July 2006

Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies

Michael J. May
University of Pennsylvania, mjmay@cis.upenn.edu

Carl A. Gunter
University of Illinois, cgunter@cs.uiuc.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies

Abstract
There is a growing interest in establishing rules to regulate the privacy of citizens in the treatment of sensitive personal data such as medical and financial records. Such rules must be respected by software used in these sectors. The regulatory statements are somewhat informal and must be interpreted carefully in the software interface to private data. This paper describes techniques to formalize regulatory privacy rules and how to exploit this formalization to analyze the rules automatically. Our formalism, which we call privacy APIs, is an extension of access control matrix operations to include (1) operations for notification and logging and (2) constructs that ease the mapping between legal and formal language. We validate the expressive power of privacy APIs by encoding the 2000 and 2003 HIPAA consent rules in our system. This formalization is then encoded into Promela and we validate the usefulness of the formalism by using the SPIN model checker to verify properties that distinguish the two versions of HIPAA.

Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies*

Michael J. May
University of Pennsylvania
mjmay@cis.upenn.edu

Carl A. Gunter
University of Illinois Urbana-Champaign
cs.uiuc.edu/~cgunter

Insup Lee
University of Pennsylvania
lee@cis.upenn.edu

Abstract

There is a growing interest in establishing rules to regulate the privacy of citizens in the treatment of sensitive personal data such as medical and financial records. Such rules must be respected by software used in these sectors. The regulatory statements are somewhat informal and must be interpreted carefully in the software interface to private data. This paper describes techniques to formalize regulatory privacy rules and how to exploit this formalization to analyze the rules automatically. Our formalism, which we call privacy APIs*, is an extension of access control matrix operations to include (1) operations for notification and logging and (2) constructs that ease the mapping between legal and formal language. We validate the expressive power of privacy APIs by encoding the 2000 and 2003 HIPAA consent rules in our system. This formalization is then encoded into Promela and we validate the usefulness of the formalism by using the SPIN model checker to verify properties that distinguish the two versions of HIPAA.*

1 Introduction

An increasing number of government agencies and enterprises are finding a need to write down privacy rules for their handling of personal information of parties who entrust such information to them. These rules are derived from a complex set of requirements laid down by diverse stakeholders; they are often complex and may contain important ambiguities and unexpected consequences. Examples include sector-specific rules like the Health Insurance Portability and Accountability Act (HIPAA) [41] and Gramm-Leach-Bliley Act [20] in the US and comprehensive privacy rules established by the European Economic Community [38, 39]. Increasing automation of data management using computer systems invariably means that these privacy rules become requirements for software systems that manage data affected by privacy rules. Analyzing conformance to these rules requires careful comparison of the 'privacy API' of a computer system with the (typically informal) regulatory rule sets. This paper develops a technique for bridging the gap between regulatory rule sets and specifications that are precise enough to be implemented without ambiguity and formal enough to be analyzed by automatic means for the criteria that stakeholders for the regulatory rule sets care about.

A variety of efforts have been made to develop languages for expressing rules for privacy. However, there have been few practical case studies for these approaches and there remain many questions about exactly which features are really needed. We examinee this issue from the bottom up by taking a practical regulatory rule set, the HIPAA consent rules, and explore whether anything beyond techniques developed almost 3 decades ago are required to express them. In particular, we ask whether the HIPAA consent rules can be expressed using a basic set of access control matrix operations from the classic access control systems literature. We find that this is not possible directly, but we identify a modest set of extensions that are capable of expressing the complete rule set. Our extensions require us to add explicit operations for notification and logging and to cover conditions and obligations that typically cannot be directly verified by a computer. In addition, it is important for purposes of readability and textual accuracy to include certain binding constructs to reflect legal phrasing where terms in one clause are taken to be 'as in' another clause. With these few additions we are able to describe sets of rules which we call *privacy APIs* that are capable of expressing the HIPAA rules with such precision that we are able to en-

*In: Computer Security Foundations Workshop (CSFW), July 2006, Venice, Italy. This research was supported in part by NSF CCF-0429948, ARO DAAD19-01-1-0473, and ARO W911NF-05-1-0158.

code them in a modelling language (Promela) and use a model checker (SPIN) to demonstrate consequences of the rule sets of the kind that concerned key stakeholders enough to precipitate rule changes from one version to the next.

The value of this contribution is two-fold. First, we are able to set a baseline of constructs sufficient to provide a formalization of practical rules that can be read by a human. This aids readers in determining conformance of the encoding to the informal regulatory rules on the one hand and the software on the other hand. Second, we are able to provide a formalization that can be used for automated analysis to explore consequences of the rule set that may be surprising and undesirable.

The rest of this paper is organized as follows. In Section 2 we present background on access control and define auditable privacy systems. In Section 3 we describe privacy APIs. In Section 4 we present a study of the HIPAA consent requirements. In Section 5 we show our formal model, queries, and results. In Section 6 we discuss related work. We conclude in Section 7.

2 Background

In this work, we use the style of commands introduced by Harrison, Ruzzo, and Ullman (HRU) [30] to create a formalism we call an *auditable privacy system* and use it to let us better understand and verify legal privacy requirements. In this section we describe the properties of the classic HRU model, define auditable privacy systems, and describe the limitations of the classic HRU model in directly modelling them.

2.1 Access Control and HRU

There are two general types of access control policies, discretionary access control (DAC) and mandatory access control (MAC). In DAC policies, an object's owner has control over which permissions are granted to others. In MAC policies, policy rules determine permissions and access to objects, normally without the input of the owner.

Originator Control (ORCON) policies [26] are a special kind of MAC that give rights to the *originators* of system objects even if those objects are no longer owned by them. For example, if Alice creates a file and Bob makes a copy of it, only Alice can grant new permissions on Bob's copy, even if Bob modifies it. The policy is mandatory since the system, not Bob, determines how the originator designation is assigned. The ORCON idiom is commonly used for policies where one principal holds or collects another principal's information.

Legal policy about the extent of ORCON rights that people have over information about them varies by country and area of law. The laws that we discuss in this work give a weak level of control to the data *subject*, the principal that an object is "about," and a stronger level of control to the data

One of the early models for access control systems is the HRU model. Rules in the language are transaction-style commands that execute in sequence on a single system. The model uses a single access control matrix to store the state of the system. *Primitive operations* of the system manage the reading and writing of rights and the creation and deletion of principals and objects in the matrix. *Commands* consist of (optional) conditions and a series of primitive operations that are executed transactionally. The policy of a system is the set of commands that the system publishes. Since the policy of the system is determined by the functionality of the full set of commands, in order to validate the properties of the system, the full set of commands must be analyzed and verified.

