) existing exactly once in each policy. The remaining pick statements can be reconciled into a concrete instance (I) using an (efficient) edge cover algorithm.

n-Policy Reconciliation Algorithm

In the case where more than one domain policy needs to be reconciled with a session policy, a simple algorithm would be to reconcile the session policy with one domain policy at a time. The policy expression resulting from each 2-party reconciliation is used as the session policy for reconciliation with the next domain policy. As a final step, a specific configuration is chosen from pick statements remaining after the final reconciliation (due to equivalent configurations). A reasonable strategy chooses the first configuration in each remaining pick statement from the session policy, assuming that the session policy lists configurations in decreasing order of preference.

The ordering of reconciliation may affect the reconciliation results; some orderings of domain policies will not be reconcilable, while others will. For example, consider the following session and domain policies:

$$\begin{array}{lc} SessionPolicy & (a \oplus b) \wedge (c \oplus d) \\ DomainPolicy\_1 & (a \oplus c) \wedge (b \oplus d) \\ DomainPolicy\_2 & (b) \wedge (d) \end{array}$$

If domain policy 1 is considered first, the policies *may* reconcile to ($a \wedge c$). Thus, domain policy 2 would not be reconcilable. If domain policy 2 were considered first, reconciliation would arrive at ($b \wedge d$), and thus be reconcilable with domain policy 1. The introduction of the third policy violates the property that a specific configuration occurs in at most two pick statements in the reconciliation expressions – the reduction to the edge cover problem fails in such a case. It can be shown that the problem is intractable by a reduction from the one-in-three satisfiability problem.

Where reconciliation is not possible, it may be desirable to find a subset of policies that *can* be reconciled. One potential reconciliation algorithm, Largest Subset Reconciliation (LSR), would attempt to find an instance reconcilable with the largest number of domain policies. LSR has the undesirable property that it may fail to allow the participation of required members (for example, by excluding the video source in a video conference). Moreover, as shown in Appendix B, LSR is also intractable.

An extension to the reconciliation algorithm establishes an ordering of domain policies. Higher prioritized policies are considered first and lower priority policies are considered only when higher priority policies provide no guidance; otherwise they may be excluded. This algorithm is polynomial time and has been used extensively to derive the security policy in the Antigone communication system [28].

Our experience in using the policy framework for a range of group communication applications indicates that often pick statements intersect with at most one pick statement of all other policies. For example, all IKE policies will define similar pick statements for $Crypto$, $Hash$, and $Exchange$ mechanisms. In this case the problem of reconciliation is tractable. Any violation of this property (over a set of session and domain policies) can be efficiently detected by a simple scan of the policies – in that case, the heuristic suggested above of prioritizing domain policies can be used.

3.3. Authorization Reconciliation

The authorization policy defined in an instance is the result of the reconciliation of action clauses of all considered policies. However, the semantics of such an operation are unclear; one may view reconciliation of access control to be an intersection (logical OR of each policy), a union (logical AND), or something else (session AND at least one domain policy). The first approach (logical OR), however, has the unfortunate side affect that a permissive domain policy can circumvent any controls stated in the session or domain policies.

Our reconciliation algorithm takes the conservative approach of accepting the logical AND of all access control policies. This approach will not allow any controls to be circumvented; however, a restrictive domain policy can cause access to be denied. We discuss our experience with this issue further in Section 6.

We now illustrate authorization reconciliation. Consider an example session policy that defines the action clause[4] ($t_i : c_1 \wedge c_2$ :: *accept*;) and two domain policies with action clauses (t_i: c_3 :: accept;) and (t_i: c_4 :: accept;), respectively (where t_1 is an action and each c_i a condition). The resulting policy from the Authentication reconciliation algorithm is:

$$t_1 : ((c_1 \vee c_2) \wedge c_3 \wedge c_4) :: accept$$

3.4. Compliance

Not all domain policies are required to (or often can) be consulted during reconciliation. Hence, before participating in a session, a participant must be able to check the compliance of its domain policy with the instance that is governing the active session. Compliance is successful if all requirements stated in the domain policy are satisfied by the instance. Note that compliance in this work serves a different purpose than the compliance algorithms in trust management [5, 10, 4]; our compliance algorithm determines whether an instance is consistent with a domain policy. In contrast, compliance in trust management systems attempts to determine if the available credentials and the current system state satisfy the trust policy.

As with reconciliation, there are two phases of compliance; provisioning and authorization. The provisioning compliance algorithm compares domain policy with a received policy instance. Each configuration and pick statement must be satisfied by the instance. A configuration is satisfied if it is explicitly stated in the instance. A pick statement is satisfied if exactly one configuration is contained in the instance. Thus, provisioning compliance is as simple as testing the containment of the evaluated domain policy by the instance. More precisely, an instance describes a truth assignment for the (configuration) variables in the domain policy expression. The instance is compliant if the expression is satisfied (evaluates to TRUE) by the truth assignment.

Several researchers have examined the problem of compliance in an authorization policy. Gong and Qian's model of a policy composition (i.e., reconciled policies) define a two-principle compliance definition [17]. The *principle of autonomy* requires that any action accepted by one policy must be accepted by the composition (reconciled instance is not less permissive). The second principle, *secure interoperability*, requires that the composition must be no more permissive than either policy. However, this two-fold definition of compliance is extremely restrictive; all policies must specify equivalent authorizations. Moreover, Gong and Qian showed that compliance detection in their model is intractable.

