Sequence Diagram Aided Privacy Policy Specification

Hui Shen, Ram Krishnan, Rocky Slavin, and Jianwei Niu

Abstract—A fundamental problem in the specification of regulatory privacy policies such as the Health Insurance Portability and Accountability Act (HIPAA) in a computer system is to state the policies precisely, consistent with their high-level intuition. In this paper, we propose UML sequence diagrams as a practical means to graphically express privacy policies. A graphical representation allows decision-makers such as application domain experts and security architects to easily verify and confirm the expected behavior. Once intuitively confirmed, our work in this article introduces an algorithmic approach to formalizing the semantics of sequence diagrams in terms of linear temporal logic (LTL) templates. In all the templates, different semantic aspects are expressed as separate, yet simple LTL formulas that can be composed to define the complex semantics of sequence diagrams. The formalization enables us to leverage the analytical powers of automated decision procedures for LTL formulas to determine if a collection of sequence diagrams is consistent, independent, etc. and also to verify if a system design conforms to the privacy policies. We evaluate our approach by modeling and analyzing a substantial subset of HIPAA rules using sequence diagrams.

Index Terms—Formal verification, sequence diagram, temporal logic, privacy policy, HIPAA

1 Introduction

A number of privacy policy regulations have been enacted as public law. Prime examples of such policies include health privacy rules of the Health Insurance Portability and Accountability Act (HIPAA) [1], consumer financial privacy rules of GLBA [3], student educational record privacy rules of FERPA [4], children online privacy rules of COPPA [2], etc. These privacy policies typically regulate sharing of private information of an individual between two or more organizations. In order to enforce such privacy policies in information systems, they must be codified precisely, consistent with the intuition behind those policies. However, in practice, the people who design such privacy policies are not experts in codifying those policies in computer systems while those who codify those policies are not experts in the legal parlance using which those privacy policies are often specified.

Consider HIPAA for example. There has been extensive prior work on formalizing HIPAA so it can be analyzed and codified in computer systems to regulate information sharing. Protection of private data often requires timely communication among multiple parties in a decentralized manner, and needs to accommodate the preferences of the subjects of private data. Several frameworks have been proposed for specifying and analyzing privacy policies using formal methods, including contextual integrity (CI) [8], Privacy APIs [26], PrivacyLFP [13], and work by Lam et al. [24]. To date, most of the work in this area has concentrated on methodologies for formally capturing privacy policies or analyzing privacy policies to determine whether they satisfy various properties. Much less emphasis is placed on whether the framework, comprising techniques and tools, enable domain experts to comprehend and check whether the codification of privacy rules is valid. In particular, such works assume that what has been codified is accurate with respect to the intuition behind legally stated HIPAA rules. This presents a fundamental security challenge that is hard to address. We propose that HIPAA domain experts should be involved while codifying those complex privacy rules.

One of the most prevalent tools that is used today for design specifications is a collection of unified modeling language (UML) sequence diagrams [28]. Our premise is that UML sequence diagrams are intuitive ways to express privacy policies such as those in HIPAA which regulate interaction between various entities. The graphical nature of sequence diagrams allow domain experts to visually confirm the intuition, for example, behind HIPAA privacy rules. Once confirmed, those sequence diagrams still need to be formally codified in computer systems and analyzed. Our technique allows the sequence diagrams to be algorithmically translated into linear temporal logic (LTL) templates that precisely characterize the intuition behind graphical specification by privacy architects. This enables automated verification using techniques such as model checking. We demonstrate these tasks using a HIPAA case-study in this article.

We focus on two aspects. The first aspect centers on conformance of system design with that of stated privacy policies. As will be discussed later, we use the notion of *security properties* as conceived by Schneider [31] to model *privacy policies*. The second aspect focuses on formal characteristics of the security properties (privacy policies) themselves. Our theory applies to the above two aspects in general, and our verification applies specifically to HIPAA—that is, if a hospital information system specification conforms to HIPAA privacy policies/rules, i.e., security properties, (aspect 1),

- *H. Shen, R. Slavin, and J. Niu are with the Department of Computer Science, University of Texas at San Antonio, Texas, TX 78249. E-mail: huishen12@gmail.com, rocky.l.slavin@ieee.org, niu@cs.utsa.edu.*
- *R. Krishnan is with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, Texas, TX 78249. E-mail: ram.krishnan@utsa.edu.*

Manuscript received 2 Dec. 2013; revised 21 Nov. 2014; accepted 11 Dec. 2014. Date of publication 18 Dec. 2014; date of current version 18 May 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2014.2384500

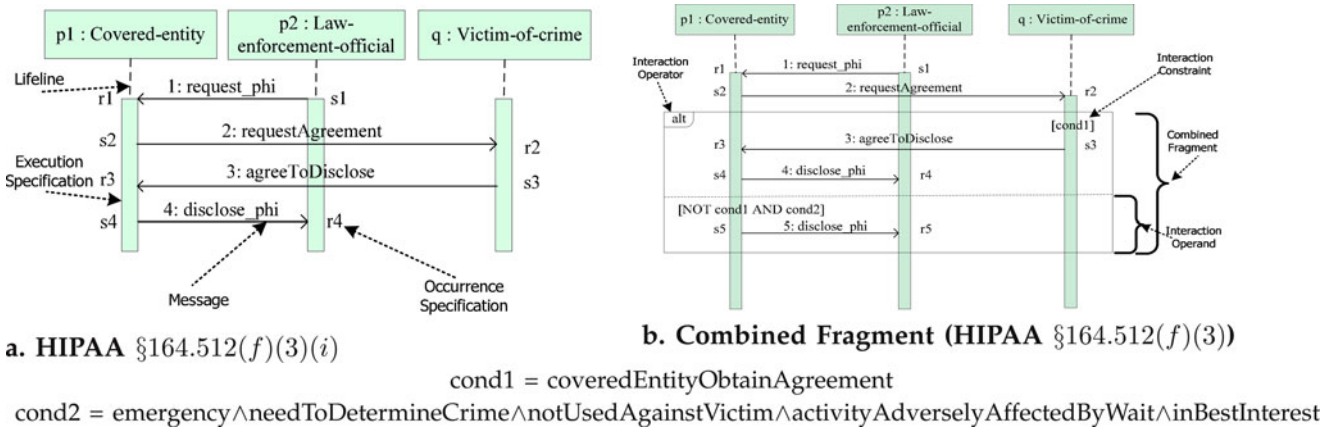

Fig. 1. Sequence diagram syntax.

if the rules are logically consistent with one another, and if they are logically independent from one another (aspect 2).

The following criteria specify our rationale behind selecting sequence diagrams as a framework of choice for security policy specification: (1) The framework should be intuitive and help bridge the gap between application domain experts and system architects or designers. (2) The framework must facilitate formal verification of security properties of the system using automated tools. (3) It would be convenient to utilize a single framework for specifying both the system design and its expected security properties.[1]

A number of prior work in specification and verification of systems satisfy some of the above criteria, but not all. For instance, the large body of research work on utilizing process algebra based approaches for this purpose satisfies criterion (2), but not (1) or (3). For example, in [16], the authors propose an approach to translate UML into process algebra models. Such approaches have many practical limitations. For example, the tool developed in [16] can only discover deadlocks. Theoretically, one might translate process algebra constructs into sequence diagrams, but this is not a feasible approach in practice. Prior work on formalizing trace semantics [34], [35] satisfy criterion (1) and aspects of (2). Formalizing large-scale privacy policies such as HIPAA at individual trace level can become very cumbersome. Furthermore, we are not aware of tools that are readily available for formal verification of properties against systems using the trace theory approach in these works.

The contributions of this article are as follows:

- We develop an approach to use LTL to formalize the semantics of UML 2 sequence diagrams with all the combined fragments (CF), including nested combined fragments and interaction constraints. These constructs allow different types of control flow for presenting complex and concurrent behaviors. In the framework, different semantic aspects are expressed as separate, yet simple LTL formulas that can be composed to define the semantics of a sequence diagram. (Sections 3 and 4). As discussed in Section 7 , prior work does not formalize all the necessary constructs of sequence diagrams *in LTL*. Our approach that formalizes in LTL permits us to leverage existing automated procedures such as model checking for rigorous formal analysis.
- We present how to use UML 2 sequence diagrams for formally specifying a substantial portion (>100 sequence diagrams) of the HIPAA privacy policies that are behavior-related. (HIPAA includes certain static definitions that we do not consider for our purpose.)
- We develop a tool suite that can generate LTL templates given the sequence diagrams as input. This tool enables system designers and security architects to conduct formal analysis. To this end, we utilize this tool and a model checker (Section 6) to verify the consistency and independence of HIPAA privacy rules as well as conformance thereto.

2 UML 2 Sequence Diagram

The semantics of sequence diagrams are described in terms of traces by the object management group [28]. However, it is not formally defined how traces are derived compared to their precise syntax descriptions [28]. As the first step of defining a sequence diagram using LTL formulas, we begin with presenting the semantics of the basic sequence diagram, then discussing the structured control constructs [32].