We can view a system's command set in the idiom of an interface or API (application programmers interface). Since the access control matrix is only accessible through the commands we can focus on the verification of the commands to gain confidence in the safety of the system as a whole.

2.2 Auditable Privacy System

We extend the theory and formal Privacy System of Gunter, *et al.* [28] with auditing to produce an *auditable privacy system*. The basic events of an auditable privacy system are:

Transfer. Transferring an object by copying it and passing the copy to the recipient's domain. Objects may be transferred with or without ORCON restrictions.

Action. Using a private object for some purpose. Uses include low level events like read and write as well as scenario based events such as treatment for medical needs and billing for services. Often we need a method of auditing what actions occurred and for what purpose.

Creation. The addition of new objects or principals to the system and access control matrix. Principals may create new objects about themselves or other others if authorized. Often we require a method of auditing the creation of objects and principals.

Rights Establishment. A subject or owner of an object granting or revoking permissions on it to principals, roles, or groups of principals.

Notification. An obligation to inform concerned

principals about events performed by other principals in the system. It is commonly imposed as an enforcement mechanism where system level verification is not sufficient.

Logging. An obligation on the system to track events–both those that succeed and those that fail. It is used for system verification and auditing.

We call a system that includes all the above events and obligations, that includes both run time and post-hoc auditing of the behavior of principals and the transfer of objects through domains, an *Auditable Privacy System.* We use the six primitive operations from the HRU model: **enter** r **into** (p, o) and **delete** r **from** (p, o) for rights establishment; **create principal** p, **create object** o, **destroy principal** p, and **destroy object** o for creation. Their semantics are given in HRU's original paper [30] and reproduced in Appendix B for reference. We add two primitive operations: **inform** p **of** t for notification; and **log** t for logging. Transfer is accomplished by creating a new object and inserting the old object's content.

Since an auditable privacy system is ORCON based and considers objects being "about" its *subject* and created by its *originator*, we need the following non-primitive commands that layer over the primitive operations:

command	CreateObject (a, s, o)
	create object o
and	**enter** originator **in** (a, o)
and	**enter** subject **in** (s, o)
end	
command	CopyObject (a, s, o, o′)
if	originator **in** (a, o)
and	subject **in** (s, o)
then	**create object** o′
and	**enter** originator **in** (a, o′)
and	**enter** subject **in** (s, o′)
end	

2.3 Limitations of the HRU model

While the HRU model gives the tools to model commands that affect the access control matrix, by itself it has shortcomings in implementing an auditable privacy system.

First, there is no baked-in notion of the initiator (actor) of a command. Commands may take arguments, one of which can be designated for the actor, but there is no primitive in the system supporting this or providing authentication. This is important because privacy policies normally predicate rules on the actor in a command.

Second, there are no roles or groups in the classic HRU system. Policies often define rules in terms of roles so omitting that functionality makes maintaining permissions significantly more complex.

Third, the commands in the HRU system are restricted to conditions and primitive operations that affect the state of the access control matrix, greatly limiting the power of the commands that we can write. We would like to include environment based conditions and operations that include logging and notification.

Fourth, in order to enforce ORCON policies we need to create non-primitive operations that act as an interface to the primitive operations. These operations enforce the MAC properties of the system by acting as an interface to the system. The HRU model does not allow commands to invoke other commands, so this functionality must be added.

Finally, since the commands are limited to reading and modification of the access control matrix, there is no way to include outside obligations in the commands.

These shortcomings require us to modify the classic HRU model in the following ways.

First, we give each command an implicit argument called *actor* which is assigned the value of the actor in a command. We thereby rely on the system's authentication for identifying the actor in a command.

Second, we use groups to provide rudimentary roles. Each group is an object in the system and principals have permission **member** on it if they are members of the group. We do not handle the general trust management issue since it is not needed here. Some roles indicate affiliations or relationships between principals, so in those cases we place indicative permissions in the matrix as appropriate (*e.g.* Alice has relationship **doctor** on Bob).

Third, we add a global append-only log and a mechanism for informing principals of events that occur in the system, the two outside operations needed for auditable privacy systems.

Fourth, we allow commands to invoke other commands and hide some primitive operations. This enables us to write commands that function as an interface to the underlying access control matrix.

Finally, we discern two kinds of obligations: (1) obligations that the computer system can perform or verify and (2) obligations that the computer system can not perform or verify. For the first type, the computer system performs the obligation or validates that the obligation is performed before the command completes. We address the second type in the next section.

By making the above adaptations we have a model that can fully implement auditable privacy systems.

We use auditable privacy systems to underly privacy APIs, which we define next.

3 Privacy APIs

Legal privacy rules are complex natural language documents written by a large group of people. Their length and complexity often make them prohibitively difficult to understand. However, by observing a similarity between legal privacy policies and APIs we can gain some traction.

Both APIs and legal privacy policies publish a set of allowed command combinations (procedures) to access a database of protected information (system state) and disallow all others. Verification consists of checking that each procedure is safe and that no combination of procedures will lead to a disallowed state.

Using this similarity, we convert legal privacy policies to access control rules that are an interface to a set of private information. The set of procedures that we derive from a legal policy document is its *privacy API* which we formally define below. The policy goals are the system invariants. We can use formal methods tools such as model checking to explore the states of the system and verify that the privacy API enforces the system invariants.

The invariants of a legal privacy policy are a function of the influences of stakeholders involved in its design. Each stakeholder's invariants reflect its ideals which are often in conflict with other stakeholders. Policy writers design a system that satisfies some of the invariants of each, hopefully creating a reasonable amalgam.

Stakeholders can use their ideals to define a tailored set of invariants that lets them find the relative acceptability of a policy as it evolves. When a policy changes, they evaluate whether the new system is more or less acceptable. We can use the privacy API model to streamline that evaluation.

Since legal privacy policies give rules for the interaction of users and information, they normally include conditions and obligations that a computer system can not perform or verify. In some cases, the conditions and obligations require an expert to evaluate properly (*e.g.* when deciding if a purpose is reasonable). They are part of the requirements, events that the system specification can not observe, so we must treat them differently in our model. Our model verifies and enforces what it can and for the rest it relies on the environment. By factoring out unenforceable conditions and obligations, we achieve a model that is easier to design and verify.