[4]The `accept` keyword closing each clause indicates that the operation are accepted where the conditions are met. `accept` is intended as syntactic sugar, and is present in all authorization clauses. There is no `deny` in the model; Authorization fails unless explicitly accepted.

Ismene adopts the Gong and Qian's secure interoperability as a definition of compliance, but not the principle of autonomy. More precisely, compliance determines if, for any action and set of conditions, an action accepted by the policy instance is accepted by the domain policy. This embodies a conservative approach to compliance, where any action that would be denied by the domain policy **must** be denied by the instance. Hence, compliant instances always respect the limitations stated in the domain policy. However, if a domain policy is more permissive than the policy instance, the policy instance's restrictions are not relaxed.

The authorization compliance algorithm assesses whether the instance logically implies the domain policy. Given an expression e_1 describing the conditionals of action clauses in an instance, and a similar expression describing a domain policy e_2, it is conceptually simple to check compliance between the policies by testing whether the expression $e_1 \Rightarrow e_2$ is a tautology. To illustrate, consider the action clauses defined in the following instance and domain policies:

Instance	$X : (c_1 \wedge c_2) \vee c_3$ :: *accept*;
Domain policy A	$X : c_1$ :: *accept*; $X : c_3$ :: *accept*;
Domain policy B	$X : c_1, c_3$ :: *accept*;

The policy instance is compliant with the domain policy A because it is less permissive (e.g., $(c_1 \wedge c_2) \vee c_3 \Rightarrow c_1 \vee c_3$). The instance is not compliant with domain policy B because the session policy is more permissive (e.g., $(c_1 \wedge c_2) \vee c_3 \not\Rightarrow c_1 \wedge c_3$). General-purpose tautology testing is intractable [11]. However, the lack of negative conditionals in Ismene allows efficient compliance testing. For brevity, we omit further details.

3.5. Analysis

While a reconciliation algorithm may be able to identify an instance satisfying the session and domain policies, our approach makes no guarantees that the instance is properly formed. A properly formed instance adheres to a set of principles defining the correct composition and configuration of security mechanisms. An analysis algorithm determines whether a policy or instance is properly formed.

Assertions are used to define the meaning of properly formed policy by declaring legal and required relations between configurations. Each assertion contains a tag (`assert`), a conjunction of conditions, and a conjunction of consequences. Conditions and consequences are restricted to pick and configuration statement, and may be negated. Semantically, assertions state that the consequences must hold where the consequences are true (i.e., condition conjunction c, consequence conjunction q, $c \Rightarrow q$). For example, an issuer may wish to assert a completeness requirement [22, 8] that confidentiality of application data always be provided. Thus, knowing that the *ssl*, *ipsec*, and *ssh* transforms are the only means by which confidentiality can be provided, the issuer states the following (conditionless) assertion expression:

$$(ssl \oplus ipsec \oplus ssh)$$

Analysis determines if an instance (or policy) satisfies the assertion: exactly one confidentiality mechanism must be configured.

Analysis techniques guaranteeing correct software construction have been studied extensively within component architectures [20, 25]. These approaches typically describe relations defining compatibility and dependence between components. A configuration is deemed correct if it does not violate these relations. For example, Hiltunen [20] defines the conflict, dependency, containment, and independence relations. The following describes assertion expressions representing these relations (where independence is assumed):

conflict (A *is incompatible with* B)	$!(A \wedge B)$
dependency (A *depends on* B)	$A \Rightarrow B$
containment (A *provides* B)	$A \Rightarrow (!B)$

An analysis algorithm assesses whether a policy can or an instance does violate the relevant assertions. The *online policy analysis algorithm* assesses an instance with respect to a set of assertions. This algorithm evaluates the assertion expressions against the truth assignment defined by the instance. Any *false* evaluation result indicates that an assertion has been violated, and the instance cannot be used. Obviously, by virtue of the tractability of expression evaluation, online analysis is efficient.

An *offline policy analysis algorithm* (OFPA) attempts to determine if **any** instance resulting from reconciliation can violate a set of assertions. Demonstrated in Appendix B, offline analysis is intractable (coNP). However, this algorithm need only be executed once (at issuance), and thus does not impact session setup. Moreover, most reasonable configurations we have encountered exhibit a degree of *independence*; the introduction of a configuration is largely the result of the reconciliation of a few clauses. Hence, the evaluation of an assertion can be reduced to the analysis of only those clauses upon which the configurations stated in the assertions are dependent. We present an optimized algorithm for OFPA in [26].

4. Ismene

This section presents a brief overview of the Ismene policy language. Ismene specifies conditional provisioning and authorization requirements through a general-purpose policy language. A thorough survey of the grammar and semantics of Ismene is presented in [26]. Ismene policies are collections of totally ordered provisioning, action, and assertions clauses. For brevity, we omit the discussion of assertion clauses (see section 3.5).

4.1 Provisioning Clauses

Each provisioning clause is defined as the tuple:

```
<tag> : <conditionals> :: <consequences>;
```

Tags are used to associate meaningful names with provisioning requirements. Conditions are predicates that identify the circumstances under which the consequences are applicable. Consequences state session provisioning requirements through *configurations* and *pick* statements, or identify relevant sub-policies through *tags*. The reserved `provision` tag is used to name the overall provisioning requirements. Consider the following simple example, where x, y, z, and w specify mechanism configurations:

```
provision: :: confidentiality, keymgmt;
confidentiality: c1, c2 :: x, y;
confidentiality: :: pick(w, z);
rekeying: :: d
```

The first (provision) clause says that the policy must provision both confidentiality and key management services (tags). The second and third clauses state that if $c1 \wedge c2$ is true, x and y must be configured; otherwise either w or z (but not both or neither) must be configured. The final clause says that d must be configured under all circumstances. Therefore, the policy expression used as input to reconciliation is $x \wedge y \wedge d$ where $c1 \wedge c2$ is true at the time of reconciliation, and $(w \oplus z) \wedge d$ where $c1 \wedge c2$ is false. Note that the ordering of clauses with the same tag (e.g., confidentiality tag) dictates the order of evaluation. If the conditionals for an earlier instance of the tag holds (e.g., $c1 \wedge c2$), those consequences (e.g., x and y) must be enforced, and the subsequent clauses for the same tag are ignored.

