# Generating Secure Workflow Designs from Requirements Goal Models Using Patterns

Sotirios Liaskos[1] ✉, Ibrahim Jaouhar[1],
Syed Muhammad Danish[1], and Shakil M. Khan[2]

[1] School of Information Technology, York University,
{liaskos,jaouhar,smdanish}@yorku.ca
[2] Dept. of Computer Science, University of Regina, shakil.khan@uregina.ca

**Abstract.** Identifying and analyzing security requirements is an essential part of the information systems engineering lifecycle. Several techniques have been introduced for comprehensively modeling such requirements. Once identified, security requirements must be translated into designs that allow domain actors to securely accomplish business tasks under given risk assumptions and contexts. Correctly translating requirements to such designs, however, can be challenging when considering both the complexity and specialized nature of security mechanisms, such as cryptography, and the role of varying practical and contextual aspects of the problem at hand in correctly applying such mechanisms. We propose a model-driven pattern-based approach for supporting the implementation of security requirements. Security requirements models, augmented with descriptions of contextual and threat assumptions, are combined with reusable domain-agnostic workflow patterns which model established ways for securely performing common business tasks. The combined models are compiled into a formal specification, whereby automated reasoning is applied for generating domain-appropriate workflows that satisfy the security requirements. Using the technique, analysts can efficiently explore the impact of different threat assumptions and pragmatic constraints to candidate security designs, while ensuring that the latter are consistent with tried-and-tested community knowledge.

**Keywords:** Goal Modeling · Security Requirements · AI Planning

## 1 Introduction

Identifying and analyzing security requirements is considered to be an essential part of the early stages of the software development lifecycle [9,16,19,36]. Specifying such requirements upfront allows the security components that implement them to become an organic part of the resulting system [11,14,23]. A wealth of techniques for modeling security requirements have been proposed, aimed at allowing precise description of both such requirements and their relationship to other dimensions of the problem at hand [29,33,39].

However, developing models of security requirements is only the first step towards a secure system. A design must be produced that appropriately implements such requirements in accordance to widely accepted security standards.

For example, to support secure exchange of information, cryptographic primitives [32] of the right kind vis-à-vis the required security guarantees need to be introduced and correctly [15] applied. Furthermore, there may be multiple options with regards to the choice and practical application of these primitives based on varying perceptions about the key threats, vulnerabilities, and practical constraints in the domain. Effectively capturing and combining these different problem aspects to devise a well-fitted solution, that also includes the correct choice and use of security mechanisms, can be challenging without support.

We propose a model-driven framework for supporting the implementation of security requirements according to recommended best practices and consistent with pragmatic constraints and threat assumptions of the problem at hand. The framework is based on encoding such best practices as patterns that compactly describe a large number of alternative workflows for fulfilling common business tasks. Each such workflow includes or does not include different kinds of security-enhancing steps, and is accompanied by a set of attack models, which encode alternative attacks against security properties of the processes modeled by the pattern. To address a specific problem, goal-oriented requirements analysis is performed using a combination of general-purpose and security-centered goal models. Patterns that constitute appropriate implementations of security-relevant goals in those models are adopted and infused into the models. The result is subsequently augmented with a threat model, that describes the level of sophistication of the attacks to be defended against, as well as other pertinent domain constraints. It is then compiled into a formal specification that can be used for automatically identifying workflows that indicate what security steps domain actors need to take while pursuing their domain-specific business tasks.

The approach allows systematic exploration and derivation of security designs that comply with requirements, community best practices, and constraints of the problem at hand. Among its contributions are the use of high-variability goal decompositions as best practice-encoding models for security and the utilization of planners for high-level reasoning about both the security mechanisms and the socio-technical processes in which these are embedded. To demonstrate the technique, we apply it to a common inter-organizational order fulfillment scenario, using a prototypical inter-actor information exchange pattern.