2.1 Basic Sequence Diagram

We refer to a sequence diagram without combined fragments as a basic sequence diagram (see Fig. 1a for an example with annotated syntactic constructs). A *Lifeline* is a vertical line representing a participating object. A horizontal line between Lifelines is a *Message*. Messages are of two types: asynchronous and synchronous. (A synchronous Message is depicted with a block arrow head and must have an explicit reply Message. Its formal semantics can be found in Section 5.6 of [17]) Each Message is sent from its source Lifeline to its target Lifeline and has two endpoints. Each endpoint is an intersection with a Lifeline and is called an *Occurrence Specification (OS)*, denoting a sending or receiving event occurrence within a certain context, i.e., a

1. In this case, sequence diagrams are used to specify both the system specification and the expected security properties. We then check if the system specification conforms to security properties. Note that this approach is not specific to verifying conformance of only security properties, but more generally to check conformance of one set of sequence diagrams (e.g. system specification) with that of another (e.g. system requirements).

sequence diagram. OSs can also be the beginning or end of an *Execution Specification* [28].

The semantics of a basic sequence diagram is defined by a set of traces. A trace is a sequence of OSs expressing Message exchange among multiple Lifelines. We identify four orthogonal semantic aspects, each of which is expressed in terms of the execution order of concerned OSs.

1) Each OS can execute only once, i.e., each OS is unique within a sequence diagram.
2) On each Lifeline, OSs execute in graphical order.
3) For a single Message, the sending OS must take place before the receiving OS does.
4) In a sequence diagram, only one object can execute an OS at a time, i.e., OSs on different Lifelines are interleaved.

2.2 Structured Control Constructs

A *Combined Fragment* is represented as a solid-outline rectangle, which consists of an *Interaction Operator* and one or more *Interaction Operands*. CFs can be nested within other CFs. Fig. 1b shows an example CF with annotated syntactic constructs. A CF can enclose all, or part of, Lifelines in a sequence diagram. The Interaction Operands are separated by dashed horizontal lines. The Interaction Operator is shown in a pentagon in the upper left corner of the rectangle. OSs, CFs, and Interaction Operands are collectively called *Interaction Fragments*. An Interaction Operand may contain a Boolean expression which is called an *Interaction Constraint* or Constraint. An Interaction Constraint is shown in a square bracket covering the Lifeline where the first OS will happen. An *Interaction Use* construct allows one sequence diagram to refer to another sequence diagram. The referring sequence diagram copies the contents of the referenced sequence diagram. We identify three orthogonal semantic rules general to all CFs.

1) OSs and CFs, are combined using weak sequencing (defined below). On a single Lifeline, a CF's preceding Interaction Fragment must complete the execution prior to the CF's execution, and the CF's succeeding Interaction Fragment must execute subsequently.
2) Within a CF, the semantics of each CF Operator determines the execution order of all the Operands. If the Constraint of the Operand evaluates to $True$, the OSs and CFs within each Operand are combined using weak sequencing. If it evaluates to $False$, the Operand is excluded.
3) The CF does not execute when the Constraints of all the Operands evaluate to $False$. Thus, its preceding and succeeding Interaction Fragments are ordered by Weak Sequencing.

2.3 Interaction Operator

Each Operator has its specific semantic implications regarding the execution of the OSs enclosed by the CF on the covered Lifelines. The Operators that are utilized in the article are summarized as follows. (We provide the complete list of Operators in [17]):

- *Alternatives*. One of the Operands whose Interaction Constraints evaluate to $True$ is nondeterministically chosen to execute.
- *Option*. Its sole Operand executes if the Interaction Constraint is $True$.
- *Parallel*. The OSs on a Lifeline within different Operands may be interleaved, but the ordering imposed by each Operand must be maintained.
- *Loop*. Its sole Operand will execute for at least the minimum count (lower bound) and no more than the maximum count (upper bound) as long as the Interaction Constraint is $True$.
- *Assertion*. OSs on a Lifeline must occur immediately after the preceding OSs.
- *Negative*. Its Operand specifies forbidden traces.
- *Weak sequencing*. *On a Lifeline*, the OSs and CFs within an Operand cannot execute until the OSs and CFs in the previous Operand complete.

2.4 Sequence Diagram Deconstruction

To facilitate codifying the semantics of sequence diagrams and nested CFs in LTL formulas, we show how to deconstruct a sequence diagram and CFs to obtain fine-grained syntactic constructs. Recall that OSs and CFs directly enclosed in the same Operand or sequence diagram are combined using weak sequencing, constraining their orders with respect to each individual Lifeline only [28]. We further deconstruct a sequence diagram into syntactic constructs on each Lifeline.

We project every CF cf_m onto each of its covered Lifelines l_i to obtain a *compositional execution unit (CEU)*, which is denoted by $cf_m{\uparrow}_{l_i}$ (e.g., OSs r3, s4, and s5 are grouped into a CEU (see Fig. 1b)). Every Operand op_n of CF cf_m is projected onto each of its covered Lifelines l_i to obtain an *execution unit (EU)* while projecting cf_m onto l_i, denoted by $op_n{\uparrow}_{l_i}$ (e.g., OSs r3 and s4 are in an EU (see Fig. 1b)). If the projected Interaction Operand contains a nested Combined Fragment, a *hierarchical execution unit (HEU)* is obtained; otherwise a *basic execution unit (BEU)* is obtained, i.e., an EU is a BEU if it does not contain any other CEUs. Projecting a sequence diagram onto each enclosing Lifeline also obtains an EU. In an HEU, we also group the OSs between two adjacent CEUs or prior to the first CEU or after the last CEU on the same level into BEUs (e.g., OSs r1 and s2 are grouped into an BEU (see Fig. 1b)).

The deconstruction enables us to focus on the order of constructs on each Lifeline, i.e., a composition of (nested) CEUs at different levels. On each Lifeline, the OSs and CEUs directly enclosed in the sequence diagram, which are considered at the highest-level, are ordered sequentially. The semantics of the CEUs are represented at a lower-level, where the order of the EUs directly enclosed in each CEU depends on its Interaction Operator. Similarly, the directly enclosed OSs and CEUs (if any) within each EU are ordered sequentially at the next level. A BEU is considered at the lowest-level. In this way, the semantics of the constructs on a Lifeline can be described recursively. The inter-Lifeline semantics is defined by the Messages and the interleaving semantics, i.e., the receiving OS must happen after the sending OS of the same Message, and only one Lifeline can execute an OS at a time. The inter-Lifeline semantics rules are independent from the intra-Lifeline semantics rules, which make the

semantics rules composable. Further details on the deconstruction are provided in [17].

3 Trace Semantics

The semantics of a sequence diagram (denoted seq which is not to be mistaken for "seq" construct for weak sequencing) with CFs is defined by two sets of traces, one containing a set of valid traces, denoted as $Val(seq)$, and the other containing a set of invalid traces, denoted as $Inval(seq)$. The intersection of these two sets is empty, i.e., $Val(seq) \cap Inval(seq) = \emptyset$. Traces specified by a sequence diagram without a Negative CF are considered as valid traces. An empty trace is a valid trace. Invalid traces are defined by a Negative CF. A trace derived from seq can be finite, denoted as $\sigma[..n] = \sigma_0\sigma_1 \ldots \sigma_n$. The trace can also be infinite (denoted as $\sigma = \sigma_0\sigma_1 \ldots \sigma_n \ldots$) if it expresses the behavior of infinite iterations in terms of Loop with infinite upper bound.

3.1 System and Policy Specification

It is well-known that sequence diagrams can be used for design specifications. One key question is that to what extent can sequence diagrams be used to specify security policies of a system. Our work allows a system and the security policies that are expected of the system both to be specified using sequence diagrams. Subsequently, we leverage model checking for verifying if a system conforms to security policies and to ensure consistency and independence of those policies. We formally define these using the trace semantics of sequence diagrams.

Let P represents a set of sequence diagrams that define the security policy that the system, which is characterized by a set of sequence diagrams S, is required to satisfy. Our work enables P and S to be translated into respectively equivalent LTL templates. Let Σ_S denote the set of events that can occur in the system. Let Σ_P denote the set of events that the policy is concerned about. For simplicity, we assume that $\Sigma_S \supseteq \Sigma_P$. Thus, Σ_S^ω denotes the set of all possible traces of system events, and Σ_P^ω denotes the set of all possible traces of policy events. Let $\Sigma = \Sigma_P \cup \Sigma_S$. Σ^ω denotes all traces of system and policy events.

We use the notions of system and security policy as characterized by Schneider [31]. Note that as per Alpern and Schneider [5], [6], the policies that are enforceable in Schneider's class of "Enforcement Monitoring" mechanisms [31] is a *property*. In our case, the system could be a hospital system, and the HIPAA rules represent the security policy. A system conforms to policy if the set of traces generated by the system is a subset of that allowed by the policy.

In our framework, the nature of security policies that can be expressed using sequence diagrams fall in the category of *safety properties* as characterized by Alpern and Schneider [6].[2] Safety properties specify "bad things" that must not happen during the execution of the system specified by S. Thus, in the context of HIPAA, we specify HIPAA rules using sequence diagrams (P). A hospital information system is also specified using sequence diagrams (S). We then verify if S conforms to P by specifying S as LTL templates (or as a NuSMV model [32]) and P as properties that are expected of the model using LTL. Conformance can then be verified using a model checker.

Note that UML 2.0 sequence diagrams are limited in expressive power for specifying *liveness properties* since the semantics of the assert construct is limited. In [20], the authors add universal semantic interpretation to UML 2.0 sequence diagrams to support specification of safety and liveness properties. In contrast, our approach utilizes UML 2.0 sequence diagrams as is to express P. In our verification, we interpret P with universal semantics. We incorporate the notion that P specifies an ordering of events that must always be preserved in S. When using sequence diagrams specifying S, we consider that all the traces derived from a sequence diagram are complete traces, where no OSs of other sequence diagrams may execute [19]. In the case of sequence diagrams specifying P, we adopt partial trace semantics, i.e., OSs in Σ_S but not in Σ_P can interleave the partial traces. In summary, sequence diagrams can be used to specify safety properties of systems, and partly liveness properties to the extent the Assertion CF allows.

