Declarative Privacy Policy: Finite Models and Attribute-Based Encryption

Peifung E. Lam
Stanford University
353 Serra Mall
Stanford, CA 94305
pflam@cs.stanford.edu

John C. Mitchell
Stanford University
353 Serra Mall
Stanford, CA 94305
mitchell@cs.stanford.edu

Andre Scedrov
University of Pennsylvania
209 South 33rd Street
Philadelphia, PA 19104
scedrov@math.upenn.edu

Sharada Sundaram
Symantec Corporation
350 Ellis Street
Mountain View, CA 94043
sharada_sundaram@symantec.com

Frank Wang
Stanford University
353 Serra Mall
Stanford, CA 94305
frankw@cs.stanford.edu

ABSTRACT

Regulations and policies regarding Electronic Health Information (EHI) are increasingly complex. Federal and State policy makers have called for both education to increase stakeholder understanding of complex policies and improved systems that impose policy restrictions on access and transmission of EHI. Building on prior work formalizing privacy laws as logic programs, we prove that for any privacy policy that conforms to patterns evident in HIPAA, there exists a finite representative hospital database that illustrates how the law applies in all possible hospitals. This representative illustrative example can support new education, new policy development, and new policy debugging tools. Addressing the need for secure transmission of usable EHI, we show how policy formalized as a logic program can also be used to automatically generate a form of access control policy used in Attribute-Based Encryption (ABE). This approach, testable using our representative hospital model, makes it possible to share policy-encrypted data on untrusted cloud servers, or send strategically encrypted data across potentially insecure networks. As part of our study, we built a prototype to secure Health Information Exchange (HIE), with automatically generated ABE policies, and measure its performance.

Categories and Subject Descriptors: K.4.1 [**Privacy**]: Policy Compliance Automation; J.3 [**Medical Information Systems**]: Trustworthy and Secure Infrastructure for Health Information Systems

General Terms: Security, Legal Aspects, Algorithms

Keywords: Privacy Policy, Compliance, EHR, ABE

1. INTRODUCTION

Emerging Electronic Health Record (EHR) systems hold great promise for empowering patients and ensuring more effective delivery of health care. Among governments, healthcare providers, and insurance companies, there is also growing awareness of the advantages of Health Information Exchange (HIE) in promoting more streamlined and effective patient treatment, payment, research data use, law enforcement, bioterrorism response and action, and government oversight. The complexity of regulations and organizational policies related to health records makes it difficult for enterprises to design and deploy effective compliance systems. We believe that technology can play an important role in ensuring that privacy policies associated with EHRs and other patient health information systems are expressed precisely and unambiguously, as well as in ensuring that processes and people in such organizations act in a manner that is compliant with the stated policies.

Our earlier work on formalization of privacy law and its enforcement in organizational processes in hospitals provides a useful starting point for this effort [1–3]. We used a stratified fragment of Prolog with limited use of negation to formalize a portion of the US Health Insurance portability and accountability Act (HIPAA). The language we chose was tractable and it enabled us to build a prototype hospital Web portal messaging system where the privacy rules were checked automatically before a message was sent.

In this paper we show that for any formalized healthcare policy of a certain form, there is a finite model of a representative hospital. This hospital, characterized by a set of individual entities such as doctors and patients, and a set of facts about relations on the entities in the domain, is representative in the sense that any situation governed by the healthcare policy arises in connection with some of the individuals represented in it. We prove that such a finite representative model exists for any set of Prolog rules representing privacy regulations and policies, under certain reasonable assumptions about the structure of the privacy policy. We also present an algorithm to create this finite representative model. The finite representative model can be used to derive training materials that allow healthcare professionals to understand how the law applies in different scenarios. It can also be used to produce cases to test and debug a system that relies on the policy, such as to ensure compliance with privacy law.

In a second related effort, we experiment with the transformation of policy in logic form into a more limited form of policy associated with cryptographic access control. Specifically, given a logic program representing the privacy law, we create a set of individual entities with the roles or other attributes referenced in the logic program, and query the logical database to determine the set of individuals that are permitted to access the EHR. The attributes of such an individual are then used to generate the policy for Attribute Based Encryption (ABE), so that the EHRs encrypted by ABE can only be decrypted by individuals with the right attributes as determined by the logic program. As part of our study, we built a prototype system for policy transformation and measure its performance.

1.1 Background

The Health Insurance Portability and Accountability Act (HIPAA) requires the establishment of national standards for electronic health care transactions and national identifiers for providers, health insurance plans, and employers. The Strategic Healthcare IT Advanced Research Projects on Security (SHARPS) [4] is a multi-institutional and multidisciplinary research project, supported by the Office of the National Coordinator for Health Information Technology, aimed at reducing security and privacy barriers to the effective use of health information technology. We summarize below a SHARPS [5] document that reported obstacles to interoperable health information exchange as compiled by 34 states within the U.S. that participated in the study. The same document proposed privacy and security solutions for these obstacles. The "solutions" are often high-level approaches with sub-goals that can in turn be accomplished with information technology or administrative policy changes. We highlight below a few areas where we think further research can contribute to more effective solutions. Although we have not carried out research in all these areas, they provide a backdrop for our current work.

Education: All 34 states recognize the need for varying levels of education to reduce variation in how policies are applied and also to increase stakeholder awareness and trust in the systems. The most common recommendations were for educational campaigns directed at patients and consumers and training programs for providers and organizations. We propose a mechanism to automatically generate interesting case studies that capture the essential behaviors of the law in allowing or denying communication of an EHR.

Secure Transmission of Health Information: Several state teams identified the secure transmission of personal health information between health care organizations, and between such organizations and consumers, as a significant issue. Reports cited the lack of interoperable solutions and the high cost of implementing appropriate forms of secure transmission that protect the data in transit and protect against inappropriate interception and modification. We propose the use of Attribute-Based Encryption (ABE) to protect EHR in transmission / external storage, so that only the authorized parties can decrypt the information.

Common Authentication and Authorization: State teams noted that the lack of a common method for authenticating individuals created mistrust between organizations. Current practices are often based on authentication by phone call or a fax from a known staff member, with authorization sometimes requiring a patient-signed a consent form (although not necessarily required by law) before the personal health information is exchanged. In moving toward efficient EHR exchange, we believe Web-based authentication mechanisms, for example, can be productively combined with policy-based authorization using techniques explored in this paper.

1.2 Related Work

A number of articles [6, 7] have explained the complex security and privacy requirements for healthcare IT. While several frameworks and concepts have been proposed [8–10], we believe that further detailed technical studies and implementations are needed.

In the area of policy specification and law formalization, there are numerous privacy languages and formalizations described in the literature, including the Enterprise Privacy Authorization Language (EPAL) [11,12], the eXtensible Access Control Markup Language (XACML) [13], and the Platform for Privacy Preferences (P3P) [14]. These languages and logics are used for some specific purposes like access control or website authorization. However, as explained in [1], for example, they have drawbacks for expressing detailed policy based on HIPAA, similar laws, or subtle privacy polices formulated by medical organizations.

There are many other formalization mechanisms proposed [2, 15–17]. While several are based on temporal logic, others allow study of the policy and law for privacy implications. Our formulation not only helps in verification and study but also acts as a tool to effectively extract the policy components needed to be compliant and secure. [18] and [19] uses a formalized model based design to assist in privacy compliance for the Vanderbilt Medical Center. Our finite model would be a good fit for their privacy component. [20] and [21] are some related and complementary works.