The paper is organized as follows. We present the modeling process in Section 2 and the formalization and analysis details in Section 3. We offer related work in Section 4 and conclude in Section 5.

## 2 Developing Security-Enhanced Goal Models

### 2.1 Motivating Example and Framework Overview

Consider the example of a medium-sized residential building contractor requesting quotes, submitting orders, and sending payments to various material suppliers. Typically, the contractor submits orders for materials and, once the materials have been delivered, an invoice is sent with a payment information,

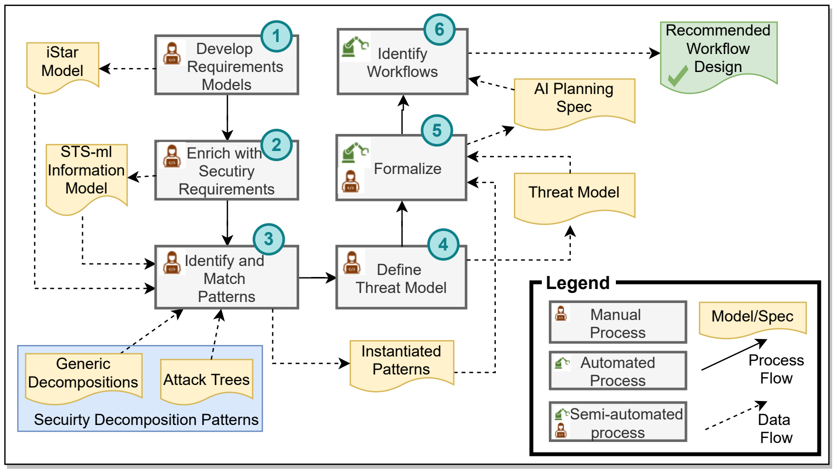

**Fig. 1.** Framework overview.

often in the form of a bank account number for sending a wire transfer. The contractor performs such interactions via email, which, however, introduces several threats, including the submission and fulfillment of unauthorized orders through spoofing, and invoice fraud, i.e., submission of a tampered invoice, where the payment information has been replaced with the attacker's account numbers.

If our contractor were a large company, a holistic B2B solution may have been devised by experts, acquired, and integrated with, e.g., the company's enterprise resource planning system. However, such solutions may not be accessible by smaller organizations that use common communication and productivity software such as word processors, emails, and shared drives to arrange common business transactions. Nevertheless, such systems offer security functions that could offer protection to at least some level. In the contractor example, simple encryption of the exchanged order and invoice documents from within their authoring software [2,35] could protect against certain kinds of attacks. However, without guidance, improper application of those tools and functions is possible. For example, sending a symmetrically encrypted invoice by email followed by a second email with the encryption key in plaintext does not protect against attack scenarios in which the email accounts are compromised. How the encryption key can be shared otherwise depends on what communication channels are available and how secure these are perceived to be, given an assessment of how motivated and capable potential attackers are. Contextual and practical aspects, such as the willingness and skill of actors across organizations to utilize specific software or take extra security steps may also affect the recommended solution.

The proposed framework is aimed at assisting analysts of cases such as the above in systematically generating secure workflow designs – i.e., sequences of steps that domain actors can take to accomplish business tasks in a secure way. The framework is based on the development and gradual refinement of a series of models of various aspects of the problem, the selection and adoption of authoritative patterns that encode multiple ways to implement security requirements, and the fusion of these developed and adopted artifacts into a formal specification that allows automated identification of solutions.