3.2 Property Definitions

In this section, given S and P, we define the notions of conformance, consistency and independence.

Conformance. We can use sequence diagrams, which are represented as LTL formulas using our framework, to express information systems as well as policy and regulation rules. A system is said to conform to a policy if each system execution satisfies the policy or regulation rules. That is, $\forall \sigma \in \Sigma_S^\omega : \sigma \models S \rightarrow P$.

Consistency. The consistency among a set of related policy rules states that there exists a trace that satisfies all the rules in the set. That is, $\exists \sigma \in \Sigma_P^\omega : \sigma \models P$, where σ is a trace and P is a set of related policy rules expressed as sequence diagrams.

Independence. The independence among a set of related policy rules requires that no policy rule nor its negation can be logically entailed by other policy rules. That is, it is *not* the case that:

$$\forall \sigma \in \Sigma_P^\omega : \sigma \models \left(\bigwedge_{i,j \in \{1 \cdots |P|\} \wedge i \neq j} P_j \right) \rightarrow P_i$$

and it is *not* the case that:

$$\forall \sigma \in \Sigma_P^\omega : \sigma \models \left(\bigwedge_{i,j \in \{1 \cdots |P|\} \wedge i \neq j} P_j \right) \rightarrow \neg P_i$$

where σ is a trace, P is a set of related policy rules, and P_i, P_j are rules in P.

3.3 LTL for Sequence Diagram

This paper presents a framework to characterize the traces of sequence diagram in linear temporal logic. LTL is a formal language for specifying the orders of events and states in

2. Sloman et al. [33] define policy informally as that they "define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves". Although we believe that sequence diagrams can be used to express security policies in general with appropriate extensions, we restrict our scope to specification and verification of the formal notion of safety properties [6] using UML 2.0 sequence diagrams.

TABLE 1
Auxiliary Functions

Function	Explantation
$LN(p)$	return the set of all Lifelines in p.
$MSG(p)$	return the set of all Messages directly enclosed in p.
$SND(j)$	return the sending OS of Message j.
$RCV(j)$	return the receiving OS of Message j.
$Reply(u)$	return the reply Message of a synchronous Message containing OS u.
$typeOS(u)$	return the type of OS u, which is a sending OS or a receiving OS.
$typeCF(u)$	return the Interaction Operator of CF u.
$TOP(u)$	return the set of Interaction Operands whose Constraints evaluate to $True$ within CF u.
$nested(u)$	return the set of CFs, which are directly enclosed in CF u's Interaction Operands whose Constraints evaluate to $True$. It can be overloaded to an Interaction Operand or a sequence diagram.
$TBEU(u)$	for CEU or EU u, return a set of directly enclosed BEUs, whose Constrains evaluate to $True$.
$AOS(q)$	return the set of OSs which are enabled (i.e., the Constraints associated with it evaluate to $True$) and chosen to execute in q.
$TOS(u)$	return the set of OSs of the BEUs directly enclosed in CEU or EU u whose Constaints evaluates to $True$.
$pre(u)$	return the set of OSs which may happen right before CEU u. The set contains an OS if a BEU whose Constraint evaluates to $True$ prior to u on the same Lifeline. If a CEU executes prior to u on the same Lifeline, the set may contain a single or multiple OSs depending on the CEU's Operator and nested CEUs (if there are any nested CEUs). If an HEU executes prior to u on the same Lifeline, the set is determined by the last CEU or BEU nested within the HEU.
$post(u)$	return the set of OSs which may happen right after CEU u, which can be calculated in a similar way as $pre(u)$.

terms of temporal operators and logical connectives. We use LTL formulas to express the semantic rules prescribed by sequence diagram constructs, each of which constrains the execution orders among OSs. Note that an LTL formula is defined over infinite traces. In the case that sequence diagram seq expresses a set of finite traces, we need to handle the mismatch between an LTL formula and a sequence diagram's finite trace semantics. We adapt the finite traces of sequence diagrams without altering their semantics by adding stuttering of a no-op, τ, which is an empty event occurrence and not observable, after the last OS σ_n of each trace [19]. We denote the set of finite traces of seq using Σ^*_{seq}, where Σ_{seq} is the set of OSs of seq. We denote the infinite adaptation using $\Sigma^*_{seq}\tau^\omega$, where τ^ω represents an infinite sequence of no-ops.

We include a summary of temporal operators that are sufficient to understand our LTL template. $\Box p$ means that formula p will continuously hold in all future states. $\Diamond p$ means that formula p holds in some future state. $\bigcirc p$ means formula p holds in the next state. $\ominus p$ means that formula p holds in the previous state. $\diamondsuit p$ means that formula p holds in some past state. $\hat{\diamondsuit} p \equiv \diamondsuit \ominus p$ means that formula p holds in some past state, excluding current state. $p\,\mathcal{U}\,q$ means that formula p holds until some future state where q becomes true, and p can be either $True$ or $False$ at that state. The macro $p\,\widetilde{\mathcal{U}}\,q \equiv p\,\mathcal{U}(q \wedge p)$ states that in the state when q becomes $True$, p stays $True$.

4 Sequence Diagram Formalization

In this section, we describe how to use LTL to formalize sequence diagrams. Codifying the semantics of a notation can be challenging, especially if we consider all semantic constraints at once. To reduce the complexity and to improve readability, we devise an LTL framework, comprised of simpler definitions, we call *templates*, to represent each semantic aspect (i.e., the execution order of event occurrences imposed by individual constructs) as a separate concern. To capture the meanings of nested CFs, we provide a recursively defined template, in which each individual CF's semantics is preserved (e.g., the inner CF's semantics is not altered by other CFs containing it). These templates can then be composed using temporal logic operators and logical connectives to form a complete specification of a sequence diagram.

To facilitate the representation of a sequence diagram in LTL, we define a collection of auxiliary functions (see Table 1) to access information of a sequence diagram. For instance, function $SND(j)$ returns the sending OS of Message j. The algorithms to calculate the auxiliary functions are provided in [17]. Functions $MSG(p)$, $LN(p)$, $AOS(q)$ are overloaded where p can be an Interaction Operand, a CF, or a sequence diagram, and q can be p, an EU, or a CEU.

4.1 Basic Sequence Diagram

We start with defining an LTL template, called Π^{Basic}_{seq} (see Fig. 2), to represent the semantics of a basic sequence diagram. The semantic rules for basic sequence diagram seq defined in Section 2.1 are codified separately using formulas α_g, β_j, and ε_{seq}.

α_g focuses on the intra-lifeline behavior to enforce rules 1 and 2. Recall that when projecting basic sequence diagram

$$\Pi^{Basic}_{seq} = \Big(\bigwedge_{\substack{i\in LN(seq)\\ g=seq\uparrow_i}} \alpha_g\Big) \wedge \Big(\bigwedge_{j\in MSG(seq)} \beta_j\Big) \wedge \varepsilon_{seq}$$

$$\alpha_g = \Big(\bigwedge_{k\in[r..r+|AOS(g)|-2]} (\neg OS_{k+1}\,\widetilde{\mathcal{U}}\,OS_k)\Big) \wedge \bigwedge_{OS_e\in AOS(g)} (\neg OS_e\,\mathcal{U}\,(OS_e \wedge \bigcirc\Box\neg OS_e))$$

$$\beta_j = \neg RCV(j)\,\widetilde{\mathcal{U}}\,SND(j)$$

$$\varepsilon_{seq} = \Box\Big(\Big(\widehat{\bigvee_{OS_m\in AOS(seq)} OS_m}\Big) \vee \Big(\bigwedge_{OS_m\in AOS(seq)} (\hat{\diamondsuit} OS_m)\Big)\Big)$$

Fig. 2. LTL templates for basic sequence diagram.

$$\Pi_{seq} = \bigwedge_{i \in LN(seq)} \Big(\bigwedge_{g \in TBEU(seq\uparrow_i)} \alpha_g \Big) \quad \wedge \quad \bigwedge_{j \in MSG(seq)} \beta_j \quad \wedge \quad \bigwedge_{CF \in nested(seq)} \Phi^{CF} \quad \wedge \quad \varepsilon_{seq}$$

Fig. 3. LTL templates for sequence diagram with combined fragments.

$$\Phi^{CF} = \begin{cases} \eta^{CF} & if |TOP(CF)| = 0 \quad (1) \\ \Psi^{CF} \wedge \bigwedge_{CF_i \in nested(CF)} \Phi^{CF_i} & if (|TOP(CF)| > 0) \wedge (typeCF(CF) \neq alt) \wedge (typeCF(CF) \neq loop) \quad (2) \\ \Psi^{CF}_{altOrloop} & if (|TOP(CF)| > 0) \wedge ((typeCF(CF) = alt) \vee (typeCF(CF) = loop)) \quad (3) \end{cases}$$

$$\eta^{CF} = \bigwedge_{i \in LN(CF)} \Big(\big(\bigwedge_{OS_{post} \in post(CF\uparrow_i)} (\neg OS_{post}) \big) \, \widetilde{\mathcal{U}} \, \big(\bigwedge_{OS_{pre} \in pre(CF\uparrow_i)} (\diamondsuit OS_{pre}) \big) \Big)$$