From a more theoretical view, outside conditions and obligations are environment variables that are invisible to the system specification [27]. The system specification needs assurance that the environmental variables are correct, however, so we must create a bridge between them. Flags accomplish this by allowing principals to make assertions about the environment that the computer system can check to impose outside conditions and obligations. The flags can be tied to particular principals in the system (*e.g.* patient gives a consent form that is signed and dated) or be non-principal-specific conditions (*e.g.* disclosure document reserves the right to change).

In our model implementation, when the model can not perform or verify a condition or obligation, it must either (1) allow the command and set a flag that indicates an obligation must be fulfilled, (2) look for an assertion that the required obligation has already been fulfilled, or (3) forbid the execution until an authorized principal authorizes it. The third case is similar to adding another condition to the command and removing the hindering condition, so we deal with it in that manner. The first and second cases are handled with environment flags. The second case finds parallels in legislative policies which use post-hoc auditing and notification for enforcement. Conditions of the last two types are an *if condition* which both returns a boolean and a checks or returns an obligation with flags. We call such a condition an *if with obligation*, or *OIF*.

Example: The following is a selection from the Gramm-Leach-Bliley Act (GLB) financial services privacy [20] requirements [15 USC Subchapter 1 §6802(b)(1)(B)]: "A financial institution may not disclose nonpublic personal information to a nonaffiliated third party unless ... (B) the consumer is given the opportunity, before the time that such information is initially disclosed, to direct that such information not be disclosed to such third party;"

Using our adapted HRU syntax[1] we translate the quote as follows. Note that since GLB includes ORCON restrictions [15 USC Subchapter 1 §6802(c)], we use the ORCON **CopyObject** command.

COM	Disclose(a, s, r, f, p, ev)
if	"third party disclose" **in** p
and	affiliated **not in** (a, r)
and	member **in** (s, consumer_group)
and	"consumer given chance to opt-out" **in** ev
and	opt-out **not in** (s, r)
then	CopyObject (a, s, f, f′)
and	**enter** own **in** (r, f′)
end	

[1] In this work we use the following abbreviations for parameters: (a)ctor, (s)ubject, (r)ecipient, (f)ile, (p)urpose, (ev)idence.

Alice executes the above command, trying to disclose a file F about Sam, a consumer, to Rick, a non-affiliated third party. First the command checks that the purpose is for a third party disclosure. Then, if it finds that **affiliated** $\notin M$[Alice, Rick], that Sam is a member of **consumer_group**, that environmental flag **consumer given opportunity to opt-out** is asserted true, and that **opt-out** $\notin M$[Sam, Rick], it does an ORCON copy of F to create F′ and transfers the new copy to the ownership of Rick. □

We can use our model to translate larger sections of legal privacy policies, giving a large set of commands and conditions.

4 Translating HIPAA Consent

Part of the US Health Insurance Portability and Accountability Act (HIPAA) [41] is a Privacy Rule which governs the management, storage, and distribution of patient health information. Since hospital electronic health care records systems are normally complex and often nonuniform, assuring compliance with the rule is hard. Health care entities are required by law to have a Privacy Officer in charge of enforcing compliance with the rule, so hospital administrators and staff need not learn the details of the rule. Instead they are trained in the basics while specialized health care software enforces the access control rules.

Our goal is to provide a model to let policy writers and enforcers better understand allowed flows of information, allow comparisons between versions of the law, and allow verification of the law's intended invariants. To do this, we translate a section of HIPAA rules into auditable privacy system commands and use formal methods model checking tools to verify properties of the system. We study the rules about patient consent for treatment, payment, and health care operations.

4.1 Consent

The rules about when health care entities must get patient consent before performing treatment, payment, and health care operations activities [§164.506] were the subject of debate during the development of HIPAA. The version of the rules from 2000 [23] contains many provisions and exceptions, so the rules are about one page long. The later version of the rules from 2003 [25] was simplified, so it is only about one third of a page long. For this study, we translate both versions of the consent rules and use a model checker to perform queries that explore their differences.

The system requirements for the HIPAA privacy rules contain many low level requirements, for example rules regarding the placement of computer terminals and password management. Our model abstracts away such low level issues and so has only the following system requirements: each principal in the system has a unique name and is authenticated; there is a database of private records in which each file is associated with its subject; the computer system has access to the date and time; the computer system has can test boolean equality, string equality, and test for the presence of an item in a set; the computer system has an append only log; the system has a method of notifying principals when needed. This abstraction lets us focus on the functionality of the law's access control requirements, those that govern the disclosure and use of private information.

We translate the English rules into the modified HRU syntax but preserve the original structure of the policy. Each paragraph or clause in the law is translated into one or more commands. The reference structure of clauses is preserved by allowing commands to call other commands in a function-like manner. Complex conditions that may be referenced by multiple clauses or rules are translated as *if* commands (or "AsIn" commands since their titles normally contain that phrase) that may be called by other if commands and can return true or false. We discuss the need for this below in section 4.3.

As noted above, the consent rule was considerably rewritten between the two versions that we considered, so the two rule sets vary considerably in length. The 2000 version rule set contains 5 helper commands and 60 regular and `if` commands, all local to [§164.506]. There are 26 environmental evidence flags included. The 2003 version rule set contains 5 helper commands and 54 regular and if commands with 21 of them local to [§164.506] and 33 of them non-local. There are 74 environmental evidence flags included.

HIPAA requires that events be logged [164.308(a)(ii)(D), v.2003]. This requires a system to audit events logs, so all commands in the system must perform an implicit logging operation. For conciseness, however, our verification model only records events that other commands later look for.

4.2 Translation Methodology

In our translation, we strove to stay as close to the structure of the text as possible. To this end, we designed a methodology which made translation straightforward and observably close to the text.

Each paragraph in the text has one or more com-

mands that execute it. Paragraphs are translated into multiple commands when they allow or deny multiple actions (*e.g.* use and disclose). A paragraph has at most one if command (AsIn) which includes or references all the conditions needed for the paragraph. The result is a rule set whose size is linear in the number of paragraphs.

When paragraphs reference each other, whether directly or implicitly, we locate and translate the referenced clause and include an explicit call to it.