Conditionals in a clause often refer to *attributes*. An *attribute* describes a single or list-valued invariant. For example, the following attributes define a single-valued version number and list-valued ACL:

```
version := < 1.0 >;
JoinACL := < {alice}, {bob}, {trent} >;
```

An occurrence of the symbol "$" signifies that the attribute should be replaced with its value. As in the KeyNote action environment [4], the *attribute set* is the set of all attributes.

```
% Ismene Provisioning Clauses
provision : PrivSession($inaddr,$ipt,$oaddr,$opt)
          :: strong_key_mgmt, confidentiality;
provision : :: weak_key_mgmt, confidentiality;
strong_key_mgmt: Manager($ent)
               :: config(dh_key(refresh,60));
strong_key_mgmt : :: config(dh_key(refresh,240));
weak_key_mgmt : :: config(lm_key(refresh,300));
confidentiality : :: pick( config(dhndlr(3des)),
                           config(dhndlr(des)) );

% Ismene Action Clauses
join : config(dhndlr(des)), In($JoinACL,$joiner),
       Credential(&cert,sgner=$ca,subj.CN=$joiner)
     :: accept;
join : Credential(&cert,sgner=$ca,delegatejoin=true),
       Credential(&tocert,sgner=$cert.pk,
                  subj.CN=$joiner)
     :: accept;
```

Figure 3. Ismene Policy - The provisioning clauses in the session and domain policies are evaluated to arrive at the policy expressions used as input to reconciliation. Action clauses are evaluated over the lifetime of the session to enforce authorization policy.

Enforcement infrastructures (e.g., applications) provide additional evaluation context by adding attributes to the attribute set. Conditional evaluation is outside the scope of Ismene; the environment in which Ismene is used is required to provide a predicate interface for each condition. This is similar to GAA API condition upcalls [29]).

Consider the provisioning clauses in Figure 3 that define requirements for public and private sessions of `tc`. If the session is private (as classified by session address attributes), then the `strong_key_mgmt` clauses are evaluated; otherwise `weak_key_mgmt` is evaluated. The `confidentiality` clause is evaluated in either case. The strong key management clause states that a Diffie-Hellman [12] keying mechanism must be used. The behavior of this mechanism is further refined to refresh the session key every 60 (240) seconds where a management is (is not) present. Where the session is not deemed private, the `weak_key_mgmt` clause simply provisions the Leighton-Micali key management mechanism [24]. The confidentiality clause instructs the data handler mechanism to use either 3DES or DES, depending on the result of reconciliation.

Note that the mechanisms indicated in the policy specification (e.g., `dh_key` and `dhndlr`) must be provided by the enforcement infrastructure. These are not keywords in the language; mechanism names are mapped to the service implementations by the enforcement infrastructure.

4.2 Action Clauses

Each action clause has the following structure:

actionName: $c_1, ..., c_n$ :: accept

The specified action (operation) is allowed if all the conditions hold when the action is attempted (i.e. at run-time). *accept* is the only allowed consequence. Hence, Ismene represents a *closed world* in which denial is assumed. The protected actions are defined by the enforcement infrastructure, and assumed known *a priori* by the policy issuer.

Used exclusively in action clauses, the reserved `credential()` conditional evaluates available credentials. All credentials are modeled by Ismene as a set of attributes. For example, an X.509 certificate [21] is modeled as attributes for `subj.O` (subject organization), issuer.CN (issuer canonical name), etc. To illustrate, consider the following action clause:

```
join : Credential(&cert,sgnr=$ca,subj.CN=$part) : accept;
```

The first argument of a credential conditional (denoted with “&” symbol) represents binding. The credential test binds the matching credentials to the (&cert) attribute. Binding is scoped to the evaluation of a single clause, and conditionals are evaluated left to right. The second and subsequent parameters of a credential conditional define a matching of credential attributes with attribute or constant values. The above example binds the credentials that were issued by a trusted CA (`sgnr=$ca`) and have the subject name of the participant (`subj.CN=$part`) to the `&cert` attribute. The conditional returns true if a matching credential can be found. The enforcement architecture is required to identify the set of credentials associated with an action.

Credential conditions are similar to trust management assertions [5, 10, 4]; evaluation determines whether the attributes of an assertion satisfy the relevant policy expression. Conditionals in action clauses can also contain checks for mechanisms that are currently provisioned in the session. Hence, authorization policy can be predicated on session provisioning.

Consider the action clauses in Figure 3. The first `join` action clause describes an ACL-based policy for admitting members to the session. The member is admitted if she is identified in the `JoinACL` attribute, she can provide an appropriate certificate credential, and the session is provisioned with the DES-enabled data handler mechanism. The second `join` is consulted only when the conditionals of first clause are not satisfied.