ABE Attribute based encryption (ABE) facilitates encryption based on a policy, which specifies who can decrypt the data in the future. This is an active field of research and many new variants have been proposed [22, 23]. The important ones being Key-Policy based Encryption [24] and Ciphertext-Policy ABE [25]. There have also been some efforts to incorporate ABE in different systems [26]. Building on our previous work [27], we provide an efficient method to build a HIPAA-compliant policy specifically for medical data. We implemented the proof of concept using the functional encryption package [28].

2. FINITE MODEL FOR PRIVACY LAWS

Privacy policies are often expressed in general terms, such as "a patient may receive a portion of the patient's psychiatric record if authorized by the patient's psychiatrist" that apply to any individuals in specific roles and relationships. The roles and relationships may change and the number of individuals governed by a policy statement may grow over time. Thus even a simple finite policy can be applied in a wide range of situations. This leads us to ask whether, for a given policy, if there is a fixed finite set of individuals, with specific roles and relationships, that exhibit all the cases of interest in all applicable hospitals (or other such organizations). Building on earlier work [27], we show that there always exists such a finite illustrative hospital, under

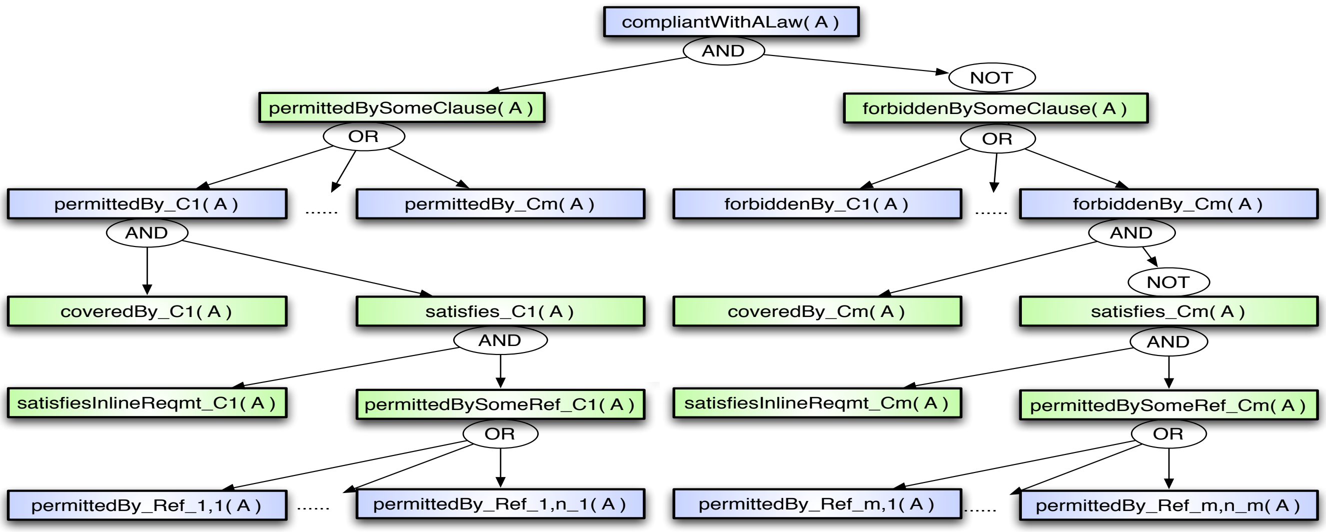


Figure 1: Initial Compliance Tree

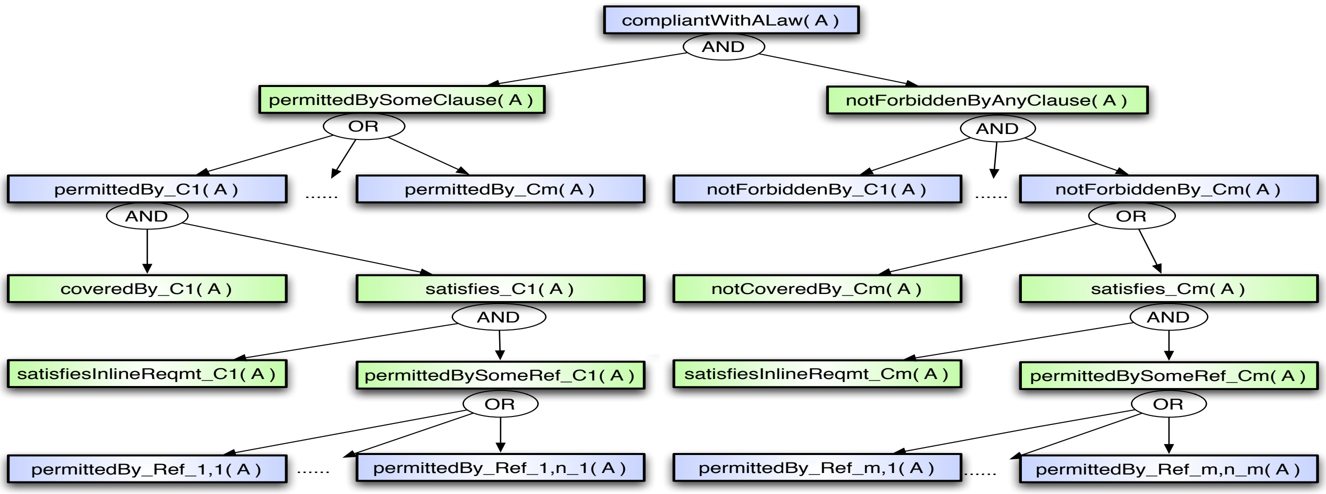


Figure 2: Normalized Compliance Tree

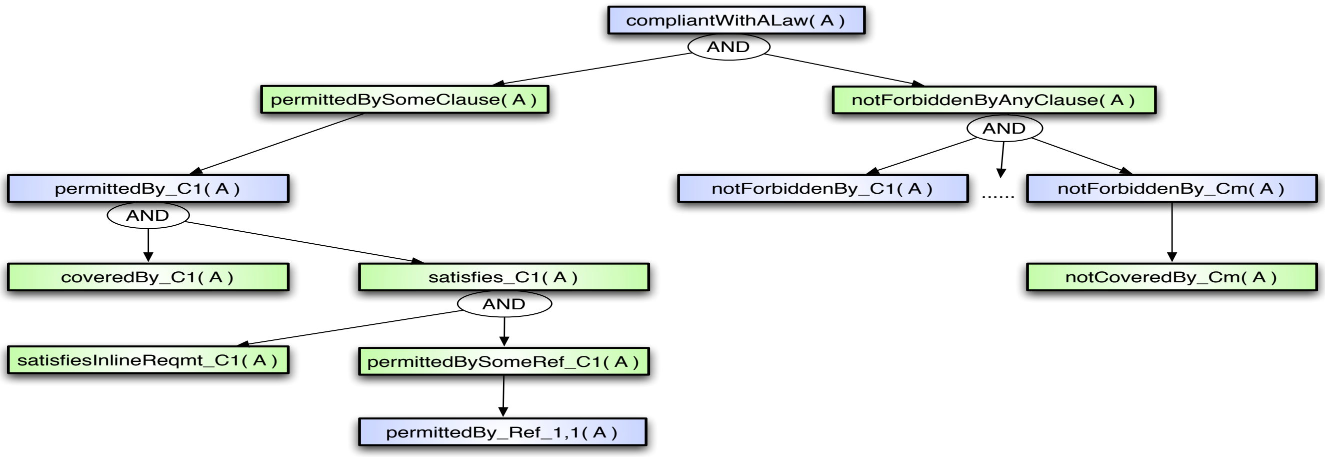


Figure 3: An Instance of Search Trees

reasonable assumptions about the structure of the law, and there is an algorithm for constructing it.