An overview can be seen in Figure 1. The analysts begin by developing goal models that describe the domain-specific functional and non-functional goals of stakeholders of the socio-technical system in question (Step 1). Subsequently, goals and dependencies that entail security-relevant actor activity are identified and the security requirements vis-à-vis those goals are specified (Step 2). For each of the security-relevant goals, analysts identify matching security decomposition patterns from a collection of such developed and supported by the community of experts. They adopt the identified patterns by instantiating them with domain specific information, and linking them to the matching security-relevant goals (Step 3). Subsequently, they develop a threat model that reflects the specific business context (Step 4). The subsequent tasks aim at identifying secure workflows, i.e., solutions of the instantiated patterns that describe actor activity to satisfy the domain goals of interest in a secure way. Specifically, the instantiated decomposition patterns are first transformed into a formal specification (Step 5) and the latter is used by an automated reasoner to generate the required solutions (Step 6). In the following we describe each of these steps.

## 2.2 Extending iStar 2.0: iStar T

The main modeling language we use in our proposal is based on an extension of the goal modeling standard iStar 2.0 [8], which we will here refer to as *iStar T*. An example of an iStar T model can be viewed in Figure 3. The model features most typical iStar 2.0 elements, such as *goals* (states of affairs that an actor wants to achieve) and *tasks* (actions that an actor can execute to achieve goals). For simplicity, the specific example omits additional iStar 2.0 elements such as *actors* and *qualities*, the latter being attributes for which an actor wants some level of achievement [8]. When a goal is AND-refined (resp., OR-refined) into a set of other goals and/or tasks, all (resp., exactly one) of the goals in that set needs to be satisfied for the parent goal to be considered satisfied.

The additional elements of *iStar T* are aimed at modeling temporal and causal constraints. Firstly, a vocabulary of *domain predicates* and *domain objects* is introduced to be used for constructing first-order *domain literals*, to, in turn, represent the state of the environment and how state changes as tasks are performed. Thus, *Has(john, encryptedDoc(invoice, publicKey(bill))* is a domain literal, constructed from predicate *Has* and objects *john*, *bill*, *encryptedDoc(·,·)*, and *publicKey(·)* – henceforth predicates and variables start with capital and individuals/functions with small letter. Two added visual elements contain domain predicates: *preconditions* and *effects*. Preconditions contain Boolean formulae of domain literals and effects contain lists of such. In both cases, arguments are either objects or variables. Three additional kinds of links are also introduced in *iStar T*: *precedence links* $\xrightarrow{pre}$, *negative precedence links* $\xrightarrow{npr}$ and *effect links* $\xrightarrow{eff}$.

Precedence (resp., negative precedence) links can be drawn from preconditions, goals, or tasks and can target goals or tasks, and mean that performance of the latter is not possible unless (resp., if), the origin is satisfied. Effect links are drawn from tasks to effects and signify that performance of the task makes the non-negated domain predicates listed in the effect true, and the negated false.

In addition, iStar T models may contain an *initialization template*. This element contains a list of patterns that describe what kinds of domain predicates need to be introduced to allow instance-level analysis. The patterns are of the form $\{P_1, P_2, \ldots : Pred(P_1, P_2, \ldots)\}$ and mean that instances of $Pred(P_1, P_2, \ldots)$ can be produced for various bindings of the corresponding variables with objects. For example, based on domain information, template {*S,R: HasEmailAddress(S, R)*}, may be instantiated to predicates *HasEmailAddress(contractor,supplier)* and *HasEmailAddress(accountant,contractor)* – or to no predicates at all.

Finally, the content of goals and tasks in iStar T are also first-order literals – e.g., *Encrypt(Sender, Recipient, Document)*. Replacing the parameters with domain objects signifies a specific instance of a goal or task, e.g., *Encrypt(john, bill, invoice)*. A subgoal, task, or effect can contain a subset of the parameters contained by the parent goal and every enabling precondition. Likewise, the parameters of literals in effects are a subset of those in the task that is associated with the effect. Typically, the leaf level elements of an iStar T decomposition model within the scope of an actor are tasks. However, it is possible that the decomposition is terminated at a predicate-containing goal.