The second `join` clause describes a delegation policy. The first credential conditional binds `&cert` to the set of credentials delegating join acceptance (the delegation certificates issued by the trusted CA), and second tests the presence of any credential signed by the delegated public key.

5. Modeling Policy

This section demonstrates the use of Ismene policy by modeling the semantics of existing policy approaches.

IKE Session Policy (Responder)

```
provision : selector(12.14.0.0,any,17,23,any,$name)
  :: pick( config(ike(idea-cbc,md5,group1)),
           config(ike(blowfish,sha1,group2)),
           config(ike(cast-cbc,sha1,group2)) ),
     pick( config(preshare()), config(kerberos()) );

auth : config(ike(preshare)),
       Credential(&cert,modulus=$prekey.mod)
     :: accept;
auth : config(kerberos()),
       Credential(&tkt,issuer=$realmtgs)
     :: accept;
```

IKE Domain Policy (Requestor)

```
provision : selector(any,12.14.9.1,17,23,any)
  :: pick( config(ike(cast-cbc,sha1,group2)),
           config(ike(cast-cbc,md5,group2)) ),
     config(preshare());

auth : config(ike(preshare)),
       Credential(&cert,modulus=$prekey.mod)
     :: accept;
```

Figure 4. IKE Policy - session (responder) and domain (requestor) policies are used to implement IKE phase one policy negotiation. The IKE SA policy (instance) is arrived at through the intersection of the responder (session) policy and requestor (domain policy) proposals.

DCCM Session Policy (CCNT)

```
provision: ::
  pick( config(conf(3DES)), config(conf(CAST)),
        config(conf(IDEA)), config(conf(RC4)) ),
  pick( config(kman(OFT)), config(kman(LKH)),
        config(kman(DH)), config(kman(pswd)) ),
  pick( config(trans(SSH)), config(trans(SSL)),
        config(trans(IPSec)) );
```

DCCM Domain Policy 1 (member)

```
provision: ::
  pick( config(conf(3DES)), config(conf(CAST)) ),
  pick( config(kman(OFT)), config(kman(LKH)) ),
  pick( config(trans(SSH)), config(trans(SSL)),
        config(trans(IPSec)) );
```

DCCM Domain Policy 2 (member)

```
provision:
  :: pick( config(conf(CAST)), config(conf(RC4)) ),
     pick( config(kman(OFT)) ),
     pick( config(trans(SSH)), config(trans(SSL)) );
```

Figure 5. DCCM Policy - Designed for policy negotiation in multi-party communication, DCCM creates a session policy through intersection of (domain) policy proposals defined over a template structure (session policy). DCCM does not specify authorization policy.

These policies serve to highlight the similarities and differences between Ismene and other policy languages and architectures.

5.1. Internet Key Exchange

The Internet Key Exchange [19] (IKE) dynamically establishes security associations (SA) for the IPSec [23] suite of protocols. The IKE phase one exchange negotiates an IKE SA for securing IPSec SA negotiation and key agreement. Policy is negotiated through of a round of policy proposals defining the algorithms and means of authentication protecting the IKE SA.

Figure 4 depicts Ismene policies whose reconciliation models an IKE phase one policy negotiation. The session policy (IKE policy of the responder) and domain policy (IKE policy proposal) are reconciled to arrive at the SA policy. Similar to IPSec selectors, the `selector` condition in the example identifies where the identified policy is relevant. Hence, by creating similar policies with different selectors, it is possible to construct policies for all IPSec traffic supported by a particular host or network; a provision clause and associated selector is created for each class of traffic that requires IKE SA negotiation.

As in IKE negotiation, the reconciliation algorithm intersects the policy proposals resulting in the provisioning of `ike(cast-cbc,sha1,group2)` and `preshare` mechanisms. The reconciliation of the action clauses results in a single `auth` (peer authentication) clause. Note that the `config` condition in the Kerberos `auth` clause is statically evaluated; Kerberos is not configured in the instance, so the clause can never be satisfied. In this case, the clause is removed during reconciliation. The `preshare` action clause (which simply tests whether the peer has proved knowledge of the pre-shared key) is identical in both policies, and thus reconciles to a single condition clause.

5.2. Dynamic Cryptographic Context Management

Designed for policy negotiation in multi-party communication, the Dynamic Cryptographic Context Management (DCCM) [13] system defines a protocol used to negotiate a group session policy. The abstract Cryptographic Context Negotiation Template (CCNT) defines a provisioning policy structure from which the session policy is negotiated [1]. Each CCNT structure is defined as a n-dimensional space of independent services. To simplify, a session policy is constructed by intersecting the points on each dimension satisfying member policy proposals. DCCM does not specify authorization policy.

The creation of session policy DCCM is operationally

GAA-API Printer Policy

Token	Authority	Value
USER	Kv5	joe@acme.edu
rights	manager	submit_job
time_window	PST	6am-8pm
printer_load	lpd	20%

Token	Authority	Value
GROUP	Kv5	operator@acme.edu
rights	manager	submit_job

Ismene Printer Policy

```
submit_job :
  Credential(&tkt,srvr=Kv5,id=joe@acme.edu),
  timeWindow(6am,8pm,pst),
  printerLoad($lp,lpd,20%) :: accept;
submit_job :
  Credential(&tkt,srvr=Kv5,id=operator@acme.edu)
  :: accept;
submit_job :
  Credential(&tkt1,srvr=Kv5,id=joe@acme.edu),
  Credential(&tkt2,srvr=Kv5,id=$id),
  Credential(&del,id=$tkt2.id,
             grantor=$tkt1.id,rghts=submit_job),
  timeWindow(6am,8pm,pst),
  printerLoad($lp,lpd,20%) :: accept;
```

Figure 6. GAA-API Policy - GAA-API defines session-independent authorization policies through extended ACL tokens. The semantics of tokens are realized in Ismene through structured action clause conditionals.

similar to that of IKE; policy is calculated from the intersection of known policy structures. However, where no such intersection exists, an undefined algorithm is used to identify which proposals to reconcile. The extended (prioritized) reconciliation algorithm provides guidance; important member policies are considered first, and others afterward. However, defining a total ordering to the policies frequently requires human intervention.