Fig. 4. LTL template for nesting combined fragments.

$$\Psi^{CF} = \theta^{CF} \quad \wedge \quad \bigwedge_{i \in LN(CF)} \gamma_i^{CF} \quad \wedge \quad \Omega^{CF}$$

$$\theta^{CF} = \bigwedge_{i \in LN(CF)} \Big(\bigwedge_{g \in TBEU(CF\uparrow_i)} \alpha_g \Big) \quad \wedge \quad \bigwedge_{j \in MSG(TOP(CF))} \beta_j$$

$$\gamma_i^{CF} = \bigwedge_{OS \in TOS(CF\uparrow_i)} \big((\neg OS \, \widetilde{\mathcal{U}} \, \big(\bigwedge_{OS_{pre} \in pre(CF\uparrow_i)} (\diamondsuit OS_{pre}) \big) \big) \quad \wedge \quad \big(\big(\bigwedge_{OS_{post} \in post(CF\uparrow_i)} (\neg OS_{post}) \big) \, \widetilde{\mathcal{U}} \, (\diamondsuit OS) \big)$$

Fig. 5. LTL template for OSs directly enclosed in combined fragment.

seq onto its covered Lifelines, $LN(seq)$, we obtain BEU g for each Lifeline i, denoted as $seq \uparrow_i$. Each BEU g contains a trace of OSs, $\sigma[r..(r + |AOS(g)| - 1)]$, where $r \geq 0$ and σ_r is the first OS in BEU g, function $AOS(g)$ returns the set of OSs within g, and $|AOS(g)|$ has its usual meaning, returning the size of set $AOS(g)$. The first conjunct of α_g enforces the total order of OSs in each BEU g. For example, in Fig. 1 a, the first conjunct of α_{p2} is $\neg r4 \, \widetilde{\mathcal{U}} \, s1$, asserting that for Lifeline $p2$, $s1$ occurs before $r4$. The second conjunct of α_g enforces that every OS in BEU g executes only once. For instance, on Lifeline $p2$, the second conjunct of α_{p2} is $(\neg s1 \, \mathcal{U} \, (s1 \wedge \bigcirc\Box\neg s1)) \wedge (\neg r4 \, \mathcal{U} \, (r4 \wedge \bigcirc\Box\neg r4))$. The semantics enforced by each α_g does not constrain each other. Thus, the intra-lifeline semantics of seq is enforced by the conjunction of α_g for each Lifeline.

The semantics rule 3 is codified by a conjunction of β_j for each Message j. Formula β_j enforces that, for Message j, its receiving OS, $RCV(j)$, cannot happen until its sending OS, $SND(j)$ happens. For example, in Fig. 1a, the sending OS $s1$ occurs before the receiving OS $r1$, denoted as $\neg r1 \, \widetilde{\mathcal{U}} \, s1$. Formula ε_{seq} enforces interleaving semantics of complete traces among all the OSs of sequence diagram seq in the fourth rule, which denotes that only one OS of seq can execute at once, and the trace should execute uninterrupted until all the OSs of seq have taken place. The trace stutters at the end with τ, which is a no-op. We define the logical operator "unique or" as "$\widehat{\bigvee}$", to denote that exactly one of its OSs is chosen. A formula with logical connectives, $\bigwedge_{a_i \in S} a_i$ returns the conjunction of all the elements a_i within the set S. It returns $True$ if S is an empty set.

4.2 Correctness of LTL Semantics

Here we provide a proof sketch showing that the LTL templates for basic sequence diagram seq capture its semantics. A complete proof is provided in [17].

Theorem 1. *$(\Sigma_{sem}^{seq})^*$ and $PRE_{2j}((\Sigma_{LTL}^{seq})^\omega)$ are equal.*

Proof Sketch. First, we show that for seq with j Messages, the prefix of each infinite trace σ in the set of traces represented by the LTL templates (denoted as $(\Sigma_{LTL}^{seq})^\omega$) must not contain no-op event occurrence τ, providing the length of the prefix is $2j$: □

Lemma 1. *If $\sigma \in (\Sigma_{LTL}^{seq})^\omega$, then σ must have the form, $\sigma = \sigma_{[0..2j-1]} \cdot \tau^\omega$, where $\sigma_{[0..2j-1]}$ contains no τ.*

Next, we show that each element in the set of traces derived from seq (denoted as $(\Sigma_{sem}^{seq})^*$) must map to the prefix of a trace in the set of traces represented by the LTL templates of seq, providing the length of each prefix is $2j$, and vice versa.

4.3 Combined Fragments

A Combined Fragment can modify the sequential execution of its enclosed OSs on each Lifeline. Moreover, a sequence diagram can contain multiple CFs that can be nested within each other. To capture these features, we generalize Π_{seq}^{Basic} to Π_{seq} for expressing sequence diagram with CFs (see Fig. 3). We introduce a new template Φ^{CF} to assert the semantics of each CF directly enclosed in seq. A detailed explanation of Figs. 3, 4, 5, and 6 is provided in [17]. We give an overview below.

Template Φ^{CF} (see Fig. 4) considers three cases. Formula (1) asserts the case that the CF contains no Operand whose Constraint evaluates to $True$. Thus, the OSs within the CF are excluded from the traces. Semantics rule 3 for CFs states weak sequencing among the CF's preceding Interaction Fragments and succeeding ones, which is enforced by formula η^{CF}. Functions $pre(CF\uparrow_i)$ and $post(CF\uparrow_i)$ return the set of OSs which may happen right before and after CEU $CF\uparrow_i$ respectively.

Formula (2) asserts the case that CF contains at least one Operand whose Constraint evaluates to $True$ and CF is not

$$\Psi^{CF}_{alt} = \bigwedge_{m \in TOP(CF)} \Psi^{m}_{alt}$$

$$\Psi^{m}_{alt} = \begin{cases} \bar{\theta}^{CF}_{m} \wedge \bigwedge_{i \in LN(CF)} \bar{\gamma}^{CF}_{i,m} \wedge \bigwedge_{CF_t \in nested(m)} \Phi^{CF_t} & if\ m\ is\ the\ chosen\ Operand \quad (1) \\ True & else \quad (2) \end{cases}$$

$$\bar{\theta}^{CF}_{m} = \bigwedge_{i \in LN(m)} \Big(\bigwedge_{g \in TBEU(m\uparrow_i)} \alpha_g \Big) \wedge \bigwedge_{j \in MSG(TOP(m))} \beta_j$$

$$\bar{\gamma}^{CF}_{i,m} = \bigwedge_{OS \in TOS(m\uparrow_i)} \Big((\neg OS\ \widetilde{\mathcal{U}}\ \big(\bigwedge_{OS_{pre} \in pre(CF\uparrow_i)} (\diamondsuit OS_{pre}) \big) \Big) \wedge \Big(\big(\bigwedge_{OS_{post} \in post(CF\uparrow_i)} (\neg OS_{post}) \big)\ \widetilde{\mathcal{U}}\ (\diamondsuit OS) \Big)$$

Fig. 6. LTL formula for Alternatives.

an Alternatives or a Loop. The first conjunct Ψ^{CF} asserts the OSs directly enclosed in CF. The second conjunct states that the semantics of each CF_i, which is directly enclosed in CF, is enforced by each Φ^{CF_i}. In this way, Φ^{CF} can be defined recursively until it has no nested CFs.

Template Ψ^{CF} captures the semantics that is common to all CFs (except Alternatives and Loop) (see Fig. 5). Sub-formula θ^{CF} states semantic rule 2, which defines the order among OSs directly enclosed in CF. θ^{CF} is a conjunction of α_gs and β_js, which have their usual meanings. Sub-formula γ^{CF}_i enforces semantic rule 1, which defines the sequential execution on every Lifeline i. The first conjunct enforces that the preceding set of OSs must happen before each OS in CF on Lifeline i, and the second conjunct enforces that the succeeding set of OSs must take place afterwards. The semantics specifics for all the different types of CF Operators are defined in [17].

Formula (3) asserts the case for Alternatives or Loop. For Alternatives, Ψ^{CF}_{alt} defines the semantics of OSs and CFs directly enclosed in CF. Ψ^{CF}_{alt} and Φ^{CF_i} for CF_i directly nested in the Alternatives form an indirect recursion (defined in next section and Fig. 6). If seq contains a Loop, the OSs of seq includes OSs in each iteration of the Loop. One way to implement these formulas is provided in [17].

4.3.1 Branching

The Alternatives chooses at most one of its Operands to execute. Each Operand must have an explicit or an implicit or an "else" Constraint. The chosen Operand's Constraint must evaluate to $True$. An implicit Constraint always evaluates to $True$. The "else" Constraint is the negation of the disjunction of all other Constraints in the enclosing Alternatives. If none of the Constraints in the Operand evaluate to $True$, the Alternatives is excluded. The translation of an Alternatives into an LTL formula must enumerate all possible choices of executions in that only OSs of one of the Operands, whose Constraints evaluate to $True$, will happen. LTL formula Ψ^{CF}_{alt} in Fig. 6 defines the semantics of Alternatives, which is a conjunction of Ψ^{m}_{alt}. Each Ψ^{m}_{alt} represents the semantics of Operand m, whose Constraint evaluates to $True$, which is achieved by function $TOP(CF)$.