2.1 Background

At a high level, our model consists of a database system that encodes the state of an organization such as a hospital and answers queries about whether a particular action of information release is permitted or forbidden by a law such as HIPAA. The system supports two types of predicates: A *rule predicate* captures the essence of the law, and clarifies how an intended action stands with respect to the law. When appropriate, there may be individual rule predicates that correspond to individual clauses or subclauses of the law, as well as top-level predicates that provide a summary answer whether a given action is permitted or forbidden by the law. A *fact predicate* captures information specific to the organization and provides answers to questions such as the roles assigned to the individuals in the organization, and relations among various entities in the organization. Our intention is to separate the database into two parts, so that the rule predicates can be reused across multiple organizations subject to the same law, and administrators only need to add facts specific to their organizations.

We review some standard concepts in logic programming and databases and refer interested readers to [29] and [30] for further information on this subject. In the definitions below, each predicate p can take on a vector of arguments. For brevity we will denote a predicate with an argument vector by $p(\mathbf{x})$, instead of showing individual arguments. We recall that a literal L is an atomic formula or its negation, and we define $|L|$ as L if it is a positive literal, and $p(\mathbf{x})$ if L is a negative literal of the form $\neg p(\mathbf{x})$, where p is a predicate symbol. For ease of notation, we also define $|p| = |\neg p| = p$ where p is a predicate symbol.

Fact A *fact* states a relation that exists between individual entities and is of the form $p(\mathbf{x})$. For example, *doctorOf(d,p)* states that d is a doctor of p. Unless stated otherwise, we assume all facts are ground in this paper.

Rule A *rule* states that a relation holds provided some other relations hold. It has the form $r(\mathbf{X_0}) \leftarrow q_1(\mathbf{X_1}), \ldots, q_n(\mathbf{X_n})$, which states that $r(\mathbf{X_0})$ is true if each of the $q_i(\mathbf{X_i})$ is true, where each $|q_i|$ is a rule or fact predicate symbol, and each $\mathbf{X_i}$ is a vector of arguments.

Stratified Program Given a program P containing facts and rules, we denote by P^q the subset of clauses in P with predicate symbol q in the head. P is defined as stratified if there is a partitioning of $P = P_1 \cup ... \cup P_n$ by n non-empty strata, such that:
· if $p(...) \leftarrow ..., q(...), ... \in P_i$, then $P^q \subset P_1 \cup ... \cup P_i$;
· if $p(...) \leftarrow ..., \neg q(...), ... \in P_i$, then $P^q \subset P_1 \cup ... \cup P_{i-1}$.
For convenience, we defined $P_i = \emptyset$ for $i < 1$ or $i > n$.
Define $Index_P(q)$ as the smallest integer such that $P^q \subset P_1 \cup ... \cup P_{Index_P(q)-1}$. I.e., $Index_P(q) - 1$ is the highest numbered stratum of P in which q appears in the head of a clause. If q never appears in any of the strata, define $Index_P(q) = 1$. Unless noted otherwise, we assume P_1 contains precisely the facts of P.

Inference Let $P = P_1 \cup ... \cup P_m$ be a stratified program. Let $Q_i = (P_1, ..., P_i)$ for $1 \leq i \leq m$. The set B_P of all ground, atomic formulas over the alphabet of P is called the Herbrand base of P. Let F_0 be a ground literal with predicate symbol q such that $|F_0| \in B_P$. An inference of F_0 from P with j application of rules within the strata Q_i, denoted by $Q_i \vdash^j F_0$, is defined as follows. Let $\mathcal{I}_0 = \emptyset$ and define $\mathcal{I}_i$ recursively in ascending order of i:
(i) $\mathcal{I}_{i-1} \cup (P_i \cap B_P) \subset \mathcal{I}_i$. Denote $Q_i \vdash^0 F$ for any $F \in \mathcal{I}_{i-1} \cup (P_i \cap B_P)$.
(ii) Negation as Failure: $F_0 \in \mathcal{I}_i$, if F_0 is a negative literal, $i = Index_P(q)$, and $\neg F_0 \notin \mathcal{I}_{i-1}$. Denote $Q_i \vdash^0 F_0$.
(iii) $F_0 \in \mathcal{I}_i$, if F_0 is a positive literal, and $\exists$ substitution θ, rule R $\in P_i$ of the form $L_0 \leftarrow L_1, ..., L_n$, literals F_j with $|F_j| \in B_P$, integers $0 \leq m_j < \infty$, such that $\theta(L_j) = F_j$ for $0 \leq j \leq n$ and $Q_i \vdash^{m_j} F_j$ for $1 \leq j \leq n$. Denote $Q_i \vdash^{m_0} F_0$ where $m_0 = max\{m_j | 1 \leq j \leq n\} + 1$.

Define $P \vdash F_0$ iff $\exists m_0$ such that $Q_{Index_P(q)} \vdash^{m_0} F_0$.

Dependency Graph Given a Prolog program P consisting of a set of rules and facts, the dependency graph $\langle V, E\rangle$ of P is defined as follows. The vertices V are the predicates occurring in P, and $(u, v) \in E$ exactly when there exists a rule in P where the predicate in the head is u and the predicate v is in the body.

Hospital A *Hospital* is a finite structure (*Domain*, *Rules*, *Facts*). Here *Domain* is a set of individual entities about the hospital, such as hospital employees, patients, business associates, and records maintained in the hospital. *Rules* is a set of rules representing the regulations to be enforced in the hospital. *Facts* is a set of ground facts about the relations on the entities in *Domain*, and is called the **extensional database (EDB)** of the HIPPA Hospital. We require that $Rules \cup Facts$ is stratified, and the body of any rule in *Rules* does not reference any constants defined in *Domain*.

Given a Hospital H, we define $Domain(H)$, $Rules(H)$, and $Facts(H)$ as the corresponding attributes of H; and define $Sym(Rules)$, $Sym(Facts)$ as the set of predicate symbols in *Rules* and *Facts*, respectively. Each predicate symbol p in $Sym(Rules) \cup Sym(Facts)$ is associated with a unique integer $arity(p)$, which represents the number of arguments of p.

We say $H \vdash F_0$ if $Rules(H) \cup Facts(H) \vdash F_0$.

Action For the purpose of determining compliance, we define an *action* $a = (u_s, u_r, u_o, m_t, m_p, c, b) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U} \times \mathcal{MT} \times \mathcal{P} \times 2^{\mathcal{U} \times \mathcal{CT}} \times 2^{\mathcal{U} \times \mathcal{BT}}$, where $\mathcal{U}$, $\mathcal{MT}$, $\mathcal{P}$, $\mathcal{CT}$, and $\mathcal{BT}$ are disjoint subsets of Domain(H). Here $\mathcal{U}$ is the set of persons, $\mathcal{MT}$ is the set of message types, $\mathcal{P}$ is the set of purposes, $\mathcal{CT}$ is the set of consent types, and $\mathcal{BT}$ is the set of belief types. We define $\mathcal{A}(H)$ as the set of actions on H. The attributes of an action have the following interpretation in our model: u_s and u_r are the sender and the recipient of the message, respectively; u_o represents the patient whose information is to be communicated; m_t and m_p are the type and purpose of the message; c is a set of persons and the consent each of them provides; and b is a set of persons and the belief each of them holds.

