Implementing Security Requirements through Automatic Generation of Secure Workflows

Ibrahim Jaouhar

A Thesis submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements for the
Degree of Master of Arts

Graduate Program in Information Systems and
Technology
York University
Toronto, Ontario
March 2022

Abstract

Modern software-intensive information systems are enormously large and complex. Prior to the design process of such systems, designers and architects need to know what kinds of stakeholder needs the system is supposed to support. This is particularly true for security requirements which must be captured and analyzed alongside all other requirements rather than treated as an afterthought. Hence, many researchers have proposed different modelling frameworks in different domain fields to address security and privacy patterns. However, most of these frameworks focus on comprehensive representation and analysis of requirements, without indicating how such requirements can be implemented within the context of a business process. Users are often at loss with regards to what security technologies they should adopt and incorporate in their workflows to reach secure business processes. In this thesis, we propose a framework for enriching goal-oriented requirements models with security controls necessitated by specified security requirements. A set of patterns are designed by security experts that associate abstract domain-independent user goals/tasks with alternative workflows that achieve those goals with various levels of security. Such translation of information is performed with the aid of an AI planner, SHOP2. Consequently, system analysts with no deep experience in security technologies can acquire a view of what steps and technologies are involved in making their designs more secure and implement accordingly.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In our modern society, the digital transformation in businesses and services introduced multiple security and privacy concerns. To address these concerns, systems analysts and designers must be able to understand what security and privacy requirements stakeholders have and introduce security controls that address these requirements. However, discovering security and privacy requirements is rather difficult and requires thorough analysis of the users and stakeholders that are involved in the system. The process of identifying and documenting such needs is referred to as security requirements engineering.

Initially, the inclusion of security requirements came after the definition of the system, leading to serious security implications and vulnerabilities where possible contradiction between functional requirements and security could arise. This problem stems from considering non-functional requirements (e.g. security and reliability) as qualities that are added to the system rather than essential mandatory requirements whereby their inclusion in the system can be ignored or addressed at a later stage [1].

The main reason why non-functional requirements lack the necessary attention is because they are hard to identify and analyze, and professionals lack the necessary expertise to address these concerns [1]. Soon enough, professionals understood the importance of non-functional requirements, and specifically security, due to the overwhelming breaches in the cyberspace that is causing business losses in billions of dollars.

To address such concerns and to avoid major complexities, structured methodologies for agent-oriented systems were developed such as Secure Tropos [2], [3]. These methodologies aim to identify and analyze security and privacy concerns throughout the development stages of systems by extending established agent-oriented approaches for requirements-driven software development such as the i^* framework [4] and Tropos [5], [6].

However, capturing security requirements alone does not guarantee the solution for such problems. Rather, a design phase follows in which security requirements are implemented by security controls, i.e. security software and tasks that are aimed at ensuring the end system and its use will fulfill the said security requirements. Oftentimes, however, such design decisions require expertise that the stakeholders do not have access to, or may not be able to afford. This is particularly the case for small organizations and individuals, who simply want secure ways to perform simple transactions and information exchange tasks, but do not know exactly how and with what tools. Several open source software are available in the market in which some do not require professional guidance (e.g. antivirus programs) while others require expert guidance (e.g. cryptographic programs), where users might feel lost or fail to understand and address the desired solution correctly if left unguided.

Consider the example of a startup translation company that employs a few

number of translators and a secretary to manage their operations. The company may receive files, documents, or information that might be sensitive to the customer and require confidentiality. Since the company is a small-sized company, the acquisition of an ERP system might be difficult, which leaves email addresses as the most convenient way of communication. As a result, business owners realize the need of information secrecy between the business and it's customers due to possible threats that could lead to the leak of the data being transferred that could breach customer's confidentiality. At the same time, if secretaries do not have access to the company's account and relies on customer invoices to forward the tasks to the translators, potential fraud attempts (e.g. sending tampered invoices) could possibly exist where payment information has been altered. In the case of the medium/large companies where the acquisition of enterprise systems is possible, a thorough analysis would have been made by experts for these systems themselves to guarantee the security of business processes. Smaller companies and individual professionals and their customers may not have access to such expertise and expensive tools.

Nevertheless, off-the-shelf and open source encryption software can assist users in providing a certain of level of protection provided the correct guidance on their usage. Even common word and document processing software or email software contain security functions that are readily usable. For instance, using encryption to preserve confidentiality may prevent adversaries from reading or tampering sensitive data. However, encryption itself contains a series of activities that must be taken depending on the type of encryption being used to achieve the desired level of security.

In this thesis, we propose a framework for automatically enriching goal-oriented requirements models with security controls necessitated by specified security requirements. The social setting of an environment is mod-

elled and analyzed with STS-ml using the STS-Tool [7]. This tool allows the analysis of stakeholders and the system-to-be while highlighting relevant goals and document processing. This tool also allows analysts to attach desired security requirements for such goals and during information exchange such as confidentiality, integrity, and non-repudiation.

In parallel to identifying and modelling domain-specific security requirements, a set of security patterns is developed that is applicable to a wide variety of domains. These security patterns contain workflows that describe the necessary activities that must be taken to attain a security objective that in turn renders a business process secure. Security patterns are used to address security vulnerabilities in a system. Such vulnerabilities are identified and mapped through attack trees that represent a breach in security requirement, whereby this information is used to consider which actions users need to take in a particular domain in their daily activities to avoid such breaches.

With the security requirements, the security workflow patterns, and the attack trees at hand, an AI planner is adopted to compile this information and generate workflows for fulfilling business requirements while also meeting the security ones by protecting against attacks that are relevant to the situation at hand. In this paper, we use HTN (Hierarchical Task Network) [8] planning that combines the aforementioned information and produces a sequence of actions that explains the steps that need to be taken to achieve a primary activity.

The workflows generated by the planner are validated in two parts. The first part covers the validation of individual security patterns, and the second part checks their integration with businesses processes, while adding necessary vulnerability assumptions, to verify that security requirements

are addressed with the correct security controls, and that security breaches are being prevented as intended.

Since one of the goals of this framework is to be usable by the user of the reasoner, it has to be carefully evaluated in terms of modelling and development time, plan generation, processing time, and time to fix unexpected errors. This framework provides several contributions listed as follows:

- We extend a state-of-the-art in security requirements specification framework, STS-ml with the ability to reason about security designs, rather than just requirements.
- We present a new method to model security workflow patterns through goal decomposition trees.
- We introduce an approach for reasoning against threats through the use of attack trees as task decomposition to be disabled through security controls.
- We present an approach to translate the goals of the stakeholders and the system-to-be to an AI planner that includes relevant security requirements, vulnerability assumptions, and user/system non-functional requirements.
- The framework can be used by non-technical users or developers with no experience on how security breaches can be prevented according to desired security requirements, in order to secure their business processes regardless of the domain being presented.

This thesis is organized as follows. We first present an overview of relevant background in Chapter 2. Our discussion includes goal modelling methodologies, security patterns, attack trees, and automated reasoning techniques

which are needed to understand our framework. A detailed explanation of the framework and its translation and reasoning into the AI planner are presented in chapter 3. We then apply and present the application of this framework in multiple domains in chapter 4. In chapter 5, the validation of the plans generated by the planner and an evaluation of the framework is explained. Lastly, we conclude and provide potential future research and improvement in chapter 6.

Chapter 2

Background and Related Work

2.1 Overview

Conceptual modelling has long been understood to be an essential tool for system designers and analysts to understand business problems and their underlying complexities, so to ultimately find solutions that conveniently address them.

In this chapter, we describe major methodologies relative to this thesis to explain and elaborate the frameworks that include goal-modelling techniques and security requirement analysis. Such frameworks include the Secure Tropos methodology, which was influenced by by other goal-modelling techniques such as KAOS, SQAURE, SREP, and MOSRE. Additionally, we discuss several frameworks that address security patterns, and attack trees, which represent a breach in security requirements from an attacker's perspective. Lastly, we discuss frameworks proposed for performing an automated reasoning with goal-models, security patterns, and attack trees.

2.2 The i* framework

One of the early methodologies to acknowledge the need for identifying, describing, and modelling processes and the rationale behind them is the i^* framework [4]. The aim of the i^* framework is to identify and model relevant processes in organizational settings through representational constructs of concepts that are then used to gain some analytical capabilities. The framework consists of two models: a Strategic Dependency (SD) model, and a Strategic Rational (SR) model. Both models are represented using the i^* modelling elements presented in Figure 2.1

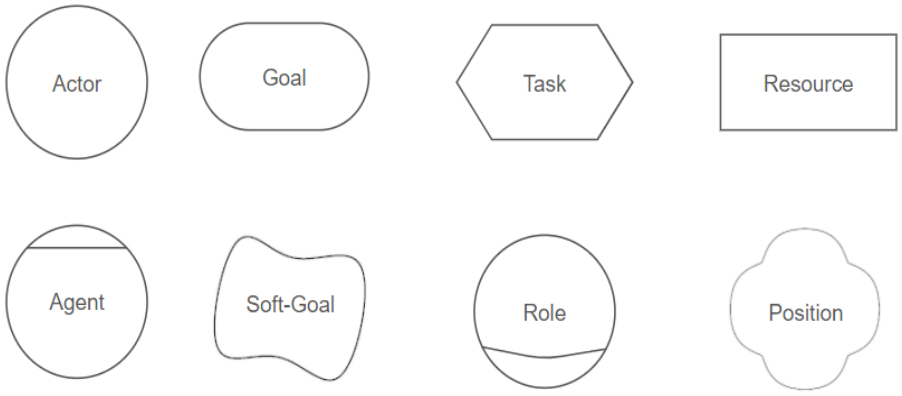

Figure 2.1: i^* conceptual elements

An actor is an entity that has strategic goals that needs to be fulfilled. An actor can be an agent (e.g. Bob), or a role (e.g. physician). A position represents a set of roles. A goal is a desired state that an actor would like to reach, while a soft-goal usually describes non-functional requirements, such as fast performance, which do not have a clear satisfaction criteria. A task is an endeavor to achieve a goal, while a resource is an asset that an actor requires to achieve a certain goal.

The purpose of the SD model is to model relational dependencies among actors to capture the intentional structure of a process, leaving non-essential

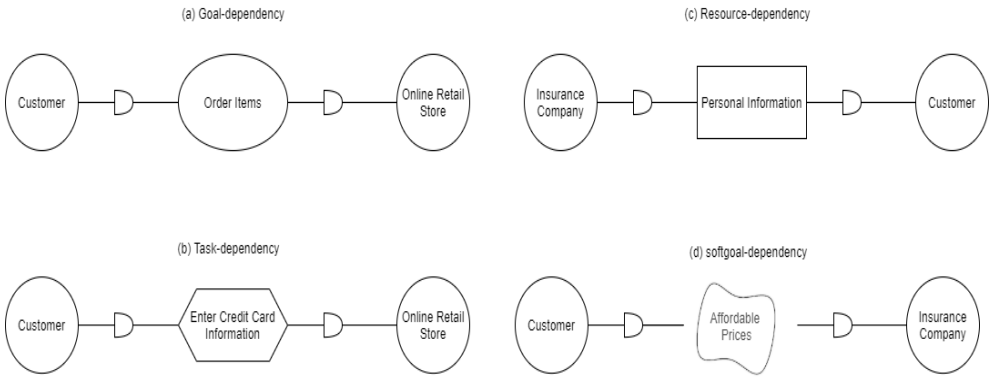

Figure 2.2: i^* dependency types

details. A dependency is created between a depender that relies on a dependee to achieve a certain objective (dependum). The outcome of this model is the identification of stakeholders and the processes that binds them together.

The purpose of the second model, which is the SR model, is to describe the processes and the rationales behind them. It expands the abstraction in the SD model by describing processes in terms of means-ends and task-decomposition links. A means-ends is the relationship to a goal, task, resource, or soft-goal that needs to be satisfied and the way to attain it, while a task-decomposition is the process of decomposing one task to further describe the means to attain that task, either through another task, goal, resource, or soft-goal.

The i^* framework distinguishes between four different types of dependencies based on the type of the dependum. The dependency types are presented in Figure 2.2

- Goal-dependency: The depender depends on the dependee to achieve a specific goal. The freedom of choice to achieve that goal is left to the dependee.

- Task-dependency: The depender depends on the dependee to achieve

a specific task. A task-dependency has a process that needs to be followed to achieve that task unlike a goal that can be attained through various methods depending on the dependee's preference.

- Resource-dependency: The depender depends on the dependee to achieve a specific resource. In other words, the availability of a resource.

- Softgoal-dependency: The depender depends on the dependee to perform a task that satisfies a soft-goal.

The i^* framework is based on the elements of goals, tasks, resources, AND/OR decomposition links, and positive/negative contribution links. An AND decomposition link denotes that all the sub-goals of a goal must be completed in order for the primary goal to be considered achieved, while an OR decomposition link requires only one sub-goal to be completed to achieve the primary one. Positive/negative contribution links can be broken as follows:

- Positive Contribution links:
  - Make: indicates a sufficient positive impact from the source to the target, implicitly denoted by the "++" sign to indicate a strong positive contribution.
  - Help: indicates a weak positive impact from the source to the target, implicitly denoted by the "+" sign to indicate a weak positive contribution.

- Negative Contribution links:
  - Hurt: indicates a weak negative impact from the source to the

target, implicitly denoted by the "-" sign to indicate a weak negative contribution.

– Break: indicates a sufficient negative impact from the source to the target, implicitly denoted by the "--" sign to indicate a strong negative contribution.

The original i^* has been refined and extended with i^* 2.0 [9]. An example using the above elements is shown in Figure 2.3 using the piStar tool [10], where a customer (role) wants to order and item (goal). Ordering an item can be done in-store or online. Ordering items online require customers to register online (Task) and to pay using a credit card (Task). Ordering items online is much more convenient for the customer than buying it in-store.

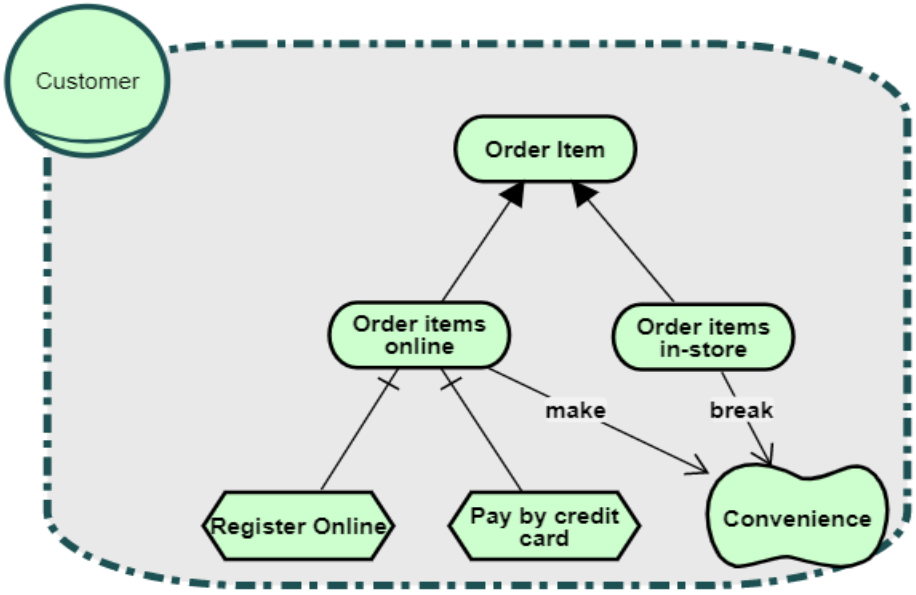

Figure 2.3: An example of i^* 2.0 language using the piStar tool

2.3 Tropos

The i^* family of languages offers a representational framework for agent oriented requirements, but does not necessarily tell us how we can fit such

representations in the software development life-cycle. Thus, various researchers aimed at introducing systems development methodologies that are based on i^*, such as Tropos and Secure Tropos. The former is a methodology that intends to support requirements analysis and design throughout the software development process in agent-orient programs, while the latter extends the Tropos methodology by incorporating security concerns throughout the system design and implementation.

Before we introduce Secure Tropos it is worth to first briefly explore its antecedent, Tropos. The Tropos methodology [5], [6] was introduced to complement proposals for agent-oriented programming environments [11], [12], and to support requirements analysis and design during all the phases of software development processes, from application domain down to system implementation [6]. This methodology is an iterative process that allows designers to not only capture critical requirements but also to redefine them as needed. This methodology covers five main development stages, which are:

- Early Requirements: The output of this phase is the organizational model, where existing organizational settings are studied and mapped to the relevant actors with their dependencies.
- Late Requirements: Where the system-to-be is designed along with its relevant functions and qualities. This phase introduces additional actors, goals, and non-functional requirements to the system.
- Architectural design: Where, as the name states, the system's global architecture is defined by the inclusion of additional actors and the decomposition of their goals and sub-goals.
- Detailed design: This phase deals with communication between agents

and their goals, beliefs, and capabilities. Practical approaches to implement the actual development are proposed and discussed using UML and AUML.

- Implementation: The implementation phase includes the implementation of the agreed approaches in the detailed design.

2.4 Secure Tropos

Although the Tropos methodology succeeds in the requirement analysis throughout the development stages, it does not explicitly integrate security concerns. Different security requirements exist across different domains such as e-commerce, health care, and cloud computing, among others. Consequently, Secure Tropos [2], [3] was introduced as a specialization to the Tropos methodology to consider not only security issues but privacy ones as well.

Secure Tropos [3] extends the Tropos methodology through the integration of security constraints that is used starting from the early requirements phase and security related concepts such as threats (e.g. Trojan horse, eavesdropping), security mechanisms (e.g. encryption), protection objective (e.g. cryptography), and security feature (e.g. privacy) that are used to make the security diagram in the late requirements phase. The mentioned elements are presented in Figure 2.4

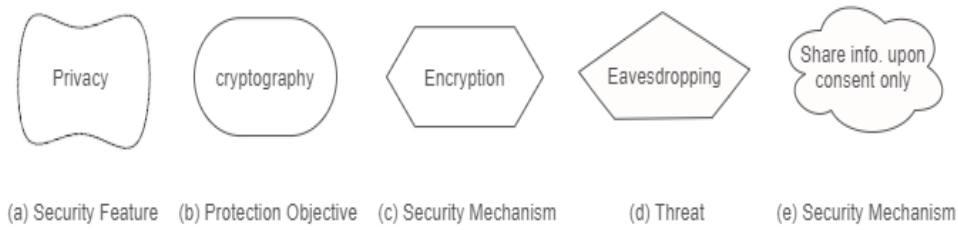

Figure 2.4: Secure Tropos Security Concepts

A security constraint is introduced by a secure dependency that needs to be fulfilled by the depender and dependee. Both actors must satisfy the security constraint in order for the secure dependency to be valid. The depender expects the dependee to satisfy a security constraint, and the dependee must agree and make an effort to comply with the secure dependency [3].

The framework distinguishes between three different types of secure dependencies; a dependee secure dependency, where the dependee introduces a security constraint that the depender has to comply with. A depender secure dependency, where the depender introduces a security constraint that the depender has to comply with. A double secure dependency, where both actors introduce a security constraint against each other. Both parties must satisfy the security constraints, otherwise, the secure dependency is deemed unsuccessful. An actor diagram using the Secure Tropos elements during early requirements is depicted in Figure 2.5.

In the late requirements phase, the system is introduced as a new actor where its functional and non-functional requirements are described taking into account the security diagram that is integrated with the security constraints. Two types of links connect the elements in this diagram. A positive contribution link that affect the element positively, and a negative contribution that affects the element negatively. An example of the

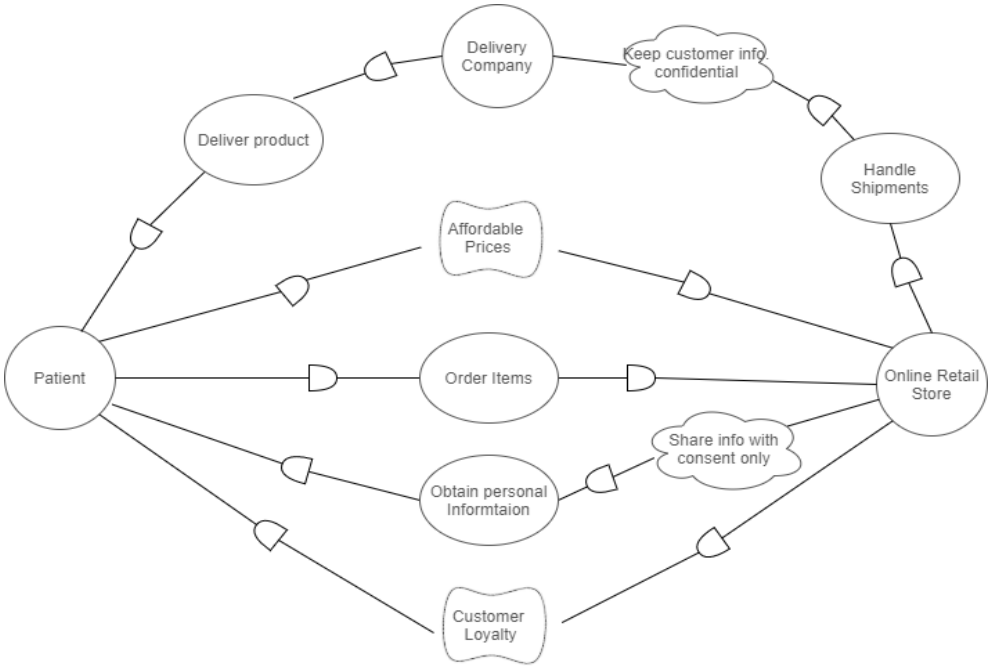

Figure 2.5: An example of an actor diagram using Secure Tropos

security diagram using Secure Tropos is presented in Figure 2.6

2.4.1 Ownership, Delegation, and Trust

Other Secure Tropos specializations (e.g., [2], [13]), on the other hand, highlight the need not only to define security concerns but also the need of compliance with privacy concerns. It has been noted that services and permissions can be delegated to people who are not necessarily trusted and that there is a need to hold the delegation of duties between them accountable [2]. Even if the actor is considered a trusted entity, permissions and authorization must be clear for both actors. Consequently, the notion ownership, provisioning, trust, and delegation has been introduced as shown in Figure 2.7 and defined as follows:

- Ownership: which indicates that an actor is the owner of a particular goal, plan, or resource labeled by by the letter “O”.

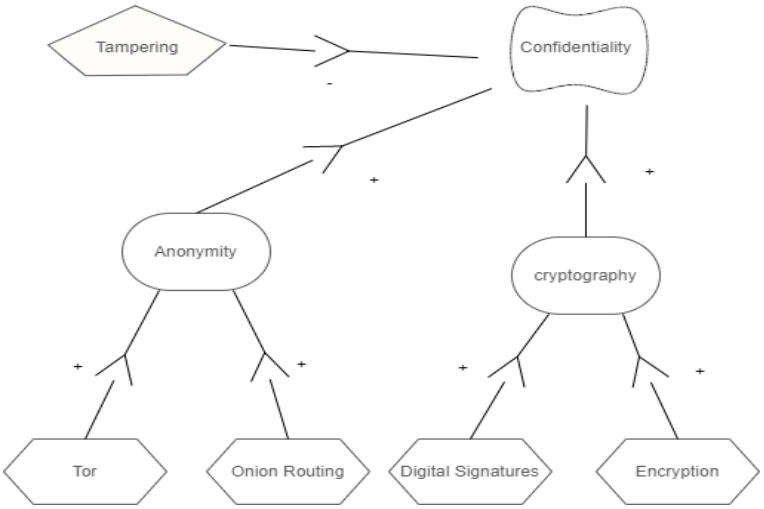

Figure 2.6: A part of the security diagram

- Delegation: includes delegation of execution and delegation of permission. The former indicates that the depender delegates the execution of a goal to the dependee (e.g. patient delegates treatment to physician), while the latter indicates the delegation of permission (e.g. delegate the permission to manage personal information to physician). Both representations are labeled as "De" and "Dp" respectively.

- Trust: includes trust of execution and trust of permission, which indicates that the depender trusts the dependee in executing a particular goal (e.g. provide medical treatment) or having permissions to carry out a certain goal (e.g. access personal information). Both representations are respectively labeled as "Te" and "Tp". Similarly, distrust of execution and distrust of permission are labeled as "Se" and "Sp" respectively. The notion of distrust introduced a new complication which is the need to monitor the actions of the distrusted party. For instance, if Bob delegates a task to Alice knowing that she might act outside her level of authorization regarding this particular task, then the need to monitor Alice's execution to the task and to monitor the

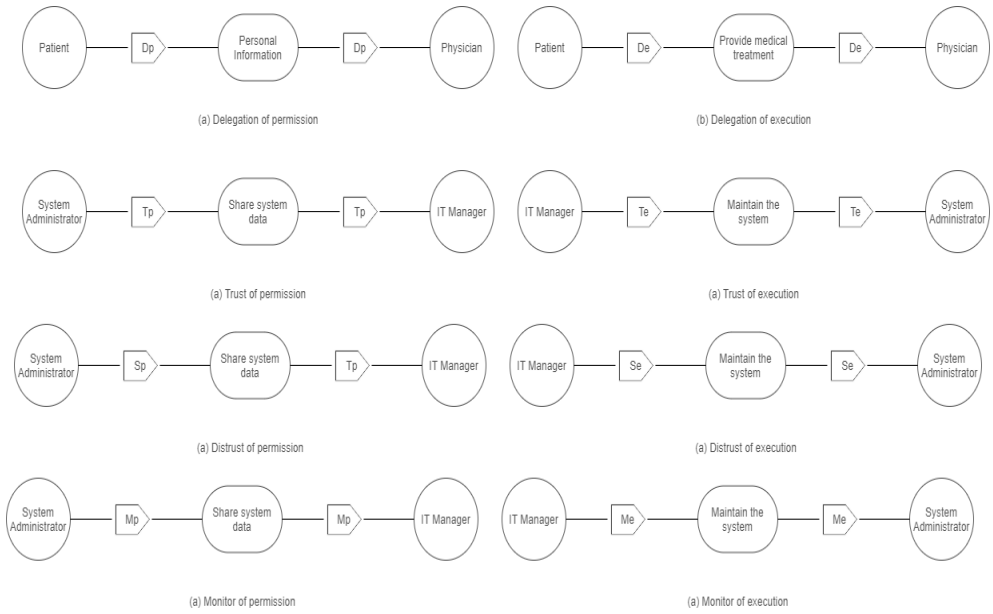

Figure 2.7: Delegations, trust, and distrust

use of permissions is required. Such representation is labeled as "Me" and "Mp" respectively.

- Provisioning: which indicates the capability of an actor to achieve a certain goal, deliver a resource, or execute a plan.

The importance of trust in requirements engineering has been recognized and extended by multiple researchers (e.g., [13]–[16]), where the notions of trust and distrust have been introduced under a single diagram, and the introduction of new attack elements such as malicious task, malicious goal, and a malicious soft-goal were able to forecast potential threats that arise from such trust relationships [16]. For instance, the impersonation of a server in an environment where the client automatically trusts the server could lead to serious corruption and exploitation of data since the client allows the complete disclosure of permissions and delegation of tasks to that server that is in fact controlled by a malicious attacker trying to sabotage the environment.

Other researchers extended Secure Tropos by adopting the elements of Se-

cure Tropos [3] to introduce the SI* framework [16] using the health care domain as an example to introduce new elements such as "Request" to denote that an actor wishes to achieve a certain goal from the dependee, and modelling the capability of the depender to the dependee throughout the system design. These elements are used in conjunction with AND/OR decomposition and means-end links to provide clear insights on how to achieve the desired end state. The framework has clear identification of objectives, entitlements and capabilities of actors by introducing two additional phases namely organizational requirements model instantiation and security mitigation to support a more precise requirement analysis and a separation of duties and conflicts.

2.4.2 Secure Tropos in Cloud Computing

The technological advancement in cloud computing made business owners and customers aware of the need to guarantee not only firm security measures but also guarantee privacy measures from cloud provider and adversaries alike. Although security and privacy are two inseparable concepts and the impact of one factor directly affects the other, the measures taken to protect each factor are different and require careful analysis. Consequently, Secure Tropos specializations in the cloud computing domain has been also introduced (e.g., [17]–[19]). One of the major reasons why organizations are avoiding cloud services is the lack of understanding of the services they provide and whether they comply with their security and privacy needs [17]. Cloud service providers offer three fundamental services: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) besides the existence of three different types of cloud services; public, private, and hybrid cloud, which turns requirement

identification and analysis into a heavy process.

Two of the Secure Tropos specializations that address cloud computing concerns both adopt the Secure Tropos language in combination with PriS framework [17], [18]. Secure Tropos mainly address the analysis of security concerns while PriS [20] is a security requirements engineering methodology that aims to address privacy requirements in early development stages to describe their effect on business processes, and to facilitate the identification of the system architecture that supports these business processes.

The first Secure Tropos specialization, presented by Kalloniatis et al. [18], introduces a security and privacy requirements engineering process for cloud that starts from organizational analysis and security and privacy requirements analysis to the selection of a deployment model. The Secure Tropos specialization presented by Mouratidis et al. [17] is based on three main activities: the Security and Privacy Cataloguing, the Security and Privacy Analysis, and the Selection of Cloud Service Provider. Although both frameworks include the concepts of the security/privacy measures (e.g. data recovery) and security/privacy mechanisms (e.g. server mirroring), security constraints are modelled as a separate entity that is enforced to a goal and satisfied by a security objective instead of being in the social dependency of actors. A graphical notation of the elements used in both frameworks is summarised in Figure 2.8, however, the framework by Mouratidis et al. [17] distinguishes between privacy and security goals where a security goal is labeled as "[S]" (e.g. [S] Integrity) and a privacy goal is labeled as "[P]" (e.g. [P] Anonymity).

Two Secure Tropos methodologies introduced tools that enables analysts to capture security and privacy requirements, namely SecTro2 [19] and STS-Tool [21], the latter being of particular interest in this thesis. The former

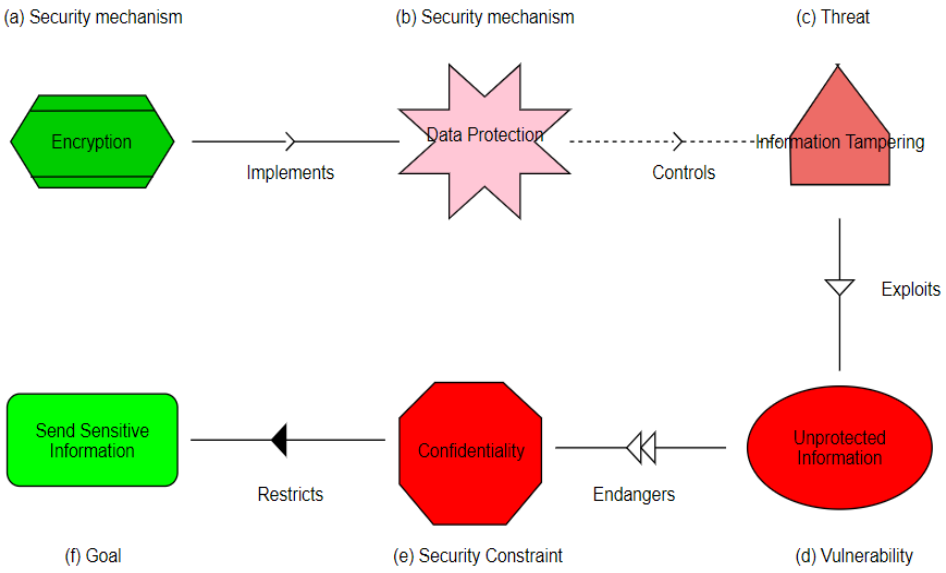

Figure 2.8: organizational and system diagram notations

was mainly introduced for the analysis of requirements in cloud computing, while the latter is a part of a bigger framework named SePTA (Security, Privacy and Trust Approach).

2.4.3 The SePTA Framework

In the previously mentioned Secure Tropos specializations, security, privacy and trust were analyzed and modelled separately, however, the SePTA framework aims to capture these concerns under one organized methodology with the assistance of several tools throughout the process. SePTA includes four different modelling languages: STS-ml, SecBPMN2, Tropos and JTrust. STS-ml and SecBPMN2 are implemented by the STS-tool, whereas Secure Tropos and JTrust by the SecTro tool.