The semantics of the chosen Operand (if clause) is described by $\bar{\theta}^{CF}_{m}$, $\bar{\gamma}^{CF}_{i,m}$ and Φ^{CF_t}, where $\bar{\theta}^{CF}_{m}$ defines the partial order of OSs within the chosen Operand and Φ^{CF_t} defines the semantics of CFs directly enclosed in the chosen Operand. Functions Ψ^{m}_{alt} and Φ^{CF_t} invoke each other to form indirect recursion. The sub-formula of the unchosen Operand (else clause) returns $True$, i.e., the unchosen Operand does not add any constraint. The Weak Sequencing of the Alternatives is represented by $\bar{\gamma}^{CF}_{i,m}$ instead of γ^{CF}_{i}, which enforces Weak Sequencing between the chosen Operand and the preceding/succeeding OSs of the Alternatives.

One way to implement the chosen Operand (if clause) is using a Boolean variable *exe* for each Operand whose Interaction Constraint evaluates to $True$, where *exe* should satisfy the following assertion.

$$\widehat{\bigvee_{i \in [1..m]}} exe_i \wedge \bigwedge_{i \in [1..m]} (exe_i \rightarrow cond_i).$$

The assertion ensures exactly one Operand whose Constraint evaluates to $True$ is chosen.

Fig. 1 b shows an example of an Alternatives with two mutually exclusive Operands. For the first case of Φ^{CF}, i.e., both $cond1$ and $cond2$ evaluate to $False$, the Alternatives is ignored as asserted by η^{CF}. The second case will not apply to Alternatives CF. For the third case of Φ^{CF}, if $cond1$ evaluates to $True$, Message *4* will occur after Message *3* occurs and Message *5* will not happen. If $cond1$ evaluates to $False$ and $cond2$ evaluates to $True$, only Message *5* happens. These are asserted by $\bar{\theta}^{CF}_{m}$. $\bar{\gamma}^{CF}_{i,m}$ asserts that for each lifeline, the OSs within the Alternatives will occur after the OSs above the Alternatives. Φ^{CF_t} is simply $True$ since there is no nested CF within the Alternatives.

4.3.2 Concurrency

The Parallel represents concurrency among its Operands. The OSs of different Operands within Parallel can be interleaved as long as the ordering imposed by each Operand is preserved,

$$\Psi^{CF}_{par} = \theta^{CF} \wedge \bigwedge_{i \in LN(CF)} \gamma^{CF}_{i}.$$

For the example shown in Fig. 11 in Section 6, in the Parallel CF, the four OSs of Messages *4* and *5* in the lower operand must maintain their order asserted by θ^{CF}, i.e., [$s4$, $r4$, $s5$, $r5$]. So do the two OSs, $s3$ and $r3$, of Message *3* in the upper operand. However, $s3$ and $r3$ ($s3$ must happen before $r3$) can interleave the subtrace of the four OSs, [$s4$, $r4$, $s5$, $r5$], in any way even if they are on the same Lifelines *p1*, and *q*. To name a few, traces [*s3, s4, r4, s5, r5, r3*] and [*s3, r3, s4, r4, s5, r5*] and [*s4, r4, s5, r5, s3, r3*] are all valid traces.

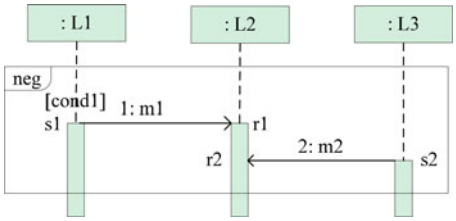

Fig. 7. Example for negative.

4.3.3 Negation

A Negative represents that the set of traces within it are invalid. For example, there are three invalid traces defined by the Negative in Fig. 7, [*s1, s2, r1, r2*], [*s2, s1, r1, r2*], and [*s1, r1, s2, r2*], if $cond1$ evaluates to $True$. Formula $\Psi^{CF}_{neg} = \theta^{CF}$ formally defines the semantics of Negative CF, asserting the order of OSs directly enclosed in it. If the Interaction Constraint of the $Negative$ evaluates to $True$, the traces with the $Negative$ are invalid traces. If the Interaction Constraint of the $Negative$ evaluates to $False$, the traces within the $Negative$ are not invalid. In this paper, we only allow $Negative$ as the outermost CF with no messages following or preceding. This is because a number of alternatives arise in the semantics of such nested CFs [17].

4.3.4 Interaction Use

Interaction Use embeds the content of the referred Interaction into the specified Interaction, thus composing a single, larger Interaction. We consider Interaction Use as a type of CF whose Interaction Operator is *ref*. Formula Ψ^{CF}_{ref} represents the LTL representation of an Interaction Use. In Ψ^{CF}_{ref}, the first conjunct describes that the OSs directly enclosed in the referred sequence diagram obey their order. The second conjunct enforces that the referred sequence diagram and its adjacent OSs are ordered by Weak Sequencing, which is represented by γ^{CF}_i,

$$\Psi^{CF}_{ref} = \theta^{CF} \wedge \bigwedge_{i \in LN(CF)} \gamma^{CF}_i.$$

See [17] for templates for CFs with other Operators.

4.4 Correctness of CFs' Semantics

Here we provide a proof sketch showing that the LTL templates for a sequence diagram with CFs, seq, capture its semantics. A complete proof is provided in [17].

Theorem 2. $(\Sigma^{seq}_{sem})^*$ *and* $PRE_{2h+2p}((\Sigma^{seq}_{LTL})^\omega)$ *are equal.*

Proof Sketch. We assume that seq has $h + p + q$ Messages, where h Messages are directly enclosed in seq. For Messages in CFs, p Messages are enclosed in one or nested Operands where all the Constraints of the Operands evaluate to *True*. For other q Messages, each Message is enclosed in one or nested Operands, where at least one Operand's Constraint evaluates to *False*. □

First, we show that for seq, the prefix of each infinite trace σ in the set of traces represented by the LTL templates (denoted as $(\Sigma^{seq}_{LTL})^\omega$) must not contain no-op event occurrence τ, providing the length of the prefix is $2h + 2p$. Note that any trace of seq does not contain the OSs enclosed in the Operands whose Constraints evaluate to *False*. This is lemma 2 below.

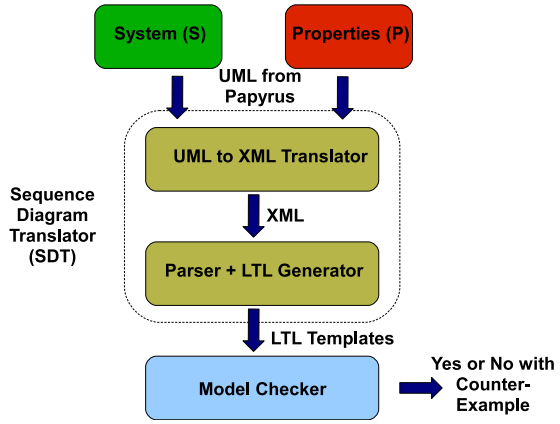

Fig. 8. SDT architecture.

Lemma 2. *If* $\sigma \in (\Sigma^{seq}_{LTL})^\omega$, *then* σ *must have the form,* $\sigma_{[0..2h+2p-1]} \cdot \tau^\omega$, *and* $\sigma_{[0..2h+2p-1]}$ *contains no* τ.

Next, we show that each element in the set of traces derived from seq (denoted as $(\Sigma^{seq}_{sem})^*$) must map to the prefix of a trace in the set of traces represented by the LTL templates of seq, providing the length of each prefix is $2h + 2p$, and vice versa.

5 Tool Suite

Our hospital authorization example presented in Section 6 consists of a simplified version of a hospital information collection form for clarity. More exhaustive sequence diagram representations would consist of a large number of traces which would be much more tedious. For example, when multiple messages are allowed to be interleaved, as in the case of the Parallel CF in Fig. 11, the set of possible traces are less obvious. To address this issue, we have designed a tool suite, sequence diagram translator (SDT), to translate sequence diagrams into LTL formulas which can be used with a model checker to identify all traces and test for security properties (Section 3.2).

First (Fig. 8), SDT takes as input a sequence diagram created in Papyrus UML, a modelling plugin for the Eclipse IDE. The sequence diagram is converted from .uml format into an XML representation (to support manual generation) by the PapyrusToXML component in SDT. Next, a Java-based parser reads the XML file and passes an internal representation to the program's LTL generator, which implements the LTL templates to generate a LTL formula representing the sequence diagram. The LTL generator is designed to take collections of sequence diagrams and generate the corresponding formulas (Section 3.2), and the necessary model to be used in NuSMV [11] model checker. SDT implements all of the necessary template formulas to handle all CFs, nested CFs, and Constraints. SDT source code is available at https://bitbucket.org/rslavin/sequence-diagram-translator.

The running time of SDT is directly related to the number of OSs, CFs, and Lifelines of a sequence diagram. For sequence diagrams with CFs, the worst case resulting formula

size is $8 * NOS + NOP * NOS + 2 * NLN * NCF + LN * NOS + 8 * NCF * NOS$, where NCF is the number of CFs, NOP is the number of Operands, NLN is the number of Lifelines and NOS is the number of OSs. Running time for model checking of sequence diagrams converted to LTL increases exponentially depending on the number of variables in the sequence diagram. The number of NuSMV variables is equal to the total number of OSs, CFs, Lifelines, and Constraints. The time to convert a sequence diagram to LTL using SDT will therefore be negligible compared to the actual model checking time. For the hospital policy example in Section 6, we tested the translation time in SDT and model checking time in NuSMV for the hospital policy example in Section 6. Over 10 runs, SDT averaged 16 ms in translation time and NuSMV averaged 97 minutes. The tests were run on Ubuntu Linux with a 3 GHz Intel Xeon CPU and 32 GB RAM.