Parent paragraphs that refer to their children include their children as references. Where appropriate, child paragraphs have their own commands. A child paragraph may reference its own specific AsIn, its parent's AsIn, or both.

Each condition or obligation in a paragraph is included in the paragraph's AsIn. If the condition or obligation is unrelated to access control (*e.g.* [§164.520] which has typographic rules for privacy practices disclosures documents) then it is checked as an environmental flag. We follow the language of the text, so unless two conditions or obligations are phrased very similarly or have obviously the same intent they have separate flags. The number of environment flags included in each model is dependent on the writing style of each paragraph.

One of the most important factors in deciding whether a request should be permitted or denied is the purpose for the request. Some requests may be made for more than one purpose, so we provide each command with a list of purposes. Commands receive a purpose set as a parameter so the set can be passed on to other commands that may be referenced. We use this method because it is similar to the textual style– paragraphs and clauses are divided and parameterized by purpose.

4.3 Challenges in Translation

It is not surprising that legal privacy policies differ in style from standard computer systems access control policies, but two common features we came upon made the distinction very sharp.

The first distinguishing feature is the way that references are used. Commonly, a paragraph refers to the conditions of another paragraph independent of its body. For example, [§164.506(a), v.2003] "Except with respect to uses or disclosures that require an authorization under [§164.508(a)(2)]" is a condition that points to the conditions of the referenced paragraph, but does not intend to activate the functionality of it. This is akin to a procedure creating a condition out of the precondition of another procedure without executing the referenced procedure. Because this is not a common programming language idiom, we implement it by dividing each paragraph into two parts: an *if command* (AsIn) which contains the conditions and one or more regular commands that reference the associated AsIn. This separation allows us to keep the reference structure of the law.

Legal references also vary in specificity. Most are unambiguous, but some are global pointers that refer to a large body of law. For example, [§164.520(b)(3), v.2000] "Except when required by law, a material change to any term..." is a deference to any other relevant legal requirement. We deal with this and other kinds of ambiguous references by creating an environmental assertion that the condition is satisfied. Non-monotonic default logic (for example, [4]) could perhaps be used here instead.

The second distinguishing feature is the use of testimonials in resolving conditions. Many environmental conditions are resolved by a testimonial from a principal in the system. For example, [§164.506(a)(3)(c), v.2000] "If ... the covered health care provider determines, in the exercise of professional judgment, that the individual's consent to receive treatment is clearly inferred from the circumstances." Our model handles testimonials in the same way that it handles other environmental conditions, but an implementation must track the testimonials that allowed a command to execute and log who asserted them.

To illustrate the translation, we give an example paragraph and the regular and if commands that we derive.

Example: Section [§164.506(c)(1), v.2003] reads:

"(c) Implementation specifications: Treatment, payment, or health care operations. (1) A covered entity may use or disclose protected health information for its own treatment, payment, or health care operations."

We separate the purpose (only for its own usage) of the paragraph into an if command:

```
OIF   AllowedAsIn506c1 (a, s, r, p, f, ev)
 if   "own use" in p
and   isTPO(p)
then  return true
else  return false
end
```

The quote contains six permissions: use and disclose for treatment, payment, and health care operations. We create two commands, **Use506c1** and **Disclose506c1**, parameterized by purpose.

COM	Use506c1 (a, s, r, p, f, ev)
if	AllowedAsIn506c1 (a, s, r, p, f, ev)
and	r == a
and	own **in** (a, f)
and	isTPO(p)
then	EnterUse(a, p, f)
end	

Use506c1 checks the paragraph's if command, that the recipient of the information is the actor, and that the actor owns the file. Own means that the file is locally visible to the actor. This condition is implicit in the legal text, that a covered entity can only look at files that it has locally.

CopyObject, **isTPO**, and **EnterDisclose** are helper commands. **CopyObject** is given above. **isTPO** returns true if the purpose set provided includes treatment, payment, or health care operations. **EnterUse** inspects the purpose set for treatment, payment, and health care operations and inserts the appropriate permission in the matrix. **EnterDisclose** is similar. **Disclose506c1** and the helper commands are provided in Appendix C. □

5 Analysis

After translating the section of legal text into commands, we convert the commands into a format suitable for input to a standard model checker. With it we can define invariants and evaluate whether the command set respects them. We use the Spin model checker (`www.spinroot.com`) although other similar tools could have been used instead. Spin's input language is Promela, a C-like language.

The translation to Promela is straightforward from our command syntax. Each command is converted into a process which listens and responds along a named public channel. A globally readable record (evidence) contains a bit flag for each environmental flag. A similar record (purpose) contains a bit flag for each purpose value. Global variables hold the values of the command's actor (`a`), subject (`s`), intended recipient (`r`), purpose (`p`), and file (`f`). The access matrix `m` is a two dimensional array with principals in rows and all system objects (principals, files, and group objects) as columns. Each entry in the array is a record with a bit flag for each possible permission. Processes refer to each other and communicate by passing messages over their named channels. The model executes as a single thread. We do not use concurrency.

Example: To illustrate the Promela model, we give an example process, Use506c1 shown above. See Appendix A for a brief background for readers unfamiliar with Promela.

```
1active proctype Use506c1(){
2 bool result = false, temp;
3 do
4  ::Use506c1_chan?request(_) ->
5  AllowedAsIn506c1_chan!request(true);
6  AllowedAsIn506c1_chan?response(temp);
7  result = temp && (r==a);
8  result=result&&(m.mat[a].obj[f].own==1);
9  if
10   ::  result ->
11    EnterUse_chan!request(true);
12    EnterUse_chan?response(temp);
13   ::  else -> skip;
14  fi;
15  Use506c1_chan!response(result);
16 od}
```

The process listens for `request` messages with a single boolean. When a message appears, line 4 executes. Line 5 transmits a request to `AllowedAsIn506c1` to check the conditions. When `AllowedAsIn506c1` responds, it assigns the response value to `temp`. Lines 7–8 checks that the recipient is the actor and that the actor owns the file. Lines 9–14 check the result. If the result is `true`, `EnterUse` is requested to update the permissions. `Use506c1` finishes and sends its result on lines 15–16. □

We next show how we can perform queries on our model using Spin.