The STS-tool is a tool that aims to capture early security requirements in sociotechnical systems. STS-ml includes three different views defined as follows:

- Social view: The aim of the social view is to analyze the dependencies between different actor within an environment. Such relationships include goals, delegations of goals, and the transmission of important documents. An agent (e.g. John) plays a certain role (e.g. physician). Goals can be decomposed into other goals through AND/OR decomposition links. Goals (e.g. obtain personal information) can produce, modify, or read from other documents (e.g. Health Card). A transmitted goal from one actor to another will be marked in dark green, while a transmitted document will be marked in dark grey. Additionally, the view allows to depict security requirements (e.g. confidentiality, non-repudiation). Such security requirements are presented under each goal or document as shown in Figure 2.9.

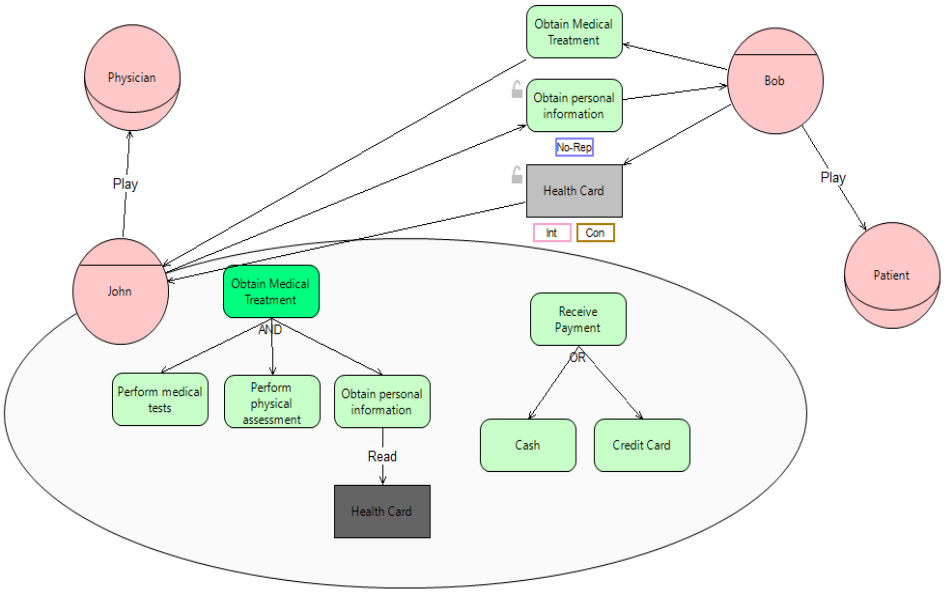

Figure 2.9: An example of the social view in the STS-tool

- Information view: The aim of the information view is to present the information that composes the documents presented in the social view. An agent/role owns a certain document (e.g. Health Card), therefore owns the information in that document (e.g. First Name, Last Name). Thus, documents refer to tangible ways to represent

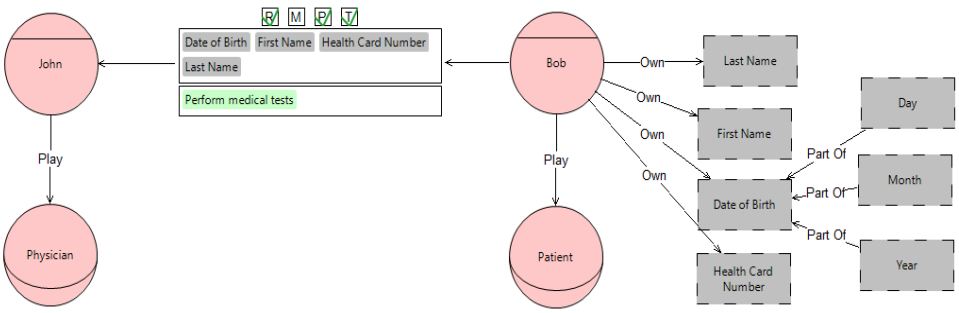

Figure 2.11: An example of the authorization view in the STS-tool

information. Information (e.g. year) can also be part of other information (e.g. date of birth). An example is presented in Figure 2.10.

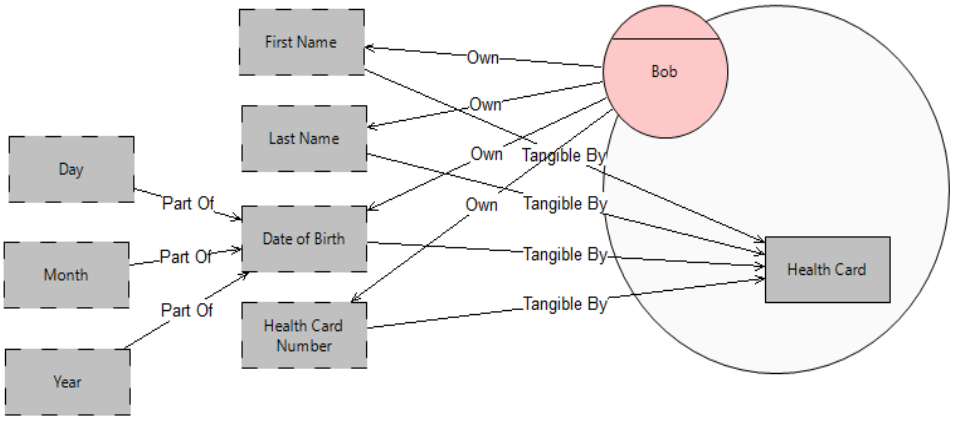

Figure 2.10: An example of the information view in the STS-tool

- Authorization view: The aim of the authorization view is to depict the authorizations and permissions granted between actor in respect to the goals and documents defined in the social and information view. Users can chose between four different permissions; read, produce, modify, and transmit to grant to other actors. An example is presented in Figure 2.11.

The framework continues with the Sectro-tool that aims to capture security and privacy requirements from the goals and delegations defined in the

first view. The security requirements can be then modelled through attack views that explain the manifestation of a certain threat that targets a vulnerability in the systems and the measures that can defend against this vulnerability. On the other hand, the trust view can be drawn to analyze whether actors will act according to the system design or not. Finally the goals and dependencies between actors can be translated to business processes through the STS-tool using the SecBPMN2 language.

The purpose of Secure Tropos specializations is to capture security and privacy requirements from early stages of the system development by analyzing the social settings of an environment and the system that is supposed to interact with it. Although these specializations introduce multiple procedures and techniques for such analyses across different domains, the security and privacy requirements are only addressed at the abstract level of the actors and the system avoiding detailed description of their implementation in practice.

Various security primitives are considered critical and were addressed with Secure Tropos such as confidentiality, integrity, and authentication. However, the measures taken to address these primitives only informed users about the relevant technologies that mitigates them such as encryption and digital signatures. Such measures require an adaptation of workflows and business processes, and assume preparatory steps which include installation of software, generation of keys and certification, secure storage and sharing of cryptographic keys.

2.5 Goal-Modelling Methodologies

Secure Tropos and Secure Tropos specializations were influenced by many other security-intensive goal modelling frameworks such as KAOS, SQAURE, SREP, and MOSRE, among others.

KAOS [22] is a well known goal-oriented requirements engineering approach whereby high level goals are elaborated. The framework introduces the notion of obstacles between software and agents that may hinder or affect a certain goal negatively and require resolution for the completion of that goal. KAOS has been extended to incorporate security requirements through the development of two models [23]. The first model depicts the system-to-be by describing the relationship between goals, agents, objects, operations and requirements, while the second model, named as "anti-model", aims to capture attackers and their relative potential threats that attack vulnerabilities in the system in order to elicit security requirements for the prevention of these threats.

SQUARE (Security Quality Requirements Engineering) [24] is a risk-driven methodology for eliciting and prioritizing security requirements within an organization by listing important functional information that are turned into artifacts. These artifacts are then categorized to meet organizational goals. The methodology also includes a risk assessment that evaluates the tolerance of the organization against various threats with regards to the categorized requirements. SQUARE has been extended [25] to support the elicitation and prioritization of privacy requirements, it also integrates a technique called Privacy Requirements Elicitation Technique (PRET) [26] for this particular purpose. SREP [27] is also a risk-driven methodology, however, it integrates common criteria and information security standards, such as ISO/IEC 27001, while eliciting security requirements. MOSRE

(Model-Oriented Security Requirements Engineering) [28] is a methodology for the elicitation, specification, and analysis of security requirements in early development stages of Web Applications.

2.6 Security Patterns

Although these goal-modelling frameworks aim to analyze security and privacy requirements, the identification of such concerns is rather difficult to develop from scratch and without expert guidance. To assist with this problem, the development and use of security and privacy catalogues (e.g., [29]–[32]) and repositories (e.g., [33]) that contain security and privacy patterns have been proposed. Security patterns aim at capturing workable solutions for security related recurring problems to be understood by developers or users who are not security professionals [33]. Privacy patterns work similarly to security patterns to guarantee the realization of privacy issues through a specific number of steps that include activities and flows connecting them together [31].

For instance, security patterns proposed by Kienzle et al. [33] are decomposed into structural pattern and procedural patterns. Structural patterns are patterns that can be implemented in the final product (e.g. encryption storage), whereas procedural are patterns that improve the process for secure development systems (e.g. log for audit). These patterns are analyzed through a template that is composed of name , abstract, problem, solution, issues, trade-Offs, related Patterns, and references. Other researchers analyzed security patterns using the Zachman framework [30]. Patterns in this framework are classified into core, perimeter, and exterior security patterns whereby spoofing, tampering, repudiation, information disclosure,

and elevation of privilege are addressed under each classification scheme.

To the best of our knowledge, no framework has presented automated security patterns in the same approach of this thesis, however, similar studies on the automation and reasoning of security patterns have been proposed to guide designers and professionals throughout various development stages. Such studies either provide a framework for the analysis of security patterns and/or model these patterns and provide some reasoning or pseudo-code that aids designers or analysts in the selection of the most appropriate pattern(s), or even derive the most appropriate patterns to use according to threat analysis and modelling.

A pattern system for security requirements engineering has been proposed, named *security problem frames* to analyze security related requirements (e.g. confidentiality and authentication) [34]. The framework defines several concretized security problem frames that include generic mechanisms to solve a particular problem (e.g. use credentials for authentication). Each frame is equipped with preconditions and postconditions that contain dependencies that should be identified. According to the framework, for a frame to be considered applicable, another frame must be applied, therefore, sub-problems would be identified throughout the process, and the generation of new sub-problems would end when all the preconditions for the security problem frames are satisfactory. For instance, a precondition for the security problem frame for confidential data transmission is that a malicious attacker should not be able to know how to send data between two parties and that a confidential path must exist between the sending party and the receiving party, whereas a postcondition states that the attacker should not be able to reveal the data sent between them. A concretized security problem frame for this case is encryption.

Weis and Mouratadis [35], on the other hand, propose a hierarchical approach for the selection of security patterns and the implication of one security pattern to various security patterns. Additionally, this approach supports the search for a combination of security patterns to meet given security requirements. The selection of security patterns is based on the Goal-oriented Requirements Language (GRL) and Prolog rules. A GRL model can show the contributions of a security pattern to security related non-functional requirements. The combination of patterns can be visualized through GRL and reasoned about through Prolog rules by a an evaluation mechanism. A satisfaction level is associated between different elements and goals through contribution links. These levels range from 0 to 1, whereby a satisfaction level of 1 indicates that a goal is fully satisfied and a satisfactory level of 0 indicates the opposite. As for the search engine of this approach, it works by generating patterns against the pattern repository that match the user's requirements. Similarly Slavin et al. [2014] propose a hierarchical approach for pattern selection through an inquiry-cycle approach by adopting a feature diagram notation for eliciting relevant patterns to select the most appropriate one according to the situation. The feature diagram includes graphical notations, such as optional, mandatory, inclusive, and exclusive, to classify patterns. These patterns are thoroughly analyzed through a pattern template that addresses different aspects of patterns.

On the other hand, Heyman et .al [36] present an approach to facilitate the selection of security metrics and to support the interpretation of measurements by associating security metrics to security patterns and leveraging these security patterns to utilize the interrelations of security patterns and security objectives and to improve secure software development. Researchers in the approach acknowledge that a good metric is a metric that is

specific and relevant to decision makers to take action. This approach is initially modelled through a dependency graph using different node types, arc types, and decomposition types, whereby security objectives (e.g. auditing) are realized by security patterns (e.g. secure logger) which in turn are realized by security metrics (e.g. Use digital signatures to achieve integrity of secure loggers). The interpretation of measurements results is achieved by going through three levels. At the first level, metrics are collected, normalized to a percentage and combined into a single value for each pattern based on the pattern description. At the second level, individual pattern scores are aggregated to form an indicator for the lowest-level objective they realize, whereby AND decomposition links propagate the minimum score of the child nodes, and OR decomposition links propagate the maximum score. At the third level, the same process is followed, however, to obtain a summarizing score for root objectives.

2.7 Automated Reasoning with Goal Models

Our approach for generating secure workflows relies on automated reasoning using an AI planner. AI planners have been used by multiple researchers for automated reasoning about goal models and security requirements and designs. Some AI planners are based on STRIPS [37], one of the first planners who also established the standard language for defining planning problem, and it's successor PDDL [38]. STRIPS consists of a set of actions based on which plans are to be built as well as a set of domain predicates describing properties of objects that are of interest in the domain. PDDL specifications include domain predicates that describe the effect of a par-

ticular action, which actions are possible, and under which circumstances an action can be performed.

PDDL has been used as a language to which a goal model can be translated to assist automated reasoning. For example Bryl et al. [39] use PDDL planners to capture the social setting of the organizational setting and the system-to-be, as well as to generate alternative solutions for the fulfillment of functional and security requirements.

Liaskos et al. [40], on the other hand, use PDDL planners to represent and reason about preferences (nice-to-have) in requirements engineering. The framework extends a goal-modelling notation to include these preferences. Researchers in the framework note that stakeholders have preferred goals apart from mandatory ones. For instance, if the stakeholders have a mandatory goal of "schedule a meeting", they might also have a preference of "using the 5th room" as a preference to that goal. To evaluate this distinction, quantitative prioritizations are constructed to the former and used as a criteria to evaluate alternative solutions to achieve the latter. To generate such solutions, an AI planner is adopted (HTN-Plan-P) which combines PDDL 3.0 and and its preference specification capabilities HTN, to generate plans that best satisfy given requirements so that analysts can understand the impact of high-level stakeholder preferences on low-level design decisions. Goal models are developed using a version of the i^* language in this framework augmented with temporal preferences and quantify preferences. The result is translated into PDDL 3.0 and HTN. Mandatory goal decomposition are translated and generated through HTN, while goal preference and priorities are translated through PDDL 3.0. The resulting combined specification is then given as input to HTN-Plan-P which identifies plans that both satisfy the main decomposition while optimizing the preference specifications.

Datalog has been used in Secure Tropos to formalize the concepts of ownership, delegation, and trust (e.g., as in [2], [14]) to automatically verify the correctness and the consistency of security requirements. A Datalog program is a logical query language that consist of a set of rules that may include recursions and negations, where negations indicate a failure to infer that the satisfaction of a certain rule is false.

2.8 Attack Trees

Attack trees are relevant in our framework in that they inform the reasoner about how a process can be compromised. Attack trees, originally suggested by Schneier [41] are hierarchical representations of an attacker's goals and methods to breach a system vulnerability. Such models allow analysts to better address security requirements based on what happens when those assumptions are broken. Attack trees do not only include goals and the methods to achieve those goals, but they can also be evaluated in terms of cost in dollars, cost in resources, and time to break a certain vulnerability. Such values allows analysts to assess the probability of success of a given attack and the possibility of an attacker to choose a given type of attack.

Researchers adopted the concept of attack trees in multiple domains to address different type of vulnerabilities. For instance, KAOS [22] included a version of this concept under the term of anti-goals to model the intentions of attackers in order to find countermeasures against security threats. Other researchers integrate attack trees with misuse cases [42] to properly analyze security requirements starting from the abstract level. The purpose of such integration is to highlight broad threats that can be technically

defined in details using attack trees. This way, users from multiple backgrounds can understand the threats identified and their countermeasures.

On the other hand, quantitative research on attack trees has been also performed to assist system administrators in providing an early warning on potential system vulnerabilities [43]. Researchers in this framework present a set of algorithms to generate a list possible attacks and their probability to occur based on different scenarios. Such scenarios include augmented and atomic attacks. The former simulates an attack that starts from the bottom leafs of an attack tree upwards, while the latter consider attacks from multiple areas within the trees that may include one or more attacker simultaneously.

Multiple researchers not only defined and presented a unique design to model and analyze attack trees (e.g., [44], [45]), but also presented practical use cases on their usage (e.g., [46]–[50]). Tong Li et al. [50] even expanded their research of attack trees in Socio-Tecnical Systems (STS) across three layers (business layer, application layer, and physical layer). They analyze patterns in the trees using three concepts (context, problem, and solution) where attack domains and impact are identified for each each target in each layer.

Chapter 3

Modeling and Reasoning with Security Workflows

3.1 Overview

In this section we present our framework for secure workflow generation using security patterns. The framework consists of the following steps and components: (a) a set of workflow patterns is developed that captures common ways to execute elementary business tasks, such as transmitting a document, using various kinds of security controls, (b) a set of attack trees is developed to describe ways by which security properties of the root of a workflow pattern can be attacked, (c) a description of domain specific requirements in STS-ml to which security patterns can be attached (d) a set of vulnerability assumptions that describe what kinds of attacks are relevant in a given context, and (e) an approach to combine information from all these models into an HTN planning specification and generate domain-specific workflows that satisfy the business and security requirements under the given vulnerability assumptions.

Figure 3.1 presents a flow chart describing the proposed framework. Initially, workflow requirement models are developed based on the elements of the i^* framework to describe domain specific actions related to actors goals, followed by security requirements analysis that are captured using the STS-tool to identify potential threats relevant to the context. Meanwhile, two additional modeling components are considered. Domain assumptions refer to pragmatic facts about the domain, such as whether the parties have each other's emails (or know each other), can install software, or whether they can meet (e.g. to exchange a password). Vulnerability assumptions, on the other hand, refer to the attacks that must be assumed to be feasible and well motivated. For example we may assume that the network that connects the domain actors is hacked which is a basic assumption, that the PCs and phones are hacked as well, which is a more strict assumption, or that the actors are actively bugged and monitored by an unusually resourceful and motivated actor, which is an extreme assumption for normal cases.

All this information is collected and modeled by requirements and business analysts with awareness but not specialized training on security. The specialized knowledge comes in a form of security workflow and attack patterns maintained by the community of experts, and applicable to a large variety of domains. The domain specific information gathered by the analysts and the patterns are then compiled into a formal specification readable by an AI planner. The compilation is taking place manually, but is easy to automate or semi-automate. Given the specification the planner is able to produce actual workflow instances that meet both security requirements and accepted security practices. This way business owners and normal users that lack high end security solutions (e.g. ERP systems) can be given the necessary steps to accomplish their goals securely using open source software

and/or using traditional methods available by the online community.

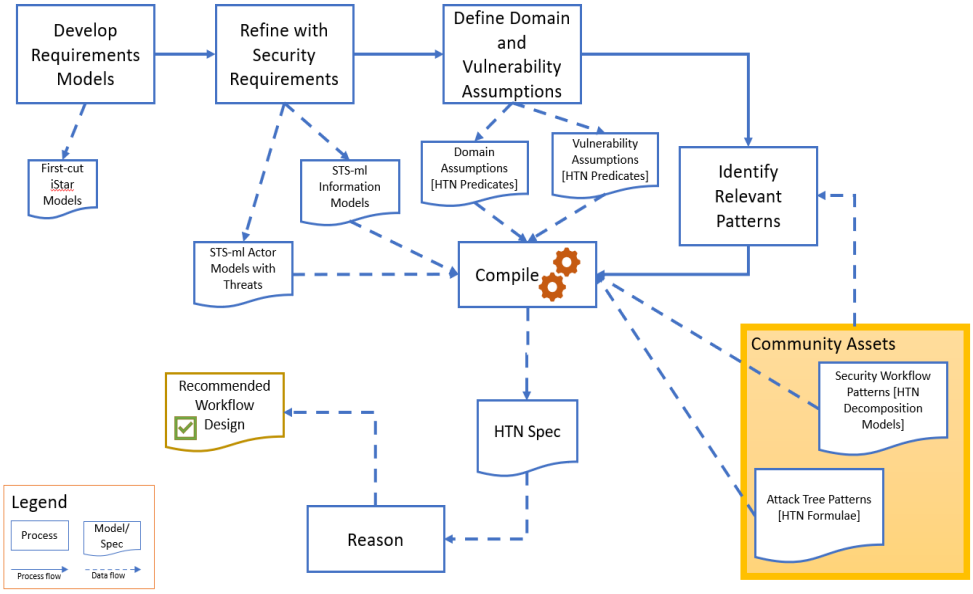

Figure 3.1: A flow chart describing the application of the framework

Note that our proposal is based on the development of a number of operational patterns that exemplify the use of basic cryptographic primitives for performing common information exchange tasks. Such patterns are meant to be built and maintained by security experts (e.g. tool providers, community, IT departments) and are problem-domain agnostic, in that they are abstract enough to be adopted to a wide variety of different domains. Each pattern is identified by a simple functional goal, and it contains various ways by which the goal can be executed in secure or less secure ways.

We begin by describing the concept of a threat that gives rise to security requirements (Section 3.2). We then discuss our proposed workflow patterns (Section 3.3) and their integration with such requirements and with security requirements (Section 3.4). Next, we define attack tree patterns that affect security requirements (Section 3.5). We then describe the adopted AI planner (Section 3.6), and finally explain the translation of these elements into the planner while reasoning about the output being generated (Section 3.7).

3.2 Threats

The purpose of security patterns is to propose ways by which common elementary business processes can be performed while mitigating or eliminating risks that threaten goals to obtain secure business processes. It is thus useful to investigate threats that necessitate such patterns. For simplicity, we specifically focus on three different attacks that threatens goals: information sniffing, information tampering, and repudiation. These threats can also include many different type of attacks [51]–[53].

- Sniffing: sniffing [54] is one of the most common attacks that aims to steal important information from users, such as passwords, emails, files, or any sensitive content that may be beneficial for the adversary. Attackers do so by reading the data transferred from point A to point B in the network. This contrasts phishing that draws users into giving their own sensitive information through fraudulent websites or emails. This type of attack breaks confidentiality of users since their data can be exposed to the public.

- Tampering: information tampering [55] refers to malicious modifications of data by unauthorized users to damage, corrupt, or gain advantage of the victim. Such data can be stored on disk or being transferred in the network. Information tampering can lead to serious damage into business operations and may cause heavy losses across different sectors of an organization, and sometimes it can put people's lives at risk if the data being tampered with is health care information.

- Repudiation: repudiation refers to the denial of an entity to committed action or behavior [56]. For instance, if a customer ordered an

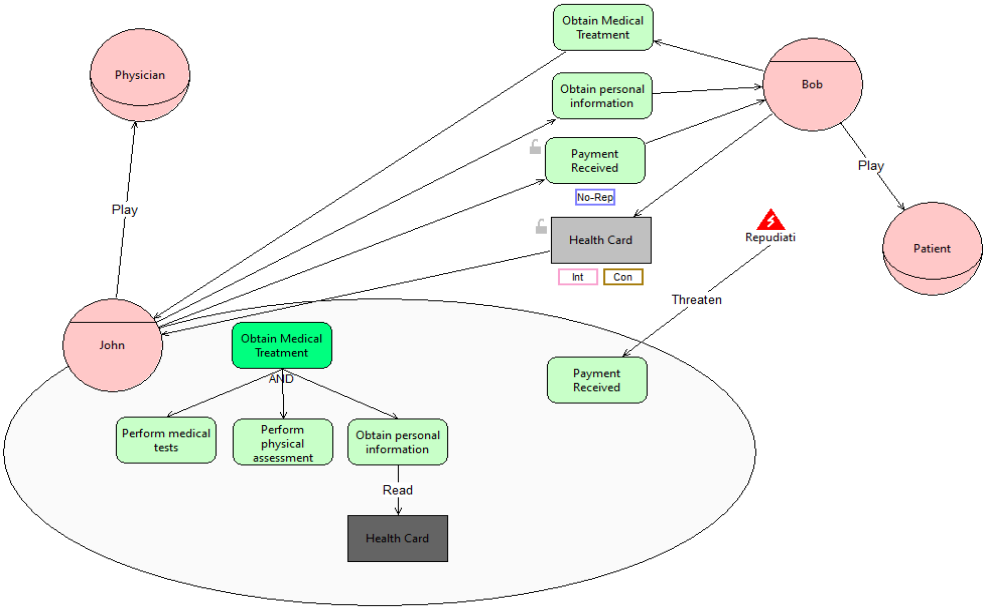

Figure 3.2: An example of the threats in the STS-tool

item online or received a payment from another person, then repudiation refers to the denial of such processes.

To model these threats, we use the event element from the STS-tool. An event can threaten a goal or a sub-goal. An example is shown in Figure 3.2 where a denial of payment from the physician to the patient can happen. Thus, the event "repudiation" threatens the goal "Receive Payment".

3.3 Modelling Security Workflow Patterns

3.3.1 Overview and Pattern Format

In the presence of possible security threats, tasks performed by domain actors to achieve their goals may need to be enriched by additional security steps, such as encryption/decryption, digital signing/verifying, authentication, etc. However, the way by which such tasks need to be mixed with domain specific tasks in order to bring about the desired security properties

is not always straightforward and may require expert advice and validation.

For instance, consider the example of the translation company introduced earlier, where a customer would like to send a confidential document to the company. To achieve confidentiality, the sender and the recipient would have to agree on a type of encryption that requires generating and sharing keys. Nonetheless, multiple encryption types exist for information exchange, and their usage depends on the security requirements users would like to achieve. Additionally, sharing such keys may expose participants to vulnerabilities, if not done properly.

Assuming that the secretary and the customer exchange their information through email services and that such services are compromised by adversaries, then sending the required keys through this communication method will also expose the keys to the attacker making the whole encryption process pointless, as sensitive documents are leaked and a breach in confidentiality is present. Thus, the need arise to find alternative ways to address security requirements taking into consideration vulnerability assumptions.

To address such events, we propose the development of high-variability workflow patterns that describe allowed ways by which such security actions can be mixed with domain-specific actions to allow secure fulfillment of actor goals. Such security patterns are usually build by security experts to address domain problems that seem difficult to analyze and may have more than one solution for a particular problem.

We note that the patterns we present in the following subsections are meant to demonstrate feasibility of the proposal rather than taken as definitive patterns addressing the corresponding security concern. As we discussed, patterns are supposed to be developed by security experts and maintained in the public domain to ensure their validity. It is thus not our purpose in

this thesis to propose the patterns as the definite ones for immediate use. We model security pattern by adopting the elements of the i^* 2.0.

3.3.2 Encryption Pattern

We now discuss in more detail some of the patterns, starting from the encryption pattern. Consider the example of the translation company earlier. A frequent task that employees use in their daily activities is the transmission of files and documents. Ideally, one would like to keep the information being transmitted from being sniffed, repudiated, or altered by an attacker that could take advantage of the stolen information. For this particular reason, we introduce the pattern of encryption to the information being transmitted to prevent such attacks.

There are two main types of encryption: symmetric encryption and asymmetric encryption [56]. Each type of encryption has its own advantages and disadvantages, and sometimes both methods can be used together to further enhance the communication channel between two parties that would like to exchange encrypted information. Both methods include keys to encrypt and decrypt information, however, the generation, type, and the exchange of keys may vary depending on context being used. Encryption methods are explained as follows:

- Symmetric Encryption: in a symmetric encryption, the same key to encrypt data is also used to decrypt it. This key can be generated by the sender or recipient as long as both parties share and work with the same key to transfer their information. This key must also be kept secret and only be shared among the communicating members, otherwise the communication is considered not confidential as the

attacker who obtained the shared secret key can decrypt and alter the information being sent.

- Asymmetric Encryption: also referred as public-key cryptography, unlike symmetric encryption, asymmetric encryption involves the generation of two types of keys: a private key and a public key. In the context of asymmetric encryption/decryption, the former is used for information decryption, while the latter is used for information encryption. Each member in this type of communication generates its own private/public keys. As the names denote, each private key is to be kept completely secret to its owner, while public keys are made public so that anyone in the network can use them to encrypt the desired data and send it to the recipient that decrypts it with its own private key.

Several advantages and disadvantages distinguish these two methods. One of the main advantages of symmetric encryption is that it includes one type of keys, and it is relatively much faster than asymmetric encryption as it can be designed to have high rates of data throughput that can encrypt hundreds of megabytes per second. However, as stated earlier, keys must remain secret at both ends, therefore sharing them securely in large networks becomes much more difficult and complex.

In public-key cryptography, on the other hand, keys are easier to maintain since the private key that is required to be secret is usually generated by the client and doesn't have to move over the network unlike symmetric keys, especially in large network communications. Additionally, key pairs can remain unchanged for several years [56]. However, this encryption method is much slower than symmetric encryption, and the public key that needs to be made public must be authenticated, usually through a trusted-third

party (explained in Section 3.2.3).

To gain advantage of both methods, a third encryption method is introduced, named as hybrid encryption (e.g. SSL and TLS) [56], whereby asymmetric encryption is used to exchange symmetric keys. Such method is mainly used between server-client communication. The private/public key pair is generated by the server. The client receives the public key from the server and generates a symmetric key that is encrypted with the server's public key and sent to the server itself to be decrypted by its private key. The main advantage of this method is to gain performance since large amount of data is being transferred between both ends.

Since the transmission of information can happen between systems and individuals, it's important to differentiate between them since online communications between clients and servers can include an extra encryption method (hybrid encryption), while the transmission of encrypted information between persons require the installation of an encryption software, followed by the generation and sharing of keys. As shown in Figure 3.3 we label the communication between individuals who require to encrypt their own information and send them over the network under "setup-security-tools [Sender, Recipient, Information]" to describe the necessary security controls that individuals need to follow to address unwanted threats, while the transmission of information from a user to a server as "system-encryption-method [System, Client]". The variables in the brackets or braces (e.g. Sender and Recipient) start with a capital letter and can be replaced according to the context being used, while constants start with a small letter (e.g. key and symmetric).

Using the encryption patterns presented in Figure 3.3, a user can decide to avoid any encryption methods or to use a type of encryption that will

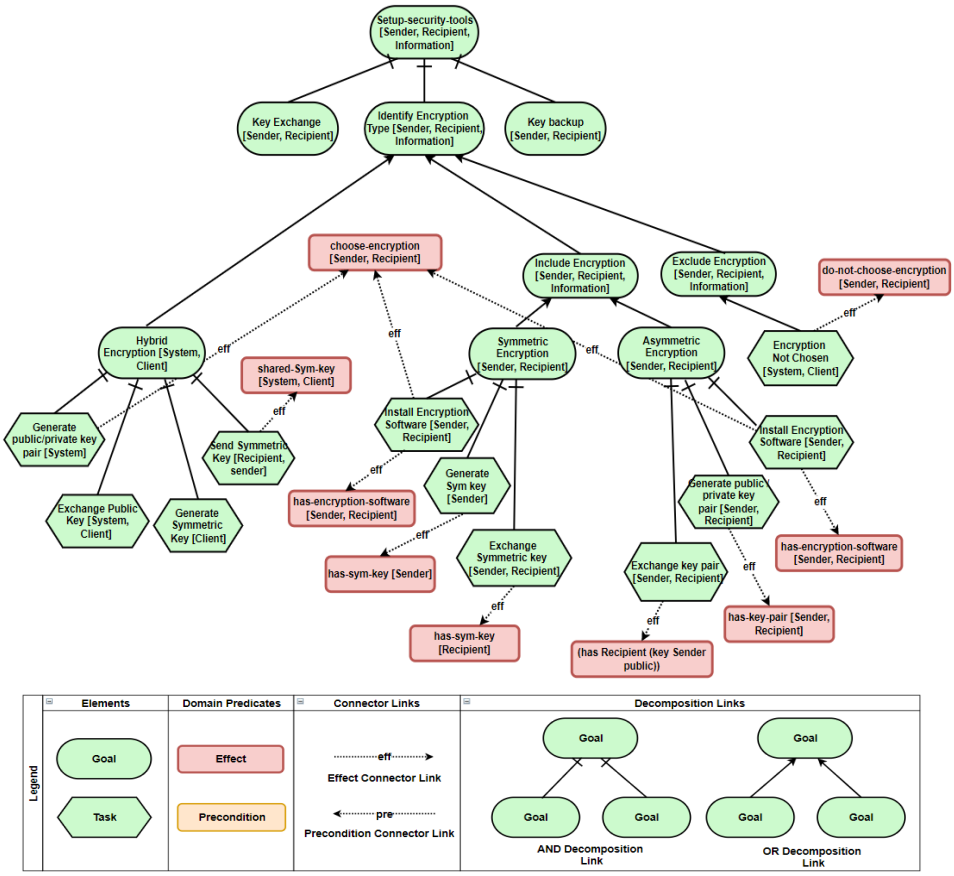

Figure 3.3: Encryption method presented using i^* 2.0 elements

result in the effect of "do-not-choose-encryption [Sender, Recipient]" and "choose-encryption [Sender, Recipient]". Accordingly, the three encryption options are presented through tasks with the OR decomposition link since one type has to be chosen during the transmission of information. Each task includes the necessary steps that should be taken for the completion of the goal. For instance, choosing symmetric encryption to encrypt a user document will require the user to install an encryption software and generate a symmetric key that will be shared with the recipient. These steps will lead to the effects of "has-encryption-software [Sender, Recipient]", "has-sym-key [Sender]", and "has-sym-key [Recipient]" respectively.