6 Case Study

In this section, we evaluate our formal framework and tool suite by modeling and analyzing a collection of HIPAA privacy rules [1] using sequence diagrams. We utilize the tool suite that takes sequence diagrams as input and generates LTL formulas for formal analysis [32]. In particular, we show how to verify consistency and independence properties amongst a collection of HIPAA rules. We also show how to verify if a healthcare provider's information sharing practices conforms with HIPAA rules. Since real data are difficult to obtain, we use a hospital's publicly available information collection and patient authorization form (Fig. 12) as an example.

HIPAA overview. HIPAA provides national standards for insurance portability, fraud enforcement and administrative simplification of the healthcare industry [9]. It regulates the use and transmission of confidential health information, which is referred to as protected health information (PHI) among covered entities. Covered entities are the organizations required to comply with HIPAA, including hospitals, insurance companies, doctors and so on. The organizational policy rules of the covered entities should comply with HIPAA regulations, failure of which may result in severe penalties. For instance, Rite Aid Corporation paid $1 million for violations of the HIPAA Privacy Rule [29]. In another case, a former UCLA Health System employee was sentenced to prison and fined for unauthorized access to organizational electronic health record system [14].

Our work addresses the concerns discussed in Section 1 in the following way. (1) Presenting the HIPAA rules using sequence diagrams is a user-friendly, yet precise, way to understand compliance requirements. In this section, we model a collection of HIPAA policy rules using sequence diagrams. (2) Our tool suite can help system architects to detect the HIPAA violations in the system automatically. For instance, sequence diagrams that reflect the system design can be generated based on system logs such that they can verify that the organizations design can conform to HIPAA relations thus preventing potential financial losses.

6.1 Strategy for Mapping HIPAA Rules

HIPAA Privacy Rule consists of 17 sections, each of which may contain *standards* and *implementation specifications*. A standard is a statement which reflects an organization's intention. Implementation specifications define processes by which the intention is implemented. We model the portion of HIPAA which are related to information sharing. Specifically, we model all transmission-related requirements in 11 of these sections. The remaining rules do not concern information transmission and are static in nature, such as definitions, conditions, and contents. The transmission-related rules can be separated into two groups. One group restricts the use or disclosure of PHI for covered entities, including Sections 164.502, 164.506, 164.508, 164.510, 164.512, and 164.514. The other group restricts the request from the individual to the covered entities, including Sections 164.520, 164.522, 164.524, 164.526, and 164.528.

HIPAA structure. Each section contains multiple paragraphs, which are listed using lower case letters, numbers, roman letters, upper case letters, and italic numbers for different levels. For example, in Section 164.512 (see [1] for a full description of this section), the standard of disclosures about victims of abuse, neglect or domestic violence is labeled using "(c)". This standard consists of two paragraphs, which are labeled as "(1) Permitted disclosures" and "(2) Informing the individual". The details of permitted disclosures are described in three paragraphs, which are labeled using roman letters "(i), (ii), (iii)". Furthermore, two paragraphs from lower level describe the details of paragraph "(iii)", where they are labeled using upper case letters "(A), (B)". To keep the diagrams clear and readable, we model each paragraph listed with a lower case letter, a number, or a roman letter using a sequence diagram. A paragraph is labeled with upper case letters or italic numbers when it is referred by other paragraphs since it mainly represents the purposes or conditions of rules.

Rule types. We categorize the paragraphs in a HIPAA rule into three groups. (1) "At Least One" rules or Permission rules (all system traces each satisfies at least one of the rules): Each paragraph provides a number of possible means to regulate the behavior, e.g., "*Section 164.512(a)(1) a covered entity* **may** *use or disclose protected health information to the extent that such use or disclosure is required by law...*". (2) "All" rules or Mandatory rule (all system traces each satisfies all such rules): Each paragraph provides mandatory means to regulate the behaviors, e.g., "*Section 164.512(a)(2) A covered entity* **must** *meet the requirements described in paragraph (c), (e), or (f) of this section for uses or disclosures required by law.*". (3) Restrictive rules or Prohibition rules (all system traces each is restrictive from violating any of such rule): Each paragraph provides a forbidden means to regulate the behaviors, e.g., "*Section 164.512(j)(2) A use or disclosure pursuant to paragraph (j)(1)(ii)(A) of this section* **may not** *be made if...*". Any of the three types of rules can be combined with exceptions, each of which enumerates the alternative cases of a rule. If system traces satisfy one alternative case, they do not need to satisfy the rule. For instance, "*Section 164.524 (b)(2)(i) Except as provided in paragraph (b)(2)(ii) of this section...*", where *Section 164.524(b)(2)(ii)* is an exception of *Section 164.524(b)(2)(i)*. In addition, a paragraph may refer to others to reuse that clause.

Mapping. We model each paragraph listed with roman letters using a sequence diagram. Particularly, a paragraph expressing a restrictive rule is modeled using a sequence

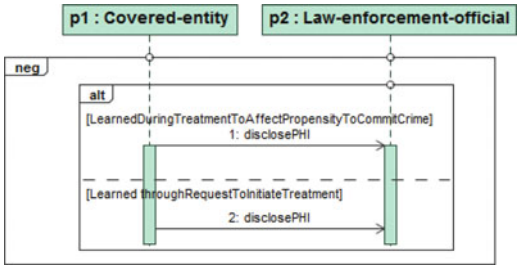

Fig. 9. Paragraph $164.512(j)(2)(i)$.

diagram with a Negative CF. A paragraph with exceptions is modeled using a sequence diagram with an Alternatives CF, where each Operand refers to the sequence diagram of each exception rule. Each paragraph listed with numbers is also modeled using a sequence diagram, which expresses the composition of lower level paragraphs with CFs. Multiple "At Least One" rules can be combined using an Alternatives CF. This procedure is recursively applied until all sections are mapped.

Each sequence diagram expressing a paragraph may have Constraints, which represent the purposes and the predicates. A predicate represents a condition which needs to be evaluated by actors. Actors are objects represented by Lifelines. where actor's role is modeled using the instance's class. For example, a Lifeline's label can be $p1 : coveredEntity$, which represents that the role of actor $p1$ is a covered entity. Roles are hierarchical, i.e., general roles can be subdivided into specific roles. For instance, the most general role in HIPAA is defined as "HIPAA-role" which can be described more specifically as a covered entity. In turn, a health plan or health provider can be a covered entity. In this way, a role can be replaced as a sub-role during verification. Messages transmitted among actors is modeled using asynchronous Message transmission, where the Message name indicates the information it carries. For policy rules with time constraints, we add a "Timer" Lifeline.

6.2 Sequence Diagrams for HIPAA

We illustrate our mapping strategies using two examples for safety and liveness properties separately. More examples are demonstrated in [17]. Section 164.512(j)(2) defines a safety property by expressing forbidden behaviors. It regulates that "A use or disclosure pursuant to paragraph (j)(1)(ii)(A) of this section *may not* be made if the information described in paragraph (j)(1)(ii)(A) of this section is learned by the covered entity: (i) in the course of treatment to affect the propensity to commit the criminal conduct that is the basis for the disclosure under paragraph (j)(1)(ii)(A) of this section, or counseling or therapy; or (ii) through a request by the individual to initiate or to be referred for the treatment, counseling, or therapy described in paragraph (j)(2)(i)." We model this paragraph using a sequence diagram with a Negative CF (see Fig. 9). The Negative CF describes that if the condition is satisfied, the scenarios expressed by either case are invalid. Paragraph Section $164.512(j)(2)$ is represented as nested combined fragment in Fig. 9.

Section 164.524(b)(2)(i) expresses a liveness property, which regulates that "(i) Except as provided in paragraph (b)(2)(ii) of this section, the covered entity must act on a request for access no later than 30 days after receipt of the request as follows. (A) If the covered entity grants the request, in whole or in part, it must inform the individual of the acceptance of the request and provide the access requested, in accordance with paragraph (c) of this section. (B) If the covered entity denies the request, in whole or in part, it must provide the individual with a written denial, in accordance with paragraph (d) of this section." This paragraph expresses a rule with exception. We model the policy rule and its exception using the Operands of an Alternatives CF (see second Alternatives CF in Fig. 10). The first Operand describes the exception case described paragraph 164.524(b)(2)(ii) regarding PHI not being available on-site. The second Operand describes the response for an access request in any other case. Two possible responses are combined using an Alternatives CF. A "timer" Lifeline is introduced to track the time interval. The timer is set when the request is sent, and it will notify the individual when the specified amount of time has elapsed.

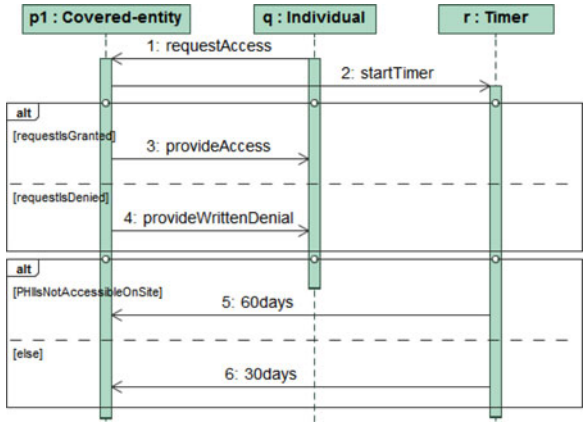

Fig. 10. Paragraph $164.524(b)(2)(i) - (ii)$.