Given a complete iStar T model there is variability in the ways by which tasks can be sequentially executed for fulfilling a root goal. A *task trajectory* $tH = \{a_1, a_2, \ldots\}$ (also: *solution* to the root goal) represents one such sequence that satisfies the added temporal constraints. It is, furthermore, *goal-satisfying* if the tasks included in $tH$ satisfy the root goal. The term *workflow* as discussed in this paper refers to such trajectories.

## 2.3 Modeling Domain and Security Requirements

We now turn our focus to how we use iStar T, in conjunction with other kinds of models, to generate security designs based on high-level security requirements.

Firstly, a goal-oriented analysis of the requirements of the problem at hand is performed using iStar T. In Figure 2, top diagram, the interaction between *Contractor* and *Supplier* is shown in the form of a hybrid strategic rationale / strategic dependency diagram [8]. The resulting model (here using only the iStar 2.0 subset for simplicity) offers the rationale and context in which security-relevant activity between the two actors needs to take place. To further identify such activity and specify the exact security associated with it we use STS-ml (Socio-Technical Security modeling language [9]). In STS-ml, a variety of views are defined for describing security requirements within a socio-technical context. We are here interested in the *social view* in which, among other things, the exchange of information between domain actors is displayed, along with security requirements that such exchange should satisfy. In Figure 2, bottom diagram, an *Invoice* document needs to be submitted to the *Contractor* containing the

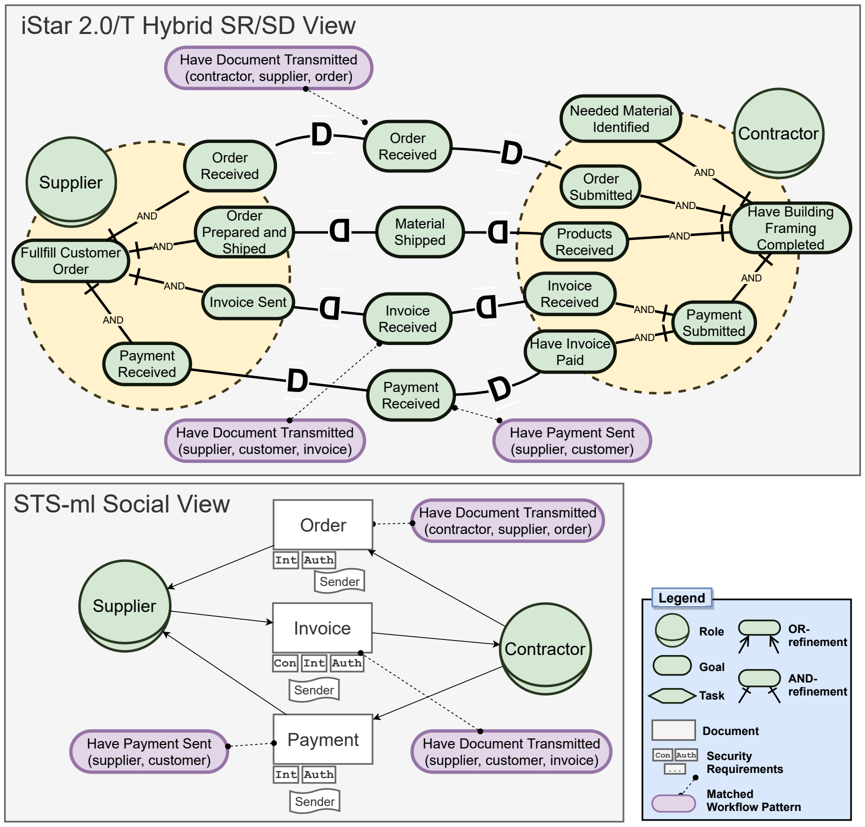

Fig. 2. iStar and STS-ml requirements models.

amount owed and a bank account in which the amount is to be deposited. The sender is to ensure that security requirements are to be met in such transmission. In our case, we require that the document needs to remain confidential (marked with the label `Con`), that it shall not be tampered with (marked with `Int`), and that the recipient shall be able to verify that the document was indeed prepared by the supplier (marked with `Auth`). This kind of analysis is performed for all dependencies identified in the iStar model that entail some form of exchange of an informational asset.