2.2 Finite Model for Privacy Laws

In this section, we construct a finite model to formalize a privacy law, subject to certain assumptions on the structure of the law. For illustration, we will primarily use as example the part of the US Health Insurance Portability and Accountability Act (HIPAA) that regulates information sharing in a healthcare provider environment. A main focus is on HIPAA Administrative Simplification, Regulation Text: 45 CFR Parts 160, 162, and 164 that regulate the use and disclosure of *Protected Health Information* (PHI). Although HIPAA is used to illustrate our results, the assumptions under which we derive our results have been abstracted out, so that the results can be applied to similar regulations satisfying the same assumptions.

We continue the approach of [27] to translate the legal clauses of a law into the corresponding rules in Prolog. Each clause of the law often states the situations it covers. For example, HIPAA regulations 164.510(b)(3) states the situations it covers as "the individual is not present, or the opportunity to agree or object to the use or disclosure cannot practicably be provided because of the individual's incapacity or an emergency circumstance". We define predicate $coveredBy\_Ci(A)$ to be true when action A is covered by clause i of the law. Each clause may state some requirements inline that need to be met to permit an action, and we capture them with the predicate $satisfiesInlineReqmt\_Ci(A)$. A clause may further require that the action is also permitted by some other clauses that it references, and we capture this using the predicate $permittedBySomeRef\_Ci(A)$. The following rule predicates are involved in determining whether an action is compliant with a law. The predicate $permittedBy\_L_i(A)$ determines whether action A is permitted by some subset L_i of the legal clauses of the law, and similarly, $forbiddenBy\_L_i(A)$ determines whether A is forbidden by the subset L_i of the legal clauses. The predicate $compliantWithALaw(A)$ is defined to be true if A is permitted by some clause of the law and not forbidden by any clause of the law.

Procedure 2.1. ***Formalization of A Privacy Law*** *Formally, let $m > 0$ be the number of clauses in the privacy law. Recall that in Prolog, the comma (,) represents the* AND *operator and the semicolon (;) represents the* OR *operator. The formalization of the law into the Prolog program P contains the following rules.*

compliantWithALaw$(A) \leftarrow$
  $permittedBySomeClause(A)$,
  $not(forbiddenBySomeClause(A))$.
permittedBySomeClause$(A) \leftarrow$
  $permittedBy\_C1(A)$;
  ...;
  $permittedBy\_Cm(A)$.
forbiddenBySomeClause$(A) \leftarrow$
  $forbiddenBy\_C1(A)$;
  ...;
  $forbiddenBy\_Cm(A)$.

For each $i \in \{1, ..., m\}$, let $n\_i \geq 0$ be the number of clauses referenced from clause i, and define Ref_i, j as the letter 'C' followed by the clause number of the jth reference in clause i. Then the following rules are also in P.

permittedBy_Ci$(A) \leftarrow$
  $coveredBy\_Ci(A)$, $satisfies\_Ci(A)$.
satisfies_Ci$(A) \leftarrow$
  $satisfiesInlineReqmt\_Ci(A)$,
  $permittedBySomeRef\_Ci(A)$.
permittedBySomeRef_Ci$(A) \leftarrow$
  $permittedBy\_Ref\_i, 1(A)$;
  ...;
  $permittedBy\_Ref\_i, n\_i(A)$.
forbiddenBy_Ci$(A) \leftarrow$
  $coveredBy\_Ci(A)$,
  $not(satisfies\_Ci(A))$.

Note the predicate names do not contain the substrings Ref_i, j as they are replaced by the actual clause references. For example, if clause 1 references only clause 2, then **permittedBySomeRef_C1**$(A) \leftarrow permittedBy\_C2(A)$.

Finally, the rules that define the predicates $coveredBy\_Ci(A)$ and $satisfiesInlineReqmt\_Ci(A)$, which are specific to the content of the law, are also part of P.

Although for ease of illustration the rules depicted here we assume only one of the n_i referenced clauses is needed to permit or forbid the action, it is straightforward to generalize them to any finite logical combination of the predicates of the other clauses using the logical connectives AND, OR and NOT. Note that in such cases, for the following algorithms and proofs to apply, intermediate predicates representing the conjunction or disjunction of a subset of predicates can be introduced to maintain the current pattern of rules where *the predicates in the body of the same rule are connected by at most one of the operators AND or OR.*

Assumption 2.2. ***Acyclicity*** *In this paper, we assume the dependency graph of the Prolog program P is acyclic, where P is constructed via the formalization procedure 2.1 from a privacy law L.*

Intuitively, this means that whether an action complies with clause i of the law may depend on whether it complies with some other clause j, but whether the action complies with clause j can't in turn depend on whether it complies with clause i.

Algorithm 2.3. *Construction of* ***Compliance Graphs*** *Given a privacy law L that satisfies Assumption 2.2 and its Prolog translation P by procedure 2.1, we construct its compliance graph $\langle V, E\rangle$ (which we prove is a tree) as depicted in Fig. 1. The compliance graph traces the execution of the Prolog program that determines the compliance of an action with respect to the law. Each node in V is labeled with a predicate in P. Initially E is empty and V contains a single node labeled with $compliantWithALaw(A)$. We recursively build up V and E as follows. Each edge (u, v) in the dependency graph of P is processed at most once as follows. Pick (u, v) such that $u \in V$. If $v \notin V$, we add v to V and (u, v) to E. Otherwise we add a new node v' to V labeled with the same predicate as v, and add the new edge (u, v') to E. For each newly added edge (u, v) in E, we label v with the NOT operator if $\neg v$ is in the rule body that defines u. Finally, we label each internal node u in V with the AND operator if its child predicates are connected by the AND operators, and with the OR operator if its child predicates are connected by the OR operators.*

Lemma 2.4. *Under Assumption 2.2, Alg.2.3 terminates, and the compliance graph constructed is a tree with the root labeled $compliantWithALaw(A)$. See Appendix for proof.*

We call such a compliance graph an **Initial Compilance Tree**.

Algorithm 2.5. ***Normalization of Compliance Trees*** *We normalize an initial compliance tree by pushing the NOT operators towards the leaves using De Morgan's laws, as illustrated in Fig. 2. The normalization step helps to simplify Algorithm 2.7 , as we explain later. Specifically, as the NOT operator is pushed across each layer of the tree towards the leaves, we interchange AND and OR operators, cancel any* *two adjacent NOT operators, and rename each predicate to a new predicate representing its negation. We call the resultant tree a* ***normalized compliance tree****, and we note that it contains only AND and OR but no NOT operators at its internal nodes.*

Lemma 2.6. *Under Assumption 2.2, the initial and normalized compliance trees of a law are finite. See Appendix for proof.*

Given the formalization of a law P, our goal is to understand the behavior of the law by studying its operation on a domain of entities. Ideally, we would like to be able to completely characterize the behavior of the law by studying its operation on a finite domain of entities. This provides a structured procedure for us to study the law by iterating the top-level predicate, *compliantWithALaw*, over a finite set of actions and inspecting which clauses of the law apply and the outcome of whether an action is permitted or prohibited by the law. It also serves as a tool for training and education purposes, as a student can learn the law by observing its operation on example actions. We show that this is possible, and provide a procedure to construct this finite domain. We construct search trees which are subtrees of the normalized compliance tree such that each subtree represents a class of instances of actions that are permitted by the law. We show that there are a finite number of such classes which together completely characterize the law.

Algorithm 2.7. *Construction of* ***Search Trees*** *We choose a selection function [30] f arbitrarily such that given each OR-node u of a normalized compliance tree T_N, f selects precisely one of u's children in iteration, prunes all subtrees rooted at the other children of u that are not selected, and removes the OR label at u. Each application of the selection function to T_N then results in a sub-tree where only the AND-nodes remain, an instance of which is illustrated in Fig. 3. We call each such subtree a* ***search tree*** *[30].*

Note the removal of *NOT* operators in algorithm 2.5 simplifies algorithm 2.7 here as the *NOT* operator when applied to the *AND* and *OR* operators changes their semantics.