6.3 HIPAA Verification

We have modeled more than one hundred HIPAA policy rules using sequence diagrams. Yet, it is hard to check manually that if the policy rules contradict each other (consistency), or if some policy rules are redundant (independence). With the help of our tool suite, we can analyze sequence diagram based policy rules automatically. However, due to the state explosion problem, it is difficult to check all the policy rules at the same time. To simplify the verification process, we group the policy rules into sets of related rules and verify each set independently. We define a set of related policy rules to (1) share the same purpose; and (2) at least two actors with the same or hierarchically related roles, e.g., "HIPAA-role" and "law-enforcement-official" are hierarchically related because "HIPAA-role" can be replaced using "law-enforcement-official" in specific rules.

Consistency checking. For each set of rules, we build a free model consisting of all variables used in the LTL formulas without defining their transition relations. Then we check the free model against the negation of conjunction of the LTL formulas. If no counter-examples are generated (i.e., there are no property violations), the policy rules are inconsistent. Otherwise, the policy rules are consistent and a counterexample which satisfies all the LTL formulas (i.e., all the HIPAA privacy rules belonging to the same purpose) is provided by the model checker. As a

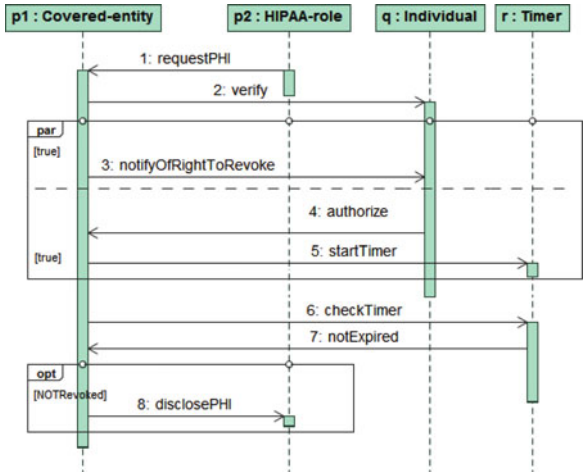

Fig. 11. Hospital information system.

proof-of-concept, we tested the consistency of rules for three purposes with 6, 5 and 8 rules respectively and they all held true. These rules are large with multiple nested paragraphs. A major verification across entire HIPAA is research endeavor in its own right.

Independence checking. We build the free model for each set of rules and check it against the independence properties. We tested independence for three purposes with 6, 5 and 8 rules respectively and they held true.

Conformance checking. We can use LTL models generated with LTL templates to verify that a system conform to, or satisfies, an existing policy or regulation. To exemplify this, we have adapted a model from forms that is common in many hospitals regarding authorization for release of PHI. Fig. 11 models the possible traces regarding authorization of PHI disclosure present in our modified version of such forms. The parallel combined fragment allows for Message 3, the notification to the individual by the covered entity of their right to revoke authorization, to occur at any time between Message 4 and 5. This means Message 3 can occur before Message 4, between Messages 4 and 5, or after Message 5. Message 8, the disclosure of PHI to the HIPAA role can only occur if authorization has not been revoked by the individual.

Section 164.508(c)(2)(i) of HIPAA describes the requirement for authorization forms to provide notice to the individual of their right to revoke authorization for disclosure of PHI. Furthermore, Section 164.508(b)(2)(i) describes the invalidity of authorizations after the expiration date as required by Section 164.508(c)(1)(v). Fig. 12 models the acceptable traces for disclosure of PHI with regard to these HIPAA regulations. The alternative combined fragment allows for two possibilities in response to the request for PHI from a HIPAA role: 1) if a previous authorization has not yet passed the expiration date and the authorization has not been revoked by the individual, Message 2, disclosure of the requested PHI can occur. 2) in any other case Messages 3, 4, and 5, verification of authorization, notification of an individual's right to revoke authorization, and authorization by the individual must occur in that exact order before disclosure of PHI to the HIPAA-role can occur. Note that Messages 2 and 6 are the same with respect to possible traces.

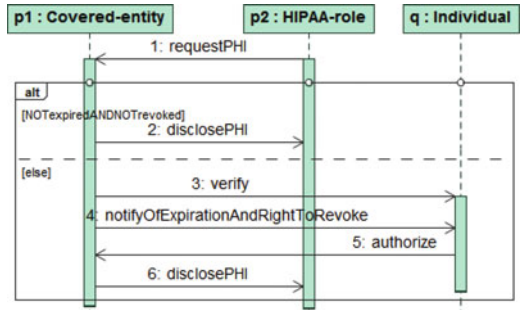

Fig. 12. Paragraphs Sections 164.508(c)(2)(i) and 164.508(b)(2)(i).

To check for conformity of the hospital policy in Fig. 11 with the HIPAA rules in Fig. 12, we must show that Fig. 12's traces contain all of the traces defined by Fig. 11 ignoring the extra OSs defined in 11 not appearing in 12 (e.g., startTimer and CheckTimer). Such OSs that are not related to the HIPAA policy are not considered as per the partial trace semantics described in Section 3. A free model defining all of the OSs in both sequence diagrams must be generated by defining Boolean variables for each OS. Then, conformance can be validated by checking if the LTL formula describing Fig. 11 implies the LTL formula describing Fig. 12. If the resulting formula returns true, this implies that all possible traces in the hospital policy (excluding irrelevant OSs) are a subset of the possible traces in the HIPAA rule, i.e., the hospital policy conforms to the HIPAA rule.

For our example, the fictional hospital policy would not conform to the HIPAA rule due to the possibility of the notification of an individual's right to revoke authorization occurring after authorization. This is apparent in Fig. 11 due to the parallel combined fragment which allows Message 3 to be interleaved in any order within the second operand of the combined fragment. This represents a single possible trace which can violate the HIPAA rule which outlines a specific ordering of the Messages in the second operand of the alternative combined fragment in Fig. 12, which the hospital policy fulfills due to its constraints.

This example shows how a PHI authorization form can be checked for conformance with two HIPAA regulations. Because the form also regards the possibility of disclosure of PHI for marketing purposes, it could also be checked against HIPAA Section 164.508(a)(3) as seen in Fig. 13, after being modified to include any relevant traces. The regulation states that "...a covered entity must obtain an authorization for any use or disclosure of protected health information for marketing, except if the communication is in the form of (A) a face-to-face communication made by a covered entity to an individual; or (B) a promotional gift of nominal value provided by a covered entity." Furthermore, "if the marketing involves direct or indirect remuneration to the covered entity from a third party, the authorization must state that such remuneration is involved."

Similar instances can be checked to validate conformance to other privacy policy regulations such as the Gramm-Leach-Bliley Act (GLBA) regulating financial institutions. For example, Section 502 of GLBA describes regulations for disclosure of personal information. Privacy policies for financial institutions can be modelled and checked for conformance to GLBA in same way as

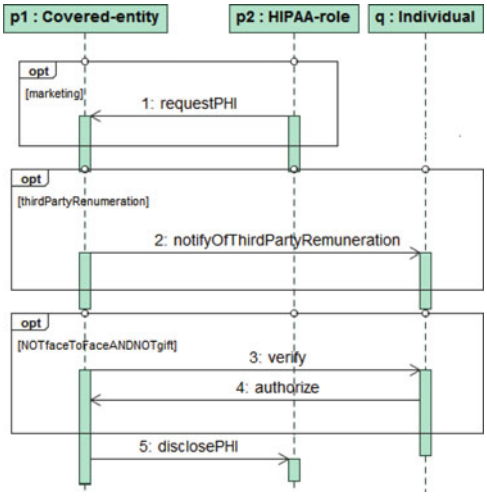

Fig. 13. Paragraph $164.508(a)(3)$.

we have modelled hospital policies and checked for conformance to HIPAA. This method provides a means to validate whether a system's design satisfies existing security policies for any relevant set of traces that can be modelled as a sequence diagram.

7 Related Work

Sequence diagram trace theory. Sequence diagrams are sufficiently expressive and they have been used to specify a variety of systems or policies (see for example [25], [32], [35]). Also, there has been prior work on formalizing sequence diagrams. In particular, Solhaug et al. [34], [35], extend UML sequence diagrams using Deontic STAIRS constructs such as obligation and permission to specify access control policies. STAIRS [21] provides a denotational trace semantics for the main constructs of interactions in UML 2 while Deontic logic enables obligations and permissions to be specified on the trace semantics. They formalize the trace theory of Deontic STAIRS that allows policy specifications to be refined incrementally. Since they build on STAIRS, their approach requires that policy satisfaction of a system be characterized at the level of traces using composition operators such as parallel, sequence and alternative. Such an approach may not be directly conducive for automated verification using widely-adopted model checking tools such as NuSMV or Cadence. The significance of our work is thus to formalize sequence diagrams in a comprehensive manner using LTL, and to demonstrate its sound application toward formal specification and analysis of regulatory privacy policies such as HIPAA, and verification of whether a healthcare information system satisfies HIPAA rules. In particular, we choose sequence diagrams due to its intuitive appeal and the ability to precisely codify them. We envision that such an intuitive approach is highly usable in practice.