Ismene session and domain policies modeling the semantics of DCCM policy creation within an example CCNT (from [13]) is depicted in Figure 5. The session policy defines the template CCNT, and domain policies represent policy proposals submitted by expected group members (domain policies). Ismene reconciliation finds the intersection of policies associated with the three essential mechanisms securing the group; confidentiality (`conf`), key management (`kman`), and key management transport (`trans`).

5.3. GAA-API

The Generic Authorization and Access Control API (GAA-API) provides a general-purpose framework for describing authorization in distributed systems [29]. Hence, policy in GAA-API is not session oriented, but used to continuously govern access to resources. Ismene, however, can be used to define non-session policy. Reconciliation and compliance approaches enable administratively disconnected communities to share resources while maintaining the integrity of independent authorization policies.

GAA-API policies, called extended ACLs (EACL), consist of tokens describing the authorization, rights, and conditions of access. Tokens are associated with resources to precisely describe to whom and under what conditions access is granted. Access is allowed where conditions are satisfied and credentials matching the policy statements are found. For example, Figure 6 describes equivalent GAA-API and Ismene authorization policies associated with `acme.edu`'s printers. These policies state that the user `joe` (authenticated by the local Kerberos service) should be allowed to submit print jobs only between 6am and 8pm and when the printer is not loaded. Moreover, the policy states that an operator can always submit a print-job.

The example delegation policy in Figure 6 demonstrates a fundamental difference between GAA-API and Ismene. While GAA-API implicitly permits delegation, Ismene requires the issuer to state a policy allowing it. The Ismene policy states that joe is allowed to delegate (through a delegation credential) the `submit_job` right to any entity authenticated by the same Kerberos service. Moreover, the clause states that conditions under which joe is allowed access are explicitly imposed on any such delegation. For brevity, we omit the operator's right to delegate job submission.

5.4. KeyNote

Central to KeyNote trust management system is the notion of credentials [4, 6]. A credential is a structured policy describing conditional delegation; an authority (authorizer) states that a principal (licensee) has the right to perform some action under a set of conditions. An action is allowed if a delegation chain can be constructed from a credential matching the requested action to a trusted local policy. Users supply credentials as is needed to gain access. Hence, KeyNote significantly eases the burden of policy management by allowing policy to be distributed to users, rather than configured at all policy enforcement points. The KeyNote policy depicted in Figure 7 delegates decisions about IPSec policy to the `ADMIN_KEY`, and restricts the provisioning to a range of cryptographic algorithms. The `ADMIN_KEY` credential encapsulates a policy that the user Bob (who is identified by a key) should be allowed access if IPsec is configured with the 3-DES or CAST encryption algorithms and SHA-1 HMACs are used for message authentication.

The Ismene policies state a similar requirement, while also providing a reconciliation algorithm for generating an acceptable policy instance to provision the session. How-

KeyNote Local Policy

```
Authorizer: POLICY
Licensees: ADMIN_KEY
Conditions: app_domain == ‘IPsec policy’
  && ( esp_enc_alg = ‘3des’ ||
       esp_enc_alg = ‘aes’ ||
       esp_enc_alg = ‘cast’ )
  && ( esp_auth_alg = ‘hmac-sha’ |
       esp_auth_alg = ‘hmac-md5’ )
```

Ismene Session Policy

```
ADMIN_KEY := < 0xba34... >;
provision : ::
  pick( config(esp_enc_alg(3des)), config(esp_enc_alg(aes)),
        config(esp_enc_alg(cast)) ),
  pick( config(esp_auth_alg(hmac-sha)),
        config(esp_auth_alg(hmac-md5)) );
accept_policy :
        Credential( &policy, policy.issuer=$ADMIN_KEY )
              :: accept;
```

KeyNote IPSec Credential

```
Authorizer: ADMIN_KEY
Licensees: Bob
Conditions: app_domain == ‘IPsec policy’
  && ( esp_enc_alg = ‘3des’ ||
       esp_enc_alg = ‘cast’ )
  &&  esp_auth_alg = ‘hmac-sha’ ;
```

Ismene Domain Policy

```
signer := < 0xba34... >;
signature := < 0x98cc... >;
id := < Bob >;
provision : ::  pick( config(esp_enc_alg(3des)),
                      config(esp_enc_alg(cast)) ),
                config(esp_auth_alg(hmac-sha));
```

Figure 7. KeyNote Policy - KeyNote credentials are only consulted where they have been explicitly delegated authority by a local policy. Conversely, Ismene regulates the acceptance of policy through the proper assignment of `accept_policy` conditions.

ever, one facet of KeyNote not captured in Ismene is the explicit delegation of policy; KeyNote credentials are only consulted where they have been explicitly delegated authority by a local policy. In contrast, Ismene does not make any assumptions about the origin and authentication of policy, but focuses on the construction of session policy. In the example, the Keynote delegation approach is partially modeled in the Ismene policies. The session policy is consulted for the `accept_policy` action prior to the acceptance of any domain policy and accepted where signed by `ADMIN_KEY`. In this case, Ismene enforces policy through reconciliation; only instances consistent with the KeyNote conditions can result from reconciliation.