We are interested in queries that legislators, lawyers, and interested parties would normally perform by hand. We want to ask queries such as "Does this loophole exist in the law?" and "Can this command execute before that?" When the result of a query violates the invariant of a stakeholder, we call it a relative stakeholder problem. The context of a query is important as is the set of procedures that are active for the test. Such queries may not need a large number of steps to be answered, but they are interesting because the one asking may not be able to look at the text directly due to its length or complexity.

In order to find interesting queries to perform, we examined the different HIPAA consent policies and the summary of public comments that motivated the policy changes between 2000 and 2003. The comments document [24] contains a short list of concerns that interested parties found in the 2000 document. Some items on the list are relative stakeholder problems, so other stakeholders might not consider them to be concerns. Others are concerns in that they forbid what is common industry practice.

The 2003 policy was written with the comments in mind, but it is not clear if the new policy solved all of the concerns. To answer that question, we take the expert discovered concerns, "discover" them in the 2000 version, and find out whether they are still present in the 2003 version. We selected the following three concerns for this test:

(1) Emergency medical providers were concerned that the requirement that they attempt to obtain consent as soon as reasonably practicable after an emergency would have required significant efforts which might have been viewed by patients as harassing, because these providers typically do not have ongoing relationships with patients.

(2) Providers that are required by law to treat were concerned about the mixed messages to patients and interference with the physician-patient relationship that would have resulted because they would have had to ask for consent to use or disclose protected health information for treatment, payment, or health care operations, but could have used or disclosed the information for such purposes even if the patient said no.

(3) The transition provisions would have resulted in significant operational problems and the inability to access health records would have had an adverse effect on quality activities because many providers were not required to obtain consent for treatment, payment, or health care operations.

We translate the above concerns to the following queries. They are designed to discover the concern in the 2000 version and check whether they return the same result in the 2003 version. (1) Are emergency providers required to obtain patient consent patients for treatment, payment, or health care operations (TPO) after the fact? (2) Can a doctor access a patient record for TPO after the patient has denied consent? (3) Can a doctor see a patient record for TPO without consent in a non-emergency situation?

We evaluate the queries by first setting the initial state of the 2000 model to have appropriate invariants, principals, files, and permissions to discover the concern and then using the same initial state in the 2003 model to see if it yields the same result. As an example, we present the setup and invariant for the second query

Example: We initialize the matrix: `Dan` is a doctor, `file1` is a private record, `Paula` is the subject of `file1`, `Dan` has permission `own` on `file1`, and `Dan` is required by law to treat `Paula`. `Dan`, `file1`, `Paula`, and `health_care_provider_group` are all names for unique integer values that index the matrix.

```
m.mat[Dan].obj[health_care_provider_group].m-
ember = 1;
m.mat[Paula].obj[file1].subject = 1;
m.mat[Dan].obj[file1].own = 1;
m.mat[Dan].obj[Paula].required = 1;
```

`Paula` then denies `Dan` consent for TPO on `file1`.

```
a = Paula; s = Paula; f = file1; r = Dan;
denyConsent506b4i_chan!request(true);
denyConsent506b4i_chan?response(_);
```

We then set the purpose flags in the purpose set `p`. There also are a large set of evidence flags that are set which we skip here for brevity.

```
p.treatment=1; p.payment=1;
p.healthcare_operations=1;
```

Then we check an invariant to see if `Dan` can get access to `file1` for TPO. We also check `f_new` which would be the index of a new copy of `file1` that Dan could perhaps gain access to through a disclosure.

```
invariant = (m.mat[Dan].obj[file1].treat==0)
&&(m.mat[Dan].obj[file1].pay==0)&&
(m.mat[Dan].obj[file1].healthops==0)&&
(m.mat[Dan].obj[f_new].treat==0)&&
(m.mat[Dan].obj[f_new].pay ==0) &&
(m.mat[Dan].obj[f_new].healthops==0);
assert(invariant==1);
```

We then allow Spin to non-deterministically execute commands, non-deterministically changing the actor and recipient variables each time. After each command completes, we evaluate and check the invariant. □

Spin found the first query to be true in the 2000 version as expected. Since there is no notion of after the fact consent in the 2003 version, the model checker couldn't evaluate the query, a trivial false.

Spin found the second query to be true for the 2000 model, but surprisingly returned true for the 2003 model as well. Upon inspection we found that there was a provision in the (current) 2003 rules stating that even though consent is not required for treatment, payment, or health care operations, health care entities optionally may request consent anyway [164.506(b)(1), v.2003]. No paragraph in the section declares that an optional consent is binding, however. To find out what this omission meant, we consulted with the Lauren Steinfeld, Privacy Officer of the University of Pennsylvania. Ms. Steinfeld remarked that a situation of denial of an optional consent request for TPO is legally complex because it has a conflict of patient expectations of privacy and potential medical necessity. In practice, this case may be affected by state laws which preempt the federal guidelines. In short, it is not resolved in this section of HIPAA.

The third query diverged for the 2000 model, but a query which limited the exploration to a depth of two rule invocations returned a false as expected. The query for the 2003 version return true as expected.

We learned a number of lessons from our experiments.

First, following the structure and style of the law let us discover a somewhat subtle policy property in query 2. We had expected that the result for the 2003 version would be negative since the the paragraph about entities required by law to provide health care was removed. However, since the 2003 version is silent on the need to respect optional consent, our query found an ambiguity in the legal text that requires deeper legal analysis to resolve.

Second, we can discover properties of the system based on the presence or absence of permissions or environmental flags in the model. For some concerns, an assertion that no command includes a particular obligation is sufficient as in our first query.

Third, we have indication that our current model may suffer from an undecidability property noted by HRU. The original HRU paper [30] proved that the general question of safety, whether granting a general right to one principal can eventually lead to it being leaked to an unauthorized principal, for systems described in their syntax is undecidable. They note that certain systems may not be subject to their conclusion, but we have not yet explored that question for our system. We will consider ways of modifying our model or queries to further explore this problem.

6 Related Work

Access control systems research has been extensively researched for decades. Recent work on formalizations of access control systems and privacy rules include Anderson's [3] principals for the design of a national patient information database, Bertino, *et al.*'s formal model for policy features and comparison [15], Fischer-Hübner and Ott's formalization of the Generalized Framework for Access Control [22], Yu, *et al.*'s formal semantics [45] for P3P [42], Jajodia *et al.*'s work on provisional authorizations [32] which underly the XML access control language XACL [29], and Wijesekera and Jajodia's propositional policy model and algebra [44, 43].