3.3.3 Key Exchange Pattern

As mentioned in the previous section, encryption methods require sharing keys. Consider the example of Alice and Bob that would like to share an important and sensitive document over the network and avoid any threats that can affect their confidentiality. Let's assume that the encryption method used is symmetric, thus they would like to share the secret key to establish their communication. Alice and Bob can of course meet in person and exchange their keys if they live in close distance [56]. But what if large distances separates them? Another choice would be to send the key over online email services (e.g. Gmail or Microsoft Outlook) or by phone, however, these methods do not always guarantee the confidentiality of the key as the key being sent is in plain text and does not prevent the attacker from eavesdropping.

In asymmetric cryptography, it is the public key that needs to be exchanged. One way of doing so is through a trusted-third party (TTP), such as a certificate authority. A trusted-third party (TTP) [56], is a trusted entity that authenticates public keys by issuing digital certificates confirming that the public key sent for verification really corresponds to the party that requires verification. Such certificates include information about the entity, such as the name and the public key of that entity, the validity of the certificate, and possibly other relevant information (e.g. address, network address, etc.). TTPs are usually used by servers to identify themselves to users as a legitimate entity that do not have any fraudulent intentions.

In the case of Alice and Bob, the most secure key exchange would be to issue their own certificates and send them to each other, nevertheless, this key exchange method can be expensive as entities that require certification

generation and verification require a certain fee for that service. Additionally, Alice and Bob can still send their public keys over email services and confirm that the public key is not tampered by phone as long as they certify (i.e. confirm that a public key corresponds to a certain entity) each other's keys in the encryption software they are using at the end of the key transfer. The key exchange pattern is depicted in Figure 3.4 with the corresponding effects. As shown in this model, the effect "choose-encryption [Sender, Recipient]" in the encryption pattern became the precondition for the key exchange pattern. In other words, for the key exchange pattern to be true, the precondition of choosing an encryption method should be true.

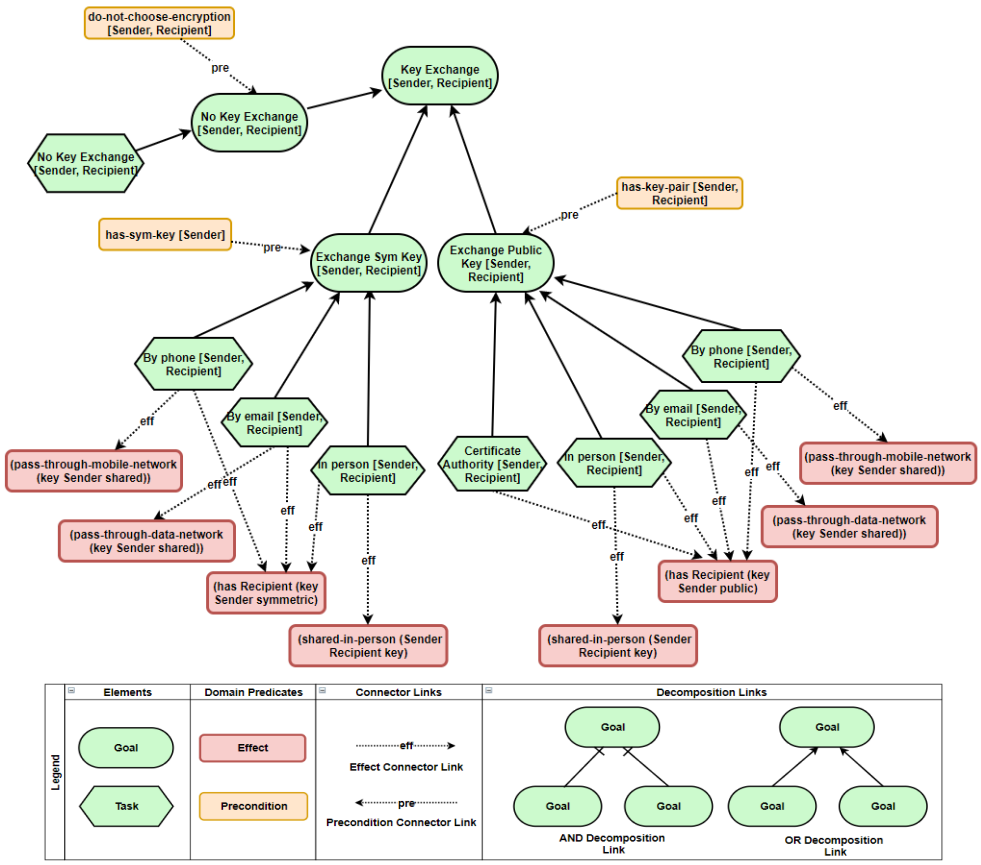

Figure 3.4: Key exchange pattern presented using i^* 2.0 elements

3.3.4 Transmit Information Pattern

Sending the information to the desired entity comes after deciding the encryption method. In case the chosen encryption method is symmetric, the sender encrypts the information using the shared secret key that was agreed upon, sends it to the recipient that will be decrypted using the same shared secret key. As for the case of asymmetric encryption, this includes the generation public/private keys and the exchange of public keys. Both encryption methods can include or exclude digital signatures.

Let us describe digital signatures by referring to the example of Alice and Bob. If Alice lends Bob $1,000, she may issue a document with the presence of a witness specifying the date, amount lent to Bob, and the penalty on Bob in case of payment failure that includes both their signatures.

A witness acts as a trusted third party to validate the contract issued between the members and/or to resolve potential disputes that requires the assistance of a third party, as well as to act as witness to cases that requires the interference of a notary service, such as a judge. As a result, this contract is considered legally binding for both members on which they cannot deny the activities and rules agreed upon since it contains their signatures and the presence of a trusted-third party.

Similarly, digital signatures in cryptography are designed to support authentication, data integrity, and non-repudiation [56]. Data integrity refers to the assurance to detect data manipulation by unauthorized parties (i.e, assurance against tampering). Authentication is also related to identification, which is the ability of the members in the communication network to identify each other.

Although digital signatures can be used in symmetric encryption, they may

provide data integrity and message authentication but not non-repudiation [56]. In order for non-repudiation to be achieved, a clear distinction between the sender and the receiver must be present. However, since a single key is shared in symmetric encryption, both parties can originate a message using the shared key. Thus, in case a dispute arises, a differentiation between both communicating parties cannot be identified. To assure non-repudiation, public-key cryptography must be used whereby the sender of a message signs it with it's private key, and sends it to the recipient to be verified with its public key (i.e. the opposite usage of keys for encryption and decryption).

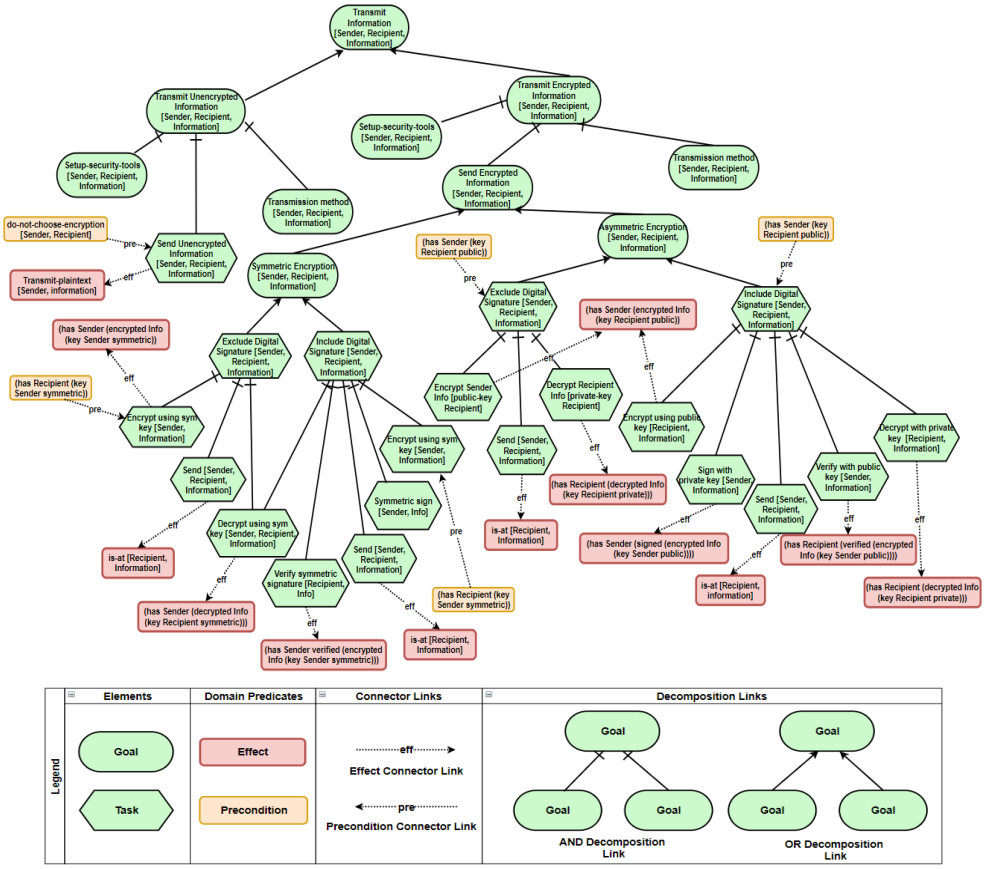

Figure 3.5: Transmit information pattern presented using i^* 2.0 elements

In the case of Alice and Bob, certificate authorities act as the trusted-third party. In fact, certificate authorities verify an entity's public key by signing it prior issuing the necessary certificate. In case users do not wish to seek

authentication through certificate authorities, they can either certify them themselves, or certify them from another user that is mutually trusted. Figure 3.5 depicts the transmit information pattern with the corresponding preconditions and effects.

Hybrid encryption will result in the transfer of information using symmetric keys that uses the same encryption and decryption steps when choosing symmetric encryption. For simplicity, we omit full presentation of the specific pattern.

3.3.5 Transmission Method

Although the pattern in this section does not represent any cryptographic primitives, it's important to include the means that users would like to exchange their information. Three general means have been included in this pattern; transmit information using online email services (e.g. Microsoft Outlook or Gmail), online file storage systems (e.g. Dropbox), or in person by putting the desired data on a media and deliver it to the recipient. The examples of which email services or online storages are used is not listed in the diagram for reasons that are explained in the validation and evaluation chapter. The transmission method pattern is presented in Figure 3.6. It's noticeable to mention that unlike other patterns this pattern has no preconditions since the information can be transmitted regardless of whether the information is encrypted or not. Trivial preconditions, such as having an account, having log-in credentials, etc. can be added depending on the use case.

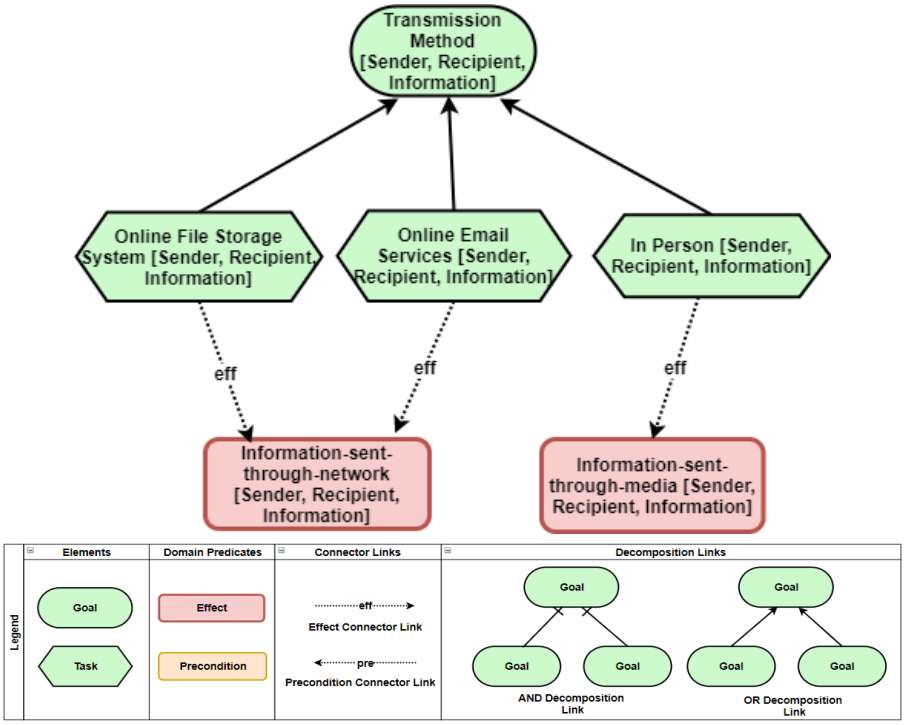

Figure 3.6: Transmission method pattern presented using i^* 2.0 elements

3.3.6 Key Backup Pattern

Let's assume that Alice and Bob chose an encryption method, exchanged their keys. An important step to be taken is to backup the generated keys [56]. Although keys are usually stored on the local drive that the encryption software is using, a sudden crash of the hard drive can lead to the loss of keys, leading to the loss of important documents, and if these files are still present on their emails or online storages (e.g. Gmail or Dropbox), the chance of decrypting them becomes null unless one of the users have the files with the ability to decrypt them. Therefore, key backup is crucial to prevent such unwanted events. Keys can be backed up using any type of media (e.g. external hard drive, USB device, CD-ROM, etc.) and stored in a safe place (e.g. safe deposit, bank, etc.). The key backup pattern is presented in Figure 3.7.

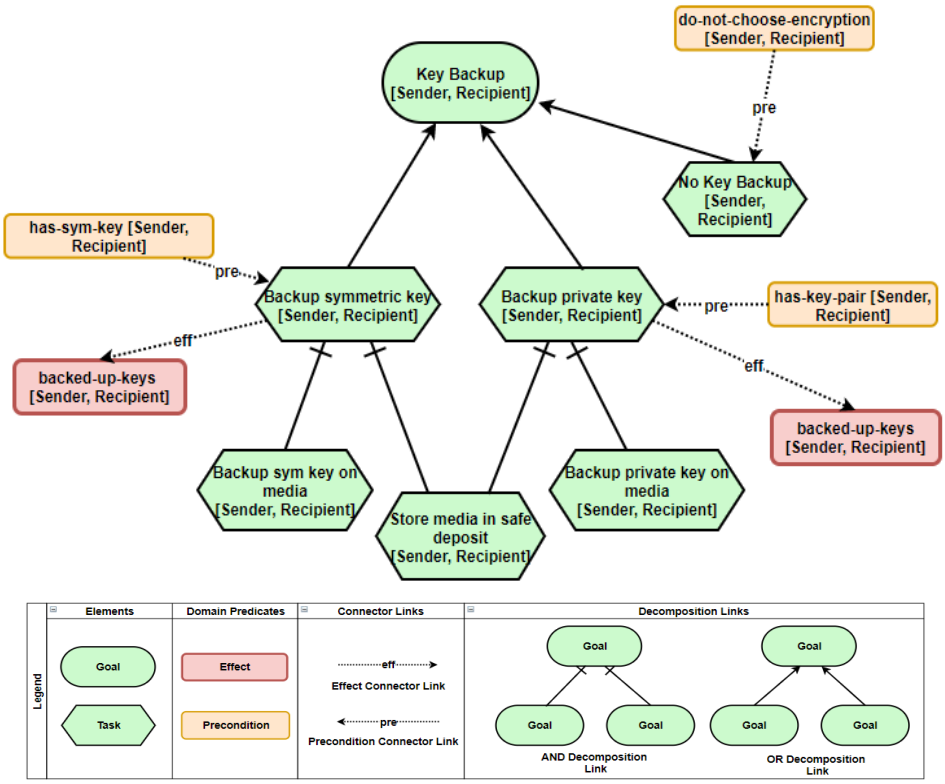

Figure 3.7: Key backup pattern presented using i^* 2.0 elements

This pattern includes the backup of symmetric and private keys. Symmetric key backup requires that users had generated symmetric keys, shown as the precondition "has-sym-key [Sender, Recipient]", while asymmetric key backup requires that parties generated public/private keys, shown as the precondition "has-key-pair [Sender, Recipient]". Such preconditions are the effects of using the corresponding encryption methods.

3.3.7 Manage Information on Disk Pattern

Let's assume that a user wants to download and store a sensitive document from the internet, and would like to keep the document safe from unauthorized access. Despite that operating systems have their own encryption mechanism, an adversary can read or alter any file on the systems once the

security controls of the operating system are compromised. Thus, adding an extra security layer for the users may prevent such threats since encryption software, such as Kleopatra, will ask users for a passphrase upon the creation of keys and at the time of the usage of those keys. Adversaries, therefore, cannot read or tamper encrypted files unless they know the passphrase.

Figure 3.8 depicts the manage on disk pattern, whereby the encryption method pattern is also included in this pattern since the user must choose an encryption method after downloading the document. However, using symmetric encryption for this particular context is sufficient to provide sufficient security because sharing keys won't be required as the user wants to store this document on disk without sending it online. The retention period have been also included in the pattern with general options, such as "delete information after end of use" and "delete information after a specific time", since such options may be defined differently across companies and organizations regardless of the domain.

3.4 Capturing Security Requirements

With a security pattern catalogue at hand, analysts can reuse each pattern to analyze requirements in various domains and contexts. For modeling such requirements and the desired security qualities, the STS-tool, that is based on modelling goal-oriented security requirements to describe sociotechnical systems, is used. As we described, this tool includes three views of which we mainly use the social view to translate the main goals and tasks of stakeholders with their desired security requirements using our chosen patterns. The social view can describe numerous security needs of which we

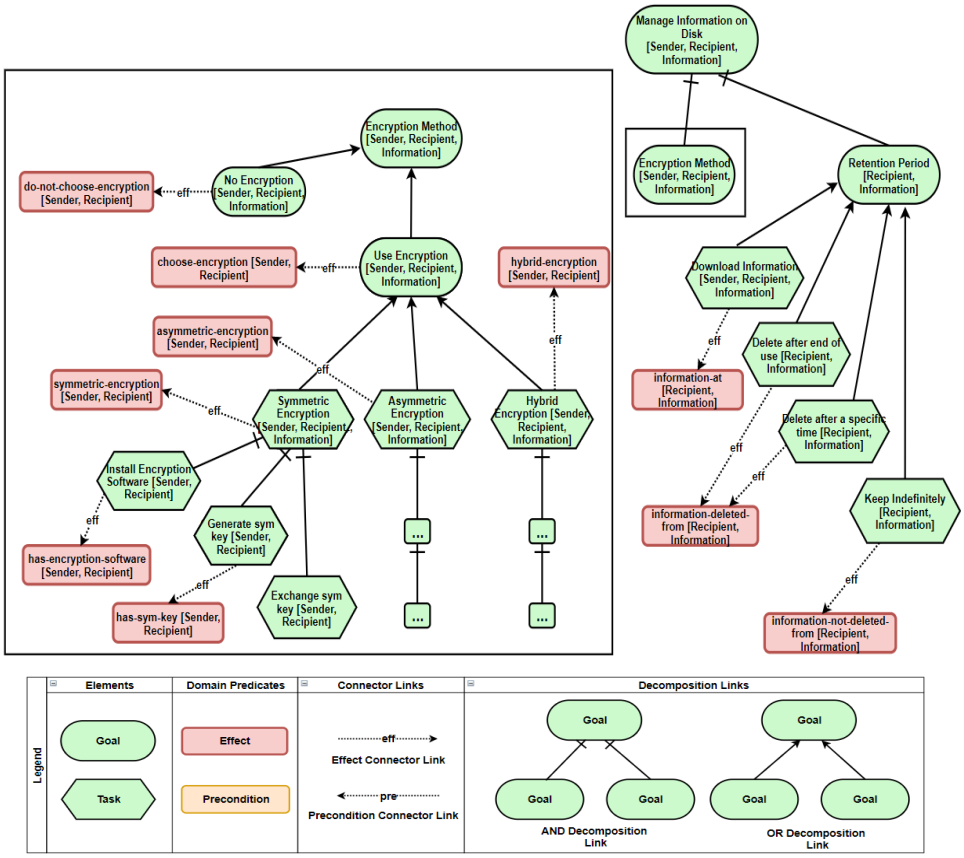

Figure 3.8: Manage information on disk pattern presented using i^* 2.0 elements

choose confidentiality, integrity, authentication, and non-repudiation that we propose a solution to through our catalogue.

The example of the translation company is depicted in Figure 3.9. The secretary is responsible for managing customers requests, sending invoices, and receiving payments. Security requirements can be attached to documents being transmitted between actors. As shown in the Figure, the invoice document has the labels "Con", "Int", and "Auth" to denote the need for confidentiality, integrity, authentication respectively, and the label "Non-Rep" under the goal payment received to denote the need for non-repudiation, which translates in this case that payments should not be denied once completed. Threats are also modelled through events that threaten documents. In this Figure, information sniffing and tampering threaten the

invoice document, while repudiation threatens both goals “send invoice” and “payment received”.

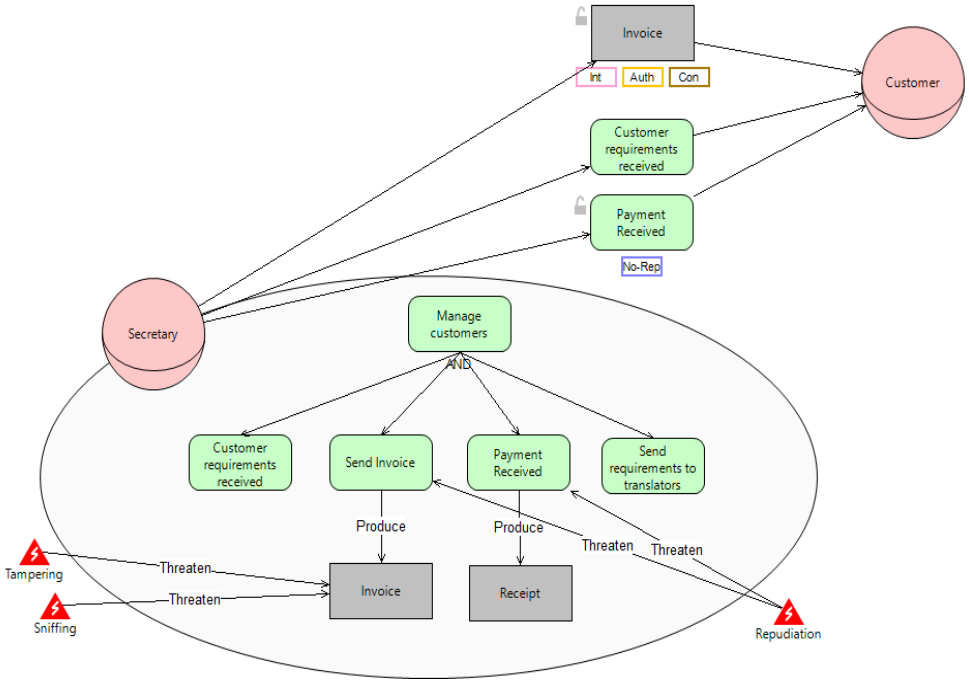

Figure 3.9: Goal modelling and document transfer using the STS-Tool

After modelling main business processes, security patterns are attached to those processes as shown in Figure 3.10 where sending an invoice from the secretary to the customer requires the pattern “Transmit Information”. The variables of [Sender, Recipient, Information] will be replaced by “Secretary, Customer, Invoice] respectively. Similarly, making a payment from the customer’s side will require the same pattern. By using the expertly developed pattern for goals that entail information transmission, the developer of the model can be confident that whatever security controls and steps included there are properly implemented and interwoven with the other tasks.

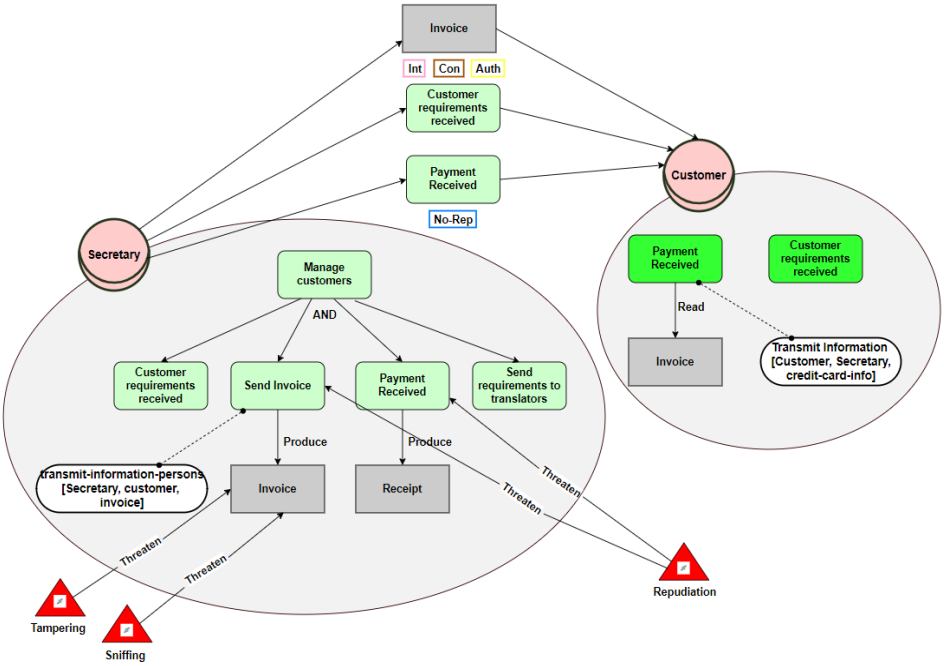

Figure 3.10: Integrating security patterns using the STS-Tool.

3.5 Attack Trees as negations of security requirements

While workflow patterns describe secure (or less secure) ways by which domain actors can achieve common goals, an additional construct will be necessary to model steps that an attacker can take in order to breach the stated security requirements. In fact, we can view major security concerns modeled in STS-ml (confidentiality, integrity, authentication, etc.) as negated attacker goals. Such attacker goals, such as confidentiality in Figure 3.10, can be decomposed in a form of an attack tree [22] or (elsewhere known as an anti-goal) as shown in Figure 3.11 using the i^* elements. The attack tree contains goals and tasks that an attacker would like to accomplish to breach the security controls of the victim. These elements are marked in red to distinguish an attacker from a regular actor. The goals

list the options that are available to the adversary to gain access over confidential data. Some of those goals contain dashed boxes to denote the desires of the attacker wished to be obtained through the actions of actors, such as acquiring confidential data from the user's mailbox after breaching a particular security control (e.g. "is-at [Sender, mailbox, Document]"), and solid ones to indicate goals that an attacker pursue actively.

Figure 3.11 describes how a breach in confidentiality can occur by either accessing the sender or receiver's conversation. The attacker would attempt to either compromise the network, an application (e.g. mailbox), or a device, and attempt to steal and decrypt information by eavesdropping keys. Eavesdropping keys is further described in Figure 3.12 to contain three possible ways an adversary can acquire keys. If the the adversary was successful in carrying out such activities, then the effect "confidentiality-breached (Sender, Recipient, Document)" is considered true. On the other hand, when an attacker goes one step further to temper the stolen information, a breach in confidentiality as well as in integrity occurs, as shown in Figure 3.13.

Moving to the third security requirement in the STS-tool, authenticity, presented in Figure 3.14, if the adversary knows that Alice and Bob communicate or would like to communicate over the network, a simple failure in sending, confirming, or certifying keys can lead to a breach of this kind. For instance, the adversary can generate a private/public key pair and claim that he is Bob, and if Alice accepts the keys without a secure key exchange method or without confirming Bob's identity, the attacker succeeds in user falsification. Whereas, if the keys were to be stolen from Bob for example, the attacker can generate any information of his choice and sends it to Alice as Bob, therefore breaching their authentication by user impersonation, i.e. stealing the identity of a legitimate user.

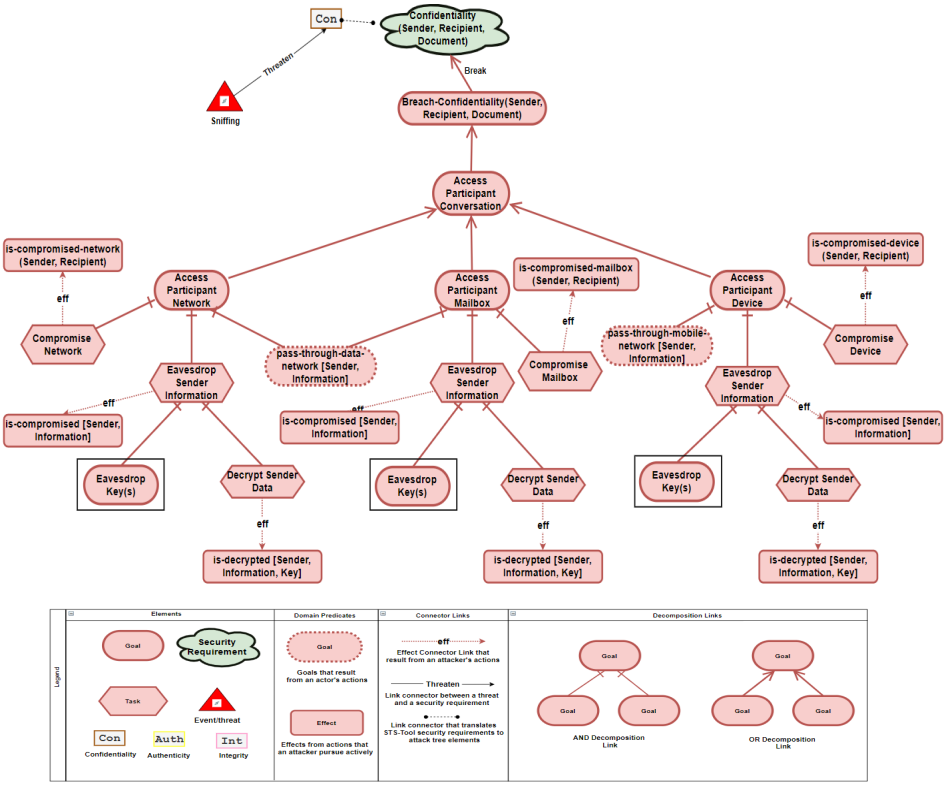

Figure 3.11: A breach in confidentiality using anti-goal decomposition.

3.6 Reasoning about Security Workflows

With attack trees at hand, we can now describe domain-specific requirements of business processes in relation with attack trees to show the effects of attacker's actions on security requirements. The usage of attack trees allows the analysis and the selection of alternative workflows within a specific business process (e.g. exchange document) to satisfy certain security requirements by blocking the adversary's actions from the attack tree. To reason about such activities, we utilize AI planning. Business processes and attack tree patterns are translated into planning domain and problem specifications to allow the planner to search for workflows that satisfy security requirements while avoiding the satisfaction of the root goal of the attack tree.