Algorithm 2.8. *Construction of* ***Proof Trees*** *and a* ***Representative Hospital*** *We construct proof trees [30] from search trees as follows. We initialize extensional database D_{EDB} to an empty set of facts. For each search tree T_S, define its associated* ***proof tree*** *T_P as (T_S, θ_S), where θ_S is a substitution of variables in T_S by new constants that do not already appear in D_{EDB}. Since initially there are no facts relating the new constants, each negative ground literal of the form $\neg p(\mathbf{b})$ in $\theta_S(T_S)$ is evaluated to true using negation as failure, where $\theta_S(T_S)$ is obtained by applying θ_S to each variable in T_S. Note here that p is necessarily a fact predicate as no negated rule predicate exists in the normalized compliance tree. For each positive ground literal $q(\mathbf{c})$ in $\theta_S(T_S)$ where q is a fact predicate that appears in the rule body of a rule predicate in T_S we add $q(\mathbf{c})$ to D_{EDB}. If we encounter both a ground literal and its negation, such as $p(\mathbf{b})$ and $\neg p(\mathbf{b})$, in $\theta_S(T_S)$, then we mark T_S as inconsistent and T_P is undefined. The* ***Representative Hospital*** *is defined as $H = (Domain, P, D_{EDB})$, where Domain is the set of constants referenced in D_{EDB}, P is the formalization of the law using procedure 2.1 and D_{EDB} is the EDB constructed above. We call the proof trees generated in this algorithm and the search trees generated in algorithm 2.7 the proof trees and search trees associated with H.*

Given T as an initial compliance tree, a normalized compliance tree, or a search tree, we say an action a satisfies T with respect to an EDB D_1, if *compliantWithALaw*(a) evalutes to true in T with respect to D_1. Formally, $P_T \cup D_1 \vdash$ *compliantWithALaw*(a), where P_T is the subset of the rules in the Prolog formalization P that define the rule predicates in T. Similarly, we say an action a evalutes to true at a node u of a tree T with respect to an EDB D_1, if $P_T \cup D_1 \vdash q(a)$, where q is the literal label of u. Here q is p where p is the predicate label of u, if u is not also labeled with the *NOT* operator; otherwise q is $\neg p$.

Theorem 2.9. *Every action a compliant with the law L satisfies some search tree T_S of a valid proof tree $T_P = (T_S, \theta_S)$, where T_S and T_P are associated with the Representative Hospital $H = (Domain, P, D_{EDB})$ constructed in algorithm 2.8. Also, each valid proof tree $T_P = (T_S, \theta_S)$ produces an action $\theta_S(A)$ that is compliant with the law in the Representative Hospital H, where A is a variable vector of the action type.*

3. SECURE HEALTH INFORMATION EXCHANGE

In this section we show how policy formalized as a logic program can be used to automatically generate a form of access control policy used in Ciphertext-Policy Attribute-Based Encryption (CP-ABE). Given a logic program representing the privacy law, we create an extensional database containing a set of individual entities with the attributes referenced in the logic program, and query the database to determine the set of individuals that are permitted to access the EHR. The attributes of such an individual are then used to generate the policy for Attribute Based Encryption (ABE), so that the EHRs encrypted by ABE can only be decrypted by individuals with the right attributes as determined by the logic program. As part of our study, we built and evaluated a prototype system for policy transformation.

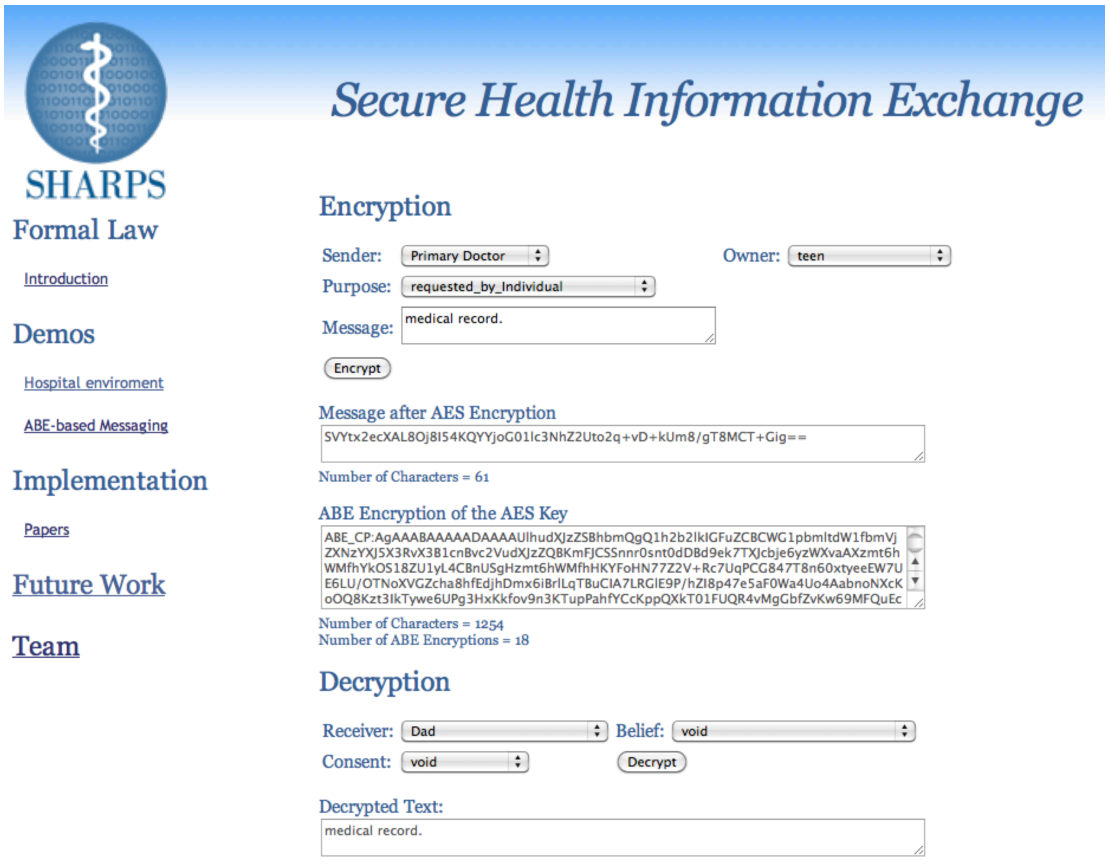
3.1 Background

In CP-ABE, a user's credentials are represented by a set of strings called "attributes." During encryption, a policy expressed as a formula over these credentials is attached to the ciphertext. Subsequently, only users whose attributes satisfy the formula are able to decrypt the ciphertext.

CP-ABE algorithms support four basic operations. The following algorithm is due to Brent Waters [31].

Setup(U) The setup algorithm takes parameter U as the number of attributes in the system. It chooses groups $\mathbb{G}$, $\mathbb{G}_T$ of prime order p, with an associated bilinear map e : $\mathbb{G}$ x $\mathbb{G} \rightarrow \mathbb{G}_T$. It chooses from $\mathbb{G}$ a generator g and U random group elements $h_1, ..., h_U$ that are associated with the U attributes. In addition, it chooses random exponents α, $a \in \mathbb{Z}_p$. The public key is $PK = g, e(g, g)^{\alpha}, g^a, h_1, ..., h_U$, and the master secret key is $MSK = g^{\alpha}$.