## 2.4 Decomposition Patterns

The models resulting from the above process describe the business context and rationale of security-relevant actor activity and the specific security requirements associated with the latter. However, although they describe what security is needed they do not say *how* it can be attained. For example, the models describe a need to transfer an invoice document from the supplier to the contractor

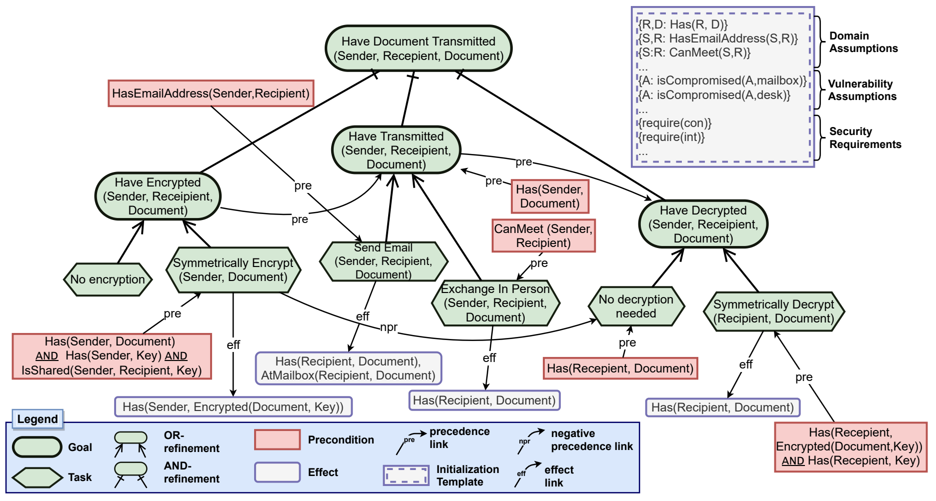
Have Document Transmitted (Sender, Recepient, Document)
HasEmailAddress(Sender,Recipient)
Have Transmitted (Sender, Receipient, Document)
Have Encrypted (Sender, Receipient, Document)
Have Decrypted (Sender, Receipient, Document)
Has(Sender, Document)
CanMeet (Sender, Recipient)
No encryption
Symmetrically Encrypt (Sender, Document)
Send Email (Sender, Recipient, Document)
Exchange In Person (Sender, Recipient, Document)
No decryption needed
Symmetrically Decrypt (Recipient, Document)
Has(Sender, Document) AND Has(Sender, Key) AND IsShared(Sender, Recipient, Key)
Has(Recipient, Document), AtMailbox(Recipient, Document)
Has(Recipient, Document)
Has(Recepient, Document)
Has(Sender, Encrypted(Document, Key))
Has(Recipient, Document)
Has(Recepient, Encrypted(Document,Key)) AND Has(Recepient, Key)
pre
eff
npr
{R,D: Has(R, D)}
{S,R: HasEmailAddress(S,R)}
{S:R: CanMeet(S,R)}
...
{A: isCompromised(A,mailbox)}
{A: isCompromised(A,desk)}
...
{require(con)}
{require(int)}
...
Domain Assumptions
Vulnerability Assumptions
Security Requirements
Legend
Goal
Task
OR-refinement
AND-refinement
Precondition
Effect
precedence link
Initialization Template
negative precedence link
effect link

**Fig. 3.** The Transmit Information decomposition pattern (simplified).

with specific security requirements (`Con`, `Int`, etc.), but no concrete steps are offered to fulfill those requirements. For example, should the supplier just email the invoice as-is? Or should they encrypt it first? Maybe also sign it? If yes, with what tools and configurations?