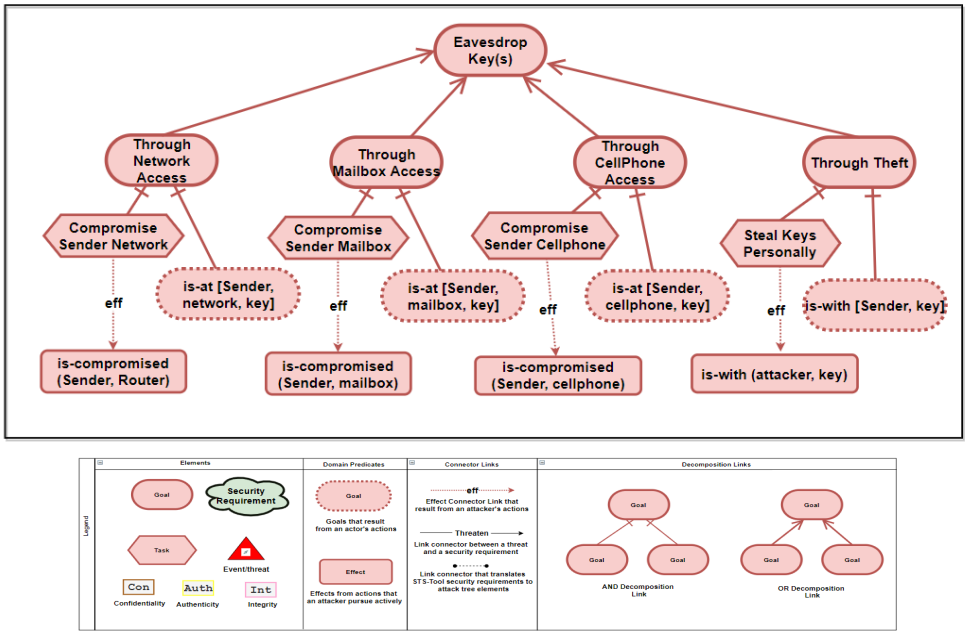

Figure 3.12: Eavesdrop attack pattern using anti-goal decomposition.

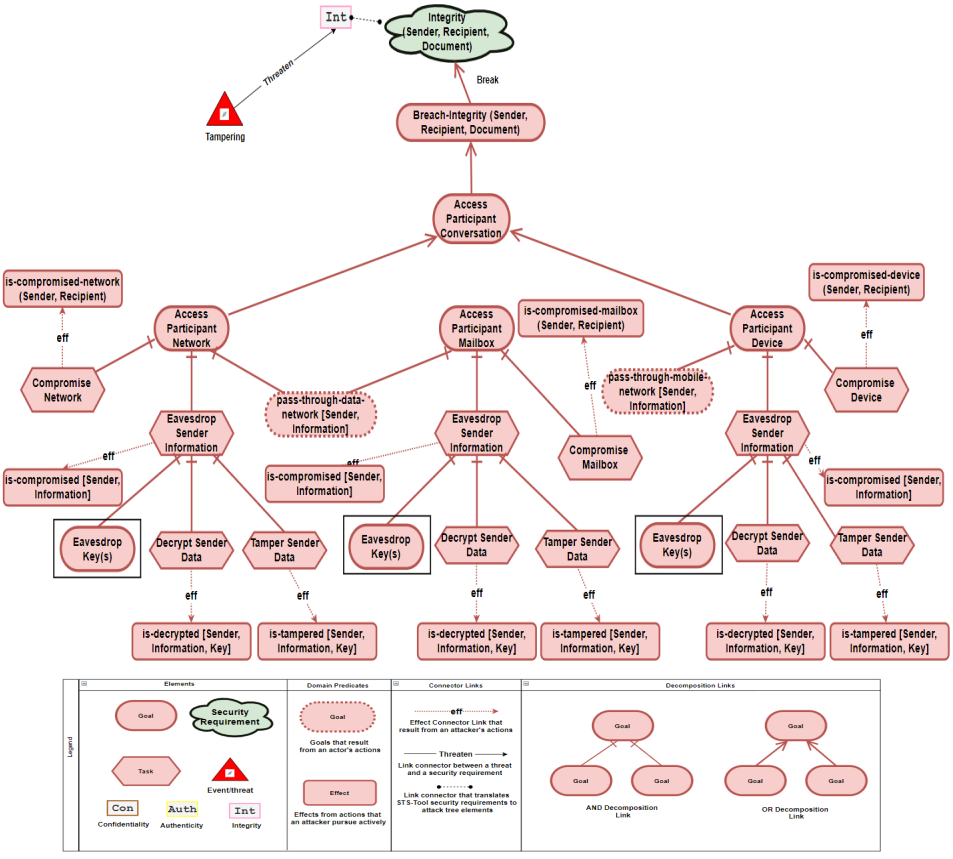

Figure 3.13: A breach in integrity sing anti-goal decomposition.

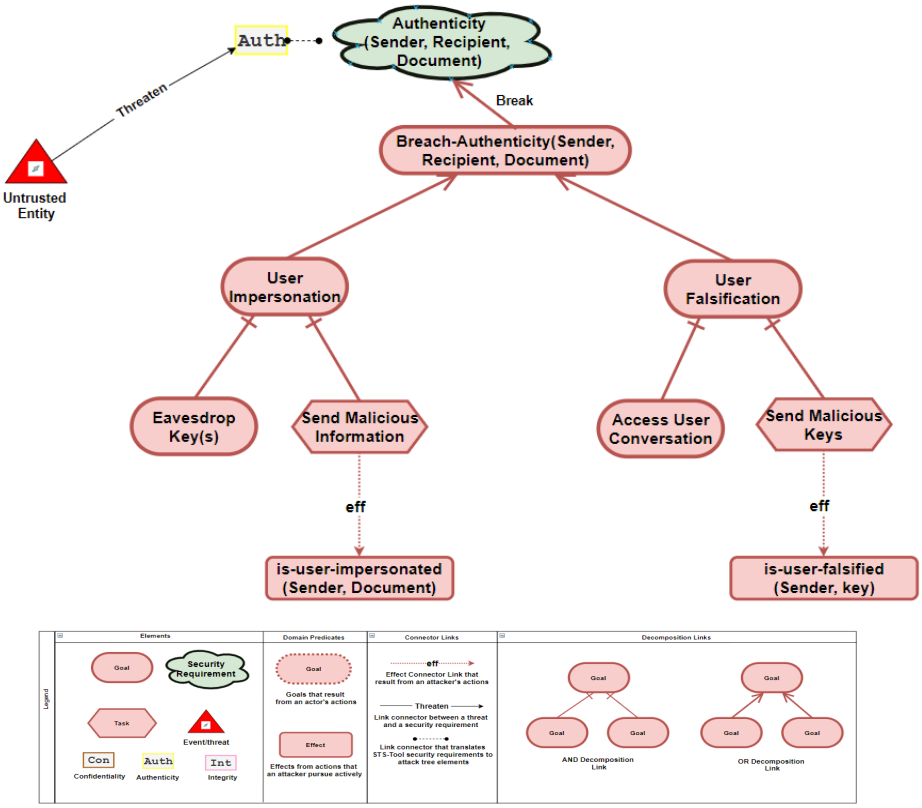

Figure 3.14: A breach in authenticity sing anti-goal decomposition

3.6.1 AI Planning

Artificial Intelligence planners are automated reasoning tools that read an action theory specification and generate sequences of actions that allow the accomplishment of a goal according to the action theory, and the description of any initial state. In this thesis, we adopt Hierarchical Task Network (HTN) planners, specifically SHOP2 [57]. In HTN the input is given in the form of a domain specification and a problem specification.

Domain specifications consist of actions (also called as *operators*) and objects (e.g. invoice, public-key, user, etc.), as well as constructs that specify the means to achieve goals (also called *methods*). Domain predicates are n-ary relations over objects that describe the object's properties (e.g. has (customer public-key)). Domain predicates can have multiple forms that

range from preconditions, deletion lists, to addition lists. Preconditions denote that domain predicates must be true for the action or goal to be satisfied, deletion lists indicate deleted domain predicates with their respective objects as opposed to addition lists. Thus, domain predicates are used to describe the state of the domain, and their turning true or false as a result of action performance signifies changing of one state to another.

Operators describe *primitive* tasks, which cannot be decomposed further, that must be performed for the completion of a goal or method through domain predicates. Methods (also referred as *compound* tasks), on the hand, describe the standard operations that one would like to perform in a particular domain as they are characterized by the way users think about the problem. They are decomposed recursively into tasks and subtasks, whereby a method can include one or more methods. However, the satisfaction of the main method requires the completion of its relative primitive and compound tasks.

It's also relevant to mention that operators, unlike methods, include an additional optional predicate that allow the specification of the task cost if desired. By default, the task cost equals to 1 if no cost is mentioned. We specifically assign non-zero cost to security-related tasks. Through such weights, the planner will be biased towards identifying workflows that include as few security-related tasks as possible, while satisfying security requirements and vulnerability assumptions.

Consider the example of an operator that describes the action of walking from Keele to Glendon campus. A precondition for walking could be to have a good weather, the deletion list would include the old location (keele campus) to indicate it's deletion from the current state of the operator, while the addition list would include the new location (Glendon campus)

to final state of the operator. A presentation of this example is shown below, whereby the words that begin with an exclamation mark describe the task name, the words that start with an interrogation mark are variables that can be replaced depending on the context, the words that contain only letters are constants, and words followed by a semi-colon are comments:

```
(:operator (!walk ?loc1 ?loc2)
           ((good-weather))    ; Precondition(s)
           ((is-at ?loc1))     ; Delete list
           ((is-at ?loc2))     ; Addition list
           5                   ; Task Cost
)
```

An example of a method can be to travel from Toronto to Ottawa, whereby the traveler would walk from home to the subway station to take a train to Ottawa. A precondition of such method would be to have money and a good weather. This method is presented with it's respective operators as follows:

```
(:operator (!walk ?home ?subway)
           ((good-weather))
           ((is-at ?home))
           ((is-at ?subway))
)

(:operator (!ride-train ?Toronto ?Ottawa)
           ((have-money))
           ((is-at ?Toronto))
           ((is-at ?Ottawa))
)


(:method (Travel ?Toronto ?Ottawa)
         ()                                ; Precondition(s)
         (!walk ?home ?subway)             ; Task-1
         (!ride-train ?Toronto ?Ottawa))   ; Task-2
```

```
)
```

HTN also includes the notion of *axioms*, that is an expression described through a *head* and a *tail*, whereby the head is true if the tail is true. Axioms can contain conjunctions, dis-junctions, numerical expressions, and external function calls. For instance, a head could describe the distance that could be walked when a weather is good or bad. Thus, if the tail that states the weather good is true, the distance that can be walked is up to 5 miles. Similarly, when the tail that states the weather is bad is true, the distance that can walked is reduced to 1 mile. This example is presented as follows:

```
(:- (walking-distance ?loc1 ?loc2)
     good ((good-weather)
           (distance ?loc1 ?loc2 ?distance)
           (call <= ?distance 5))
     bad ((distance ?loc1 ?loc2 ?distance)
           (call <= ?distance 1)))
```

Problem specification, on the other hand, consists of defining the initial state of objects in the domain (if any exists), such as "good-weather" and "distance Keele Glendon 3", to indicate that the weather is good, and that the distance between the locations is 3 miles, as well as a goal formula that is described through a method (e.g. travel Keele Glendon). Defining the problem domain starts with the function "*defproblem*" followed by the problem name and the script name prior defining the initial state and the goal name. An example of such representation is shown as follows:

```
(defproblem problem1 simple-example
    ((good-weather)(distance Keele Gelendon 3))
    ((travel Keele Gelendon)))
```

The planner executes plans by reading the function *find-plans*, which is used in this thesis, or *do-problems* followed by a mandatory argument for both functions which is the plan name, and other mandatory/optional arguments depending on which function is used. By default, SHOP2 prints the plan with the least cost, i.e. the plan with the least number of tasks. The most used optional arguments in our plans include the arguments "`:verbose`" and "`:plans`" to print the plan(s) found rather than just listing the number of plans, and "`:which :all`" to print all the available plans rather than just printing the plan with the least cost. Additionally, the planner allows the selection of plans based on a number specified by the user of the reasoner through the argument "`:optimize-cost`", that forces the planner to print the first plan obtained which no other plan has a lower total cost. An example of each representation is presented below:

```
1: (find-plans 'problem1 :verbose :plans)

2: (find-plans 'problem1 :verbose :plans :which :all)

3: (find-plans 'problem1 :verbose :plans :which :all :optimize-cost 50)
```

When SHOP2 executes the plan, it returns several values, such as the number of plans and the CPU time taken to generate the plans, etc., as well the description of each plan. A full example of finding a plan to go from Keele to Gelendon campus is presented below with its respective plan result:

```
(in-package :shop-user)
(defdomain simple-example (

 (:operator (!walk ?loc1 ?loc2)
            ((good-weather))
            ((is-at ?loc1))
            ((is-at ?loc2)) )

(:method (travel ?x ?y)
         ((walking-distance ?x ?y))
```

```
        ((!walk ?x ?y)))

(:- (walking-distance ?loc1 ?loc2)
     good ((good-weather)
           (distance ?loc1 ?loc2 ?distance)
           (call <= ?distance 5))
     bad ((distance ?loc1 ?loc2 ?distance)
           (call <= ?distance 1)))

(defproblem problem1 simple-example
    ((good-weather)(distance Keele Gelendon 3))
    ((travel Keele Gelendon)))

(find-plans 'problem1 :verbose :plans)

 Plans   Mincost   Maxcost   Expansions   Inferences   CPU time   Real time
   1       1.0       1.0          3            7         0.001      0.001

Plans:

(((!WALK KEELE GELENDON)))

0.001
```

In our framework, we translate high level and intermediate goals, such as transmitting a document, into methods, and actions into tasks, such as transmitting an encrypted document, and map vulnerability assumptions of attack trees through axioms. The former includes the necessary steps to establish the main goal through a series of actions. For example, on of these actions may require the user to send an encrypted document, whereby a precondition for this action can be to have the required keys generated, the deletion list will be empty, as transmission of a document does not turn any domain fact false, and the add list will include a predicate that signifies that the recipient now has the document.

Table 3.1: Goal model elements translation into HTN constructs

HTN Constructs / Goal Models	Methods	Operators	Preconditions	Delete Lists	Addition Lists	Axioms
Goals	X					
Actions		X				
Security Requirements					X	
Threats					X	
Preconditions			X			
Effects					X	
Vulnerability Assumptions						X

3.6.2 Developing Planning Domain Specifications

We now turn to how we translate security requirements, workflow patterns, attack trees, and qualities into HTN specifications, so that SHOP2 can offer us plans that satisfy the security requirements while also avoiding successful attacks. In short, the visual representation of top goals in the social view are translated to methods, while leaf level tasks are translated into HTN tasks. High-level security pattern goals are also translated into HTN methods with their respective preconditions and effects that result from the addition list after completing the necessary tasks and sub-tasks. Vulnerability assumptions and attacker's actions are translated using axioms, while security requirements are translated using addition lists of operators. A summary for these translated elements is shown in Table 3.1. We now present the translation procedure in more detail.

3.6.3 Translating Workflow Patterns

Translating security patterns into HTN constructs can be shown through the example of the translation company presented earlier. Firstly high-level (non-leaf) goals are translated into one or more HTN methods. For example, the top-level goal "Manage-customers" of the secretary in Figure 3.10 includes several tasks that describe unique business processes. This top-level goal is translated into an HTN method as follows:

```
(:method (manage-customers ?secretary ?customer ?documents)

         ()

         ((!requirements-received ?secretary ?customer ?documents)

         (!send ?secretary ?customer ?invoice)

         (!payment-received ?secretary ?customer)

         (!send-requirements ?secretary ?translators ?documents))

)
```

Depending on how the goal is decomposed, the method(s) is constructed in a different way. In general, goals that are AND-decomposed are translated as methods that contain HTN tasks, which, in turn, consist of other methods or operators. Those other methods or operators are, likewise, translations of the sub-goals or sub-tasks of the goal being translated. Goals that are OR-decomposed are translated as in as many methods as the sub-goals of the goal in question. Each such method contains a task with only one element, which is a method or operator resulting from translating the sub-goals of the OR-decomposed goal.

An example of the AND-decomposed goals is illustrated through the "Setup Security Tools" goal is Figure 3.3, where the identification of the encryption method takes place. Since the completion of this goal requires the completion of three sub-goals ("Identify User Encryption Type, "Key Exchange", and "Key Backup"), the former is translated to an HTN method along with the sub-goals, however, including the latter as compound tasks (i.e. methods) under the primary goal. This example does not only show the translation AND-decomposed goals, but it also presents an example of translating goals that are recursively decomposed into other goals into methods that are defined through task lists recursively decomposed into other tasks lists, an outcome which we refer to here as nesting. This representation is shown as follows:

```
(:method (setup-security-tools ?sender ?recipient ?info)

         ((identify-user-encryption-type ?sender ?recipient ?info)

         (key-exchange ?sender ?recipient)

         (key-backup ?sender ?recipient))

)
```

An example of the OR-decomposed goals, on the other hand, is illustrated through the goal "transmit-information [Sender, Recipient, Information]" shown in Figure 3.5. This goal is further decomposed into two other goals through an OR-decomposition link, in which information can be sent unencrypted or encrypted depending on the precondition that is satisfied, translated into HTN methods as follows:

```
(:method (transmit-information ?sender ?recipient ?info)
         ()
         (transmit-unencrypted-information ?sender ?recipient ?info))
)


(:method (transmit-information ?sender ?recipient ?info)

         ()

         (transmit-encrypted-information ?sender ?recipient ?info))

)
```

In the example above, "`(transmit-unencrypted-information ?sender ?recipient ?info)`" and "`(transmit-encrypted-information ?sender ?recipient ?info)`" are also methods. In case, the precondition of sending encrypted information is satisfied, alternate methods are present to describe either the use of symmetric or asymmetric encryption by translating their leaf level tasks into primitive HTN tasks, such as in the example of symmetric encryption presented as follows:

```
 (:method (transmit-encrypted-information ?sender ?recipient ?info)

          ()
```

```
        (!encrypt-using-shared-symmetric-key ?sender ?info)

        (!send-encrypted-information ?sender ?recipient ?info)

        (!decrypt-using-shared-symmetric-key ?recipient ?info)

)
```

This time, the task in the method include operators, recognized by the exclamation mark "!" in front of the name.

Leaf-level tasks of the workflow patterns are translated into such operators and used in methods as above. As we saw, operators have a precondition, an addition list and a delete list. These are added as follows:

- If a precondition element in the graph is pointing towards the task, the content of the precondition element is added as a precondition.
- If the task is pointing towards an effect element through an effect link, then the content of the effect link is added in the addition list.
- The delete list is generally left empty.

Going back to our example, for the task "`(!encrypt-using-shared-symmetric-key ?sender ?info)`" to be valid, the sender and the receiver (i.e. the secretary and the customer) must have generated and shared the required symmetric key. The requirement of having obtained the required key is translated through the precondition list of the first primitive task, whereas the effect of having an encrypted document once the task is completed is shown through the addition list as follows:

```
(:operator (!encrypt-using-shared-symmetric-key ?sender ?info)

        ((has ?sender sym-key ?info))

        ()
```

```
        ((has ?sender (encrypted ?info (key ?sender symmetric))))
)
```

The process of identifying which encryption method to be used for information transfer, where the necessary keys need to be generated and shared, is described in another pattern showing how users setup their security tools through the "setup-security-tools [Sender Recipient]" goal, translated into HTN method, as shown in Figure 3.3. Thus, for information transfer to be complete, users must have installed the necessary security tools. Such integration can be seen using the goal "`manage-customers ?secretary ?customer ?documents`" of the secretary as follows:

```
(:method (manage-customers ?secretary ?customer ?documents)
        ()
        (!requirements-received ?secretary ?customer ?documents)
        (!send ?secretary ?customer ?invoice)
        (setup-security-tools ?secretary ?customer ?invoice)
        (transmit-information ?secretary ?customer ?invoice)
        (!payment-received ?secretary ?customer)
        (!send-requirements ?secretary ?translators ?documents)
)
```

3.6.4 Translating Vulnerability Assumptions and Attack Trees

Let us know describe the translation of vulnerability assumptions and attack trees using axioms. Attack trees describe vulnerabilities that attackers managed to compromise to gain advantage of sensitive information that users want to protect. Vulnerability assumptions are parts (effects) of the

attack tree that are considered to be possible in the application context. Vulnerability assumptions are, thus, effects that the analysis assumes that attackers are motivated enough to bring about. For example, the attacker's successful physical access to a target's phone or computer, despite being a step that could bring about a security requirement breach, it may or may not be assumed to be as likely depending on the application context. Nevertheless, the success of an attack is bound to the actions taken by actors, such as using email services to transmit information. Consequently, actors transferring sensitive information can decide which vulnerability assumption is true according to the business process being carried out, such as the assumption of a compromised network or a hacked device.

We translate attack trees into HTN axioms. The head of the axiom describes the security requirement that the attack compromises. The tail is an AND/OR formula consisting of the leaf level effects of the attack tree that need to be brought about for the root of the attack tree to be satisfied. As we saw, these are the results of user actions combined with actions taken by the attacker (the vulnerability assumptions), as described in the attack tree.

Four different possible attacks were introduced through attack trees: (a) an attacker can access a participant's network to steal sensitive information being transferred through internet network, (b) an attacker can access a participant's mobile network to steal sensitive information being transferred over mobile network, (c) an attacker can compromise a participant's application or device to access sensitive information, and (d) an attacker can physically assault a participant to steal sensitive information. For the ease of description, we provide a simplified description of this translation into HTN axioms for each of the mentioned attacks as follows:

(a) The data network is compromised:

```
(:- (interception-successful ?sender ?recipient ?info)
    (is-compromised-data-network ?sender ?recipient)
    (pass-through-data-network ?info))
```

The axiom denotes that key information transfer between participants happens through data network, and that such network is compromised by the attacker.

(b) Likewise for the mobile network:

```
(:- (interception-successful ?sender ?recipient ?info)
    (is-compromised-mobile-network ?sender ?recipient)
    (pass-through-mobile-network ?info))
```

The axiom denotes that key information transfer between participants happens through mobile network, and that such network is compromised by the attacker,

(c) The application/device is hacked:

```
(:- (interception-successful ?sender ?recipient ?info)
    (is-hacked ?sender device) (is-at ?sender device ?info))
```

The axiom denotes that the application/device where the key resides is compromised.

(d) The attacker is following a participant to physically steal information:

```
(:- (interception-successful ?sender ?recipient ?info)
    (is-followed ?sender)
    (shared-in-person ?sender ?recipient (key ?sender shared))
```

The axiom denotes that key information transfer between participants is shared in person, and that and that the participant is being followed by an attacker who aims to steal such information physically. Thus, that interception is deemed successful when any of the tails of the axioms become true.

3.6.5 Translating Qualities

To translate qualities, such as cost, convenience, and performance into HTN, we created additional operators and methods that translate these qualities using the negative/positive graphical links of qualities presented in the i^* 2.0 language. For instance, given a plan that contains no encryption in it's activities will impact performance positively. Such quality can be presented using the "make" link from no encryption to performance which can be also labeled as "++" sign for it's positive impact. Accordingly, the quality will receive quality points based on the number of negative/positive signs. Consequently, the user of the reasoner can compare plans according to total quality points.

Figure 3.15 shows a simplified example of modelling such qualities according to different encryption methods using the piStar tool. For instance, avoiding encryption during information transfer has a positive impact on cost and performance (make link), and a negative impact on convenience (break link).

To define such qualities in HTN, operators are used to define the mathematical computation of such values. Since three qualities are being presented in this framework, one operator has to be defined for each quality. The operator will define the variable of the number of performance points that

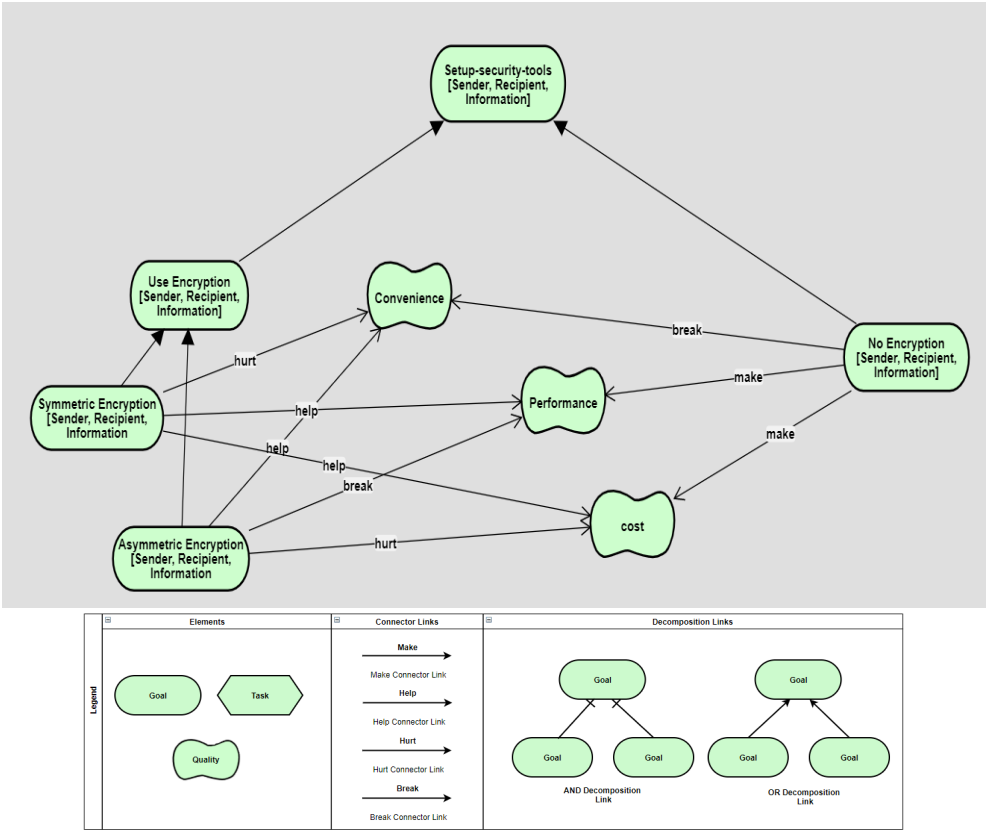

Figure 3.15: A simplified pattern of quality illustration

the planner has to start with in the preconditions list, and the variable regarding the number of performance points at the end of the computation by removing the initial value specified in the delete list. An example of a quality operator is presented below:

```
(:operator (!performance-points ?old ?new)

           ((have-performance ?old))

           ((have-performance ?old))

           ((have-performance ?new))

)
```

After defining operators, methods have to be also defined for each quality to specify the number of increments/decrements depending on the quality type. Consider the "make performance" quality that indicates an increment of 2 points every time this quality is used. The method requires that a

number of performance points has to be specified in the initial state of the problem domain, such as zero, presented as a precondition for the method, and the number that needs to be incremented in the operator's header. This example is presented as follows:

```
(:method (make-performance ?p)

        ((have-performance ?p))

        ((!performance-points ?p (call + ?p 2)))

)
```

After defining the necessary operators and methods, qualities can be attached to necessary security patterns. Take the example of performance and using no encryption during the information transfer. Avoiding any type of encryption automatically denotes that the information transfer is faster than using one. To translate such representation, the "make-performance ?p" method is nested under the method that indicates what type of encryption should be used when setting up the security tools between the sender and the receiver, such as in the example below:

```
(:method (setup-security-tools ?sender ?recipient ?info)

        ()

        (!encryption-method-is-no-encryption ?sender ?recipient ?info)

        (make-performance ?p)

)
```

The process of defining qualities, however, is not complete yet. The defined operators and methods only specify the number of points that should be incremented or decremented. An additional operator and method is required for each quality to display its total number points at the end of each plan, such as the case of performance below:

```
(:operator (!total-performance-points ?new)

           ()

           ()

           ((print-performance-points ?new))

)
```

To further give users insights about the plans that may be more relevant in some situations, we also include the total sum of quality points. Such representation requires an additional operator and method that follow the same logic of defining qualities and calculating their increments/decrements, as shown below:

```
(:operator (!total-quality-points ?old ?new)
           ((have-quality ?old))
           ((have-quality ?old))
           ((have-quality ?new))
)


(:method (quality-points ?q ?cost ?c ?p)

         ((have-quality ?q)

         (have-cost ?cost)

         (have-convenience ?c)

         (have-performance ?p))

         ((!total-quality-points ?q (call + ?q ?cost ?c ?p)))

)
```

Since HTN translates business workflows and security patterns to provide secure business processes, the planner, at this point, will generate all the possible plans of information transmission, which includes using and not using encryption - Thus, to translate the security requirements presented in the STS-tool, such business processes must be accompanied by security constraints that guides the planner into generating secure plans according

Table 3.2: A summary of HTN constructs presented in the initial state

HTN Argument Type	HTN Arguments	Explanation
Domain Fact	willing-to-install-software ?user	The user is willing to install the encryption software
Domain Fact	willing-to-generate-keys ?user	The user is willing to generate the required encryption keys
Domain Fact	willing-to-pay ?user certificate	The user is willing to pay to obtain digital certificates
Domain Fact	can-meet ?sender ?receiver	The sender and the receive can meet in person
Domain Fact	knows ?sender ?receiver	Users know each other and have exchanged contact information
Domain Fact	2-or-less ?sender ?receiver	The number of communicating members is less than 2
Domain Fact	more-than-2 ?sender ?receiver	The number of communicating members is greater than 2
Domain Fact	has ?user encryption-software	The user has already installed an encryption software
Quality Predicate	have-convenience 0	The number of quality points associated with convenience start from 0
Quality Predicate	have-cost 0	The number of quality points associated with cost start from 0
Quality Predicate	have-performance 0	The number of quality points associated with performance start from 0
Quality Predicate	have-quality 0	The number of quality points associated with quality start from 0
Vulnerability Assumption	is-compromised-data-network	The data network has been compromised by the attacker
Vulnerability Assumption	is-compromised-mobile-network	The mobile network has been compromised by the attacker
Vulnerability Assumption	is-hacked ?user phone	The user's phone has been compromised
Vulnerability Assumption	is-followed ?user	The user is being followed by a potential attacker

to the user's security requirements. We do so at the level of planning problem specification as we discuss next.

3.6.6 Security requirements as plan constraints

We discussed above how we produce the domain specification. We now turn to how we translate the integration of security requirements and vulnerability assumptions in planning problem specification as plan constraints.

3.6.7 Initial State

The initial state of problem specification addresses two main parts: the facts that describe domain predicates and vulnerability assumptions. Domain predicates can address security related concerns in the form of vulnerability assumptions (e.g. hacked [secretary phone]) and non-security concerns (e.g. willing-to-install-software [secretary]).

As described earlier, vulnerability assumptions are part of attack trees, and users of the reasoner can decide which vulnerability assumptions would be meaningful depending on the context. Table 3.2 provides a summary for HTN constructs presented in the initial state of problem specification.

Consider the example of transferring a document between the secretary and a customer, the initial state of the problem specification would include: (a) domain predicates, such as specifying the actors willingness to install and generate keys as well as a domain predicate specifying the number of participants, (b) vulnerability assumption(s) that actors deem to be true, such as a hacked mobile device, and (c) quality predicates to specify the starting point of each quality as follows:

(1): `(willing-to-install-software secretary)`

(2): `(willing-to-install-software customer)`

(3): `(willing-to-generate-keys secretary)`

(4): `(willing-to-generate-keys customer)`

(5): `(2-or-less secretary customer)`

(6): `(can-meet customer secretary)`

(7): `(can-meet secretary customer)`

In the above domain predicates, we describe a context in which there are two roles "secretary" and "customer" and they are both willing to install encryption software and generate keys, and are also able to meet (are e.g., in the same city) in order to securely exchange keys.

(b) Vulnerability assumption construct:

(8): `(is-hacked secretary phone)`

In the above vulnerability assumption we assume that the secretary's phone is hacked, meaning that all activities taking place through that phone can be compromised (intercepted, changed etc.). In other words we want the reasoner to generate workflows that allow achievement of operational goals such that security requirements are guaranteed despite this assumed compromise.

(c) Quality points constructs:

(9): `(have-convenience 0)`

(10): `(have-cost 0)`

(11): `(have-performance 0)`

(12): `(have-quality 0)`

The elements presented in the initial can change how plans are generated depending on what actions the actors are willing to take, and which vulnerability assumptions are considered true.