The formalization and analysis in this work uses a classic access control model to achieve its goals. We chose HRU syntax to begin from the ground up, adding only exactly what is necessary. We could perhaps have used any of a long list of more modern systems such the Enterprise Privacy Authorization Language (EPAL) [5], the Platform for Privacy Preferences (P3P) [42], XACL [29], UCON [37], Cassandra [13], or Fischer-Hübner and Ott's formalization [22], however we did not find the necessity. We give detailed reasoning for two models that are close to our own: EPAL and Cassandra.

EPAL [5, 7] allows the writing of rule based policies that include obligations, similar to our model. Policies consist of two documents: (1) a vocabulary document which contains principals, purposes, actions, "data" (i.e. information that the policy discusses), and obligations and (2) a policy document that contains *rules* that give approved (or disapproved) combinations of vocabulary terms and conditions that can inspect flags and parameters provided in a special invocation "container" and return boolean values. Policies are queried for a ruling and, like our model, can return true or false with an optional inclusion of obligations. There has been extensive work on the formal semantics of EPAL [6], policy composition [9, 11] and comparison [10], and translation to other languages [33, 8]. As a study of the power of EPAL, Powers, *et al.*translate a section of Ontario's Freedom of Information an Protection of Privacy Act [36] to EPAL [40]. As with our model, their resulting policy closely mirrors the legal text, having one rule per textual paragraph. They show how the policy allows them to process a request and return a ruling in a similar manner to how a human would, but do not analyze or verify their policy formulation.

Certain properties of EPAL's rules, conditions, system model, and vocabularies, however, made it difficult for us to use. Rules do not have an explicit representation of access to system and object state, so rules that inspect the state of objects and rights or modify state (*e.g.* data anonymization, disclosure of minimum necessary information, etc.) must rely on complex conditions and obligations. They also do not include a parameter for "recipient," so rules that depend on the information recipient can not be written easily. They also can not query or invoke other rules, so they can not collaborate or query each other for rulings or obligations. Conditions can not access rule invocation parameters such as purpose, actor, and data. They are confined to flags and parameters included in a "container." There also is no provision for writing to or inspecting a system log. Vocabularies include functionality for hierarchies of principals, data, and purposes (but not actions) and rule priorities, functionality that we did not need.

We could perhaps have used EPAL by devising larger sets of conditions, obligations, containers, and rules, but we find our representation more scalable for legal texts. In particular, since EPAL has no structure for rules referring each other, the size of an EPAL

rule set would grow exponentially with the number of references in the legal text.

In the early 2000's, the British government produced a set of documents describing how its Integrated Care Records Service should work, including an output based specification (OBS) [34]. Becker and Sewell use their Cassandra trust management language to model the OBS' rules for the national data Spine [14]. Cassandra is a Datalog and credential based access control language. It provides rules for performing actions, activating roles, deactivating roles, requesting credentials, and submitting credentials over a distributed computing environment.

Cassandra differs from this work in several ways. First, since Cassandra is concerned with trust management, it does not address the flow of objects in the system, an important concern in our model. Conversely, since we are concerned with permissions and object distribution, we provide only rudimentary support for roles in the form of groups. This difference of focus reflects a difference in legal policy design. While the UK's law uses roles and credentials extensively, the US' health and financial privacy laws do not. Additionally, we desire the smallest set of features that will accomplish our modelling goal. Cassandra, for instance, includes a credential distribution system, significantly more than is needed here. Finally, they are concerned with implementing policy specifications as given and thereby showing the sufficiency and flexibility of their language, not investigating policy permissiveness or performing policy comparison using queries and invariants.

Earp *et al.* [21] provide a framework for comparing web site privacy policies from different perspectives. Their framework could be used to compare the provisions in different versions of privacy regulations as well.

Breaux and Antón [17] analyze one of the HIPAA fact sheet summaries [35] to find patterns and semantics of rights, obligations, rules, and constraints in natural language privacy regulations. They use a formal semantic language KTL to model and query simple references. Our work differs in its focus on the implementation and modelling of an intricate legal source, rather than the extraction of semantic patterns. Our technique let us to discover omissions in the law, but it is unclear if KTL can do so too.

Barth, *et al.* [12] present a theoretical model of privacy policies with respect to contextual integrity. Their theoretical model is limited to the passing of messages between agents, ignoring action and purpose, but they present theorems for checking future satisfiability of policy obligations. They give several short examples from legal privacy policies, including GLB, HIPAA, and the Childrens Online Privacy Protection Act (`www.coppa.org`).

There has been extensive work on the formalization of obligations in contexts including from businesses processes [2] and contracts [1], access control policies [31, 37, 16], and enterprise privacy policies [19, 18]. Obligations in our model are handled rudimentarily with flags and checks by policy rules. A full treatment of the classification, tracking, enforcement, and management of the different types of obligations that arise in privacy laws is beyond the scope of this paper.

7 Conclusions

Our work contributes in four important ways to the formal analysis of legal privacy policies.

First, we define *auditable privacy systems*, a formalism for understanding the basic operations of legal privacy policies, and shown the minimum amount of change required to implement them using classical access control models.

Second, we have created a language suitable for modelling legal privacy policies and developed a methodology of translating natural language text to it. Our method closely follows the structure of the text, preserving references to other rules and environmental conditions. The payoff for this method comes from preserving the subtleties of the law during modelling and analysis. In translation, we reveal essential differences between the underlying policy frameworks of HIPAA (and, from our readings, legal privacy policies in general), and classical access control matrix based systems.

Third, our model is designed to allow for users to do policy evaluation by "executing the law." This approach differs from the standard methods of inspection and expert consultation by allowing non-experts to explore the rights and permissions of legal policies.

Finally, by observing a bridge between legal privacy policies and formal analysis, we open the door for verification of more legal privacy rules. Our queries uncovered an unexpected ambiguity of the law, a feature that some commenters on the 2000 law considered a problem and a HIPAA expert saw as legally tricky. By expanding the range and scope of our work in the future, we hope to open more legal policies to scrutiny.

Acknowledgements We are grateful for discussions about Privacy APIs that we had with Mike Hicks, Harvey Rubin, Lauren Steinfeld, Alan Abrahams, and Nikhil Dinesh.

References

[1] Alan S. Abrahams. *Developing and Executing Electronic Commerce Applications with Occurrences*. PhD thesis, University of Cambridge, 2002.