6. Implementing Ismene

The *Ismene Applications Programming Interface* (IAPI) defines interfaces for the creation, parsing, reconciliation, and analysis of Ismene policies[5] . The Ismene policy compiler, `ipcc`, validates the syntax of a session and domain policies and implements the algorithms presented in Section 3. We have further integrated IAPI with Antigone communication system [27], and used it as the basis for several non-trivial diverse group applications [28]. These include a group white-board, file-system mirror, and reliable group services. Our experience indicates Ismene is sufficiently powerful to capture a wide range of application-specific policies. The investigation also suggested areas of further study:

[5]All source code and documentation for the Ismene language, the augmented Antigone communication system, and applications are freely available from `http://antigone.eecs.umich.edu/`.

Performance - The enforcement of fine-grained access control can negatively affect performance. For example, one file-system mirroring policy requires the evaluation of `send` action clauses prior to each packet transmission. Such evaluation slowed file transfers. We noted that because action clause evaluation was often invariant, results could be cached. We present the design of a policy evaluation cache and a comprehensive study of enforcement performance in [26]. Caching significantly mitigated the cost of policy enforcement.

Authorization Reconciliation - As authorization policies defined by an instance are constructed from the conjunction of the session and domain policies, clauses can become restrictive. For example, consider the case where the session policy requires, for some action, the presentation of an X.509 certificate, and a domain policy require the presentation of a Kerberos ticket. In this case, the resulting instance requires that both a certificate and a ticket be presented. We are currently investigating ways in which overly-restrictive or unsatisfiable authorization policies can be detected at reconciliation time or at run-time.

Policy Dependencies - The effectiveness of analysis is predicated on the correct construction of policy assertions. In practice, mechanisms and configurations have complex relationships. Assertion construction requires a comprehensive knowledge of use of the cryptographic algorithms, protocols, and services. This knowledge must be reflected in the policy construction. This situation is not unique to Ismene; any policy infrastructure must ensure that unsafe instances are rejected.

7. Conclusions

In this paper, we have presented a model and language for the specification and reconciliation of security policies. We show that the general problem of reconciliation is intractable. However, by restricting the language, we show that reconciliation of two policies becomes tractable. Reconciliation of three or more policies remains intractable. We identify heuristics that detect situations where intractability is likely to occur and prioritize policies during reconciliation to achieve efficient reconciliation.

A compliance algorithm determines whether a policy instance is consistent with a participant's domain policy. The analysis algorithm determines whether the provisioning of a session adheres to a set of assertions that express correctness constraints on a policy instance. We identify efficient algorithms for both compliance and analysis. We demonstrate that the more general problem of determining if any instance generated from a policy can violate a set of correctness assertions is intractable (in coNP).

Based on the model, we presented an overview of the Ismene policy language and demonstrated its expressiveness and limitations through the representation of policies defined in several policy languages. The language has been implemented and is being used in several non-trivial applications.

Networks are becoming more open and heterogeneous. This stands in stark contrast to the singular nature of contemporary security infrastructures; communication participants have limited ability to affect session policy. Hence, the participant security requirements are only addressed inasmuch as they are foreseen by policy issuers. Ismene, and works similar to it, seek to expand the definition and usage of policy such that run-time policy is the result of the requirements evaluation, rather than dictated by the policy issuers.

8. Acknowledgments

We would like to thank Peter Honeyman for his many contributions to this work within the Antigone project. We would also like to thank Avi Rubin, Sugih Jamin, Trent Jaeger, Paul Resnick, and the anonymous reviewers for there many thoughtful comments.

References

[1] D. Balenson, D. Branstad, P. Dinsmore, M. Heyman, and C. Scace. Cryptographic Context Negotiation Template. Technical Report TISR #07452-2, TIS Labs at Network Associates, Inc., February 1999.

[2] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31, 1999.

[3] S. Bellovin. Distributed Firewalls. *;login:*, pages 39–47, 1999.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, volume 1603, pages 185–210. Springer-Verlag Lecture Notes in Computer Science State-of-the-Art series, 1999. New York, NY.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, November 1996. Los Alamitos.

[6] M. Blaze, J. Feignbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System - Version 2. *Internet Engineering Task Force*, September 1999. RFC 2704.

[7] D. C. Blight and T. Hamada. Policy-Based Networking Architecture for QoS Interworking in IP Management. In *Proceedings of Integrated network management VI, Distributed Management for the Networked Millennium*, pages 811–826. IEEE, 1999.

[8] D. Branstad and D. Balenson. Policy-Based Cryptographic Key Management: Experience with the KRP Project. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX '00)*, pages 103–114. DARPA, January 2000.

[9] L. Cholvy and F. Cuppens. Analyzing Consistancy of Security Policies. In *1997 IEEE Symposium on Security and Privacy*, pages 103–112. IEEE, May 1997. Oakland, CA.

[10] Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust Management for Web Applications. In *Proceedings of Financial Cryptography '98*, volume 1465, pages 254–274, Anguilla, British West Indies, February 1998.

[11] S. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of 3th Annual ACM Symposium on Theorey of Computing*, pages 151–158. ACM, 1971.