3.6.8 Goal Formula

The final goal or method that analysts would like to reach can be decomposed into two parts: An operational part and a security part. The former describes the necessary business actions that should be taken in a specific domain, whereas the latter describes what must be true in the end of the process to preserve the security of the goal and avoid any breaches.

HTN planners only address goals through methods without the ability to specifically define plan constraints otherwise. To achieve the conjunction between domain tasks and security conditions, a new task is introduced within the main method that contain the security breaches users would like to have negated. This task includes security constraints as preconditions to deny the effects of the attack tree. This way the top level security method won't be achieved unless the latter is successfully completed. Table 3.3 provides a summary for HTN constructs presented in the goal formula of problem specification.

HTN constructs of problem specification vary from one domain to another

Table 3.3: A summary of HTN constructs presented in the goal formula.

HTN Argument Type	HTN Element Type	HTN Argument	Explanation
Security Requirement	Operator	(!security-requirement)	Dummy task containing the necessary security requirements as negation of domain predicates
Security Requirement	Precondition	(info-may-not-be-sniffed)	The information being transferred between participants should not be sniffed (stolen) by an attacker
Security Requirement	Precondition	(info-may-not-be-tampered)	The information being transferred between participants should not be tampered by an attacker
Security Requirement	Precondition	(info-may-not-be-repudiated)	The information being transferred between participants should not be repudiated by an attacker
Security Requirement	Precondition	not (interception-successful)	The information being transferred between participants should not be intercepted by an attacker
Quality	Method	(total-convenience-points ?c)	The method that prints the total number of convenience points at the end of each plan
Quality	Method	(total-performance-points ?p)	The method that prints the total number of convenience points at the end of each plan
Quality	Method	(total-cost-points ?cost)	The method that prints the total number of cost points at the end of each plan
Quality	Method	(quality-points ?q ?cost ?c ?p)	The method that prints the total sum of quality points at the end of each plan

and depend on the requirements users would like to achieve. Consider the example of the translation company. Assuming that the network is compromised by the attacker (vulnerability assumption), the secretary and the customer have to setup the necessary security tools before exchanging documents to prevent a breach in integrity (which is the security requirement). The problem specification of this example would include: (a) domain facts to denote the willingness of the participants to install an encryption software, generate and share they necessary keys, and the number of participants, (b) vulnerability assumptions to indicate that the network is compromised by the attacker, and (c) quality points to allow the user of the reasoner to analyze plans according to qualities if desired. Given such requirements, the HTN constructs used in the initial state are as follows:

(a) Domain facts constructs:

(1): `(willing-to-install-software secretary)`

(2): `(willing-to-install-software customer)`

(3): `(willing-to-generate-keys secretary)`

(4): `(willing-to-generate-keys customer)`

(5): `(2-or-less secretary customer)`

(6): `(knows customer secretary)`

(7): `(knows secretary customer)`

(b) Vulnerability assumption construct:

(8): `(is-compromised-data-network secretary customer)`

(c) Quality points constructs:

(9): `(have-convenience 0)`

(10): `(have-cost 0)`

(11): `(have-performance 0)`

(12): `(have-quality 0)`

The goal formula, on the other hand, should include: (d) the method or goal that participants are trying to achieve, (e) the dummy task containing the necessary security requirements, and (f) the quality methods for the necessary computation of the quality points. Given such requirements, the HTN constructs used in goal formula are as follows:

(d) Method or goal to be achieved:

(13): `(setup-security-tools secretary customer invoice)`

(14): `(transmit-information secretary customer invoice)`

(e) Dummy task containing the necessary security requirements:

(15): `(!security-requirements secretary customer)`

(f) Quality methods for the necessary computation of the quality points:

(16): `(total-convenience-points ?c)`

(17): `(total-performance-points ?p)`

(18): `(total-cost-points ?cost)`

(19): `(quality-points ?q ?cost ?c ?p))`

The dummy task in point (15) is described in the domain specification to contain security requirements as negation of: (a) potential breaches by the attacker, and (b) a negation of the attack axiom to denote that the user would like address the vulnerability at hand. Such representation is shown

as follows:

```
 (:operator (!security-requirement ?sender ?recipient ?info)

            (((info-may-not-be-tampered ?sender ?recipient ?info))

            (not (interception-successful ?sender ?recipient ?info)))

            ()

            ()

)
```

Combining all the HTN constructs will lead to the complete representation of the problem specification as follows:

```
(defproblem p1 translation-company

 ((willing-to-install-software customer)

 (willing-to-install-software secretary)

 (willing-to-generate-keys customer)

 (willing-to-generate-keys secretary)

 (knows customer secretary) (knows secretary customer)

 (2-or-less secretary customer)

 (is-compromised-data-network secretary customer)

 (have-convenience 0) (have-cost 0)

 (have-performance 0) (have-quality 0))

((setup-security-tools secretary customer invoice)

 (transmit-information secretary customer invoice)

 (!security-requirements secretary customer)

 (total-convenience-points ?c) (total-performance-points ?p)

 (total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))
```

The dummy security requirements task along with the vulnerability assumptions in the initial state with the domain facts will force SHOP2 to generate plan(s) that meet the user's requirements. Optionally, users can

take advantage of the cost of tasks explained previously, allowing the planner to filter plans according to specific criteria. Such feature guides the user into selecting plans that only seem reasonable or affordable to practically implement. For instance, a security analyst might assign a cost for choosing a symmetric encryption plan, while assigning a higher cost to an asymmetric encryption plan since the latter requires more effort to complete the secured task. Through such weights, the planner will be biased towards identifying workflows that include as few security-related tasks as possible, while satisfying security requirements and vulnerability assumptions.

3.6.9 Reasoning about Security Workflows

As discussed in the previous sections, the planner will generate plans according to security requirements given vulnerability assumptions and other domain facts. We now offer some examples of how plans are generated given different inputs and how they are interpreted.

Assume that the user of the planner did not define any security requirements or vulnerability assumptions during a document transfer. The problem specification is presented with it's respective results as follows:

```
(defproblem p1 translation-company
    ()
    (transmit-information secretary customer invoice))
```

The plan is generated as follows:

```
{1:  (!encryption-method-is-no-encryption secretary customer)
 2:  (!send-unencrypted-information secretary customer invoice)}
```

Thus, the plan with the cheapest cost will be generated with no security

measures taken.

Consider now a different the scenario where security requirements are desired during the information transfer, such as confidentiality with the presence of vulnerability assumptions, such as a compromised mobile network. The security condition task and the problem specification of such representation is presented as follows:

```
 (:operator (!security-requirement ?sender ?recipient ?info)
            ((info-may-not-be-sniffed ?sender ?recipient ?info)
            (not (interception-successful ?sender ?recipient ?info)))
            ()
            ()
)


(defproblem p1 translation-company
 ((willing-to-install-software secretary)
 (willing-to-install-software customer)
 (willing-to-generate-keys secretary)
 (willing-to-generate-keys customer)
 (knows secretary customer) (knows customer secretary)
 (2-or-less secretary customer)
 (is-compromised-mobile-network customer secretary))
 ((transmit-information secretary customer invoice)
 (!security-requirement secretary customer)))
```

The plan is generated as follows:

```
{1: (!symmetric-encryption-method secretary customer)
 2:  (!install-encryption-software secretary)
 3:  (!install-encryption-software customer)
 4:  (!generate-symmetric-key secretary)
 5:  (!share-symmetric-key secretary customer)
```

```
6:  (!share-keys-by-email secretary customer)
7:  (!no-key-backup secretary)
8:  (!no-key-backup customer)
9:  (!encrypt-using-shared-symmetric-key secretary invoice)
10: (!send-encrypted-information secretary customer invoice)
11: (!decrypt-using-shared-symmetric-key customer invoice)
12: (!security-requirements secretary customer)}
```

The planner returns a plan with the necessary encryption method that satisfies the desired security requirement with the least possible cost. Thus, it includes sending the invoice as an encrypted document, in this case using symmetric encryption, although asymmetric encryption is also an option, while avoiding sending information using SMS.

Since the initial state of the problem specification includes the willingness of the actors to install an encryption software, and to generate keys, the planner prints the actions that meet those criteria (steps 1-4). The inclusion of "`knows secretary customer`" and "`knows customer secretary`" denotes that actors already shared their contact details, which explains the key transmission method by email in step 5 and 6. Two options regarding key backup are present in our catalogue (Figure 3.7), however, since both options include the same number of tasks in HTN, the planner will consider the first option written in the HTN script (No key backup), which is printed in step 7 and 8.

In some cases, the analyst might combine multiple security requirements with multiple vulnerability assumptions. Consider an example in which, in addition to requiring confidentiality, integrity is required while making the additional vulnerability assumption of having a hacked mobile aside from having a compromised data network. Therefore, such assumptions must

be included in the initial state of problem specification, and in the security requirements as negations in the precondition of the security condition. The problem specification of such representation is shown as follows:

```
 (:operator (!security-requirement ?sender ?recipient ?info)
            ((info-may-not-be-sniffed ?sender ?recipient ?info)
            (info-may-not-be-tampered ?sender ?recipient ?info)
            (not (interception-successful ?sender ?recipient ?info)))
            ()
            ()
)


(defproblem p1 translation-company

 ((willing-to-install-software secretary)

 (willing-to-install-software customer)

 (willing-to-generate-keys secretary)

 (willing-to-generate-keys customer)

 (knows secretary customer) (knows customer secretary)

 (can-meet secretary customer) (can-meet customer secretary)

 (2-or-less secretary customer)

 (is-compromised-data-network secretary customer)

 (is-hacked secretary phone))

 ((transmit-information secretary customer invoice)

 (!security-requirement secretary customer)))
```

In this example, the domain facts “can-meet secretary customer” and “can-meet customer secretary” are added to denote the ability of the participants to physically exchange information if necessary. Although the domain facts “knows secretary customer” and knows “customer secretary” are not needed in this case because the key exchange will not happen through email or phone SMS, the user of the reasoner does not have the knowledge to make such distinction. Nevertheless, the user states possible domain facts, leading the planner to search the best fitting plan given the user’s

requirements.

Note that in cases of basic domain facts being absent in the problem specification, such as the refusal of the participants to meet personally (e.g. "can-meet secretary customer) or the refusal of the customer to install an encryption software ("willing-to-install-software customer"), the planner will generate no plans given such vulnerability assumption and security requirements. Assuming the mentioned domain facts are instead true, SHOP2 will generate the following plan:

```
{1: (!symmetric-encryption-method secretary customer invoice)
 2: (!install-encryption-software secretary)
 3: (!install-encryption-software customer)
 4: (!generate-symmetric-key secretary invoice)
 5: (!share-symmetric-key secretary customer)
 6: (!share-keys-in-person secretary customer)
 7: (!no-key-backup secretary) (!no-key-backup customer)
 8: (!encrypt-using-shared-symmetric-key secretary invoice)
 9: (!symmetric-sign secretary customer invoice)
 10: (!send-encrypted-information secretary customer invoice)
 11: (!verify-symmetric-signature customer invoice)
 12: (!decrypt-using-shared-symmetric-key customer invoice)
 13: (!security-requirements30 secretary customer invoice)}
```

Accordingly, the planner generates a plan using symmetric encryption with the use of digital signatures, and no longer considers the plan that includes exchanging keys using email addresses or phone SMS, thus searching for alternative solutions that avoid such means of information transfer, such as sharing keys in person in this case.

As described previously, information sniffing directly threatens confidentiality since attackers may be able to access sensitive information being sent, as shown in Figure 3.11. The use of any encryption method satisfies this security requirement, nevertheless, the need of information integrity forces to planner to address this requirement by using digital signatures. The

steps required for transmitting information using the identified encryption method is described from step 8 to 12, as shown in figure 3.5

Next, consider an example where there is a slight change in the domain facts and vulnerability assumptions, in which the customer and the secretary are willing to pay for stronger security measures, the customer consists of three members, and that participants are being followed by potential attackers. Thus, the domain facts "willing-to-pay customer" and "willing-to-pay secretary", "more-than-2 secretary customer", and "is-followed customer" are added to the initial state of the problem specification. Considering that participants are only exchanging documents that require confidentiality, the security requirement "((info-may-not-be-sniffed)) is only added in the dummy task. Such translation of user requirements are translated in the problem specification as follows:

```
 (:operator (!security-requirement ?sender ?recipient ?info)
            ((info-may-not-be-sniffed ?sender ?recipient ?info)
            (not (interception-successful ?sender ?recipient ?info)))
            ()
            ()
)


(defproblem p1 translation-company
 ((willing-to-install-software secretary)
 (willing-to-install-software customer)
 (willing-to-generate-keys secretary)
 (willing-to-generate-keys customer)
 (willing-to-pay customer certificate)
 (willing-to-pay secretary certificate)
 (knows secretary customer) (knows customer secretary)
 (can-meet secretary customer) (can-meet customer secretary)
 (more-than-2 secretary customer) (is-followed customer))
```

```
((transmit-information secretary customer invoice)
(!security-requirement secretary customer invoice))
```

Ideally, since the document being exchanged requires only confidentiality, the plan with the least cost (i.e. the least number of tasks) is using symmetric encryption, however, since the initial state contains a domain fact that states a number of participants higher than two, the planner gives priority to asymmetric encryption. Given that the users are willing to pay for extra security measures, the planner also provides a plan that includes certifying keys using certificate authorities as follows:

```
{1: (asymmetric-encryption-method secretary customer invoice)
 2: (!install-encryption-software secretary)
 3: (!install-encryption-software customer)
 4: (!generate-public-private-key-pair secretary)
 5: (!generate-public-private-key-pair customer)
 6: (!certify-keys-using-certificate-authority secretary invoice)
 7: (!certify-keys-using-certificate-authority customer invoice)
 8: (!share-digital-certificate customer secretary)
 9: (!share-digital-certificate secretary customer)
 10: (!no-key-backup secretary)
 11: (!no-key-backup customer)
 12: (!choice-is-to-include-digital-signature secretary customer invoice)
 13: (!encrypt secretary invoice (public-key customer))
 14: (!sign secretary invoice (private-key secretary))
 15: (!send-encrypted-information secretary customer invoice)
 16: (!verify customer invoice (public-key secretary))
 17: (!decrypt customer invoice (private-key customer))
 18: (!transmit-using-online-file-storage-systems secretary customer invoice)
 19: (!security-requirements secretary customer invoice)}
```

Having presented plans that include various combinations of security requirements and vulnerability assumptions, we now move to present plans with the inclusion of qualities. Let's take an example where a document being sent from the secretary to the customer requires confidentiality and integrity (i.e. the use of symmetric encryption) while considering that the mobile network is compromised, and the participants are potentially being monitored by attackers. The problem specification will include the following:

```
 (:operator (!security-requirement ?sender ?recipient ?info)
            ((info-may-not-be-sniffed ?sender ?recipient ?info)
            ((info-may-not-be-tampered ?sender ?recipient ?info)
            (not (interception-successful ?sender ?recipient ?info)))
            ()
            ()
)

(defproblem p1 translation-company
 ((willing-to-install-software secretary)
 (willing-to-install-software customer)
 (willing-to-generate-keys secretary)
 (willing-to-generate-keys customer)
 (knows secretary customer) (knows customer secretary)
 (2-or-less secretary customer)
 (have-convenience 0) (have-cost 0)
 (have-performance 0) (have-quality 0)
 (is-compromised-mobile-network secretary customer)
 (is-followed customer))
 ((transmit-information secretary customer invoice)
 (!security-requirement customer invoice)
 (total-convenience-points ?c) (total-performance-points ?p)
 (total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))
```

The plan is generated as follows:

```
{1: (!symmetric-encryption-method secretary customer invoice)
 2: (!install-encryption-software secretary)
 3: (!install-encryption-software customer)
 4: (!generate-symmetric-key secretary invoice)
 5: (!share-symmetric-key secretary customer)
 6: (!share-keys-by-email secretary customer)
 7: (!performance-points 0 1) (!convenience-points 0 1) (!cost-points 0 2)
 8: (!no-key-backup secretary) (!no-key-backup customer)
 9: (!performance-points 1 2) (!convenience-points 1 0) (!cost-points 2 3)
 10: (!encrypt-using-shared-symmetric-key secretary invoice)
 11: (!symmetric-sign secretary customer invoice)
 12: (!send-encrypted-information secretary customer invoice)
 13: (!verify-symmetric-signature customer invoice)
 14: (!decrypt-using-shared-symmetric-key customer invoice)
 15: (!transmit-using-online-file-storage-systems secretary customer invoice)
 16: (!performance-points 2 4) (!convenience-points 0 2) (!cost-points 3 5)
 17: (!security-requirements30 secretary customer invoice)
 18: (!total-convenience-points 2) (!total-performance-points 4)
 19: (!total-cost-points 5) (!total-quality-points 0 11)}
```

The following generated plan starts with each quality having 0 points, which is defined in the initial state of the plan as [(have-convenience 0), (have-cost 0), (have-performance 0), (have-quality 0)]. Step 7 shows the qualities that result from sharing the key by email. In our patterns, we state that sharing keys by email will help performance and convenience since this method is relatively faster and more convenient than other methods, and make cost since it requires no additional fees that the user is usually paying for such services. Thus, performance and convenience will have one point

each, while cost will have 2 points.

Step 9 shows the qualities that result from using symmetric encryption as an encryption method. In our patterns, such as in Figure 3.15, this method "help" performance, "hurt" convenience, and "help" cost. Therefore, performance and cost will gain 1 additional point each, and convenience will lose 1 point from their original number of points presented in step 7. Consequently, the number of quality points will increase or decrease as the plan is being executed. The sum of total number of quality points is then added at then end, as shown in step 19.

The total quality points in step 19 may help the user of the reasoner to filter the best plans whenever a high number of plans is generated. For instance, if the planner generates 4 plans where 2 plans contain a negative number of quality points, then it might help the user to consider the plans that have a positive number of points and ignore the rest of the plans. However, this might not always be the case. Even though some plans might have a negative total number of quality points, it might be the most convenient plan to use depending on the desired level of security and the comfortability of the user in carrying out such activities.

Consider finally an example where the customer is willing to pay some money for extra security measures. In case integrity is only required, the planner will generate a plan that includes symmetric encryption with digital signatures. Nevertheless, the planner will also generate other plans that contain stronger security measures against the user's security concerns that include the use of asymmetric encryption, such as in the following plan where the total number of quality points equals to "-14" compared to the plan with the least cost that has a total number of quality points equals to "1".

{1: ((asymmetric-encryption-method doc patient info)

2: (!install-encryption-software doc)

3: (!install-encryption-software patient)

4: (!generate-public-private-key-pair doc)

5: (!generate-public-private-key-pair patient)

6: (!share-public-key doc patient info)

7: (!share-public-key patient doc info)

8: (!share-keys-in-person doc patient)

9: (!performance-points 0 -2)(!convenience-points 0 -2)(!cost-points 0 -2)

10: (!no-key-backup doc)

11: (!no-key-backup patient)

12: (!performance-points -2 -4!convenience-points -2 -1)(!cost-points -2 -3)

13: (!choice-is-to-include-digital-signature doc patient info)

14: (!performance-points -4 -6) (!convenience-points -1 1)

15: (!encrypt doc info (public-key patient))

16: (!sign doc info (private-key doc))

17: (!send-encrypted-information doc patient info)

18: (!verify patient info (public-key doc))

19: (!decrypt patient info (private-key patient))

20: (!transmit-info-in-person doc patient info) (!performance-points -6 -8)

21: (!convenience-points 1 -1) (!cost-points -3 -5)

22: (!security-requirements30 doc patient info)

23: (!total-convenience-points -1)(!total-performance-points -8)

24: (!total-cost-points -5)(!total-quality-points 0 -14))}

Chapter 4

Applications

4.1 Overview

In the following section, we offer a preliminary evaluation through applying our framework to different domains and use scenarios. The purpose of such scenarios is to measure the efficiency and effectiveness of security patterns and attack trees with vulnerability assumptions in multiple domains to address user threats and security concerns. Through different representation of domains, we can identify the limitations of such development, and whether our patterns are indeed reusable by security experts regardless of the domain being presented.

We assess the effectiveness of our framework by analyzing; (a) whether meaningful security requirements of a domain can be drawn through the analysis of business processes of the social view and security patterns, (b) if such patterns are applicable and reusable across multiple domains, (c) whether security requirements can be analyzed using the attack trees and reused as necessary, (d) whether domain facts and vulnerability assump-

tions can be successfully modelled, and (e) whether generated plans are meaningful for the domain. Efficiency, on the other hand, is measured by the time taken for SHOP2 to compute and generate such plans.

Since the planner allows the user to print all the possible solutions, for simplicity, we discuss the plan that is generated with the least cost that meets security requirements and vulnerability assumptions unless specified otherwise. We discuss such representation of domains in the following sections.

4.2 Health Care Domain

4.2.1 Overview

The first case we consider is from the health care domain. The case is a real wold service offered by a Toronto based company for identifying and contacting health care professionals in the Greater Toronto Area. The company helps patients seek reliable physicians to provide medical services that meet their expectations through transparent public reviews. The company provides an application where patients can search for physicians and specialists according to location and reviews. Once a physician is found, business and academic information of the physician can be downloaded (in PDF format). After obtaining the necessary medical treatment and paying the consultation fees, the patient receives a receipt that can be uploaded to the system following the physician's treatment confirmation to submit a review.

4.2.2 Social View and Attack Trees

To model stakeholder intentions and their dependencies, the STS-ml social view is first constructed. Security requirements are then analyzed and attached on goals and tasks whenever required.

As shown in Figure 4.1, the social view is composed of four main goals that define the activities needed to be taken by the patient as follows: (1) the patient needs to register to the application, (2) the patient searches for a physician and obtains the desired business and academic details, (3) once found, the patient can book an appointment to obtain the medical treatment and receives an online receipt from physician's secretary after paying for the service obtained, and (4) the patient posts a review of the health care practitioner on the application. Before posting the review, the patient proves the usage of the practitioner's service by uploading the receipt received by the secretary, assuming that the receipt is a digitally signed document that the health care system automatically verifies using the physician's public key.

After modelling the required business activities, threats of such activities are now considered. We address two activities that may pose security concerns. The first activity involves sending the invoice from the secretary to the patient after receiving the medical treatment, while the second activity involves sending the receipt (excluding the fees) which is also required to complete the review of the health care practitioner on the health care system.

The first basic security concern that targets sensitive information being sent is their accessibility to potential adversaries for a breach in confidentiality. Thus, as shown in Figure 4.1, information sniffing by attackers, labeled

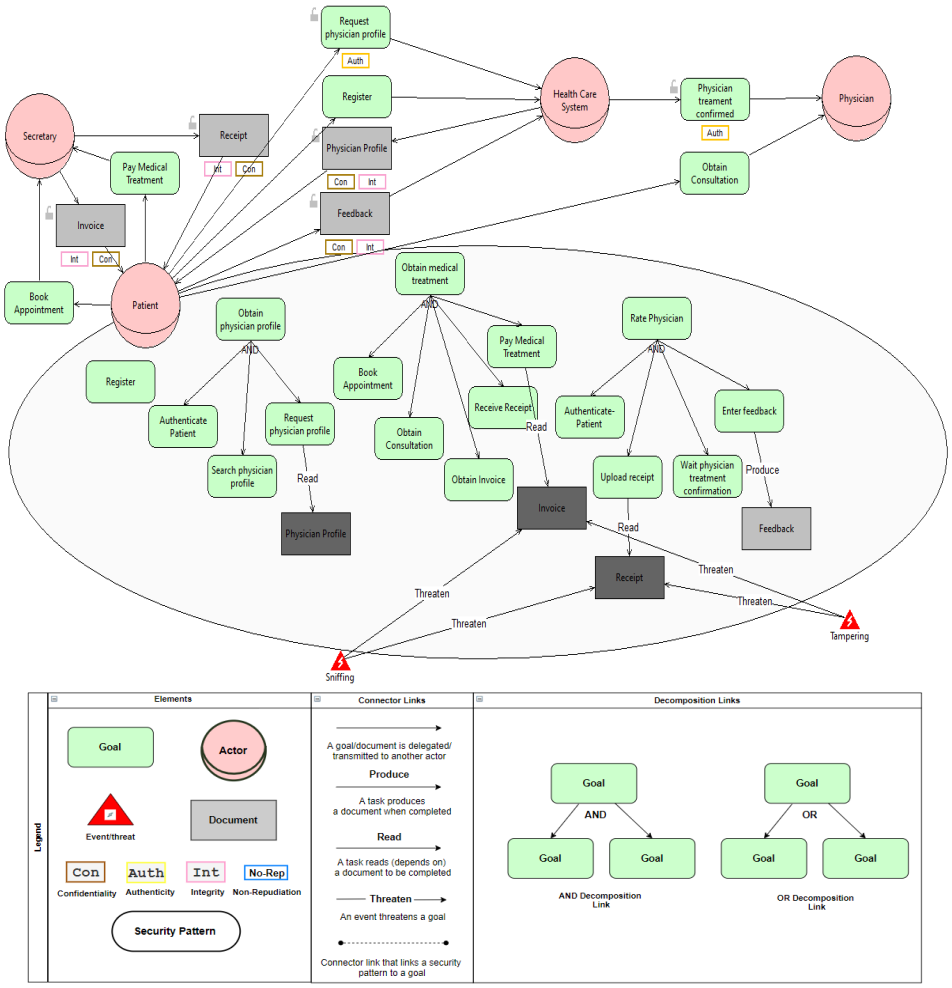

Figure 4.1: Health Care domain Social View using STS-tool.

as "sniffing", is a direct threat to both activities discussed. Whereas the second security concern, information tampering, labeled as "tampering", leads to a breach in integrity, and is a threat to both activities as well.

The analyzed threats now determine the necessary security requirements needed at each stage to maintain secure business processes. In both mentioned activities, members want to ensure confidentiality. Therefore, to address information sniffing, confidentiality labeled as "Con", is added to the invoice and the receipt, while integrity, labeled as "Int", is also added to these documents to address information tampering, i.e., to assure that the invoice is not tampered by the patient or any other potential adversary, and to denote the desire of the patient to receive a receipt that is

not tampered with so that it can be used on the application to submit a review.

4.2.3 Workflow Patterns and Vulnerability Assumptions

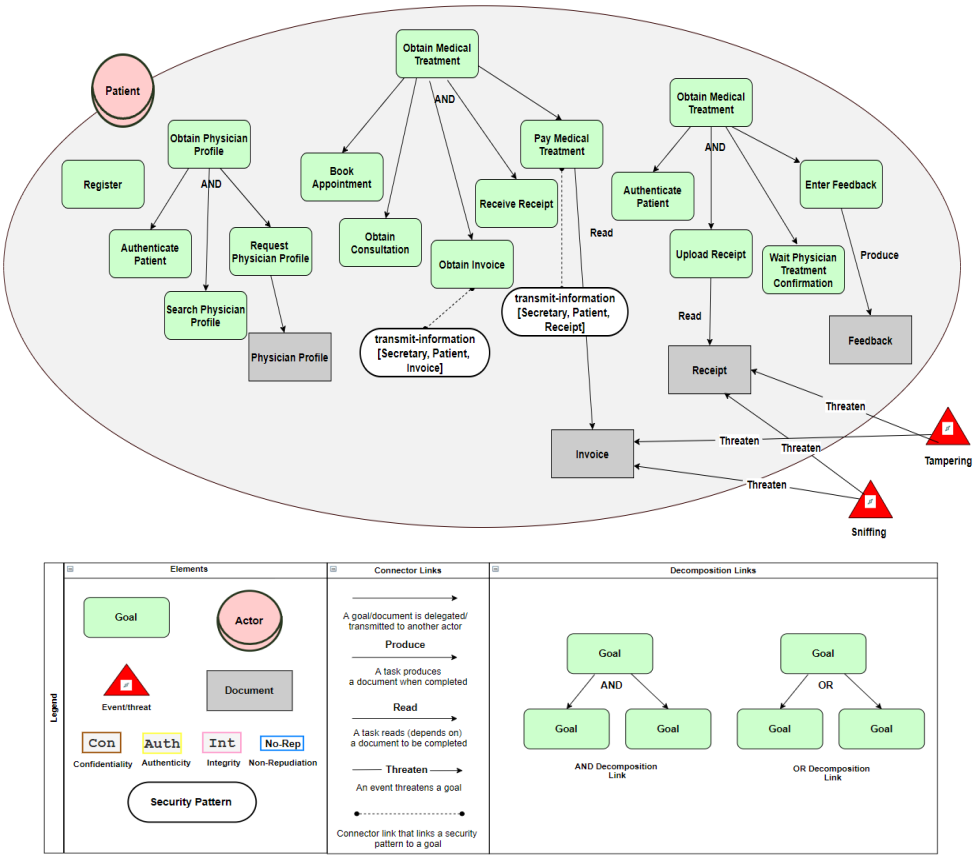

Figure 4.2: Health care domain including workflow patterns.

Having analyzed the business processes and the necessary threats and security requirements of each goal, we now move to identify the security patterns with the relevant attack trees that describe the threats identified in the social view. As shown in Figure 4.2, the activity "Obtain Invoice" is linked to the pattern "transmit-information [Secretary, Patient, Invoice]" (Section 3.3.4, Figure 3.5) that in turn calls the "setup-security-tools [Secretary, Patient, Invoice]" pattern (Section 3.3.2, Figure 3.3), while

the activity "Receive Receipt" is directly linked to the pattern "transmit-information [Secretary, Patient, Receipt]" (Section 3.3.4, Figure 3.5) that in turn calls the "setup-security-tools [Secretary, Patient, Receipt]" pattern (Section 3.3.2, Figure 3.3) since the transmission of confidential information requires members to setup their software.

Considering that the secretary and the patient communicate via email services (e.g. to share the invoice and the receipt) and mobile phones (e.g. to book appointments and/or access the documents sent), the attack trees representing confidentiality (Figure 3.11) and integrity (Figure 3.13) may be breached if the adversary is able to access their network (i.e. the data is being transferred through data network) or gain access on their mobile devices, resulting in a potential eavesdrop (Figure 3.12) of the information being transferred between the members.

In a simple process as that of our health care domain, we were able to develop STS diagrams and identify at least two threats (information sniffing and tampering) and corresponding security requirements (confidentiality and integrity) that were relevant to our domain. The purpose of confidentiality in this domain is to prevent any unauthorized user to gain access to the documents (i.e. invoice and receipt) being sent between the secretary and the patient, while also preventing any alteration (aka. tampering) to such documents in order to preserve integrity. Hence, modelling such business processes with its corresponding security patterns supports our research goal that domains can be analyzed through the social view (research goal a in Section 4.1).

In the application of workflow patterns to the activities of the domain, we were able to evoke the "transmit-information [Sender, Recipient, Information]" (Section 3.3.4, Figure 3.5) pattern that requires the "setup-security-

tools [Sender, Recipient, Information]" pattern (Section 3.3.2, Figure 3.3) at least twice when the receipt and the invoice were exchanged. Additionally, the "setup-security-tools [Sender, Recipient, Information]" pattern (Section 3.3.2, Figure 3.3) calls other workflow patterns such as the "key-exchange [Sender, Recipient, Information]" pattern (Section 3.3.3, Figure 3.4) and the "key-backup [Sender, Recipient, Information]" pattern (Section 3.3.6, Figure 3.7), which supports the position that reusable patterns can be developed as we describe here (research goal b in Section 4.1).

We saw however that there are other threats and nuanced security requirements such as anonymity of the user vis-a-vis the service (e.g. our provider to have zero knowledge of both the information (invoice and receipt) and the identity of the user and provider performing the transaction) that are not addressed by our so-far patterns. This indicates that more work needs to be done to establish if our framework can completely deal with a security workflow design or whether there are expressiveness constraints imposed by, e.g., the planner or the goal decomposition approach.

4.2.4 Domain and Problem Specification Analysis

Up to this point we have modeled the basic processes, the threats related to parts of those processes and the security requirements in response to those threats. We now move to the translation of such goals and activities to HTN to address the identified security concerns and vulnerability assumptions in the domain and problem specification.