[2] Alan S. Abrahams, David M. Eyers, and Jean M. Bacon. An asynchronous rule-based approach for business process automation using obligations. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Rule-Based Programming (RULE'02)*, pages 93–103, Pittsburgh, Pennsylvania, October 2002.

[3] Ross J. Anderson. A security policy model for clinical information systems. In *Proceedings of IEEE Security and Privacy*, pages 30–43, May 1996.

[4] Grigoris Antoniou. A tutorial on default logics. *ACM Comput. Surv.*, 31(4):337–359, 1999.

[5] Paul Ashley, Santoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Langugae (EPAL 1.2). W3C Member Submission, `www.w3c.org/Submission/EPAL`, November 2003.

[6] Paul Ashley, Satoshi Hada, Günter Karjoth, and Matthias Schunter. E-p3p privacy policies and privacy authorization. In *Proceeding of the ACM workshop on Privacy in the Electronic Society*, pages 103–109. ACM Press, 2002.

[7] Paul Ashley, Calvin Powers, and Matthias Schunter. From privacy promises to privacy management: a new approach for enforcing privacy throughout an enterprise. In *Proceedings of the 2002 Workshop on New Security Paradigms*, pages 43–50. ACM Press, 2002.

[8] Michael Backes, Markus Dürmuth, and Günter Karjoth. Unification in privacy policy evaluation - translating epal into prolog. In *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, 2004.

[9] Michael Backes, Markus Dürmuth, and Rainer Steinwandt. An algebra for composing enterprise privacy policies. In *European Symposium on Research in Computer Security (ESORICS)*, 2004.

[10] Michael Backes, Günter Karjoth, Walid Bagga, and Matthias Schunter. Efficient comparison of enterprise privacy policies. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 375–382, New York, NY, USA, 2004. ACM Press.

[11] Michael Backes, Birgit Pfitzmann, and Matthias Schunter. A toolkit for managing enterprise privacy policies. In *European Symposium on Research in Computer Security (ESORICS)*, 2003.

[12] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*. IEEE, May 2006.

[13] Moritz Y. Becker and Peter Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks*, June 2004.

[14] Moritz Y. Becker and Peter Sewell. Cassandra: flexible trust management, applied to electronic health records. In *17th IEEE Computer Security Foundations Workshop*, June 2004.

[15] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 41–52, New York, NY, USA, 2001. ACM Press.

[16] Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy management and security applications. In *28th Conference on Very Large Data Bases (VLDB'02)*, Aug 2002.

[17] Travis Breaux and Annie I. Antón. Mining rule semantics to understand legislative compliance. In *CCS Workshop on Privacy in the Electronic Society*. ACM, Nov 2005.

[18] Marco Casassa-Mont. Dealing with privacy obligations in enterprises. Tech Report HPL-2004-109, HP Labs, 2004.

[19] Marco Casassa-Mont. A system to handle privacy obligations in enterprises. Tech Report HPL-2005-180, HP Labs, 2005.

[20] Senate Banking Committee. Gramm-Leach-Bliley act - disclosure of nonpublic personal information. *15 USC, Subchapter I, Sec. 6801-6809*, 1999.

[21] Julia B. Earp, Annie I. Antón, and Olli Jarvinen. A social, technical and legal framework for privacy management and policies. In *Americas Conference on Information Systems (AMCIS)*, pages 605–612, August 2002.

[22] Simone Fischer-Hübner and Amon Ott. From a formal privacy model to its implementation. In *Proceedings of the 21st National Information Systems Security Conference*, Oct 1998.

[23] Office for Civil Rights. Standards for privacy of individually identifiable health information; final rule. *Federal Register*, 65(250):82462–82829, December 28 2000. Unamended Version.

[24] Office for Civil Rights. Standards for privacy of individually identifiable health information; final rule. *Federal Register*, 67(157):53182–53273, August 14 2002. Final modifications to privacy rule.

[25] Office for Civil Rights. Standards for privacy of individually identifiable health information. Regulation Text (Unofficial Version) 45 CFR Parts 160 and 164, U.S. Department of Health and Human Services, August 2003. As amended: May 31, 2002, Aug 14, 2002, Feb 20, 2003, and Apr 17, 2003.

[26] Richard Graubart. On the need for a third form of access control. In *12th National Computer Security Conference Proceedings*, pages 296–303, October 1989.

[27] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May 2000.

[28] Carl A. Gunter, Michael J. May, and Stuart Stubblebine. A formal privacy system and its application to location based services. In *Privacy Enhancing Technologies (PET)*, 2004.

[29] Satoshi Hada and Michiharu Kudo. XML access control language (XACL): Provisional authorization for XML documents. Standard, OASIS, 2001. `xml.coverpages.org/xacl.html`.

[30] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[31] Manuel Hilty, David Basin, and Alexander Pretschner. On obligations. In *European Symposium on Research in Computer Security (ESORICS)*, 2005.

[32] Sushil Jajodia, Michiharu Kudo, and V.S. Subrahmanian. Provisional authorizations. In *Recent Advances in Secure and Private E-Commerce*, pages 133–159. Kluwer Academic Publishers, 2001.

[33] Günter Karjoth, Matthias Schunter, and Els Van Herreweghen. Translating privacy practices into privacy promises -how to promise what you can keep. In *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pages 135–146, 2003.

[34] UK Department of Health. *Output based specification for Integrated Care Record Service*. UK Crown Copyright, August 2003. Version 2.

[35] U.S. Department of Health and Human Services. *Fact Sheet: Protecting the privacy of patients' health information*, 14 April 2003. `www.hhs.gov/news/facts/privacy.html`.

[36] Government of Ontario. Freedom of information and protection of privacy act. *Revised Statues of Ontario 1990, Chapter F.31*, 2005. `www.e-laws.gov.on.ca/DBLaws/Statutes/English/90f31_e.htm`.

[37] Jaehong Park and Ravi Sandhu. The $UCON_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[38] European Parliament. Directive 95/46/ec on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities L 281*, 24 October 1995.

[39] European Parliament. Directive 2002/58/ec concerning the processing of personal data and the protection of privacy in the electronic communications sector (directive on privacy and electronic communications). *Official Journal of the European Communities L 201*, 12 July 2002.

[40] Calvin Powers, Steve Adler, and Bruce Wishart. EPAL translation of the freedom of information and protection of privacy act. `www.ipc.on.ca/docs/EPAL%20FI1.pdf`, March 2004. White Paper.