[12] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[13] P. Dinsmore, D. Balenson, M. Heyman, P. Kruus, C. Scace, and A. Sherman. Policy-Based Security Management for Large Dynamic Groups: A Overview of the DCCM Project. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX '00)*, pages 64–73. DARPA, January 2000. Hilton Head, S.C.

[14] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. RFC 2748, The COPS (Common Open Policy Service) Protocol. *Internet Engineering Task Force*, January 2000.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractibility, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, NY, first edition, 1979.

[16] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theoretical Computer Science*, (1):237–267, 1976.

[17] L. Gong and X. Qian. The Complexity and Composability of Secure Interoperation. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 190–200, Oakland, California, May 1994. IEEE.

[18] R. Greenlaw, H. Hoover, and W. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, first edition, 1995.

[19] D. Harkins and D. Carrel. The Internet Key Exchange. *Internet Engineering Task Force*, November 1998. RFC 2409.

[20] M. Hiltunen. Configuration Management for Highly-Customizable Software. *IEE Proceedings: Software*, 145(5):180–188, 1998.

[21] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. *Internet Engineering Task Force*, January 1999. RFC 1949.

[22] S. Jajodia, P. Samarati, and V. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA, March 1997.

[23] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. *Internet Engineering Task Force*, November 1998. RFC 2401.

[24] T. Leighton and S. Micali. Secret-key Agreement without Public-Key Cryptography. In *Proceedings of Crypto 93*, pages 456–479, August 1994.

[25] X. Liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable. Building Reliable High-Performance Communication Systems from Components. In *Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, volume 33, pages 80–92. ACM, 1999.

[26] P. McDaniel. *Policy Management in Secure Group Communication*. PhD thesis, Univeristy of Michigan, Ann Arbor, MI, August 2001.

[27] P. McDaniel, A. Prakash, and P. Honeyman. Antigone: A Flexible Framework for Secure Group Communication. In *Proceedings of the 8th USENIX Security Symposium*, pages 99–114, August 1999.

[28] P. McDaniel, A. Prakash, J. Irrer, S. Mittal, and T. Thuang. Flexibly Constructing Secure Groups in Antigone 2.0. In *Proceedings of DARPA Information Survivability Conference and Exposition II*, pages 55–67. IEEE, June 2001.

[29] T. Ryutov and C. Neuman. Representation and Evaluation of Security Policies for Distributed System Services. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 172–183, Hilton Head, South Carolina, January 2000. DARPA.

[30] T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of 10th Annual ACM Symposium on Theorey of Computers*, pages 216–226. ACM, 1978. New York, New York.

[31] T. Woo and S. Lam. Authorization in Distributed Systems; A New Approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.

[32] T. Woo and S. Lam. Designing a Distributed Authorization Service. In *Proceedings of INFOCOM '98*, San Francisco, March 1998. IEEE.

[33] J. Zao, L. Sanchez, M. Condell, C. Lynn, M. Fredette, P. Helinek, P. Krishnan, A. Jackson, D. Mankins, M. Shepard, and S. Kent. Domain Based Internet Security Policy Management. In *Proceedings of DARPA Information Survuvability Conference and Exposition*, pages 41–53. DARPA, January 2000.

Appendix A - Unrestricted Policy Reconciliation (UPR)

The following construction reduces Positive, ONE-IN-THREE 3SAT to UPR in polynomial time. We begin with definitions these algorithms.

Definition 1 (Unrestricted Policy Reconciliation (UPR))
Given: A session policy g .
Question: What is an instance satisfying all configuration and pick statements in g?

Definition 2 (Positive, ONE-IN-THREE 3SAT (13SAT+))
Given: Set U variables, expression $e = C$ disjunctions over U such that each $c \in C$ has $|c| = 3$, no negated literals.
Question: Is there a truth assignment for U such that each clause in C has exactly one *true* literal?

Construction: Assume $U = \{x_1, x_2, \ldots, x_n\}$. For each $c_i \in C, c_i = (x_1 \vee x_2 \vee x_3)$, create the pick statement $pick(x1, x2, x3)$. For example, the expression $(a \vee b \vee c) \wedge (a \vee c \vee d) \wedge (b \vee d \vee e)$ would generate the following policy:

$$g = pick(a, b, c), pick(a, c, d), pick(b, d, e)$$

Now assume a polynomial-time algorithm for UPR exists. Any instance resulting from UPR must specify exactly one configuration from each pick statement. Trivially, such an instance represents satisfying truth assignment for e. Hence, because 13SAT+ is NP-complete [30], so is UPR. □

Appendix B - Largest Subset Reconciliation (LSR)

The following construction reduces MAX2SAT to LSR in polynomial time. We begin with definitions for LSR and MAX2SAT.

Definition 3 (Largest Subset Reconciliation (LSR))
Given: A session policy g and a set of domain policies L to be considered by reconciliation.
Question: What is the largest $\hat{L} \subseteq L$ such that g and all policies $l_i \in \hat{L}$ are successfully reconciled?

Definition 4 (MAX2SAT)
Given: The set U variables, conjunction of C disjunctions over U such that each $c \in C$ has $|c| = 2$, and a positive integer $K \leq |C|$.
Question: Is there a truth assignment for U that simultaneously satisfies at least K of the clauses in C?