As described in Chapter 3, goals are translated to HTN methods, and tasks to HTN operators in the domain specification. The integration of security patterns with the business processes is done through method nesting. The

decomposition of the primary goal 'Obtain Medical Treatment", as shown in Figure 4.2, with it's respective tasks and patterns is presented as follows:

```
(:method (obtain-medical-treatment ?patient ?secretary ?physician ?receipt ?invoice)
         ()
         (:ordered (!obtain-medical-treatment-plan)
         (!book-appointment ?patient ?physician)
         (!obtain-consultation ?patient ?physician)
         (!send ?secretary ?patient ?invoice)
         (!security-requirements2 ?secretary ?patient ?invoice)
         (!pay-for-medical-treatment ?patient ?physician))
         (!send ?secretary ?patient ?receipt)
         (transmit-information ?secretary ?patient ?receipt)
         (!security-requirements3 ?secretary ?patient ?receipt)
)
```

As shown above, transferring the invoice and the receipt fall under the same primary goal having each their own security requirement task. Despite the presence of two security requirements tasks, they both include the same requirements, which are the negations of identified threats (i.e. to achieve confidentiality and integrity), as well as the negation of a successful interception by the attacker (as described in Section 3.6.8), presented in the domain specification as follows:

```
(:operator (!security-requirements2 ?secretary ?patient ?invoice)
           ((info-may-not-be-sniffed ?secretary ?patient ?invoice)
           (info-may-not-be-tampered ?secretary ?patient ?invoice)
           (not (interception-successful ?secretary ?patient ?invoice)))
           ()
           ()
)

(:operator (!security-requirements3 ?secretary ?patient ?receipt)
           ((info-may-not-be-sniffed ?secretary ?patient ?receipt)
           (info-may-not-be-tampered ?secretary ?patient ?receipt)
           (not (interception-successful ?secretary ?patient ?receipt)))
           ()
           ()
)
```

Having translated the goals, patterns and security requirements in the domain specification, we now move to the initial state and the goal formula of the problem specification. The initial state (as described in Section 3.6.7), contains domain facts, vulnerability assumptions, and static quality predicates (examples shown in Table 3.2).

Starting with the domain facts, we acknowledge that the number of participating members is less than two, translated as "`2-or-less secretary patient`", both members are willing to install and generate keys, translated as "`willing-to-install-software secretary/patient`" and "`willing-to-generate-keys secretary/patient`", assume they already shared their contact information beforehand, translated as "`knows secretary patient`" and "`knows patient secretary`", and willing to meet in person if required, translated as "`can-meet secretary patient`" and "`can-meet patient secretary`". Since the participating members will already have installed an encryption software when sharing the receipt, the domain fact "`has secretary/patient encryption-software receipt`" is also added.

As described earlier, by analyzing the confidentiality and integrity attack trees, two possible vulnerabilities can be taken in consideration: since participants communicate frequently through emails and mobile phones, (1) an assumption that a compromised network can be made, translated as "`is-compromised-data-network patient secretary`", and (2) an assumption that a compromised mobile device of participants, translated as "`is-hacked secretary/patient phone`", whereby if the attackers goals are carried out with the ability of eavesdropping keys, a breach in confidentiality and/or integrity can occur. In this example, we only consider the first vulnerability.

The goal formula (as described in Section 3.6.8), on the hand, contains the goal desired to be achieved, which is "obtain-medical-treatment", and static quality methods (examples shown in Table 3.3). The problem specification is, therefore, translated as follows:

Table 4.1: Problem Specification Translation to HTN Constructs.

Problem Specification Translation to HTN Constructs	Explanation of HTN Constructs
(defproblem p1 domains	Problem Definition Syntax
(willing-to-install-software secretary) (willing-to-install-software patient) (willing-to-generate-keys patient) (willing-to-generate-keys secretary) (knows secretary patient) (knows patient secretary) (can-meet secretary patient) (can-meet patient secretary) (has secretary encryption-software receipt) (has patient encryption-software receipt) (2-or-less secretary patient)	Domain Facts
(is-compromised-data-network patient secretary) (is-compromised-data-network secretary patient)	Vulnerability Assumption
(have-convenience 0) (have-cost 0) (have-performance 0) (have-quality 0))	Quality Predicates
((obtain-medical-treatment patient secretary physician receipt invoice)	Goal Formula
(total-convenience-points ?c) (total-performance-points ?p) (total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))	Quality Methods

As a result, SHOP2 generates the following plan:

```
1: (!obtain-medical-treatment-plan)
2: (!book-appointment patient physician)
3: (!obtain-consultation patient physician)
4: (!send secretary patient invoice)
5: (!symmetric-encryption-method secretary patient invoice)
6: (!install-encryption-software secretary)
7: (!install-encryption-software patient)
8: (!generate-symmetric-key secretary invoice)
9: (!share-symmetric-key secretary patient)
10: (!performance-points 0 1)(!convenience-points 0 -1)(!cost-points 0 1)
11: (!share-keys-by-phone-sms secretary patient)
12: (!performance-points 1 0)(!convenience-points -1 -2)(!cost-points 1 3)
13: (!no-key-backup secretary) (!no-key-backup patient)
14: (!encrypt-using-shared-symmetric-key secretary invoice)
15: (!symmetric-sign secretary patient invoice)
```

```
16: (!send-encrypted-information secretary patient invoice)
17: (!transmit-using-online-file-storage-systems secretary patient invoice)
18: (!performance-points 0 2)(!convenience-points -2 0)(!cost-points 3 5)
19: (!verify-symmetric-signature patient invoice)
20: (!decrypt-using-shared-symmetric-key patient invoice)
21: (!security-requirements2 secretary patient invoice)
22: (!pay-for-medical-treatment patient physician)
23: (!send secretary patient receipt)
24: (!symmetric-encryption-method secretary patient receipt)
25: (!generate-symmetric-key secretary receipt)
26: (!share-symmetric-key secretary patient)
27: (!performance-points 2 3)(!convenience-points 0 -1)(!cost-points 5 6)
28: (!share-keys-by-phone-sms secretary patient)
29: (!performance-points 3 2)(!convenience-points -1 -2)(!cost-points 6 8)
30: (!no-key-backup secretary) (!no-key-backup patient)
31: (!encrypt-using-shared-symmetric-key secretary receipt)
32: (!symmetric-sign secretary patient receipt)
33: (!send-encrypted-information secretary patient receipt)
34: (!transmit-using-online-file-storage-systems secretary patient receipt)
35: (!performance-points 2 4)(!convenience-points -2 0)(!cost-points 8 10)
36: (!verify-symmetric-signature patient receipt)
37: (!decrypt-using-shared-symmetric-key patient receipt)
38: (!security-requirements3 secretary patient receipt)
39: (!total-convenience-points 0) (!total-performance-points 4)
40: (!total-cost-points 10) (!total-quality-points 0 14)
```

The planner provides a recommendation to the secretary and the patient to use symmetric encryption (step 5-15 and step 24-32) with digital signatures (step 15-20 and step 32-37) during the transmission of both documents by sharing the encryption/decryption keys using phone SMS (step 11 and step 28). In other words, for the secretary and the patient to achieve confidentiality and prevent potential information access to adversaries, SHOP2 provides the necessary security controls to be followed by the participants to achieve their security objectives, through the use of symmetric symmetric encryption that assures information confidentiality (as described in Section 3.3.2) including digital signatures that protects against information tampering to achieve information integrity (as described in Section 3.3.4).

The transfer of such documents between the members confidentially helps the secretary avoid potential fraud and/or financial loss, and allows the patient to provide a transparent feedback on the health care system to the intended physician.

We now move to another example, whereby the second vulnerability assumption of a compromised mobile device is also included (translated as "`is-hacked secretary/patient phone`"). The problem specification is, therefore, translated as follows:

Table 4.2: Problem Specification Translation to HTN Constructs.

Problem Specification Translation to HTN Constructs	Explanation of HTN Constructs
`(defproblem p2 domains`	Problem Definition Syntax

`(willing-to-install-software secretary)`
`(willing-to-install-software patient)`
`(willing-to-generate-keys patient)`
`(willing-to-generate-keys secretary)`
`(knows secretary patient) (knows patient secretary)`
`(can-meet secretary patient) (can-meet patient secretary)`
`(has secretary encryption-software receipt)`
`(has patient encryption-software receipt)`
`(2-or-less secretary patient)`
	Domain Facts

`(is-compromised-data-network patient secretary)`
`(is-compromised-data-network secretary patient)`
`(is-hacked secretary phone) (is-hacked patient phone)`
	Vulnerability Assumption

`(have-convenience 0) (have-cost 0)` `(have-performance 0) (have-quality 0))`	Quality Predicates
`((obtain-medical-treatment patient secretary physician receipt invoice)`	Goal Formula
`(total-convenience-points ?c) (total-performance-points ?p)` `(total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))`	Quality Methods

Consequently, SHOP2 generates the following plan:

```
1: (!obtain-medical-treatment-plan)
2: (!book-appointment patient physician)
3: (!obtain-consultation patient physician)
4: (!send secretary patient invoice)
5: (!symmetric-encryption-method secretary patient invoice)
6: (!install-encryption-software secretary)
7: (!install-encryption-software patient)
```

```
8: (!generate-symmetric-key secretary invoice)
9: (!share-symmetric-key secretary patient)
10: (!performance-points 0 1)(!convenience-points 0 -1)(!cost-points 0 1)
11: (!share-keys-in-person secretary patient)
12: (!performance-points 1 -1)(!convenience-points -1 -3)(!cost-points 1 -1)
13: (!no-key-backup secretary) (!no-key-backup patient)
14: (!encrypt-using-shared-symmetric-key secretary invoice)
15: (!symmetric-sign secretary patient invoice)
16: (!send-encrypted-information secretary patient invoice)
17: (!transmit-using-online-file-storage-systems secretary patient invoice)
18: (!performance-points -1 1)(!convenience-points -3 -1)(!cost-points -1 1)
19: (!verify-symmetric-signature patient invoice)
20: (!decrypt-using-shared-symmetric-key patient invoice)
21: (!security-requirements2 secretary patient invoice)
22: (!pay-for-medical-treatment patient physician)
23: (!send secretary patient receipt)
24: (!symmetric-encryption-method secretary patient receipt)
25: (!generate-symmetric-key secretary receipt)
26: (!share-symmetric-key secretary patient)
27: (!performance-points 1 2)(!convenience-points -1 -2)(!cost-points 1 2)
28: (!share-keys-in-person secretary patient)
29: (!performance-points 2 0)(!convenience-points -2 -4)(!cost-points 2 0)
30: (!no-key-backup secretary) (!no-key-backup patient)
31: (!encrypt-using-shared-symmetric-key secretary receipt)
32: (!symmetric-sign secretary patient receipt)
33: (!send-encrypted-information secretary patient receipt)
34: (!transmit-using-online-file-storage-systems secretary patient receipt)
35: (!performance-points 0 2)(!convenience-points -4 -2)(!cost-points 0 2)
36: (!verify-symmetric-signature patient receipt)
37: (!decrypt-using-shared-symmetric-key patient receipt)
38: (!security-requirements3 secretary patient receipt)
39: (!total-convenience-points -2) (!total-performance-points 2)
40: (!total-cost-points 2) (!total-quality-points 0 2)}
```

The planner proposes the same plan as the previous example, however, with a difference in steps 11 and 28, whereby sharing the key is suggested in person instead. Considering a compromised data network and a hacked mobile device, sending any information using those means could lead to an unauthorized access to adversaries, and/or a potential eavesdropping of keys leading to a breach in confidentiality and/or integrity. Hence, avoiding

Table 4.3: Summary of plan(s) generation in the health care domain.

Goal	Vulnerability Assumption	Security Requirement	Cryptographic Recommendation	Key Exchange Method	Plan Generation Time (in seconds)	Number of Methods-Tasks
Obtain Medical Treatment	Compromised Data Network	Confidentiality and Integrity	Symmetric or Asymmetric Encryption with Digital Signature	Phone SMS or In Person	1.461	30-56
Obtain Medical Treatment	Compromised Data Network & Hacked Mobile Device	Confidentiality and Integrity	Symmetric or Asymmetric Encryption with Digital Signature	In Person Only	1.907	30-56

such methods by proposing a different sharing method (as proposed in step 11 and 29). A summary of both activities and the plan generation time in SHOP2 is displayed in Table 4.3.

To draw possible attack patterns and their complications that match the goals and the activities of this domain, confidentiality and integrity attack trees (as seen in Section 3.5) were used as a reference. Considering the means that participants use to exchange information (email services and mobile phones), vulnerabilities were then taken into consideration to construct the problem specification, which offers us some confidence that such attack trees can be constructed to be reusable across domains (research goal c in Section 4.1).

To translate vulnerability assumptions derived from attack trees, and the actions that participants are willing to consider or already taken, along with the nature of their interaction, domain facts are included in the initial state (as describes in Section 3.6.4 and Section 3.6.7). To carry out a secure transfer of documents between the secretary and the patient, participants had to accept actions that lead to successful secure business process, e.g. their willingness to install an encryption software (translated into HTN according to Section 3.6.7), and state their choices of exchanging information

(e.g. "knows patient secretary"), while stating possible vulnerabilities, such as a compromised mobile network (translated according to Section 3.6.4). Thus, such translation of domain facts and vulnerability assumptions supports our research goal that the state of stakeholders in a domain can be reflected to the initial state of the problem specification (research goal d in Section 4.1).

Changes in domain facts and vulnerability assumption impacted the plans generated by the planner. For instance, in the first example, the domain facts "knows patient/secretary" and "can-meet secretary/patient" were included as means of information exchange while taking in consideration a compromised data network of participants, as a result, SHOP2 gave a recommendation of sharing encryption/decryption keys through phone SMS excluding the use of email services. However, in the second example where a compromised mobile network is added, although domain facts remained unchanged, the planner ignores the domain fact "knows patient/secretary" (that allows information exchanges through emails or mobile phones) and proposes a plan that includes sharing keys in person only to address the second vulnerability at hand.

In practice, if the secretary and the patient would like to exchange the invoice and the receipt while maintaining confidentiality and integrity, the planner suggests the use of symmetric encryption with the use of digital signatures to encrypt and decrypt their documents based on their readiness to engage in activities that achieve such security requirements. Of course, it is up to the members to choose the plan they see fit and can change their plan when a change in vulnerabilities or a change in the preparatory steps that a member should take. In other words, SHOP2 is able to produce plans that satisfy the security requirements of business processes and address vulnerability assumptions while taking in consideration the elements of the

initial state. Hence, reinforcing our research goal that plans generated by the planner are meaningful to the domain (research goal e in Section 4.1).

As shown in Table 4.3, the same goal is discussed while highlighting different security requirements and vulnerability assumptions. To generate the plans previously discussed in the table, SHOP2 goes through 30 methods that include 56 tasks for both goals with a generation time that is roughly 2 seconds. Considering that the planner generates these plans in a fairly low runtime, we can consider that it is efficient in computing the desired output.

For the presented examples, although the plan with the least cost provided by SHOP2 was the use of symmetric encryption (to preserve confidentiality) with the use of digital signatures (to preserve integrity), other possible plans exist, such as the use of asymmetric encryption. In this case, the analyst can filter and choose the best fitting plan according to qualities (as described in 3.6.5). However, when the number of generated plans is high, it becomes time consuming and practically inefficient to go through all the plans, which means more work need to done around this point.

4.3 Academic Domain

In this section, we present the hiring process process of a full time faculty member in a university. The process undergoes a series of activities between faculty members before reaching the final stage. Roughly, when an academic position needs to be filled, a job posting is created and posted on the university's portal and shared across different job boards to gather potential candidates. Once the application deadline has passed, candidate applications are collected, and are examined by a hiring committee, which

evaluates applications based on the specified criteria. At the end of the process, a hiring recommendation letter is prepared by the chair of the committee, reviewed by the members and once all are agreed, submitted to the Dean of the faculty for further processing. Note, that while there are many other nuances and details pertinent to this process, we focus here on the main aspects as described.

The specific parts of the process we are interested in here are three. One is after the expiration of the deadline, when the administrative coordinator (henceforth: secretary) needs to share the confidential CVs of the applicants to the hiring committee members. The second is when the Chair of the hiring committee needs to circulate to the committee the final recommendation report, soliciting comments from committee members, to whom the members respond with such comments; this whole correspondence must also be confidential. Lastly, after several cycles of this process, the chair of the committee must submit confidentially the final recommendation to the dean of the faculty.

In all these steps, sensitive information is circulating (CVs, in-camera deliberations) that needs to be protected. Furthermore the authenticity of the final document needs to be established. These requirements makes the particular steps of the process particularly suitable for application of our approach.

4.3.1 Social View and Attack Trees

As presented in the first domain, we start modelling stakeholder's intentions and dependencies through the STS-Tool. Security requirements are then analyzed and attached on goals and tasks whenever required.

Figure 4.3 depicts three primary goals that describe the mentioned activities as follows: (1) When the secretary processes the applications by gathering them and sending them to the committee, which includes the chair, (2) When the chair processes applications recommendations by sending his recommendation to the committee where, in return, they respond accordingly, and (3) When the chair creates a final report by consolidating his recommendations with the committee's recommendations and sends it to the dean.

Now that business activities have been identified, threats affecting information transfer are discussed. Four major documents are transferred between members of this domain; the CVs sent from the secretary to the committee, the chair draft recommendation report sent to the committee, the committee recommendation report feedback sent to the chair, and the final recommendation report sent from the chair to the dean. Faculty members highly acknowledge the need of of confidentiality on the documents being circulated since potential adversaries and possibly members of the university (e.g., IT Department) may have access to such documents. Therefore, information sniffing poses a direct threat to these documents, as shown in Figure 4.3. Accordingly, the "Con" label is placed respectively on each of the presented documents to indicate the need of information confidentiality.

Information integrity, on the other hand, is considered optional since the applicants will be interviewed based on their resumes, and the recommendations sent between the faculty members will be discussed before reaching a final decision, thus any modifications made to these documents will be automatically exposed. We, however, consider this security concern to be mandatory on the final report sent from the chair to the dean since it's a one time file transfer that may impact the decision on hiring the best fitting applicant. Consequently, the "Int" label is attached to that document to

indicate the need of information integrity.

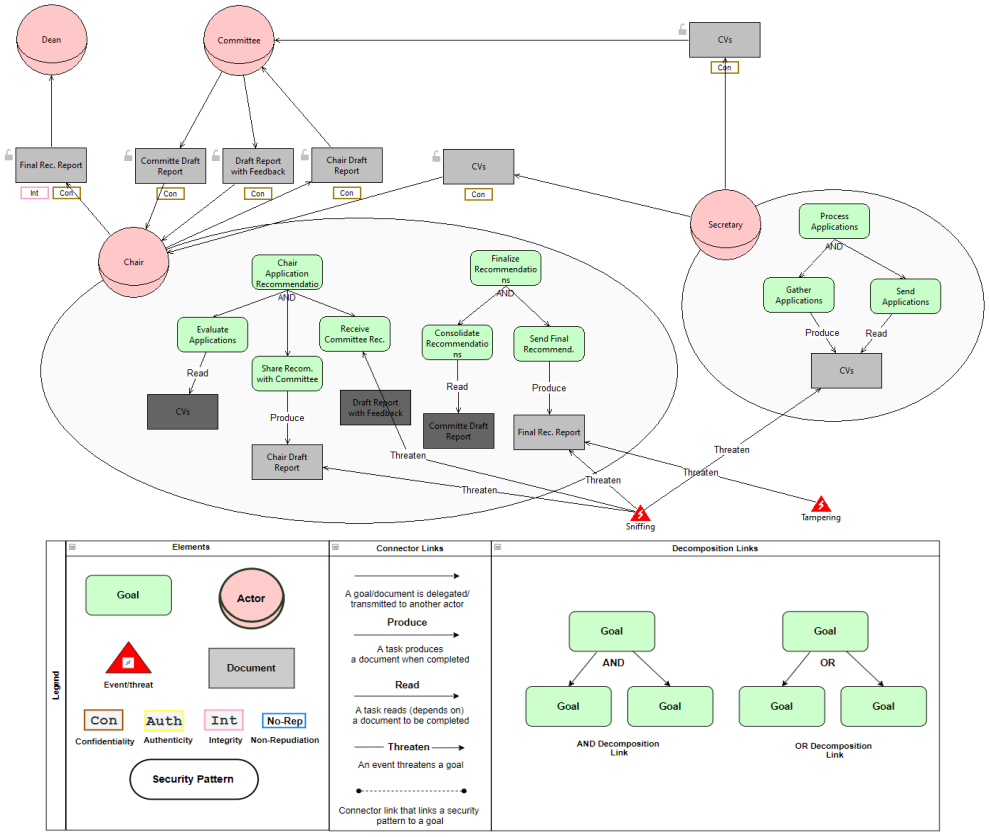

Figure 4.3: Academic domain Social View using STS-tool.

4.3.2 Workflow Patterns and Vulnerability Assumptions

After developing the models for business processes, threats, and security requirements, corresponding workflow patterns are now analyzed with their respective attack trees to identify potential vulnerability assumptions. Figure 4.4 shows the integration of these workflows under each goal where information transfer is required. All the discussed goals include tasks that require document transmission, thus requiring the "`transmit-information`" pattern (Section 3.3.4, Figure 3.5) to describe the necessary steps that achieve a secure business workflow.

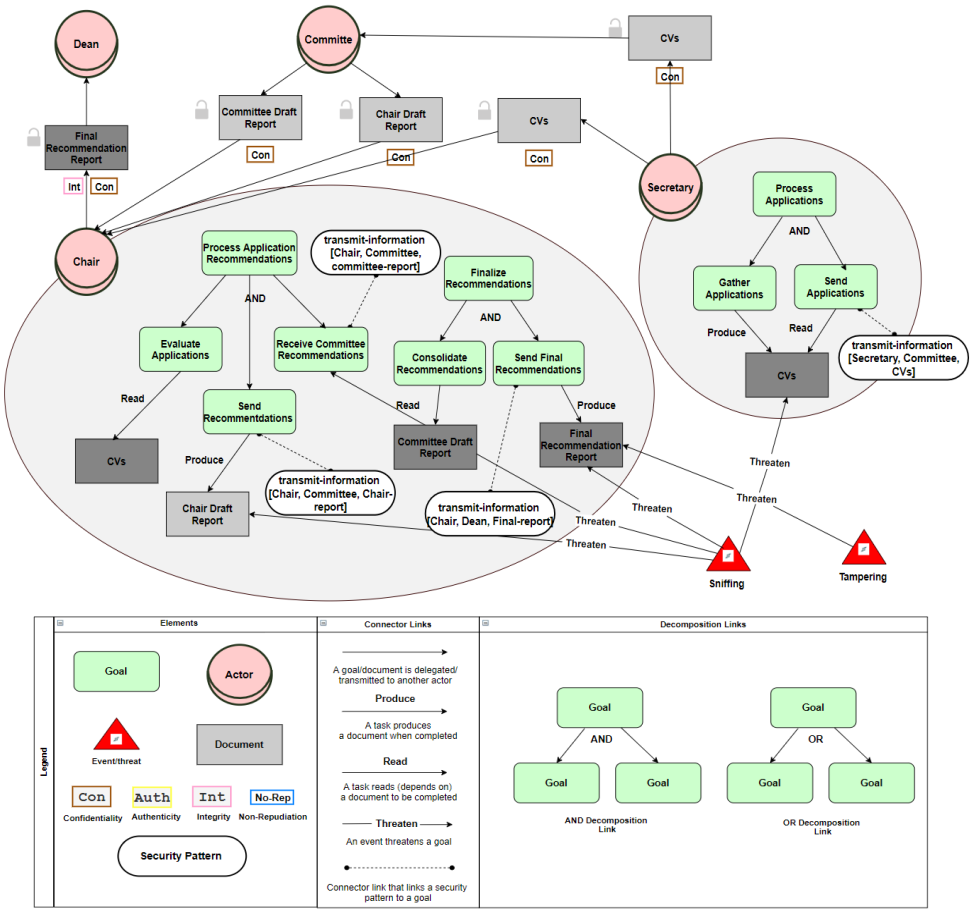

Figure 4.4: Academic domain including workflow patterns.

As described earlier, a breach in confidentiality and integrity threaten the information transmitted. The vulnerabilities that lead to such breaches in this case are now discussed. One of the vulnerabilities that attack trees (Figure 3.11 and Figure 3.13) discuss is a breach in participant's mailbox or mobile device, which are of our concern, assuming that institution's network is safe. Such vulnerabilities are mostly a result of either:

1. Stolen credentials of faculty members, or a result of employees with high system administrative rights accessing such information, leading to a breach in participants mailboxes.

2. Unauthorized access using highly administrative rights that can potentially lead to a breach in the confidentiality of participants mail-

boxes or devices, if for instance, the institution's email service is installed on their mobile device, especially when such applications are granted special permissions can have access to sensitive user information.

Recall the evaluation goals regarding the efficiency of our framework, the first goal (research goal a of Section 4.1) aims to verify whether meaningful security requirements of a domain can be drawn through the analysis of business processes of the social view and security patterns. By analyzing the process of hiring a full time faculty member using the STS-tool, we were able to model two major threats (information sniffing and tampering) and their correspondent security requirements (confidentiality and integrity) on sensitive documents being transferred between the faculty members, such as the circulation of CVs between the secretary and the committee, and the circulation of the recommendation reports between the Chair and the committee. Such representation of elements supports our goal that the social view can be indeed used to describe the processes of this domain while highlighting key security requirements.

In response to our research goal to evaluate the incorporation of patterns and their reusability pertaining to this case study, we are interested to see if the faculty hiring process and related threats will evoke any of our patterns. Indeed, as seen in Figure 4.4, we used the workflow "transmit-information [Sender, Recipient, Information]" (Section 3.3.4, Figure 3.5) pattern and its subsequent patterns "setup-security-tools [Sender, Recipient, Information]" (Section 3.3.2, Figure 3.3), "key-exchange [Sender, Recipient, Information]" pattern (Section 3.3.3, Figure 3.4) and the "key-backup [Sender, Recipient, Information]" pattern (Section 3.3.6, Figure 3.7) four times when exchanging the CVs, draft reports, and the final report. Such evocation of

patterns supports the utility and applicability of patterns across domains (research goal b of Section 4.1).

4.3.3 Domain and Problem Specification Analysis

Having modelled business processes with their respective threats, security requirements, and workflow patterns, we move to translate these elements into domain and problem specification in HTN.

The first goal "Process Applications" by the Secretary is translated as a method and its tasks as primitive tasks with their respective workflow patterns and security requirement constraint in the domain specification (as described in Section 3.6.3) as follows:

```
(:method (process-applications ?secretary ?committee ?cvs)
         ()
         (:ordered (!process-applications-plan ?secretary)
                   (!gather-applications ?secretary)
                   (!send ?secretary ?committee ?cvs)
                   (transmit-information ?secretary ?committee ?cvs)
                   (!security-requirements4 ?secretary ?committee ?cvs))
)
```

As explained in the previous section, confidentiality is required when transferring the CVs from the secretary to the committee, thus, the security requirement constraint for this transfer includes the negation of the threat, translated as "`(info-may-not-be-sniffed ?secretary ?committee ?cvs)`" along with the negation of a successful attack, translated as "`(not (interception-successful ?secretary ?committee ?cvs)` (as described in Section 3.6.8) as follows:

```
(:operator (!security-requirements4 ?secretary ?committee ?cvs)
           ((info-may-not-be-sniffed ?secretary ?committee ?cvs)
```

```
            (not (interception-successful ?secretary ?committee ?cvs)))
            ()
            ()
)
```

Moving to the problem specification of this goal, we start translating domain facts and vulnerability assumptions along with the static quality predicates of the initial state (as described in Section 3.6.7).

For this document transfer, the domain fact related to the number of participants that exceeds two is translated as "`(more-than-2 secretary committee)`" aside from considering their willingness to install and generate keys, which are translated to "`(willing-to-install-software secretary/committee)`" and "`(willing-to-generate-keys secretary/committee)`" respectively. Since members communicate through the emails of the academic institution, the domain facts "`(knows secretary committee)`" and "`(knows committee secretary)`" are added to denote their knowledge of each other's contact information. The vulnerability assumption of having a compromised mailbox is then translated as "`(is-hacked secretary/committee mailbox)`" (as shown in Section 3.6.7, Table 3.2). By adding the primary goal "Process Applications" to the goal formula, the problem specification is, therefore, presented as follows:

Table 4.4: Problem Specification Translation to HTN Constructs.

Problem Specification Translation to HTN Constructs	Explanation of HTN Constructs
(defproblem p3 domains	Problem Definition Syntax
(willing-to-install-software secretary) (willing-to-install-software committee) (willing-to-generate-keys committee) (willing-to-generate-keys secretary) (knows secretary committee) (knows committee secretary) (more-than-2 secretary committee)	Domain Facts
(is-hacked secretary mailbox) (is-hacked committee mailbox)	Vulnerability Assumption
(have-convenience 0) (have-cost 0) (have-performance 0) (have-quality 0))	Quality Predicates
(process-applications secretary committee cvs)	Goal Formula
(total-convenience-points ?c) (total-performance-points ?p) (total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))	Quality Methods

Accordingly, SHOP2 generates the following plan:

```
1: (!process-applications-plan secretary)

2: (!gather-applications secretary)

3: (!send secretary committee cvs)

4: (!asymmetric-encryption-method secretary committee cvs)

5: (!install-encryption-software secretary)

6: (!install-encryption-software committee)

7: (!generate-public-private-key-pair secretary)

8: (!generate-public-private-key-pair committee)

9: (!share-public-key secretary committee cvs)

10: (!share-public-key committee secretary cvs)

11: (!performance-points 0 -2)(!convenience-points 0 1)(!cost-points 0 -1)

12: (!share-keys-by-phone-sms secretary committee)

13: (!performance-points -2 -3)(!convenience-points 1 0)(!cost-points -1 1)

14: (!no-key-backup secretary) (!no-key-backup committee)

15: (!choice-is-exclude-digital-signature secretary committee)
```

```
16: (!performance-points -3 -4) (!convenience-points 0 1)

17: (!encrypt secretary cvs (public-key committee))

18: (!send-encrypted-information secretary committee cvs)

19: (!transmit-using-online-file-storage-systems secretary committee cvs)

20: (!performance-points -4 -2)(!convenience-points 1 3)(!cost-points 1 3)

21: (!decrypt committee cvs (private-key committee))

22: (!security-requirements4 secretary committee cvs)

23: (!total-convenience-points 3) (!total-performance-points -2)

24: (!total-cost-points 3) (!total-quality-points 0 4)))
```

The planner provides a recommendation to the members to use asymmetric encryption excluding digital signatures during the online transfer of CVs from the secretary to the committee by sharing the public key using phone SMS between each other. In other words, if participants want to achieve confidentiality, they are advised to follow the necessary security controls provided by the planner to achieve asymmetric encryption by installing an encryption software (step 5-6), generating private/public keys (step 7-8), and sharing the created public keys through phone SMS (step 9-12) to address the assumed vulnerability. Once completed, the secretary can encrypt the CVs using the public keys shared by the committee (step 14-16), without the use of digital signatures since information integrity is not a requirement (as described in 3.3.4), that would in return be decrypted by the latter using their own private keys.

We now move to translate the second goal "Process Application Recommendations" by the chair whereby the recommendations of the chair and the committee are being discussed for the selection of the best fitting applicant. Similarly to first goal, the primary goal is translated with it's respective tasks, patterns, and security requirements (as shown in Figure 4.4) as follows:

```
(:method (process-applications-recom ?chair ?committee ?chair-recom ?committee-recom)
         ()
         (:ordered (!process-application-recommendations-plan ?chair)
         (!evaluate-applications ?chair)
         (!share ?chair ?committee ?chair-recom)
         (transmit-information ?chair ?committee ?chair-recom)
         (!receive ?committee ?chair ?committee-recom)
         (transmit-information ?committee ?chair ?committee-recom)
         (!security-requirements6 ?committee ?chair ?committee-recom))
)
```

The security requirement constraints contain the security concerns to the previous goal and presented as follows:

```
(:operator (!security-requirements5 ?chair ?committee ?chair-recom)
           ((info-may-not-be-sniffed ?chair ?committee ?chair-recom)
            (not (interception-successful ?chair ?committee
            ?chair-recom))) () ()
)

(:operator (!security-requirements6 ?committee ?chair ?committee-recom)
           ((info-may-not-be-sniffed ?committee ?chair ?committee-recom)
           (not (interception-successful ?committee ?chair
           ?committee-recom))) () ()
)
```

The initial state of the problem specification, on the other hand, contain slight changes in its domain facts. In this goal, we add the second vulnerability discussed earlier of having a hacked mobile device, translated as "`(is-hacked chair phone)`", that members can meet with each other personally, translated as "`(can-meet chair committee)`" and "`(can-meet committee chair)`", and that participants already installed an encryption software considering the completion of the first goal, translated as "`(has chair/committee encryption-software)`". After the addition of the goal "`(process-application-recommendations)`" to the goal formula, the problem specification is, therefore, presented as follows:

Table 4.5: Problem Specification Translation to HTN Constructs.