[41] Health Resources and Services Administration. Health insurance portability and accountability act of 1996. *Public Law 104-191*, 1996. H.R. 3103.

[42] W3C. The Platform for Privacy Preferences 1.0 (P3P1.0). Technical report, World Wide Web Consortium, 2001. `www.w3c.org/P3P`.

[43] Duminda Wijesekera and Sushil Jajodia. Policy algebras for access control: the propositional case. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 38–47, New York, NY, USA, 2001. ACM Press.

[44] Duminda Wijesekera and Sushil Jajodia. A propositional policy algebra for access control. *ACM Trans. Inf. Syst. Secur.*, 6(2):286–325, 2003.

[45] Ting Yu, Ninghui Li, and Annie I. Antón. A formal semantics for P3P. In *SWS '04: Proceedings of the 2004 workshop on Secure web service*, pages 1–8, New York, NY, USA, 2004. ACM Press.

A Promela Basics

The following is a short primer to let the reader understand the Promela code included in this work. For a full tutorial see `spinroot.com`.

Promela has two selection structures - do and if. They are closed by their names reversed (do/od if/fi). Both structures select nondeterministically from a set of options that begin with the :: symbol. The first statement in an option is its guard, indicated by the -> symbol which normally separates it from the other statements in the option. If the guard is true or executable, the option may be selected by the structure. If it is not, the option is excluded. The `skip` statement is a no-op and is always executable.

Processes talk over named channels using listen (?) and transmit (!). We use only synchronous channels in our model, so a transmitting process blocks until there is a process willing to listen and vice versa. Transmit and listen commands are always executable. Channels are typed by the kinds of messages that can be passed over them. All the messages in our model have the same pattern - `mtype(bool)` - where `mtype` is a message type (here `request` and `response`) and the body is a boolean. A process can listen for a message that fits particular parameters and when it accepts a message it assigns the variables in the listen statement the appropriate values. For example, `chan1?request(t)` listens on `chan1` for messages of type `request` that contain only one parameter (here `t` is a boolean) and assigns

`t` the value of the boolean in the message. The special write only variable _ accepts any type and is used for throwaway values.

Promela does not have built in multi-dimensional arrays, so we build a two dimensional array by creating an array of records which themselves contain an array. Each location in the array is a record with bit flags for each potential permission. The result is a syntax like this: `m.mat[a].obj[f].own == 1` which refers to the array location [a,f] in the array and assigns 1 to the flag `own` there.

Like C's ?: operator, Promela includes a three argument operator a->b:c where a is a boolean formula. If a is true, b is the result of the operator. If a is false, c is the result of the operator. For example, `x = (true -> 1 : 2)` assigns `x` the value 1.

B Semantics of HRU model

Let r be a generic right from a finite set of generic rights R. Let $X_0, \ldots, X_k$ be formal parameters to a command. Let $p, p_1, \ldots, p_n$ and $o, o_1, \ldots, o_m$ be integers between 1 and k. Let t be a text string.

The operations in the system are: **enter** r **into** (X_p, X_o), **delete** r **from** (X_p, X_o), **create principal** X_p, **create object** X_o, **destroy principal** X_p, and **destroy object** X_o.

The configuration of the system is a tuple (P, O, M) where P is the set of principals in the system, O is the set of objects in the system ($P \subseteq O$), and M is the access matrix.

A formal description of the effect of the primitive operations on the system configuration is as follows. Let (P, O, M) and (P', O', M') be configurations of the system. A step $(P, O, M) \xrightarrow{op} (P', O', M')$ can occur if either:

1. $op =$ **enter** r **into** (p, o) and $P' = P$, $O' = O$, $p \in P$, $o \in O$, $M'[p_1, o_1] = M[p_1, o_1]$ if $(p_1, o_1) \neq (p, o)$ and $M'[p, o] = M[p, o] \cup \{r\}$, or
2. $op =$ **delete** r **from** (p, o) and $P' = P$, $O' = O$, $p \in P$, $o \in O$, $M'[p_1, o_1] = M[p_1, o_1]$ if $(p_1, o_1) \neq (p, o)$ and $M'[p, o] = M[p, o] - \{r\}$, or
3. $op =$ **create principal** p' where p' is a new symbol not in O, $P' = P \cup \{p'\}$, $O' = O \cup \{p'\}$, $M'[p, o] = M[p, o]$ for all $(p, o) \in P \times O$, $M'[p', o] = \emptyset$ for all $o \in O'$, and $M'[p, p'] = \emptyset$ for all $p \in P'$, or
4. $op =$ **create object** o' where o' is a new symbol not in O, $P' = P$, $O' = O \cup \{o'\}$, $M'[p, o] = M[p, o]$ for all $(p, o) \in P \times O$, and $M'[p, o'] = \emptyset$ for all $p \in P$, or
5. $op =$ **destroy principal** p' where $p' \in P$, $P' = P - \{p'\}$, $O' = O - \{p'\}$, and $M'[p, o] = M[p, o]$ for all $(p, o) \in P' \times O'$, or
6. $op =$ **destroy object** o' where $o' \in O$, $P' = P$, $O' = O - \{o'\}$, and $M'[p, o] = M[p, o]$ for all $(p, o) \in P' \times O'$

C Extra commands

Disclose command for [§164.506(c)(1), v.2003]

COM Disclose506c1 (a, s, r, p, f, ev)
if AllowedAsIn506c1 (a, s, r, p, f, ev)
and own **in** (a, f)
then CopyObject(a, s, f, f′)
and **enter** own **in** (r, f′)
and EnterDisclose(a, p, f)
end

Helper commands:

COM EnterUse(a, p, f)
if "treatment" **in** p
then **enter** treat **in** (a, f)
if "payment" **in** purpose
then **enter** pay **in** (a, f)
if "healthcare operations" **in** p
then **enter** healthops **in** (a, f)
and **return**
end

COM EnterDisclose(a, p, f)
if "treatment" **in** p
then **enter** treatDisclose **in** (a, f)
if "payment" **in** p
then **enter** payDisclose **in** (a, f)
if "healthcare operations" **in** p
then **enter** healthopsDisclose **in** (a, f)
and **return**
end

IF isTPO(p)
if "treatment" **in** p
or "payment" **in** p
or "healthcare operations" **in** p
then **return true**
end