Encrypt$(PK, (M, \rho), \mathcal{M})$ The encryption algorithm takes as input the public parameters PK and a message $\mathcal{M}$ to encrypt. In addition, it takes as input an access structure (M, ρ) representing the cipher policy, where M is an l x n matrix and the function ρ associates rows of M to



Secure Health Information Exchange

Encryption

Sender: Primary Doctor
Owner: teen
Purpose: requested_by_Individual
Message: medical record.

Message after AES Encryption

```
SVYtx2ecXAL8Oj8I54KQYYjoG01lc3NhZ2Uto2q+vD+kUm8/gT8MCT+Gig==
```

Number of Characters = 61

ABE Encryption of the AES Key

```
ABE_CP:AgAAABAAAAADAAAAUlhudXJzZSBhbmQgQ1h2b2lkIGFuZCBCWG1pbmltdW1fbmVj
ZXNzYXJ5X3RvX3B1cnBvc2VudXJzZQBKmFJCSSnnr0snt0dDBd9ek7TXJcbje6yzWXvaAXzmt6h
WMfhYkOS18ZU1yL4CBnUSgHzmt6hWMfhHKYFoHN77Z2V+Rc7UqPCG847T8n60xtyeeEW7U
E6LU/OTNoXVGZcha8hfEdjhDmx6iBrlLqTBuCIA7LRGlE9P/hZI8p47e5aF0Wa4Uo4AabnoNXcK
oOQ8Kzt3IkTywe6UPg3HxKkfov9n3KTupPahfYCcKppQXkT01FUQR4vMgGbfZvKw69MFQuEc
```

Number of Characters = 1254
Number of ABE Encryptions = 18

Decryption

Receiver: Dad
Belief: void
Consent: void

Decrypted Text: medical record.

Figure 4: Prototype Screen Shot

attributes. The algorithm chooses a random vector $\vec{v} = (s, y_2, ..., y_n) \in \mathbb{Z}_p^n$. These values will be used to share the encryption exponent s. For $i = 1$ to l, it calculates $\lambda_i = \vec{v} \cdot M_i$, where M_i is ith row of M. In addition, the algorithm chooses random $r_1, ..., r_l \in \mathbb{Z}_p$. The ciphertext CT is the following list of elements: $C = \mathcal{M}e(g,g)^{\alpha s}$, $C' = g^s$, $(C_1 = g^{a\lambda_1}h_{\rho(1)}^{-r_1}, D_1 = g^{r_1})$, ... , $(C_l = g^{a\lambda_l}h_{\rho(l)}^{-r_n}, D_l = g^{r_l})$, and (M, ρ).

KeyGen(MSK, S) The key generation algorithm takes as input the master secret key and a set S of attributes. The algorithm chooses a random $t \in \mathbb{Z}_p$ and creates the user private key as $K = g^{\alpha} g^{at}$, $L = g^t$, and $K_x = h_x^t \forall x \in S$.

Decrypt(CT, SK) The decryption algorithm takes as input a ciphertext CT for access structure (M, ρ) and a user private key for a set S. Suppose that S satisfies the access structure and let $I \subset \{1, 2, ..., l\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, let $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of any secret s according to M , then $\sum_{i \in I} \omega_i \lambda_i = s$. The decryption algorithm first computes $e(C', K) / \prod_{i \in I} (e(C_i, L) e(D_i, K_{\rho(i)}))^{\omega_i} = e(g,g)^{\alpha s} e(g,g)^{ast} / \prod_{i \in I} e(g,g)^{ta\lambda_i \omega_i}) = e(g,g)^{\alpha s}$. The decryption algorithm can then divide out this value from C and obtain the message M.

3.2 Design for Secure HIE and Technical Challenges

Our design for Secure Health Information Exchange has two main components: a logic representation of privacy policy and an encryption module that generates the access control policy for CP-ABE. The CP-ABE component provides three main functions: ABE key distribution, access policy generation and encryption, and decryption. Each CP-ABE function can be performed by a different party, although for the convenience of demonstration we put them on the same web page currently.

3.2.1 Logic Representation of Privacy Policy

A privacy law such as HIPAA states the conditions under which an entity can share someone's data with another entity. Under suitable assumptions such as Assumption 2.2, the law can be formalized and expressed as a Prolog program based on the technique described in [27]. Given an extensional database (EDB) containing facts about the entities, the Prolog program computes the set of permitted health information exchanges, with each exchange represented by an eight-tuple: $a = (Sender, Receiver, Owner, Type, Purpose, ReplyTo, Consent, Belief)$.
This logic system composed of the EDB and the program can serve as part of an access control layer to support secure exchange of health information, which we explain in further detail below.

In the subsequent discussion on permitted health information exchanges, we ignore the ReplyTo element (which represents a previous information exchange that caused the current exchange) and set the Type element to a constant value of *Protected Health Information* (PHI). Also, we sometimes call each instance of information exchange as a message. In our prototype we break up the six relevant elements of permitted exchanges into two parts: *HIPAAQuery* which contains elements specified during the sharing process and *HIPAAResult* which specifies who can legally access the shared message. The *HIPAAQuery* is a 3-tuple consisting of *(Sender, Owner, Purpose)* while the *HIPAAResult* consists of *(Receiver, Belief, Consent)*.

Note that for the purpose of prototyping, testing or debugging a general system, an extensional database (EDB) that demonstrates all interesting information exchange scenarios described by the law can be generated from the Prolog program itself using the approach described in 2. On the other hand, a physical deployment may have an extensional database that limits the possible exchanges, based on the factual constraints in the EDB at the time of exchange. However, the case of exporting sensitive data in encrypted form, and allowing decryption by the rightful entities at a later time presents an interesting challenge. This is because an individual entity who will have the right attributes to decrypt the CP-ABE ciphertext at a future time may not be known to the system at the time of encryption. To address this challenge, similar to the creation of the proof tree from a search tree in 2, we create in the EDB a representative entity for each class of receivers permitted to receive the sensitive information based on some particular search tree. An individual that belongs to such a class of rightful receivers will be issued an ABE user private key with the set of attributes corresponding to that class, and therefore will be able to decrypt the ciphertext later.

While it is straightforward to represent standard roles such as a "Surgeon" or "Nurse" as an attribute, by assigning a unique ID for each role, it is not as obvious how we can represent relations as attributes. For example, some HIPAA clauses state that a psychotherapy note can be sent if the recipient is the original author of that note. As a further example, the parents of a teenage patient may be permitted to receive sensitive information about their own child. In our current design we represent these relations as complex attributes such as (noteId, author), and (patientId, dad), (patientId, mom). While this design may work in a theoretical model, in a real-world deployment we may face the issue of patients being assigned different Ids in different hospitals, and so further refinement of the design will be needed.

3.2.2 Creating the Access Control Policy

For each value of *HIPAAQuery*, there is a set A of *HIPAAResult*s such that each combination of *HIPAAQuery* and a member of this set A is allowed according to the HIPAA law. The person sharing the message specifies the values for the three elements of *HIPAAQuery*. Using those values, we query a precomputed HashMap object (generated by our policy engine) to obtain the set of legal *HIPAAResults*. The person accessing the message provides the values for the elements in the *HIPAAResult* tuple. The individual can access the message only if the tuple of three values provided matches one of the elements in the set. This is the basic concept of the access control that satisfies HIPAA law.

We are enforcing this access control using Attribute-based Encryption as described in the next section. This allows us to encrypt a particular message and publish it publicly. Only those who have the attributes that allow them to legally access the message can decrypt it. This is one way of distributing the access control.