Problem Specification Translation to HTN Constructs	Explanation of HTN Constructs
(defproblem p4 domains	Problem Definition Syntax
(willing-to-install-software chair) (willing-to-install-software committee) (willing-to-generate-keys chair) (willing-to-generate-keys committee) (knows chair committee) (knows committee chair) (can-meet chair committee) (can-meet committee chair) (more-than-2 committee chair) (more-than-2 chair committee) (has chair encryption-software chair-recom) (has committee encryption-software chair-recom) (has chair encryption-software com-recom) (has committee encryption-software com-recom)	Domain Facts
(is-hacked chair phone) (is-hacked committee phone) (is-hacked chair mailbox) (is-hacked committee mailbox)	Vulnerability Assumption
(have-convenience 0) (have-cost 0) (have-performance 0) (have-quality 0))	Quality Predicates
(process-applications-recom chair committee chair-recom com-recom)	Goal Formula
(total-convenience-points ?c) (total-performance-points ?p) (total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))	Quality Methods

Accordingly, SHOP2 generates the following plan:

```
1: (!process-application-recommendations-plan chair)

2: (!evaluate-applications chair)

3: (!share chair committee chair-recom)

4: (!asymmetric-encryption-method chair committee chair-recom)

5: (!generate-public-private-key-pair chair)

6: (!generate-public-private-key-pair committee)

7: (!share-public-key chair committee chair-recom)

8: (!share-public-key committee chair chair-recom)

9: (!performance-points 0 -2)(!convenience-points 0 1)(!cost-points 0 -1)

10: (!share-keys-in-person chair committee)

11: (!performance-points -2 -4)(!convenience-points 1 -1)(!cost-points -1 -3)

12: (!no-key-backup chair) (!no-key-backup committee)
```

```
13: (!choice-is-exclude-digital-signature chair committee)

14: (!performance-points -4 -5) (!convenience-points -1 0)

15: (!encrypt chair chair-recom (public-key committee))

16: (!send-encrypted-information chair committee chair-recom)

17: (!transmit-using-online-file-storage-systems chair committee chair-recom)

18: (!performance-points -5 -3)(!convenience-points 0 2)(!cost-points -3 -1)

19: (!decrypt committee chair-recom (private-key committee))

20: (!security-requirements5 chair committee chair-recom)

21: (!receive committee chair com-recom)

22: (!asymmetric-encryption-method committee chair com-recom)

23: (!generate-public-private-key-pair committee)

24: (!generate-public-private-key-pair chair)

25: (!share-public-key committee chair com-recom)

26: (!share-public-key chair committee com-recom)

27: (!performance-points -3 -5)(!convenience-points 2 3)(!cost-points -1 -2)

28: (!share-keys-in-person committee chair)

29: (!performance-points -5 -7)(!convenience-points 3 1)(!cost-points -2 -4)

30: (!no-key-backup committee) (!no-key-backup chair)

31: (!choice-is-exclude-digital-signature committee chair)

32: (!performance-points -7 -8) (!convenience-points 1 2)

33: (!encrypt committee com-recom (public-key chair))

34: (!send-encrypted-information committee chair com-recom)

35: (!transmit-using-online-file-storage-systems committee chair com-recom)

36: (!performance-points -8 -6)(!convenience-points 2 4)(!cost-points -4 -2)

37: (!decrypt chair com-recom (private-key chair))

38: (!security-requirements6 committee chair com-recom)

39: (!total-convenience-points 4) (!total-performance-points -6)

40: (!total-cost-points -2) (!total-quality-points 0 -4)))
```

The planner suggests the same plan used between the secretary and the committee (the use of asymmetric encryption without digital signatures), however, sharing the keys is proposed to be in person between the members. Since a hacked mailbox and a hacked phone are taken in consideration, sharing the public keys using such means could lead to a potential eavesdrop of information, hence, key sharing is suggested in person instead.

Similarly to the first and second goal, the translation of the third goal "Finalize Recommendations" by the chair is presented with it's respective tasks and patterns as follows:

```
(:method (finalize-recommendations ?chair ?dean ?final-recom)
         ()
         (:ordered (!finalize-recommendations-plan ?chair)
         (!consolidate ?chair recommendations)
         (!send ?chair ?dean ?final-recom)
         (transmit-information ?chair ?dean ?final-recom)
         (!security-requirements5 ?chair ?dean ?final-recom))
)
```

The security requirement constraint, however, contains an additional security concern (as shown in Figure 4.3), which is the need for information integrity, translated as follows:

```
(:operator (!security-requirements5 ?chair ?dean ?final-recom)
           ((info-may-not-be-sniffed ?chair ?dean ?final-recom)
            (info-may-not-be-tampered ?chair ?dean ?final-recom)
            (not (interception-successful ?chair ?dean ?final-recom)))
           ()
           ()
)
```

Moving to the initial state of the problem specification. Since the chair already installed an encryption software considering the completion of the other goals, the dean only then has to install a software, translated as "

`(willing-to-install-software dean)`", and the domain fact related to the number of participants is adjusted to less than two, translated as " `(2-or-less chair dean)`". The remaining domain facts and vulnerability assumptions remain unchanged. After adding the primary goal "finalize-recommendations" to the goal formula, the problem specification is, therefore, translated as follows:

Table 4.6: Problem Specification Translation to HTN Constructs.

Problem Specification Translation to HTN Constructs	Explanation of HTN Constructs
`(defproblem p5 domains`	Problem Definition Syntax

`(willing-to-install-software dean)`
`(willing-to-generate-keys chair)`
`(willing-to-generate-keys dean)`
`(knows chair dean) (knows dean chair)`
`(can-meet chair dean) (can-meet dean chair)`
`(2-or-less chair dean)`
`(has chair encryption-software final-recom)`
	Domain Facts

`(is-hacked chair phone) (is-hacked dean phone)` `(is-hacked chair mailbox) (is-hacked dean mailbox)`	Vulnerability Assumption
`(have-convenience 0) (have-cost 0)` `(have-performance 0) (have-quality 0))`	Quality Predicates
`(finalize-recommendations chair dean final-recom)`	Goal Formula
`(total-convenience-points ?c) (total-performance-points ?p)` `(total-cost-points ?cost) (quality-points ?q ?cost ?c ?p)))`	Quality Methods

Consequently, SHOP2 generates the following plan:

```
1: ((!finalize-recommendations-plan chair)

2: (!consolidate chair recommendations)

3: (!send chair dean final-recom)

4: (!symmetric-encryption-method chair dean final-recom)

5: (!install-encryption-software dean)

6: (!generate-symmetric-key chair final-recom)

7: (!share-symmetric-key chair dean)

8: (!performance-points 0 1)(!convenience-points 0 -1)(!cost-points 0 1)
```

Table 4.7: Plan Summary of goals in the Academic domain.

Goal	Task	Vulnerability Assumption	Security Requirement	Cryptographic Recommendation	Key Exchange Method	Plan Generation Time (in seconds)	Number of Methods-Tasks
Process Applications	Send CVs	Hacked Mailbox	Confidentiality	Asymmetric Encryption Excluding Digital Signature	Phone SMS and In Person	0.045	25-34
Process Application recommendations	Send Chair Recommendations	Hacked Mailbox & Hacked Mobile Phone	Confidentiality	Asymmetric Encryption Excluding Digital Signature	In Person	0.791	23-27
	Send Committee Recommendations	Hacked Mailbox & Hacked Mobile Phone	Confidentiality	Asymmetric Encryption Excluding Digital Signature	In Person	0.791	21-31
finalize-recommendations	Send Final Recommendation	Hacked Mailbox & Hacked Mobile Phone	Confidentiality & Integrity	Symmetric Encryption Including Digital Signature	In Person	0.070	15-35

```
9: (!share-keys-in-person chair dean)

10: (!performance-points 1 -1)(!convenience-points -1 -3)(!cost-points 1 -1)

11: (!no-key-backup chair) (!no-key-backup dean)

12: (!encrypt-using-shared-symmetric-key chair final-recom)

13: (!symmetric-sign chair dean final-recom)

14: (!send-encrypted-information chair dean final-recom)

15: (!transmit-using-online-file-storage-systems chair dean final-recom)

16: (!performance-points -1 1)(!convenience-points -3 -1)(!cost-points -1 1)

17: (!verify-symmetric-signature dean final-recom)

18: (!decrypt-using-shared-symmetric-key dean final-recom)

19: (!security-requirements7 chair dean final-recom)

20: (!total-convenience-points -1) (!total-performance-points 1)

21: (!total-cost-points 1) (!total-quality-points 0 1)
```

The planner provides a recommendation to the users to use symmetric encryption including digital signatures, by sharing the keys in person as a solution to the assumed vulnerabilities. In other words, considering the number of participants that is composed of two, the planner suggests a plan that provides them with the necessary steps to achieve symmetric encryption including digital signatures, that when followed, leads to a confidential exchange of documents while preserving information integrity. A summary of the plans generated in the academic domain is presented in Table 4.7.

Having analyzed the business processes, security requirements with its corresponding attack trees, vulnerabilities assumptions, and domain facts using SHOP2, we now move to address the remaining research goals of this framework.

To draw possible attack patterns and their complications that match the goals and the activities of this domain, the confidentiality and integrity attack trees (as seen in Section 3.5) were used as a reference. As stated earlier, considering that the institution's network is safe, the attacker's goal stating a compromised network in the attack trees can be ruled out, leaving two areas of a possible vulnerability: (1) a compromised mailbox that as a result of stolen credentials or through the interference of a employee with highly administrative rights, and (2) subsequently having a compromised device (i.e. mobile phone). The instantiation of such information from attack trees reinforces the evaluation goal of this framework that constructed attack trees are also reusable in this domain (research goal c in Section 4.1).

Domain facts, on the other hand, were successfully reflecting the initial state of the business process, such as when describing the number of participants (e.g. less than or more than two members), or whether they already had installed an encryption software during the presentation of the second and third goal. The identified vulnerability assumptions were effectively translated in the problem specification in accordance with attack trees. Hence, supporting the point that domain facts and vulnerability assumptions can be translated into the problem specification of the planner (research goal d in Section 4.1).

Domain facts along with vulnerability assumptions were able to impact the plans generated by the planner according to the goal being addressed.

For instance, during the transfer of the CVs between the secretary and the committee, confidentiality was only required, yet SHOP2 provided a plan that includes asymmetric encryption instead of symmetric encryption to achieve the security requirement, due to the presence of the domain fact stating a number of participants larger than two.

Additionally, the planner was changing how participants would share their encryption keys depending on the vulnerability assumptions at hand. For example, during the transfer of the final report between the chair and the dean, while considering a hacked mailbox and mobile phone, SHOP2 suggested the sharing keys in person, however, when only a hacked mailbox was present, all the other options were suggested (by phone and in person). Thus, the planner is generating plans against the desired security requirements while taking in consideration domain facts and addressing vulnerability assumptions at hand. Therefore, supporting the research goal that the plans generated by SHOP2 are actually meaningful to the domain (research goal e in Section 4.1).

As shown in Table 4.7, three major primary goals were discussed in this domain. Considering the low generation runtime taken for SHOP2 to generate plans for these goals, a change in domain facts and/or vulnerability assumptions is slightly increasing this factor by merely few milliseconds. Consequently, analysts can almost instantly receive the output of the desired goals. Hence, confirming that SHOP2 is being efficient in generating the plans of this domain.

It's noticeable to note that although SHOP2 was able to generate plans in a timely manner, the leaf trees in the workflow patterns that represent actions or HTN tasks had to be reduced. For instance, the workflow pattern "Transmission Method" (Section 3.3.5) had specific examples such as

Google Drive, OneDrive, Gmail, Microsoft, and others. However, when leaf trees contained numerous options, SHOP2 was taking around 5 minutes to generate a plan, and might take additional time depending on the security requirements and the number of tasks to be achieved under a primary goal.

Chapter 5

conclusion

5.1 Summary and Contributions

We presented a framework to enhance goal oriented requirement models with the elicitation of security requirements described by relevant security controls. The purpose of such framework is to assist normal users and inexperienced individuals in the development of secure business processes by providing different workflow patterns that include the necessary activities to attain a desired security objective. Such workflows contain security and non-security patterns, in which the former is used not only to explain relevant security controls but to address vulnerabilities in the system as well. These vulnerabilities are identified through attack trees representing the attacker's goals, that when successfully carried out, leads to a breach in security requirements, such as confidentiality and integrity.

Modelling business processes with their relevant security requirements is carried out using the social view of the STS-tool, while workflow patterns and attack trees are created by adopting the elements of the i* framework.

Following the incorporation of business activities and security requirements with the relevant workflow patterns, a hierarchical AI planner, SHOP2, is used to translate and compile these information. Once realized, the planner generates plans that fulfill business activities while meeting the desired security objectives and addressing the vulnerabilities at hand.

The translation of the aforementioned information to HTN is done through preparing a domain specification and a problem specification. The former contains the translation of goals and tasks/actions, whereas the latter is composed of the initial state (that contains domain facts, vulnerability assumptions, and quality facts), and the goal formula (i.e. the primary goal) to be addressed along it's relevant security requirements and quality methods.

To assess the effectiveness of the framework, we offer a preliminary evaluation through the application of different domains and use scenarios. This application is made using the health care domain and the academic domain. The first domain highlights the exchange of the receipt and the invoice between the patient and the secretary, in order for the patient to legitimately provide a feedback to the physician who provided the medical treatment. Whereas the second domain focuses on the exchange of candidate's resumes and relevant recommendations reports between faculty members to fill an academic vacancy, in order to prevent unauthorized faculty members or third-party members to access such information.

In both domains, we were able to model the business activities using the STS-tool with their relevant security requirements, which are analyzed through attack trees, and attach their corresponding workflow patterns accordingly. Such application provides us some confidence that workflow patterns and attack trees can developed and reused across different do-

mains. We translated the state of the actors and their business processes along with the activities they are willing to take in order to achieve a secure business activity through domain facts, and relevant vulnerability assumptions by analyzing potential activities of attack trees that can be considered according to the context of each domain. Accordingly, the adopted planner, SHOP2, was able to generate plans according to changes made in the initial state and/or in vulnerability assumptions. Consequently, actors of the domain can follow up with the security controls suggested by the planner to achieve their desired secure business activity.

This framework provides several contributions: (a) we extend an established goal-oriented security requirements framework with the capability of reasoning about security designs rather than just security requirements elicitation, (b) we provide an approach to model and develop workflow patterns by adopting the elements of the i* framework, (c) we present a way to interpret security requirements specifications as negated attacker goals, which in turn are analyzed into AND/OR attack trees to be used for reasoning about attacker's actions and security counter-actions, and (d) we translate the goals and tasks of stakeholders through an AI planner to include security requirements, vulnerability assumptions, and non-functional requirements.

5.2 Limitations

Although we provided a preliminary evaluation on the effectiveness of the framework, several limitations are present in the framework and during the application of the domains. Such limitations are presented as follows:

(a) Whilst we present security patterns based on bibliographical refer-

ences that address major security requirements obtained from the literature, patterns need to be further rectified, extended, and validated by the community security professionals to address the identified security requirements. Thus, despite the promise the security properties of the patterns per-se require much more investigation and validation effort.

(b) The application of patterns in the presented domains were limited to one major pattern ("transmit-information") that in turns calls other major patterns in the process. Several unused workflow patterns (e.g. "Manage information on disk" as presented in Section 3.3.7, Figure 3.8) and security requirements (e.g. authenticity, as presented in Section 3.5, Figure 3.14) exist. Hence, the range of security concerns that can be materialized into new patterns in the future is much wider.

(c) The plans generated by the planner (SHOP2) in each domain were built on the translation of security patterns found in the bibliographical references based on our understanding of cryptographic primitives and not validated by security experts. Such patterns were designed to be incorporated with simple business activities that are carried out by inexperienced users, which supports their reusability across different domains.

(d) Although the generation of plan(s) using SHOP2 was done in a timely manner. Some leaf trees that represent actions (translated as HTN tasks) had to be reduced (e.g. using email services instead of listing Google and Microsoft). Such behavior made SHOP2 generate a much larger number of plans considering that each option would be considered as a plan. Additionally, the generation time of plans was significantly higher as the number tasks and methods increase under

a primary goal.

(e) Comparing plans according to non-functional requirements (such as performance and convenience) was only feasible when the number of generated plans was low. A high number of plans renders the use of this option impractical. It may be possible to use the cost of HTN tasks to filter plans. For example, the application of a higher cost to operators that include extra security measures will signify that plans with the highest number of cost denote a much more secure plan and plans with the lowest cost denote a plan that focuses on performance instead. Nevertheless, this method also has its limitations as only two opposite qualities can be compared (e.g. security vs performance).

(f) SHOP2 does not print the goal in the output of the generated plan (e.g. "process-applications" as shown in Figure 4.3), hence we included a task in the beginning of each plan being generated (e.g. "process-applications-plan") as a reference to the goal to be achieved.

5.3 Future Work

Throughout the framework, we address security requirements such as confidentiality and integrity, however, security requirements such as the anonymity of users exchanging information or the identity of users to the provider performing the service is yet to be achieved. This indicates more research and work needs to be done to assess whether the framework successfully incorporates privacy requirements and security requirements, and whether such representation and translation of requirements is attainable by the planner.

The domains presented in this paper offer only a preliminary evaluation of

the framework. Additional domains can be also included to support our evaluation points, and use case scenarios taking place between the actors provided in each domain has yet to be validated. On the other hand, the application of this framework still needs to be verified by normal users to assess whether the framework is actually feasible, and practically efficient rather than time consuming with no or little benefits gained.

The content of the problem specification currently includes domain facts about the status of actors (e.g. "knows secretary patient" or "willing-to-generate-keys patient"). However, additional domain facts representing the state of the documents being transmitted can be also included, such as "has patient encrypted invoice", allowing the planner to generate plans from this point onward rather than printing all the security controls that need to be taken by the actors.

The framework itself is semi-automated at this point. The translation of goals and tasks of stakeholders including the workflow of information has to currently be manually translated into HTN, as well the translation of axioms and vulnerability assumptions. Additional work can be done to offer an automatic translation of the elements modelled from the STS-tool to the HTN script to minimize any unexpected errors and enhance users experience.

References

[1] H. Mouratidis, P. Giorgini, G. Manson, and I. Philp, "A Natural Extension of Tropos Methodology for Modelling Security," *Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA 2002, Seattle, WA, USA*, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.3357.

[2] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements through ownership, permission and delegation," *Proceedings of the IEEE International Conference on Requirements Engineering*, no. July, pp. 167–176, 2005. DOI: 10.1109/re.2005.43.

[3] H. Mouratidis and P. Giorgini, "Secure Tropos: A security-oriented extension of the Tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 2, pp. 285–309, 2007. DOI: 10.1142/S0218194007003240.

[4] E. S. YU and J. MYLOPOULOS, "From E-R To "a-R" — Modelling Strategic Actor Relationships for Business Process Reengineering," *International Journal of Cooperative Information Systems*, vol. 04, no. 02n03, pp. 125–144, 1995. DOI: 10.1142/s0218843095000056.

[5] J. Castro, M. Kolp, and J. Mylopoulos, "A Requirements-Driven Development Methodology," pp. 108–123, 2001.

[6] P. Bresciani and A. Perini, "Tropos : An Agent-Oriented Software Development Methodology," pp. 203–236, 2004.

[7] E. Paja, F. Dalpiaz, and P. Giorgini, "Modelling and reasoning about security requirements in socio-technical systems," *Data and Knowledge Engineering*, vol. 98, pp. 123–143, 2015. DOI: 10.1016/j.datak.2015.07.007. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2015.07.007.

[8] D. S. Nau, S. J. Smith, and K. Erol, "Control strategies in HTN planning: Theory versus practice," *Innovative Applications of Artificial Intelligence - Conference Proceedings*, vol. 1, pp. 1127–1133, 1998.

[9] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 Language Guide," pp. 1–15, 2016. arXiv: 1605.07767. [Online]. Available: http://arxiv.org/abs/1605.07767.

[10] E. Gonçalves, G. Rodrigues, P. Miranda, J. Pimentel, J. Araujo, and J. Castro, "piStar-ext: Supporting the creation of iStar extensions with the piStar tool," *CEUR Workshop Proceedings*, vol. 2641, pp. 31–36, 2020.

[11] M. Wooldridgey and P. Ciancarini, "Agent-oriented software engineering: The state of the art," in *Agent-Oriented Software Engineering*, P. Ciancarini and M. J. Wooldridge, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–28.

[12] A. Sturm and O. Shehory, "Agent-oriented software engineering: Revisiting the state of the art," *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, vol. 9783642544323, pp. 13–26, 2014. DOI: 10.1007/978-3-642-54432-3_2.

[13] F. Massacci, J. Mylopoulos, and N. Zannone, "Security requirements engineering: The si*modeling language and the Secure Tropos methodology," *Studies in Computational Intelligence*, vol. 265, pp. 147–174, 2010. DOI: 10.1007/978-3-642-05183-8_6.

[14] P. Giorgini, F. Massacci, and N. Zannone, "Security and trust requirements engineering," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3655, pp. 237–272, 2005. DOI: 10.1007/11554578_8.

[15] A. Siena, J. Mylopoulos, A. Perini, and A. Susi, "Law-Compliant Software Requirements," no. 2005, pp. 472–486, 2009.

[16] G. Elahi and E. Yu, "Trust trade-off analysis for security requirements engineering," *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 243–248, 2009. DOI: 10.1109/RE.2009.12.

[17] H. Mouratidis, S. Islam, C. Kalloniatis, and S. Gritzalis, "A framework to support selection of cloud providers based on security and privacy requirements," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2276–2293, 2013. DOI: 10.1016/j.jss.2013.03.011. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.03.011.

[18] C. Kalloniatis, H. Mouratidis, and S. Islam, "Evaluating cloud deployment scenarios based on security and privacy requirements," *Requirements Engineering*, vol. 18, no. 4, pp. 299–319, 2013. DOI: 10.1007/s00766-013-0166-7.

[19] D. Karagiannis, H. C. Mayr, and J. Mylopoulos, "Domain-specific conceptual modeling: Concepts, methods and tools," *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, pp. 1–594, 2016. DOI: 10.1007/978-3-319-39417-6.

[20] C. Kalloniatis, E. Kavakli, and S. Gritzalis, "Addressing privacy requirements in system design: The PriS method," *Requirements Engineering*, vol. 13, no. 3, pp. 241–255, 2008. DOI: 10.1007/s00766-008-0067-3.

[21] M. Salnitri, K. Angelopoulos, M. Pavlidis, V. Diamantopoulou, H. Mouratidis, and P. Giorgini, "Modelling the interplay of security, privacy and trust in sociotechnical systems: a computer-aided design approach," *Software and Systems Modeling*, vol. 19, no. 2, pp. 467–491, 2020. DOI: 10.1007/s10270-019-00744-x. [Online]. Available: https://doi.org/10.1007/s10270-019-00744-x.

[22] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 249–261, 2001. DOI: 10.1109/isre.2001.948567.

[23] A. V. Lamsweerde, "IEEE Xplore Abstract - Elaborating security requirements by construction of intentional anti-models," 2004.

[24] N. R. Mead and T. Stehney, "Security quality requirements engineering (SQUARE) methodology," *SESS 2005 - Proceedings of the 2005 Workshop on Software Engineering for Secure Systems - Building Trustworthy Applications*, pp. 1–7, 2005. DOI: 10.1145/1082983.1083214.

[25] A. Bijwe and N. R. Mead, "Adapting the SQUARE Process for Privacy Requirements Engineering," no. July, pp. 1–20, 2010. [Online]. Available: https://resources.sei.cmu.edu/asset_files/technicalnote/2010_004_001_15185.pdf.

[26] S. Miyazaki, N. Mead, and J. Zhan, "Computer-aided privacy requirements elicitation technique," *Proceedings of the 3rd IEEE Asia-*

Pacific Services Computing Conference, APSCC 2008, pp. 367–372, 2008. DOI: 10.1109/APSCC.2008.263.

[27] D. Mellado, E. Fernández-Medina, and M. Piattini, "A common criteria based security requirements engineering process for the development of secure information systems," *Computer Standards and Interfaces*, vol. 29, no. 2, pp. 244–253, 2007. DOI: 10.1016/j.csi.2006.04.002.

[28] P. Salini and S. Kanmani, "Model Oriented Security Requirements Engineering (MOSRE) framework for web applications," *Advances in Intelligent Systems and Computing*, vol. 177 AISC, no. VOL. 2, pp. 341–353, 2013. DOI: 10.1007/978-3-642-31552-7_36.

[29] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management," *Technical University Dresden*, pp. 1–98, 2010. [Online]. Available: http://dud.inf.tu-dresden.de/Anon_Terminology.shtml%5Cnhttp://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf.

[30] M. Hafiz, P. Adamczyk, and R. Johnson, "Growing a pattern language (for security)," *SPLASH 2012: Onward! 2012 - Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 139–158, 2012. DOI: 10.1145/2384592.2384607.

[31] V. Diamantopoulou, C. Kalloniatis, S. Gritzalis, and H. Mouratidis, "Supporting privacy by design using privacy process patterns," *IFIP Advances in Information and Communication Technology*, vol. 502, pp. 491–505, 2017. DOI: 10.1007/978-3-319-58469-0_33.

[32] M. Peixoto, C. Silva, H. Maia, and J. Araújo, "Towards a catalog of privacy related concepts," *CEUR Workshop Proceedings*, vol. 2584, 2020.

[33] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, "Security patterns repository version 1.0," *DARPA, Washington DC*, 2002.

[34] D. Hatebur, M. Heisel, and H. Schmidt, "A pattern system for security requirements engineering," *Proceedings - Second International Conference on Availability, Reliability and Security, ARES 2007*, pp. 356–365, 2007. DOI: 10.1109/ARES.2007.12.

[35] M. Weiss and H. Mouratidis, "Selecting security patterns that fulfill security requirements," *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08*, pp. 169–172, 2008. DOI: 10.1109/RE.2008.32.

[36] T. Heyman, R. Scandariato, C. Huygens, and W. Joosen, "Using security patterns to combine security metrics," *ARES 2008 - 3rd International Conference on Availability, Security, and Reliability, Proceedings*, no. March 2014, pp. 1156–1163, 2008. DOI: 10.1109/ARES.2008.54.

[37] N. J. Nilsson and R. E. Fikes, "STRIPS : A New Approach to the Application of Theorem Proving to," *Artificial Intelligence*, vol. 8, no. October, pp. 189–208, 1971.

[38] C. Aeronautiques *et al.*, "Pddl— the planning domain definition language," Technical Report, Tech. Rep., 1998.

[39] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone, "Planning," pp. 33–47, 2006.

[40] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering," *Requirements Engineering*, vol. 16, no. 3, pp. 227–249, 2011. DOI: 10.1007/s00766-011-0129-9.

[41] B. Schneier, "Attack trees: Modeling security threats," *Dr. Drobb's Journal*, pp. 21–29, 1999.

[42] I. A. Tøndel, J. Jensen, and L. Røstad, "Combining misuse cases with attack trees and security activity models," *ARES 2010 - 5th International Conference on Availability, Reliability, and Security*, pp. 438–445, 2010. DOI: 10.1109/ARES.2010.101.

[43] I. Ray and N. Poolsapassit, "Using attack trees to identify malicious attacks from authorized insiders," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3679 LNCS, pp. 231–246, 2005. DOI: 10.1007/11555827_14.

[44] S. Mauw and M. Oostdijk, "LNCS 3935 - Foundations of Attack Trees," *Lecture Notes in Computer Science*, vol. 3935, pp. 186–198, 2006.

[45] L. Pietre-Cambacedes and M. Bouissou, "Beyond attack trees: Dynamic security modeling with Boolean logic Driven Markov Processes (BDMP)," *EDCC-8 - Proceedings of the 8th European Dependable Computing Conference*, no. May, pp. 199–208, 2010. DOI: 10.1109/EDCC.2010.32.

[46] C. W. Ten, C. C. Liu, and M. Govindarasu, "Vulnerability assessment of cybersecurity for SCADA systems using attack trees," *2007 IEEE Power Engineering Society General Meeting, PES*, no. November 2014, 2007. DOI: 10.1109/PES.2007.385876.

[47] V. Saini, Q. Duan, and V. Paruchuri, "Threat modeling using attack trees," *Journal of Computing Sciences in Colleges*, vol. 23, no. 4, pp. 124–131, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1352100.

[48] P. A. Khand, "System level security modeling using attack trees," *2009 2nd International Conference on Computer, Control and Communication, IC4 2009*, 2009. DOI: 10.1109/IC4.2009.4909245.

[49] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, "Using attack-defense trees to analyze threats and countermeasures in an ATM: A case study," *Lecture Notes in Business Information Processing*, vol. 267, pp. 326–334, 2016. DOI: 10.1007/978-3-319-48393-1_24.

[50] T. Li, E. Paja, J. Mylopoulos, J. Horkoff, and K. Beckers, "Security attack analysis using attack patterns," *Proceedings - International Conference on Research Challenges in Information Science*, vol. 2016-Augus, 2016. DOI: 10.1109/RCIS.2016.7549303.

[51] M. V. Pawar and J. Anuradha, "Network security and types of attacks in network," *Procedia Computer Science*, vol. 48, no. C, pp. 503–506, 2015. DOI: 10.1016/j.procs.2015.04.126. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.04.126.

[52] J. Zhou and D. Gollmann, "Observations on non-repudiation," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 1996, pp. 133–144.

[53] J. Zhou and D. Gollmann, "Evidence and non-repudiation," *Journal of Network and Computer Applications*, vol. 20, no. 3, pp. 267–281, 1997.

[54] A. Kulshrestha and S. K. Dubey, "A Literature Review on Sniffing Attacks in Computer Network," *International Journal of Advanced Engineering Research and Science (IJAERS)*, vol. 1, no. 2, pp. 67–73, 2014.

[55] M. Iqbal and R. Matulevičius, "Comparison of Blockchain-Based Solutions to Mitigate Data Tampering Security Risk," *Lecture Notes in Business Information Processing*, vol. 361, no. August 2020, pp. 13–28, 2019. DOI: 10.1007/978-3-030-30429-4_2.

[56] B. Schneier, "Applied Cryptography," *Electrical Engineering*, vol. 1, no. [32, pp. 429–455, 1996. DOI: 10.1.1.99.2838. arXiv: arXiv:1011.1669v3. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.2838&rep=rep1&type=pdf.

[57] D. Nau *et al.*, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003. DOI: 10.1613/jair.1141.