Sequence diagram formalization. Formalization of sequence diagrams has received significant attention in the research community [15], [27], [36]. However, as discussed in Section 1, they do not satisfy all of our criteria. We discuss a few relevant works below. Working towards similar goals, Kugler et al. [22] and Kumar et al. [23] have presented translations from a live sequence chart (LSC) to temporal logic formulas, which support concurrency, iteration, condition, and both synchronous and asynchronous Messages. LSC extends message sequence chart (MSC) using universal charts and existential charts. They expressed universal charts using both LTL and Computation tree logic (CTL), and expressed existential charts using CTL since existential charts only consider possible behaviors. Their approach formalizes global behaviors using synchronous Messages, while our work focuses on the execution of events on each Lifeline and supports both asynchronous and synchronous Messages.

Harel and Maoz [20] propose a modal sequence diagram (MSD) to specify semantics of Negation and Assertion Operators, providing an avenue for us to define liveness and safety properties. To specify and formalize temporal logic properties, Autili et al. [7] propose the property sequence chart (PSC), which is an extension of UML 2 sequence diagrams. Their approach eases software engineers' efforts for defining properties. Our method can be adapted for PSC to support a larger set of properties. By providing translation rules for all the aforementioned constructs into formal notations in a single framework, we form a basis for tool builders to create scenario-based tools for verifying information systems against security policies by leveraging model checking techniques.

HIPAA verification approaches. Several frameworks have been proposed for specifying and analyzing privacy rules [8], [10], [12], [13], [18], [24], [26], [30]. As mentioned in Section 1, these works translate HIPAA rules into formal frameworks directly. Our work is complementary to many of these works. Our approach helps engineers to understand complex security policies such as the HIPAA Privacy Rules by visually representing them, yet formal enough to be automatically verified. Furthermore, our tool suite can also check if the organizational policy rules designed by the engineers comply with rules such as HIPAA.

8 Conclusion

We demonstrated that sequence diagrams can be utilized to specify the system design of an organization and also its security policies. We showed how to verify conformance, consistency, and independence in the concrete application domain of HIPAA. In the future, we plan to apply our work to other policies such as GLBA [3], FERPA [4] and COPPA [2].

References

[1] PUBLIC LAW 104-191, "Health insurance portability and accountability act of 1996," in *Proc. 104th Congress*, 1996, pp. 743–785.
[2] PUBLIC LAW 105-277, "Children's online privacy protection act of 1998," in *Proc. 105th Congress*, 1998, pp. 728–742.
[3] PUBLIC LAW 106-102, "The gramm-leach-bliley act of 1999," in *Proc. 106th Congress*, 1999, pp. 1436–1450.
[4] PUBLIC LAW 93-380, "The family educational rights and privacy act of 1974," in *Proc. 93rd Congress* 1974, pp. 556–576.
[5] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distrib. Comput.*, vol. 2, pp. 117–126, 1986.
[6] B. Alpern and F. B. Schneider, "Defining liveness," *Inform. Process. Lett.*, vol. 21, no. 4, pp. 181–185, 1985.
[7] M. Autili, P. Inverardi, and P. Pelliccione, "Graphical scenarios for specifying temporal properties: An automated approach," *Autom. Softw. Eng'.*, vol. 14, no. 3, pp. 293–340, 2007.

[8] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum, "Privacy and contextual integrity: Framework and applications," in *Proc. IEEE Symp. Secur. Privacy*, 2006, pp. 184–198.
[9] K. Beaver and R. Herold, *The Practical Guide to HIPAA Privacy and Security Compliance*. New York, NY, USA: Auerbach, 2003.
[10] T. Breaux and A. Antón, "Analyzing regulatory rules for privacy and security requirements," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 5–20, Jan. 2008.
[11] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, "NuSMV: A new symbolic model checker," *Int. J. Softw. Tools Technol. Transfer*, vol. 2, pp. 410–425, 2000.
[12] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The PONDER policy specification language," in *Proc. Int. Workshop Policies Distrib. Syst. Netw.*, 2001, pp. 18–38.
[13] H. DeYoung, D. Garg, D. Kaynar, and A. Datta, "Logical specification of the GLBA and HIPAA privacy laws," CMU, Pittsburgh, PA, USA, Tech. Rep. CMU-CyLab-10-007, 2010.
[14] C. Dimick. Californian sentenced to prison for HIPAA violation. [Online]. Available: http://journal.ahima.org/2010/04/29/californian-sentenced-to-prison-for-hipa a-violation, Apr. 2010.
[15] C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, and C. Stehno, "Compositional semantics for UML 2.0 Sequence diagram using Petri Nets," in *Proc. 12th Int. Conf. Model Driven*, 2005, pp. 133–148.
[16] D. Remenska J. A. Templon, T. A. C. Willemse, P. Homburg, K. Verstoep, A. Casajus, and H. E. Bal, "From UML to process algebra and back: An automated approach to model-checking software design artifacts of concurrent systems," in *Proc. 5th Int. Symp. NASA Formal Methods*, 2013, pp. 244–260.
[17] H. Shen, R. Krishnan, and J. Niu, "Sequence diagram aided security policy specification," UTSA, San Antonio, TX, USA, Tech. Rep. CS-TR-2014-005, 2014.
[18] D. Garg, L. Jia, and A. Datta, "Policy auditing over incomplete logs: Theory, implementation and applications," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 151–162.
[19] R. Grosu and S. A. Smolka, "Safety-liveness semantics for UML 2.0 Sequence diagrams," in *Proc. Int. Conf. Appl. Concurrency Syst. Des.*, 2005, pp. 6–14.
[20] D. Harel and S. Maoz, "Assert and negate revisited: Modal semantics for UML sequence diagrams," *Softw. Syst. Model.*, vol. 7, no. 2, pp. 237–252, 2008.
[21] O. Haugen, K. E. Husa, R. K. Runde, and K. Stolen, "STAIRS towards formal design with sequence diagrams," *Softw. Syst. Model.*, vol. 4, no. 4, pp. 355–357, 2005.
[22] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps, "Temporal logic for scenario-based specifications," in *Proc. 11th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2005, pp. 445–460.
[23] R. Kumar, E. G. Mercer, and A. Bunker, "Improving translation of live sequence charts to temporal logic," *Electron. Notes Theoretical Comput. Sci.*, vol. 250, no. 1, pp. 137–152, 2009.
[24] P. E. Lam, J. C. Mitchell, and S. Sundaram, "A formalization of HIPAA for a medical messaging system," in *Proc. 6th Int. Conf. Trust, Privacy Secur. Digit. Bus.*, 2009, pp. 73–85.
[25] V. S. W. Lam and J. Padget, "Consistency checking of sequence diagrams and statechart diagrams using the π-calculus," in *Proc. 5th Int. Conf. Integrated Formal Methods*, 2005, pp. 347–365.
[26] M. J. May, C. A. Gunter, and I. Lee, "Privacy APIs: Access control techniques to analyze and verify legal privacy policies," in *Proc. 19th IEEE Workshop Comput. Secur. Found.*, 2006, pp. 85–97.
[27] Z. Micskei and H. Waeselynck, "The many meanings of UML 2 Sequence Diagrams: A survey," *Softw. Syst. Model.*, vol. 10, no. 4, pp. 489–514, 2011.
[28] Object management group. (2010). Unified modelling language (Superstructure), v2.3. [Online]. Available: www.omg.org
[29] HHS Press Office. Rite aid agrees to pay $1 million to settle HIPAA privacy case. [Online]. Available: http://www.hhs.gov/news/press/2010pres/07/20100727a.html, Jul. 2010.
[30] IBM research. (Nov., 2003). Enterprise privacy authorization language (EPAL) version 1.2. [Online]. Available: http://www.zurich.ibm.com/pri/projects/epal.html
[31] F. B. Schneider. (2000, Feb.). Enforceable security policies. *ACM Trans. Inform. Syst. Secur.* [Online]. *3(1)*, pp. 30–50. Available: http://doi.acm.org/10.1145/353323.353382
[32] H. Shen, M. Robinson, and J. Niu, "Formal analysis of sequence diagram with combined fragments," in *Proc. Int. Conf. Softw. Paradigm Trends*, Jul. 2012, pp. 44–54.
[33] M. Sloman, "Policy driven management for distributed systems," *J. Netw. Syst. Manage.*, vol. 2, no. 4, pp. 333–360, 1994.
[34] B. Solhaug, D. Elgesem, and K. Stolen, "Specifying policies using UML sequence diagrams–an evaluation based on a case study," in *Proc. Int. Workshop Policies Distrib. Syst. Netw.*, 2007, pp. 19–28.
[35] B. Solhaug and K. Stolen, "Compositional refinement of policies in UML–exemplified for access control," in *Proc. 13th Eur. Symp. Res. Comput. Secur.: Comput. Secur.*, 2008, pp. 300–316.
[36] J. Whittle, "Precise specification of use case scenarios," in *Proc. 10th Int. Conf. Fundam. Approaches Softw. Eng.*, 2007, pp. 170–184.

Hui Shen received the BE degree in software engineering from the Beijing Institute of Technology, Beijing, China, and the MS and PhD degrees in computer science from the University of Texas at San Antonio. Her research interests include requirements engineering and formal methods.

Ram Krishnan is an assistant professor of electrical and computer engineering at the University of Texas at San Antonio. His current research interests include access control, security policy analysis, and its applications to cloud computing.

Rocky Slavin received the BS degree in computer science from the University of Texas at San Antonio. He is currently working towards the PhD degree in computer science at the University of Texas at San Antonio with research focusing on software engineering, cyber security, and formal methods.

Jianwei Niu received the PhD degree in computer science from the University of Waterloo in 2005. She is an associate professor of computer science at the University of Texas at San Antonio. Her research interests include formal methods, software engineering, and cyber security.