3.3 Prototype Implementation

Policy Generation using JLog JLog [32] is a Java-based Prolog interpreter. We use it to construct Prolog queries, evaluate query results, and generate access control policies.

Attribute-Based Encryption The Functional Encryption library *libfenc* is a general purpose framework for implementing *functional encryption* schemes [28]. It implements a wide range of encryption technologies including Attribute-Based Encryption (ABE), Identity-Based Encryption and others.

Prototype HIE We developed a prototype HIE to demonstrate the enforcement of access control according to HIPAA law using Attribute-Based Encryption. Our HIE acts as a central exchange point where messages can be encoded and decoded by any number of participants with the proper attributes to access the messages. Any message encrypted using our HIE can be safely stored in an untrusted cloud and only legitimate users with valid keys can access their contents.

Our prototype was built with Apache Tomcat 6 and the encryption module is powered by *libfenc* [28]. We demonstrate the entire process from encryption to decryption on the HIE. As previously mentioned, the person sharing the message provides the values for the parameters *Sender, Owner* and *Purpose*. The message is first encrypted using Advanced Encryption Standard (AES). The AES key is thereafter encrypted using the policy corresponding to the values given.

The workflow of the HIE is:

1. Documents are encrypted using AES encryption.

2. We then encrypt the AES key of a document generated from the previous step using ABE. The attribute inputs and the boolean conditions between them is provided by the Prolog implementation.

3. Each user receives a key generated on the basis of his or her attributes which allows him to access the documents that he is entitled to. This key is generated with the help of the ABE master key. Thus if two or more hospitals want to use ABE based encryption to exchange information amongst themselves, they need to use the same ABE master key.

4. When the user wishes to access a particular document, we use his or her private ABE key to decrypt the ABE encrypted AES key of the document. If the decryption is successful, the decrypted AES key is used to decrypt the main document. The information contained in the decrypted document is then displayed to the user.

3.4 Prototype HIE Performance

In our prototype, ABE is only used to encrypt an AES encryption key of 16 Bytes. We find that AES encryption and most parts of our prototype incurs negligible overhead, and most of the time is spent on ABE operations. Perhaps this is because our message sizes are relatively small. The time required to perform an ABE encryption depends on the size of the access policy roughly linearly, whereas decryption time can vary based on not just the size of the access policy but also the specific user private key used for decryption. The reason is that a policy may contain sub-policies joined by the OR logical operator, and a private key that satisfies any of these sub-policies satisfies the joint policy. Depending on the order of evaluation of these sub-policies, 2 private

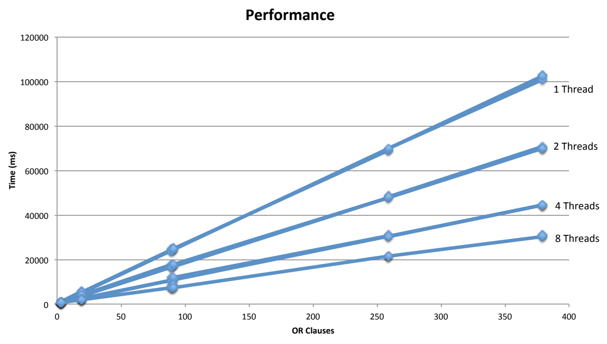


Figure 5: Performance

keys satisfying different sub-policies may incur different decryption times.

Figure 5 shows the encryption time for policies containing different numbers of disjunctive clauses combined together with the OR logical operator. We plot the runtime against the number of OR clauses. Each disjunctive clause is of the form (*Recipient Role* AND *Necessary Consent* AND *Necessary Belief*). For example, when a doctor intends to send the PHI of a teenage patient at request by that teenager, the policy computed states that the health information exchange is permitted if any of the following conditions apply: a surgeon can receive the PHI without any further consent by any person, if it is believed that the PHI contains only the minimal information necessary for the purpose of the exchange; the parents of the patient can receive the PHI without any further consent, with no restriction on any beliefs held by any person; and so on. The measurements were made using an extra-large instance on Amazon EC2 with 8 cores and 16 GB memory. To measure scalability, we repeat the experiment for one, two, four and eight threads. Our experiments suggest that our architecture is scalable and suitable for a cloud-based HIE service.

4. CONCLUSION AND FUTURE WORK

Working with a declarative framework for expressing HIPAA and healthcare privacy policies as logic programs [27], we considered two related problems. First, we investigated whether there is a simple, representative hospital that illustrates all of the interesting cases associated with HIPAA or other policy. For example, if we were to build a web-based portal allowing employees or patients to ask HIPAA questions and receive answers, would this portal depend on the hospital they are affiliated with, or can we use one representative hospital example that illustrates all cases that could occur in all hospitals? Using the concept of *compliance tree,* defined and illustrated in Section 2, we show that there is one representative hospital example for each privacy policy that satisfies a natural set of acyclicity conditions satisfied by HIPAA. An associated algorithm lets us construct this representative example for any given policy. This gives us a useful way to build education or demonstration portals, or to develop test cases to understand a policy when it is being written or applied in any system.

The second problem we considered has to do with using a general declarative privacy policy, such as the HIPAA representation in Prolog/Datalog we developed earlier [27]. If we have a privacy policy presented in this form, can we use it for other purposes besides compliance and audit? In particular, attribute-based encryption (ABE) is a promising cryptographic method for encrypting sensitive data. When data is encrypted using ABE, only individuals with decryption keys that satisfy a given access policy can decrypt that data. The specification determining which credentials are sufficient to decrypt specific data is determined by a policy decision tree that is supplied to the ABE encryption algorithm. Therefore, we investigate the problem of deriving an ABE policy decision tree from our "universal" privacy policy. We implemented the algorithm we developed for doing so in an example system that can be used to put encrypted medical data on an untrusted cloud server, or other publicly available server. We believe that this mechanism is useful to hospital information exchanges (HIE), placing medical research data on servers for download by researchers, and other "meaningful use" purposes.

As future work, we plan to expand and improve our example-generation algorithm, apply it to other classes of policies, and incorporate the results into our online demonstration systems. We will continue to improve our ABE software and produce a system that can be used to place encrypted data on Dropbox or other publicly available servers.

Acknowledgment

We gratefully acknowledge support from the National Science Foundation, the DHHS SHARP Program and SHARPS Project, the Air Force Office of Scientific Research, the Office of Naval Research, and Tata Consultancy Services. The contents of this paper are solely the responsibility of the authors and do not necessarily represent the official views of any sponsoring organization.

We thank Brent Waters for helpful discussion, and Nipun Dave, Divye Khilnani and Vivian Shen for their help in coding, and Ellick Chan for his help with reviewing and revising this paper.

5. REFERENCES

[1] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum, "Privacy and contextual integrity: Framework and applications," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 184–198.
[2] A. Barth, J. Mitchell, A. Datta, and S. Sundaram, "Privacy and Utility in Business Processes," *Computer Security Foundations Symposium, IEEE*, vol. 0, pp. 279–294, 2007.
[3] P. E. Lam, J. C. Mitchell, and S. Sundaram, "A Formalization of HIPAA for a Medical Messaging System," *Lecture Notes in Comp. Sci.*, vol. 5695, pp. 73–85, 2009.
[4] SHARPS, "Strategic Healthcare IT Advanced Research Projects on Security," http://sharps.org.
[5] J. White, J. Daniel, S. Posnack, and L. L. Dimitropoulos, "Privacy and Security Solutions for Interoperable Health Information Exchange - Assessment of Variation and Analysis of Solutions," 2007. [Online]. Available: http://healthit.hhs.gov/portal/server.pt/document/877834/avas_508.pdf
[6] A. Appari, D. Anthony, and M. E. Johnson., "HIPAA Compliance: An Examination of Institutional and Market Forces." *Proceedings of the 8th Workshop on Economics of Information Security, London*, 2009.
[7] M. I. Harrison, R. Koppel, and S. BarLev, "Unintended Consequences of Information Technologies in Health Care - An Interactive Sociotechnical Analysis." *Journal of American Medical Informatics Association*, 2007.