Appendices

Appendix A

Operators

```
;**** quality points ***
(:operator (!convenience-points ?old ?new)
           ((have-convenience ?old))
           ((have-convenience ?old))
           ((have-convenience ?new))
)
(:operator (!performance-points ?old ?new)
           ((have-performance ?old))
           ((have-performance ?old))
           ((have-performance ?new))
)
(:operator (!cost-points ?old ?new)
           ((have-cost ?old))
           ((have-cost ?old))
           ((have-cost ?new))
)
(:operator (!total-convenience-points ?new)
           () ()
           ((print-convenience-points ?new))
)
(:operator (!total-performance-points ?new)
```

```
           () ()
           ((print-performance-points ?new))
)
(:operator (!total-cost-points ?new)
           () ()
           ((print-cost-points ?new))
)
(:operator (!total-quality-points ?old ?new)
           ((have-quality ?old))
           ((have-quality ?old))
           ((have-quality ?new))
)
 ;**** no encryption ***
(:operator (!no-encryption-method ?sender ?recipient ?info)
           () ()
           ((do-not-choose-encryption ?sender ?recipient ?info))
)
(:operator (!send-unencrypted-information ?sender ?recipient ?info)
           ((do-not-choose-encryption ?sender ?recipient ?info))
           ()
           ((shared-publicly-unencrypted ?info))
)
;**** symmetric Encryption ***
(:operator (!symmetric-encryption-method ?sender ?recipient ?info)
           () ()
           ((choose-encryption ?sender ?recipient ?info)
           (info-may-not-be-sniffed ?sender ?recipient ?info))
)
(:operator (!install-encryption-software ?user)
           ((willing-to-install-software ?user))
           ()
           ((has ?user encryption-software)
            (choose-encryption ?sender ?recipient ?info))
)
```

```
(:operator (!generate-symmetric-key ?user ?info)
           ((has ?user encryption-software))
           ()
           ((has ?user sym-key ?info))
)
(:operator (!share-symmetric-key ?sender ?recipient)
           ((has ?sender sym-key ?info))
           ()
           ((has ?recipient (key ?sender symmetric)))
)
(:operator (!encrypt-using-shared-symmetric-key ?sender ?info)
           ((or (has ?sender sym-key ?info)
            (has ?sender decrypted (sym-key ?sender private))))
           ()
           ((has ?sender (encrypted ?info (key ?sender symmetric)))
           (info-may-not-be-sniffed ?sender ?info))
)
;system-encryption
(:operator (!encrypt-using-shared-sym-key ?sender ?info for ?recipient)
           ((or (has ?sender sym-key ?info)
            (has ?sender decrypted (sym-key ?sender private))))
           ()
           ((has ?sender (encrypted ?info (key ?sender symmetric)))
           (info-may-not-be-sniffed ?sender ?info))
)
(:operator (!symmetric-sign ?sender ?recipient ?info)
           ((has ?sender (encrypted ?info (key ?sender symmetric))))
           ()
           ((has ?sender signed (encrypted ?info (key ?sender
           symmetric)))(info-may-not-be-tampered ?sender ?recipient
           ?info))
)
(:operator (!send-encrypted-information ?sender ?recipient ?info)
           ((or (has ?sender (encrypted ?info (key ?sender symmetric)))
```

```
            (has ?sender (encrypted ?info (key ?recipient public)))))
            ()
            ((is-at ?recipient ?info))
)
(:operator (!verify-symmetric-signature ?recipient ?info)
            ((has ?sender signed (encrypted ?info (key ?sender
            symmetric))))
            ()
            ((has ?sender verified (encrypted ?info (key ?sender
            symmetric))))
)
(:operator (!decrypt-using-shared-symmetric-key ?recipient ?info)
            ((has ?sender (encrypted ?info (key ?sender symmetric)))
            (or (has ?recipient (key ?sender symmetric))
            (has ?system decrypted (sym-key ?system private))))
            ()
            ((has ?sender (decrypted ?info (key ?recipient symmetric))))
)
;*** asymmetric encryption ***
(:operator (!asymmetric-encryption-method ?sender ?recipient ?info)
            () ()
            ((choose-encryption ?sender ?recipient ?info)
            (info-may-not-be-sniffed ?sender ?recipient ?info))
)
(:operator (!generate-public-private-key-pair ?user)
            ((willing-to-generate-keys ?user))
            ()
            ((has ?user public-key) (has ?user private-key))
)
(:operator (!generate-system-public-private-key-pair ?system ?info)
            () ()
            ((has ?system public-key) (has ?system private-key))
)
(:operator (!share-public-key ?sender ?recipient ?info)
```

```
            ((has ?sender public-key))
            ()
            ((has ?sender (key ?recipient public)))
)
(:operator (!choice-is-exclude-digital-signature ?sender ?recipient
            ?info)
            ((choose-encryption ?sender ?recipient ?info)
            (has ?sender public-key)(has ?sender private-key)
            (has ?recipient public-key)(has ?recipient private-key)
            (or (has ?sender (key ?recipient public))
            (shared (key ?sender public))(shared (key ?recipient
            public)))) () ()
)
(:operator (!choice-is-to-include-digital-signature ?sender ?recipient
            ?info)
            ((choose-encryption ?sender ?recipient ?info)
            (has ?sender public-key)(has ?sender private-key)
            (has ?recipient public-key) (has ?recipient private-key)
            (or (has ?sender (key ?recipient public))
            (shared (key ?sender public))(shared (key ?recipient
            public)))) () ()
)
(:operator (!encrypt ?sender ?info (public-key ?recipient))
            ((or (has ?sender (key ?recipient public))
            (shared (key ?user public))))
            ()
            ((has ?sender (encrypted ?info (key ?recipient public)))
            (info-may-not-be-sniffed ?sender ?recipient ?info)
            (shared-publicly ?info))
)
(:operator (!sign ?sender ?info (private-key ?sender))
            ((has ?sender (encrypted ?info (key ?recipient public))))
            ()
            ((has ?sender (signed (encrypted ?info (key ?sender
```

```
           public))))(info-may-not-be-repudiated ?sender ?recipient
           ?info) (info-may-not-be-tampered ?sender ?recipient ?info))
)
(:operator (!verify ?recipient ?info (public-key ?sender))
           ((has ?sender (signed (encrypted ?info (key ?sender
           public))))) ()
           ((has ?recipient (verified (encrypted ?info
           (key ?sender public)))))
)
(:operator (!decrypt ?recipient ?info (private-key ?recipient))
           ((has ?sender (encrypted ?info (key ?recipient public))))
           ()
           ((has ?recipient (decrypted ?info (key ?recipient private))))
)
;*** hybrid encryption ***
(:operator (!encryption-method-is-hybrid-encryption ?sender ?recipient
            ?info)
           () ()
           ((choose-encryption ?sender ?recipient ?info))
)
(:operator (!create-symmetric-key ?client ?info)
           () ()
           ((has ?client sym-key ?info))
)
(:operator (!encrypt-sym-key-using-public-key ?system ?info)
           ((has ?client sym-key ?info))
           ()
           ((has ?client (encrypted-sym-key (public-key ?system))))
)
(:operator (!certify-and-share-keys-using-certificate-authority ?user)
           ((willing-to-pay ?user)(has ?user public-key))
           ()
           ((obtained-digital-certificate ?user))
)
```

```
(:operator (!send-symmetric-key ?client ?system ?info)
           ((has ?client (encrypted-sym-key (public-key ?system))))
           ()
           ((is-at ?system (encrypted-sym-key (public-key ?system))))
)
(:operator (!decrypt-symmetric-key-with-private-key ?system ?info)
           ((is-at ?system (encrypted-sym-key (public-key ?system))))
           ()
           ((has ?system decrypted (sym-key ?system private)))
)
;*** key exchange***
(:operator (!no-key-exchange ?sender ?recipient)
           ((do-not-choose-encryption ?sender ?recipient ?info))
           () ()
)
(:operator (!share-keys-by-email ?sender ?recipient)
           ((knows ?sender ?recipient)(knows ?recipient ?sender))
           ()
           ((or (has ?recipient (key ?sender symmetric))
           (has ?sender (key ?recipient public)))
           (pass-through-data-network key)
           (is-at ?recipient mailbox key))
)
(:operator (!share-keys-by-phone-SMS ?sender ?recipient)
           ((knows ?sender ?recipient)(knows ?recipient ?sender))
           ()
           ((or (has ?recipient (key ?sender symmetric))
           (has ?sender (key ?recipient public)))
           (shared-publicly (key ?sender shared))
           (pass-through-mobile-network key)
           (shared-by-phone-SMS ?sender ?recipient key)
           (is-at ?recipient phone key))
)
(:operator (!share-keys-in-person ?sender ?recipient)
```

```
            ((can-meet ?sender ?recipient)(can-meet ?recipient ?sender))
            ()
            ((or (has ?recipient (key ?sender symmetric))
            (has ?sender (key ?recipient public)))
            (shared-in-person ?sender ?recipient key)
            (shared-in-person ?sender ?recipient (key ?sender shared))
            (is-with ?recipient key))
)
;*** disk encryption ***
(:operator (!download-info-on-disk ?user ?info)
            () ()
            ((is-at ?user disk))
)
(:operator (!encryption-choice-is-no-encryption-on-disk ?user ?info)
            () ()
            ((do-not-choose-encryption-on-disk ?user))
)
(:operator (!encryption-choice-is-symmetric-encryption-on-disk ?user
             ?info)
            ()()()
)
(:operator (!encrypt-info-using-symmetric-key ?user ?info)
            ((has ?user sym-key ?info))
            ()
            ((has ?user (encrypted ?info (key ?user symmetric))))
)
(:operator (!decrypt-info-using-symmetric-key ?sender ?info)
            ((has ?recipient (encrypted ?info (sym-key ?sender))))
            ()
            ((has ?recipient (decrypted ?info (sym-key ?sender))))
)
;*** key backup ***
(:operator (!no-key-backup ?user)
            ((or (do-not-choose-encryption ?sender ?recipient ?info)
```

```
            (choose-encryption ?sender ?recipient ?info)
            (has ?user sym-key ?info)
            (do-not-choose-encryption-on-disk ?user)))
            ()
            ((key-at local-drive))
)
(:operator (!backup-sym-key-on-media ?user)
           ((has ?recipient (key ?sender symmetric)))
           ()
           ((key-backup-successful ?user))
)
(:operator (!backup-private-key-on-media ?user)
           ((has ?user public-key) (has ?user private-key))
           ()
           ((key-backup-successful ?user))
)
(:operator (!store-media-in-safe-deposit ?user)
           ((key-backup-successful ?user))
           ()
           ((key-at-safe-deposit ?user))
)
;*** transfer method ***
(:operator (!transmit-using-online-file-storage-systems ?sender
            ?recipient ?info)
           ()()()
)
(:operator (!transmit-using-online-email-services ?sender ?recipient
            ?info)
           ()()()
)
(:operator (!transmit-info-in-person ?sender ?recipient ?info)
           ()()()
)
(:operator (!choice-is-to-delete-information-after-a-specific-time ?user
```

```
            ?info)
           ()()()
)
(:operator (!choice-is-to-delete-information-after-end-of-use ?user
            ?info)
           ()()()
)
(:operator (!choice-is-to-keep-information-indefinitely ?user ?info)
           ()()()
)
;*** Health Care Domain ***
; Obtain-physician-profile
(:operator (!obtain-physician-profile-plan)
           ()()()
)
(:operator (!authenticate ?user ?system)
           ()()
           ((logged-on ?user ?system))
)
(:operator (!search-physician-profile ?user ?system)
           ()()()
)
(:operator (!download-physician-profile ?user ?system)
           ()()
           ((profile-downloaded ?user ?system))
)
; Obtain-medical-treatment
(:operator (!obtain-medical-treatment-plan)
           ()()()
)
(:operator (!book-appointment ?patient ?physician)
           ()()()
)
(:operator (!obtain-consultation ?patient ?physician)
```

```
          ()()()
)
(:operator (!pay-for-medical-treatment ?patient ?physician)
          ()()()
)
(:operator (!enter-credit-card-information ?user ?terminal)
          ()()()
)
; *** University Domain ***
(:operator (!process-applications-plan ?secretary)
          ()()()
)
(:operator (!gather-applications ?secretary)
          ()()()
)
(:operator (!send ?sender ?recipient ?info)
          ()()()
)
(:operator (!obtain-invoice ?advertiser ?chair)
          ()()()
)
(:operator (!process-application-recommendations-plan ?chair)
          ()()()
)
(:operator (!evaluate-applications ?chair)
          ()()()
)
(:operator (!share ?sender ?recipient ?info)
          ()()()
)
(:operator (!receive ?committee ?chair ?com-recom)
          ()()()
)
(:operator (!finalize-recommendations-plan ?chair)
```

```
           ()()()
)
(:operator (!consolidate ?chair recommendations)
           ()()()
)
;*** Security Requirements ***
(:operator (!security-requirement ?sender ?recipient ?info)
           ((info-may-not-be-sniffed  ?sender ?recipient ?info)
            (not (interception-successful  ?sender ?recipient ?info)))
            ()()
)
(:operator (!security-requirements11  ?sender ?recipient ?info)
           ((info-may-not-be-sniffed  ?sender ?recipient ?info)
            (info-may-not-be-tampered  ?sender ?recipient ?info)
            (not (interception-successful  ?sender ?recipient ?info)))
            ()()
)
(:operator (!security-requirements2 ?secretary ?patient ?invoice)
           ((info-may-not-be-sniffed ?secretary ?patient ?invoice)
           (info-may-not-be-tampered ?secretary ?patient ?invoice)
           (not (interception-successful ?secretary ?patient ?invoice)))
           ()()
)
(:operator (!security-requirements3 ?secretary ?patient ?receipt)
           ((info-may-not-be-sniffed ?secretary ?patient ?receipt)
           (info-may-not-be-tampered ?secretary ?patient ?receipt)
           (not (interception-successful ?secretary ?patient ?receipt)))
           ()()
)
(:operator (!security-requirements4 ?secretary ?committee ?cvs)
           ((info-may-not-be-sniffed ?sender ?recipient ?info)
           (not (interception-successful ?secretary ?committee ?cvs)))
           ()()
)
```

```
(:operator (!security-requirements5 ?chair ?committee ?chair-recom)
           ((info-may-not-be-sniffed ?chair ?committee ?chair-recom)
           (not (interception-successful ?chair ?committee ?chair-
           recom))) ()()
)
(:operator (!security-requirements6 ?committee ?chair ?com-recom)
           ((info-may-not-be-sniffed ?committee ?chair ?com-recom)
           (not (interception-successful ?committee ?chair ?com-recom)))
           ()()
)
(:operator (!security-requirements7 ?chair ?dean ?final-recom)
           ((info-may-not-be-sniffed ?chair ?dean ?final-recom)
           (info-may-not-be-tampered ?chair ?dean ?final-recom)
           (not (interception-successful ?chair ?dean ?final-recom)))
           ()()
)
```

Appendix B

Methods

```
;*** Quality Points Methods ***
;*** Convenience ***
(:method (help-convenience ?c)
         ((have-convenience ?c))
         ((!convenience-points ?c (call + ?c 1)))
)
(:method (make-convenience ?c)
         ((have-convenience ?c))
         ((!convenience-points ?c (call + ?c 2)))
)
(:method (hurt-convenience ?c)
         ((have-convenience ?c))
         ((!convenience-points ?c (call - ?c 1)))
)
(:method (break-convenience ?c)
         ((have-convenience ?c))
         ((!convenience-points ?c (call - ?c 2)))
)
;*** Performance ***
(:method (help-performance ?p)
         ((have-performance ?p))
```

```
        ((!performance-points ?p (call + ?p 1)))
)
(:method (make-performance ?p)
        ((have-performance ?p))
        ((!performance-points ?p (call + ?p 2)))
)
(:method (hurt-performance ?p)
        ((have-performance ?p))
        ((!performance-points ?p (call - ?p 1)))
)
(:method (break-performance ?p)
        ((have-performance ?p))
        ((!performance-points ?p (call - ?p 2)))
)
;*** Cost ***
(:method (help-cost ?cost)
        ((have-cost ?cost))
        ((!cost-points ?cost (call + ?cost 1)))
)
(:method (make-cost ?cost)
        ((have-cost ?cost))
        ((!cost-points ?cost (call + ?cost 2)))
)
(:method (hurt-cost ?cost)
        ((have-cost ?cost))
        ((!cost-points ?cost (call - ?cost 1)))
)
(:method (break-cost ?cost)
        ((have-cost ?cost))
        ((!cost-points ?cost (call - ?cost 2)))
)
;*** Total Quality Points ***
(:method (total-performance-points ?p)
        ((have-performance ?p))
```

```
        ((!total-performance-points ?p))
)
(:method (total-convenience-points ?c)
        ((have-convenience ?c))
        ((!total-convenience-points ?c))
)
(:method (total-cost-points ?cost)
        ((have-cost ?cost))
        ((!total-cost-points ?cost))
)
(:method (quality-points ?q ?cost ?c ?p)
        ((have-quality ?q)
        (have-cost ?cost)
        (have-convenience ?c)
        (have-performance ?p))
        ((!total-quality-points ?q (call + ?q ?cost ?c ?p)))
)
;*** Setup Security Tools ***
(:method (setup-security-tools ?sender ?recipient ?info)
        ()
        (:ordered (identify-encryption-type ?sender ?recipient ?info)
                  (key-exchange ?sender ?recipient)
                  (key-backup ?sender ?recipient))
)
(:method (identify-encryption-type ?sender ?recipient ?info)
        ()
        (:ordered (exclude-encryption ?sender ?recipient ?info))
)
(:method (identify-encryption-type ?sender ?recipient ?info)
        ()
        (:ordered (include-encryption ?sender ?recipient ?info))
)
(:method (include-encryption ?sender ?recipient ?info)
        ()
```

```
        (:ordered (symmetric-encryption ?sender ?recipient ?info))
)
(:method (include-encryption ?sender ?recipient ?info)
        ()
        (:ordered (asymmetric-encryption ?sender ?recipient ?info))
)
(:method (include-encryption ?sender ?recipient ?info)
        ()
        (:ordered (hybrid-encryption ?sender ?recipient ?info))
)
; No user encryption
(:method (exclude-encryption ?sender ?recipient ?info)
        ()
        (:ordered (!no-encryption-method ?sender ?recipient ?info)
                  (make-performance ?p)
                  (break-convenience ?c)
                  (make-cost ?cost))
)
; Symmetric user encryption
(:method (symmetric-encryption ?sender ?recipient ?info)
 ((2-or-less ?sender ?recipient)(and (has ?sender encryption-software
  ?info) (has ?recipient encryption-software ?info)))
(:ordered (!symmetric-encryption-method ?sender ?recipient ?info)
          (!generate-symmetric-key ?sender ?info)
          (!share-symmetric-key ?sender ?recipient)
          (help-performance ?p)
          (hurt-convenience ?c)
          (help-cost ?cost))
((2-or-less ?sender ?recipient)(has ?sender encryption-software ?info))
(:ordered (!symmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?recipient)
          (!generate-symmetric-key ?sender ?info)
          (!share-symmetric-key ?sender ?recipient)
          (help-performance ?p)
```

```
          (hurt-convenience ?c)
          (help-cost ?cost))
((2-or-less ?sender ?recipient)
 (has ?recipient encryption-software ?info))
(:ordered (!symmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
          (!generate-symmetric-key ?sender ?info)
          (!share-symmetric-key ?sender ?recipient)
          (help-performance ?p)
          (hurt-convenience ?c)
          (help-cost ?cost))
((2-or-less ?sender ?recipient))
(:ordered (!symmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
          (!install-encryption-software ?recipient)
          (!generate-symmetric-key ?sender ?info)
          (!share-symmetric-key ?sender ?recipient)
          (help-performance ?p)
          (hurt-convenience ?c)
          (help-cost ?cost))
)
; Asymmetric user encryption
(:method (asymmetric-encryption ?sender ?recipient ?info)
 ((more-than-2 ?sender ?recipient)
 (and (has ?sender encryption-software ?info)
 (has ?recipient encryption-software ?info)))
 (:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
           (!generate-public-private-key-pair ?sender)
           (!generate-public-private-key-pair ?recipient)
           (!share-public-key ?sender ?recipient ?info)
           (!share-public-key ?recipient ?sender ?info)
           (break-performance ?p)
           (help-convenience ?c)
           (hurt-cost ?cost))
```

```
((more-than-2 ?sender ?recipient)
 (has ?sender encryption-software ?info))
(:ordered  (!asymmetric-encryption-method ?sender ?recipient ?info)
           (!install-encryption-software ?recipient)
           (!generate-public-private-key-pair ?sender)
           (!generate-public-private-key-pair ?recipient)
           (!share-public-key ?sender ?recipient ?info)
           (!share-public-key ?recipient ?sender ?info)
           (break-performance ?p)
           (help-convenience ?c)
           (hurt-cost ?cost))
((more-than-2 ?sender ?recipient)
 (has ?recipient encryption-software ?info))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
((more-than-2 ?sender ?recipient))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
          (!install-encryption-software ?recipient)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
((2-or-less ?sender ?recipient)
```

```
 (and (has ?sender encryption-software ?info)
      (has ?recipient encryption-software ?info)))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
((2-or-less ?sender ?recipient)(has ?sender encryption-software ?info))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?recipient)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
((2-or-less ?sender ?recipient)
 (has ?recipient encryption-software ?info))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
((2-or-less ?sender ?recipient))
(:ordered (!asymmetric-encryption-method ?sender ?recipient ?info)
          (!install-encryption-software ?sender)
```

```
          (!install-encryption-software ?recipient)
          (!generate-public-private-key-pair ?sender)
          (!generate-public-private-key-pair ?recipient)
          (!share-public-key ?sender ?recipient ?info)
          (!share-public-key ?recipient ?sender ?info)
          (break-performance ?p)
          (help-convenience ?c)
          (hurt-cost ?cost))
)
; hybrid encryption
(:method (hybrid-encryption ?sender ?recipient ?info)
 ()
 (:ordered (!encryption-method-is-hybrid-encryption ?sender ?recipient
            ?info)
           (!generate-system-public-private-key-pair ?sender ?info)
           (!certify-and-share-keys-using-certificate-authority ?sender)
           (!share-public-key ?sender ?recipient ?info)
           (!create-symmetric-key ?recipient ?info)
           (!encrypt-sym-key-using-public-key ?sender ?info)
           (!send-symmetric-key ?recipient ?sender ?info)
           (!decrypt-symmetric-key-with-private-key ?sender ?info)
           (help-performance ?p)
           (make-convenience ?c)
           (break-cost ?cost))
)
;*** Key Exchange Methods ***
(:method (key-exchange ?sender ?recipient)
         ()
         (:ordered (no-key-exchange ?sender ?recipient))
)
(:method (key-exchange ?sender ?recipient)
         ()
         (:ordered (exchange-sym-key ?sender ?recipient))
)
```

```
(:method (key-exchange ?sender ?recipient)
         ()
         (:ordered (exchange-asym-key ?sender ?recipient))
)
(:method (no-key-exchange ?sender ?recipient)
         ()
         (:ordered (!no-key-exchange ?sender ?recipient))
)
; email key exchange
(:method (exchange-sym-key ?sender ?recipient)
 ((has ?sender sym-key ?info))
 (:ordered (!share-keys-by-email ?sender ?recipient)
           (help-performance ?p)
           (help-convenience ?c)
           (make-cost ?cost))
)
(:method (exchange-asym-key ?sender ?recipient)
 ((has ?sender public-key))
 (:ordered (!share-keys-by-email ?sender ?recipient)
           (help-performance ?p)
           (help-convenience ?c)
           (make-cost ?cost))
)
; phone key exchange
(:method (exchange-sym-key ?sender ?recipient)
 ((has ?sender sym-key ?info))
 (:ordered (!share-keys-by-phone-SMS ?sender ?recipient)
           (hurt-performance ?p)
           (hurt-convenience ?c)
           (make-cost ?cost))
)
(:method (exchange-asym-key ?sender ?recipient)
 ((has ?sender public-key))
 (:ordered (!share-keys-by-phone-SMS ?sender ?recipient)
```

```
            (hurt-performance ?p)
            (hurt-convenience ?c)
            (make-cost ?cost))
)
; in person key exchange
(:method (exchange-sym-key ?sender ?recipient)
 ((has ?sender sym-key ?info))
 (:ordered (!share-keys-in-person ?sender ?recipient)
            (break-performance ?p)
            (break-convenience ?c)
            (break-cost ?cost))
)
(:method (exchange-asym-key ?sender ?recipient)
 ((has ?sender public-key))
 (:ordered (!share-keys-in-person ?sender ?recipient)
            (break-performance ?p)
            (break-convenience ?c)
            (break-cost ?cost))
)
; Certificate Authority
(:method (exchange-asym-key ?sender ?recipient)
 ()
 (:ordered (!certify-and-share-keys-using-certificate-authority ?sender)
            (!certify-and-share-keys-using-certificate-authority
            ?recipient)
            (make-performance ?p)
            (make-convenience ?c)
            (make-cost ?cost))
)
;*** Transmit Information ***
(:method (transmit-information ?sender ?recipient ?info)
    ()
    (:ordered (setup-security-tools ?sender ?recipient ?info)
    (transmit-unencrypted-information ?sender ?recipient ?info))
```

```
)
(:method (transmit-information ?sender ?recipient ?info)
    ()
    (:ordered (setup-security-tools ?sender ?recipient ?info)
    (transmit-encrypted-information ?sender ?recipient ?info))
)
; unencrypted
(:method (transmit-unencrypted-information ?sender ?recipient ?info)
    ()
    (:ordered (!send-unencrypted-information ?sender ?recipient ?info)
    (transfer-method ?sender ?recipient ?info))
)
;encrypted
(:method (transmit-encrypted-information ?sender ?recipient ?info)
    ()
    (:ordered (send-encrypted-information ?sender ?recipient ?info))
)
; symmetric + not signed
(:method (send-encrypted-information ?sender ?recipient ?info)
         ()
         (:ordered (!encrypt-using-shared-symmetric-key ?sender ?info)
         (!send-encrypted-information ?sender ?recipient ?info)
         (transfer-method ?sender ?recipient ?info)
         (!decrypt-using-shared-symmetric-key ?recipient ?info))
)
; symmetric + signed
(:method (send-encrypted-information ?sender ?recipient ?info)
         ()
         (:ordered (!encrypt-using-shared-symmetric-key ?sender ?info)
         (!symmetric-sign ?sender ?recipient ?info)
         (!send-encrypted-information ?sender ?recipient ?info)
         (transfer-method ?sender ?recipient ?info)
         (!verify-symmetric-signature ?recipient ?info)
         (!decrypt-using-shared-symmetric-key ?recipient ?info))
```

```
)
; asymmetric + not signed
(:method (send-encrypted-information ?sender ?recipient ?info)
         ()
         (:ordered (!choice-is-exclude-digital-signature ?sender
         ?recipient ?info)
         (hurt-performance ?p)
         (help-convenience ?c)
         (!encrypt ?sender ?info (public-key ?recipient))
         (!send-encrypted-information ?sender ?recipient ?info)
         (transfer-method ?sender ?recipient ?info)
         (!decrypt ?recipient ?info (private-key ?recipient)))
)
; asymmetric + signed
(:method (send-encrypted-information ?sender ?recipient ?info)
         ()
         (:ordered (!choice-is-to-include-digital-signature ?sender
         ?recipient ?info)
         (break-performance ?p)
         (make-convenience ?c)
         (!encrypt ?sender ?info (public-key ?recipient))
         (!sign ?sender ?info (private-key ?sender))
         (!send-encrypted-information ?sender ?recipient ?info)
         (transfer-method ?sender ?recipient ?info)
         (!verify ?recipient ?info (public-key ?sender))
         (!decrypt ?recipient ?info (private-key ?recipient)))
)
;*** Download Online Information ***
(:method (download-information ?user ?system ?info)
         ()
         (:ordered (disk-encryption-method ?user ?info)
         (Retention-period ?user ?info))
)
;*** Disk Encryption Methods ***
```

```
; No disk encryption
(:method (disk-encryption-method ?user ?info)
         ()
         (:ordered (!encryption-choice-is-no-encryption-on-disk
         ?user ?info)
         (make-performance ?p)
         (break-convenience ?c)
         (make-cost ?cost))
)
; Symmetric disk encryption
(:method (disk-encryption-method ?user ?info)
         ()
         (:ordered  (!encryption-choice-is-symmetric-encryption-on-disk
         ?user ?info)
         (!install-encryption-software ?user)
         (!generate-symmetric-key ?user ?info)
         (!encrypt-info-using-symmetric-key ?user ?info))
)
(:method (disk-encryption-method ?user ?info)
         ()
         (:ordered  (!encryption-choice-is-symmetric-encryption-on-disk
         ?user ?info)
         (!install-encryption-software ?user)
         (!generate-symmetric-key ?user ?info)
         (!encrypt-info-using-symmetric-key ?user ?info)
         (!symmetric-sign ?user ?recipient ?info))
)
;*** Key Backup Methods ***
; no key backup
(:method (key-backup ?sender ?recipient)
         ()
         (:ordered (!no-key-backup ?sender)
         (!no-key-backup ?recipient))
)
```

```
; symmetric key backup
(:method (key-backup ?sender ?recipient)
         ()
         (:ordered (!backup-sym-key-on-media ?sender)
         (!backup-sym-key-on-media ?recipient)
         (!store-media-in-safe-deposit ?sender)
         (!store-media-in-safe-deposit ?recipient))
)
; private key backup
(:method (key-backup ?sender ?recipient)
         ()
         (:ordered (!backup-private-key-on-media ?sender)
         (!backup-private-key-on-media ?recipient)
         (!store-media-in-safe-deposit ?sender)
         (!store-media-in-safe-deposit ?recipient))
)
;*** Retention Period Methods ***
(:method (Retention-period ?user ?info)
         ()
         (:ordered (!choice-is-to-delete-information-after-end-of-use
         ?user ?info)
         (make-convenience ?c))
)
(:method (Retention-period ?user ?info)
         ()
         (:ordered (!choice-is-to-delete-information-after-a-specific-
         time ?user ?info)
         (help-convenience ?c))
)
(:method (Retention-period ?user ?info)
         ()
         (:ordered (!choice-is-to-keep-information-indefinitely
         ?user ?info)
         (break-convenience ?c))
```

```
)
;*** Transfer Information Methods ***
(:method (transfer-method ?sender ?recipient ?info)
         ()
         (:ordered (!transmit-using-online-file-storage-systems ?sender
         ?recipient ?info)
         (make-performance ?p)
         (make-convenience ?c)
         (make-cost ?cost))
)
(:method (transfer-method ?sender ?recipient ?info)
         ()
         (:ordered (!transmit-using-online-email-services ?sender
         ?recipient ?info)
         (make-performance ?p)
         (make-convenience ?c)
         (make-cost ?cost))
)
(:method (transfer-method ?sender ?recipient ?info)
         ()
         (:ordered (!transmit-info-in-person ?sender ?recipient ?info)
         (break-performance ?p)
         (break-convenience ?c)
         (break-cost ?cost))
)
;*** Health Care Domain ***
(:method (obtain-medical-treatment ?patient ?secretary ?physician
          ?receipt ?invoice)
         ()
         (:ordered (!obtain-medical-treatment-plan)
         (!book-appointment ?patient ?physician)
         (!obtain-consultation ?patient ?physician)
         (!send ?secretary ?patient ?invoice)
         (transmit-information ?secretary ?patient ?invoice)
```

```
        (!security-requirements2 ?secretary ?patient ?invoice)
        (!pay-for-medical-treatment ?patient ?physician)
        (!send ?secretary ?patient ?receipt)
        (transmit-information ?secretary ?patient ?receipt)
        (!security-requirements3 ?secretary ?patient ?receipt))
)
; *** University Domain ***
(:method (process-applications ?secretary ?committee ?cvs)
        ()
        (:ordered (!process-applications-plan ?secretary)
        (!gather-applications ?secretary)
        (!send ?secretary ?committee ?cvs)
        (transmit-information ?secretary ?committee ?cvs)
        (!security-requirements4 ?secretary ?committee ?cvs))
)
(:method (process-application-recommendations ?chair ?committee
         ?chair-recom ?com-recom)
        ()
        (:ordered (!process-application-recommendations-plan ?chair)
        (!evaluate-applications ?chair)
        (!share ?chair ?committee ?chair-recom)
        (transmit-information ?chair ?committee ?chair-recom)
        (!security-requirements5 ?chair ?committee ?chair-recom)
        (!receive ?committee ?chair ?com-recom)
        (transmit-information ?committee ?chair ?com-recom)
        (!security-requirements6 ?committee ?chair ?com-recom))
)
(:method (finalize-recommendations ?chair ?dean ?final-recom)
        ()
        (:ordered (!finalize-recommendations-plan ?chair)
        (!consolidate ?chair recommendations)
        (!send ?chair ?dean ?final-recom)
        (transmit-information ?chair ?dean ?final-recom)
        (!security-requirements7 ?chair ?dean ?final-recom)))
```

Appendix C

Axioms

```
; Attack Axiom
(:- (interception-successful ?sender ?recipient ?info)
(shared-publicly-unencrypted ?info)
(and (is-compromised-data-network ?sender ?recipient)
     (pass-through-data-network key)
 (or (is-at ?sender mailbox key)(is-at ?recipient mailbox key)))
(and (is-compromised-data-network ?sender ?recipient)
     (pass-through-data-network key)
     (or (is-at ?sender network key)(is-at ?recipient network key)))
(and (is-compromised-mobile-network ?sender ?recipient)
     (pass-through-mobile-network key)(or (is-at ?sender phone key)
     (is-at ?recipient phone key)))
(and (or (is-hacked ?sender ?something)
         (is-hacked ?recipient ?something))
     (or (is-at ?sender ?something key)
         (is-at ?recipient ?something key)))
(and (or (is-followed ?sender) (is-followed ?recipient))
     (shared-in-person ?sender ?recipient key)(or (is-with ?sender key)
     (is-with ?recipient key))))
```