Construction: Assume $U = \{x_1, x_2, \ldots, x_n\}$. For each $c_i \in C, c_i = (x_1 \vee x_2)$, create three domain policies:

$$\begin{array}{l} l_{c_{11}} : pick(x_1), pick(\bar{x_2}), pick(x_3, \bar{x_3}), \ldots, pick(x_n, \bar{x_n}) \\ l_{c_{12}} : pick(\bar{x_1}), pick(x_2), pick(x_3, \bar{x_3}), \ldots, pick(x_n, \bar{x_n}) \\ l_{c_{13}} : pick(x_1), pick(x_2), pick(x_3, \bar{x_3}), \ldots, pick(x_n, \bar{x_n}) \end{array}$$

Note that each policy describes *mandatory* configurations (pick statements containing only one configuration). Negative variables are inverted. For example, the following domain policies are generated for the expression $c_1 = (a \vee \bar{b})$ over $U = \{a, b, c\}$:

$$\begin{array}{l} l_{c_{11}} : pick(a), pick(b), pick(c, \bar{c}) \\ l_{c_{12}} : pick(\bar{a}), pick(\bar{b}), pick(c, \bar{c}) \\ l_{c_{13}} : pick(a), pick(\bar{b}), pick(c, \bar{c}) \end{array}$$

Create the session policy by creating a pick statement for each variable in U as follows:

$$g = \forall v_i \in U : pick(v_i, \bar{v_i})$$

Returning to the example above (where $U = \{a, b, c\}$),

$$g = pick(a, \bar{a}), pick(b, \bar{b}), pick(c, \bar{c}).$$

Note by this construction, reconciliation g with the set of all domain policies (L) satisfies at most 1 of the clauses associated with each c_i. Each domain policy represents the (mutually exclusive) ways in which each clause c_i can be satisfied, and the reconciliation of g with D is simply a truth assignment for U.

Assume a polynomial time algorithm exists for LSR. Answering MAX2SAT simply becomes the process of reconciling the policies resulting from the construction. If $|\hat{L}| \geq K$, then MAX2SAT returns *true*, and *false* otherwise. Thus, because MAX2SAT is a known NP complete problem [16], LSR is NP complete. □

Appendix C - Offline Policy Analysis (OFPA)

The following construction reduces VALIDITY to OFPA in polynomial time. We begin with definitions for VALIDITY and OFPA.

Definition 5 (Offline Policy Analysis (OFPA))
Given: A session policy g and set of assertions S.
Question: Would any reconciliation of g with arbitrary domain policies violate an assertion in S?

Definition 6 (VALIDITY)
Given: An arbitrary Boolean expression e defined over the variables U. For convenience, we assume e is in DNF.
Question: Is e valid?

Construction: Create g by defining a *provision* clause containing the tag consequence (l_1), and four clauses for each variable $x_i \in U$ as follows;

$$\begin{array}{l} provision : :: l_1; \\ l_1 : x_1, \overline{x_1} :: fail; \\ l_1 : x_1 :: l_2; \\ l_1 : \overline{x_1} :: l_2; \\ l_1 : :: fail; \\ l_2 : x_2, \overline{x_2} :: fail; \\ \ldots \\ l_i : x_i :: p; \\ l_i : \overline{x_i} :: p; \\ l_i : :: fail; \end{array}$$

Note that the last set of clauses for $x_i \in U$ references a tag to the clauses for p. For each conjunct $c_i \in e$, create the clause $p : c_i :: r;$, where the conditionals enumerate the (possibly negated) variables of c_i, and r is a arbitrary configuration. Appending a default clause containing a single f configuration ($p : \ :: f;$), and a fail clause ($fail : \ :: t;$). Complete the construction by creating a single assertion ($assert : ::!f;$). To illustrate, an expression $(a \wedge b \wedge c) \vee (\overline{a} \wedge \overline{b} \wedge d) \vee (\overline{c} \wedge \overline{d} \wedge e)$ would result in the following g and S;

$$\begin{array}{rl} g = & provision : :: l_1; \\ & l_a : a, \overline{a} :: fail; \\ & l_a : a :: l_b; \\ & l_a : \overline{a} :: l_b; \\ & l_a : :: fail; \\ & l_b : b, \overline{b} :: fail; \\ & \ldots \\ & l_e : :: fail; \\ & p : a, b, c :: t; \\ & p : \overline{a}, \overline{b}, d :: t; \\ & p : \overline{c}, \overline{d}, e :: t; \\ & p : :: f; \\ & fail : :: t; \\ S = & assert : :: \ !f; \end{array}$$

Now consider the possible evaluations of g. Each positive or negative assignment of variable $x_1 \in U$ is defined as a unique condition. The evaluation of the clauses l_1 has two possible results; if the condition x_1 and $\overline{x_1}$ are both true or neither is, the evaluation algorithm will immediately drop to the $fail$ clause which defines a single condition t. In this case, the assertion test will trivially be satisfied by this evaluation. If exactly one of the conditions x_1 and $\overline{x_1}$ is TRUE, then the clauses associated with x_2 are consulted. This process repeats until either the $fail$ clause or the first clause associated with p is reached. If the first p clause is reached, then the conditions represent a legal truth assignment for U. Moreover, it is clear that no legal truth assignment for U arrives at $fail$.

Now, consider the evaluation of the clauses of p. Because e is represented in DNF, any truth assignment for U must satisfy at least one conjunct for e to be valid. The evaluation of some p clause will arrive at configuration t if any conjunct is satisfied by the truth assignment for U, and f otherwise. If e is valid, the final p clause can never be reached (because all legal truth assignments satisfy at least one conjunct of e), and the assertion can never be violated. Hence, the negation of the answer returned by OFPA is the answer for VALIDITY (OFPA returns *false*, where e is valid and *true* otherwise). Because VALIDITY is a known to be in coNP-complete, so is OFPA. $\square$