[8] L. O. Gostin, J. G. Hodge, and R. O. Valdiserri, "Informational Privacy and the Publicï£¡fs Health: The Model State Public Privacy Act." *American Journal of Public Health*, 2001.

[9] Markle Foundation, "The Connecting for Health Common Framework." http://www.connectingforhealth.org/.

[10] M. E. F. et al., "A Regional Health Information Exchange: Architecture and Implementation." *Proceedings of AMIA Annual Symposium*, 2008.

[11] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (EPAL 1.1)," http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/, 2003.

[12] M. Backes, G. Karjoth, W. Bagga, and M. Schunter, "Efficient Comparison of Enterprise Privacy Policies," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004, pp. 375–382.

[13] A. Anderson, D. E. A. Nadalin, B. Parducci, F. S. E. Coyne, M. M. H. Lockhart, P. H. M. Kudo, S. P. R. Jacobson, S. A. S. Godik, and T. Moses, "Extensible Access Control Markup Language (XACML) v2.0," 2004.

[14] J. Reagle and L. F. Cranor, "The Platform for Privacy Preferences," *Commun. ACM*, vol. 42, pp. 48–55, 1999.

[15] H. Deyoung, D. Garg, D. Kaynar, and A. Datta, "PrivacyLFP: A Logic of Privacy with Fixed Points 5," 2010.

[16] T. Breaux and A. Anton, "Analyzing Regulatory Rules for Privacy and Security Requirements," *IEEE Transactions on Software Engineering*, vol. 34, pp. 5–20, 2008.

[17] M. J. May, C. A. Gunter, and I. Lee, "Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies," in *Proceedings of the 19th IEEE workshop on Computer Security Foundations*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 85–97.

[18] J. Mathe, J. Martin, P. Miller, A. Ledeczi, L. Weavind, A. Nadas, A. Miller, D. Maron, and J. Sztipanovits., "A Model-Integrated, Guideline Driven, Clinical Decision Support System." *IEEE Software*, 2009.

[19] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema., "Developing Applications Using Model Driven Design Environments." *IEEE Computer*, 2006.

[20] L. E. Olson, C. A. Gunter, and P. Madhusudan., "A Formal Framework for Reflective Database Access Control Policies." *ACM Conference on Computer and Communications Security*, 2008.

[21] L. Olson, C. A. Gunter, and S. P. Olson., "A Medical Database Case Study for Reflective Databases Access Control." *Security and Privacy in Medical and Homecare Systems*, 2009.

[22] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based Encryption with Non-Monotonic Access Structures," in *Proceedings of the 14th ACM conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 195–203.

[23] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-Based Encryption with Partially Hidden Encryptor-Specified Access Structures," in *Proceedings of the 6th International Conference on Applied Cryptography and Network Security*, ser. ACNS'08. Springer-Verlag, 2008, pp. 111–129.

[24] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data," in *Proceedings of the 13th ACM conference on Computer and Communications Security*, ser. CCS '06. ACM, 2006, pp. 89–98.

[25] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 321–334.

[26] R. Bobba, O. Fatemieh, F. Khan, A. Khan, C. A. Gunter, H. Khurana, and M. Prabhakaran, "Attribute-Based Messaging: Access Control and Confidentiality," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 31:1–31:35, December 2010.

[27] P. E. Lam, J. C. Mitchell, and S. Sundaram, "A Formalization of HIPAA for a Medical Messaging System," in *Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business*, ser. TrustBus '09. Springer-Verlag, 2009, pp. 73–85.

[28] "FEATURES: Functional Encryption Applicable to Usable and Really Effective Security," 2010, http://spar.isi.jhu.edu/features/index.html.

[29] U. Nilsson and J. Maluszynski, *Logic, Programming, and PROLOG*. New York: John Wiley & Sons, Inc., 1995.

[30] S. Ceri, G. Gottlob, and L. Tanca, *Logic programming and databases*. New York: Springer-Verlag, 1990.

[31] B. Waters, "Ciphertext-Policy Attribute-Based Encryption: an Expressive, Efficient, and Provably Secure Realization," in *Public Key Cryptography (PKC 2011)*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 53–70.

[32] "JLog - Prolog in Java," 2011. [Online]. Available: http://jlogic.sourceforge.net/

Proofs

Proof Sketch of Lemma 2.4 By procedure 2.1, every predicate in the dependency graph of the Prolog program P is reachable from the predicate $compliantWithALaw(A)$. Hence Each edge (u, v) in the dependency graph of P is processed exactly once by algorithm 2.3, i.e. the algorithm terminates. In the compliance graph $\langle V, E\rangle$, a new edge (u, v) is added to E precisely when a new node v is added to V. This ensures that each node in $\langle V, E\rangle$ has at most 1 parent. In addition, by construction every node in $\langle V, E\rangle$ is connected from the node labeled with $compliantWithALaw(A)$. Hence each node in the compliance graph has a unique path to the node labeled with $compliantWithALaw(A)$. □

Proof Sketch of Lemma 2.6 It is easy to check that the normalization algorithm 2.5 does not change the number of nodes in the compliance trees. For both the initial and normalized compliance trees, the branching factors of the internal nodes are bounded above by the number of legal clauses in the law, which we denote as m. The only place where a clause references another clause is via the predicates of the type *permittedBy_Ci*. Since the dependency graph of the law is acyclic, for any i each *permittedBy_Ci* predicate appears at most once in a chain of references, so the height of the tree is bounded above by $3m + 2$. □

Proof Sketch of Theorem 2.9 Given an action a that is compliant with the law L in a context represented by the facts of an EDB D_1, we have $P_{T_I} \cup D_1 \vdash compliantWithALaw(a)$, where T_I is the initial compliance tree, P is the Prolog formalization, and $P = P_{T_I}$ as they contain the same set of rule predicates. This in turn is true iff for each internal node u in the normalized compliance tree T_N the following is true with respect to D_1: if u's children are connected by AND a is evaluted to true at all of them, and if u's children are connected by OR a is evaluted to true at at least one of them. We can use a selection function f that picks for each OR node u exactly one of the children nodes where a is evaluated to true. f can then be applied to T_N to generate the required search tree T_S, and θ_S is the substitution that maps the variable vector A to the satisfying action a, such that $P_{T_S} \cup D_1 \vdash compliantWithALaw(a)$.

Given $H = (Domain, P, D_{EDB})$ and proof tree $T_P = (T_S, \theta_S)$, observe that $P_{T_S} \cup D_{EDB} \vdash compliantWithALaw(\theta_S(A))$. By the construction of T_S as a subtree of the normalized compliance tree T_N, $P_{T_N} \cup D_{EDB} \vdash compliantWithALaw(\theta_S(A))$. And by the construction of T_N from the initial compliance tree T_I, $P_{T_I} \cup D_{EDB} \vdash compliantWithALaw(\theta_S(A))$, where $P = P_{T_I}$. Hence $\theta_S(A)$ is compliant with the law in the context of D_{EDB}, where $H = (Domain, P, D_{EDB})$. □