To capture authoritative ways by which such questions can be answered, a number of *security decomposition patterns* embodying best practices for fulfilling common business tasks are maintained and available for adoption and use. Each pattern consists of (a) a goal decomposition model in iStar T with a mandatory initialization template, together called the *generic decomposition*, and (b) a set of *attack trees* and *satisfaction conditions* to be discussed below. The pattern is identified by the root of the generic decomposition. Figure 3 shows a (partial for brevity) generic decomposition for pattern *Have Document Transmitted(Sender, Recipient, Document)*. The model encodes various ways by which the goal can be executed in secure or less secure ways. Further, its initialization template seen at the top right of Figure 3, offers a vocabulary of predicates to be used to define instance-level: (a) *domain assumptions*, (b) *vulnerability assumptions* and (c) pertinent *security requirements*.

Domain assumptions are practical aspects of the domain at hand that may affect the applicability of security controls and/or attacks against them. Examples of instances of such assumptions include whether the involved parties have each other's handles (email address, phone number, etc.), e.g., *HasEmailAddress(contractor, supplier)*, or statements about the format and size of the information to be exchanged, e.g., *DigitalFile(invoice)*. Vulnerability assumptions, on the other hand, represent a state of affairs in which possible attacks have been successfully carried out. For instance, we may assume that the email account of either party has been compromised, that their PCs or phones run a rootkit, or that their physical space is compromised, using predicates such as *IsCompromised(supplier, emailAccount)*, *IsHacked(contractor, cellphone)*, or *IsCompromised(supplier,desk)*. Asserting a vulnerability assumption indicates that an attacker is motivated enough to bear the cost of the corresponding attack, given the business context and anticipated benefit. The set of concrete domain and vulnerability assumptions that result from instantiating the corresponding initialization templates constitute the *threat model*.

Finally, the initialization template of the generic decomposition contains predicates of the form *requires([SR])*, where *[SR]* is a security requirement of interest, represented through the appropriate individual (e.g., *con* or *int*).

The generic decomposition of Figure 3 describes various ways by which a *Document* can be sent from a *Sender* to a *Recipient*. It includes the option to send in plaintext with no encryption or to symmetrically encrypt, send, and subsequently decrypt. It also shows two ways by which the message (encrypted or otherwise) can be transmitted. A more complete model includes subgoals for digitally signing, performing asymmetric encryption, exchanging keys, and additional information exchange methods (e.g., by specific messenger, SMS, snail mail etc.). An example – in formalized form, more below – can be found in our supporting page [30]. The model, hence, implies a great number of alternative goal-fulfilling task trajectories $tH$, each containing all, some, or none of the security enabling tasks in the model. Which one of the implied trajectories is recommended for the problem at hand will depend on the specific threat model and security requirements, i.e., how the initialization template is instantiated, as well as a model of possible attacks. We discuss the latter next.

## 2.5 Attack Trees and Satisfaction Conditions

While decomposition patterns capture a family of workflows by which goals of domain actors can be fulfilled, associated *attack trees* [43] present a family of ways by which an attacker can compromise such workflows from a point of view of a specific security requirement. We model attack trees using iStar T as seen in Figure 4. The model contains a quality goal whose breach is the root goal of the attack tree. Each decomposition pattern is associated with a number of attack trees, each showing how a specific security requirement (e.g., confidentiality, integrity, or authentication) can be breached.

The attack tree contains several steps (and alternatives thereof) an attacker needs to take in order to achieve that root goal. In the case of document transmission, the attacker can compromise the recipient's mailbox (e.g., crack their email password) in order to retrieve the document itself or the key to its decryption, if it has been encrypted. Alternatively, they can gain physical access to the computer of the recipient and install tracking devices (e.g., keyloggers or cameras). Importantly, for some attacks to be successful, the victim must have performed enabling actions. For example, for the encryption key to be retrieved from the recipient's mailbox following successful cracking of their password, the recipient must have received such keys in plaintext by email. These are repre-

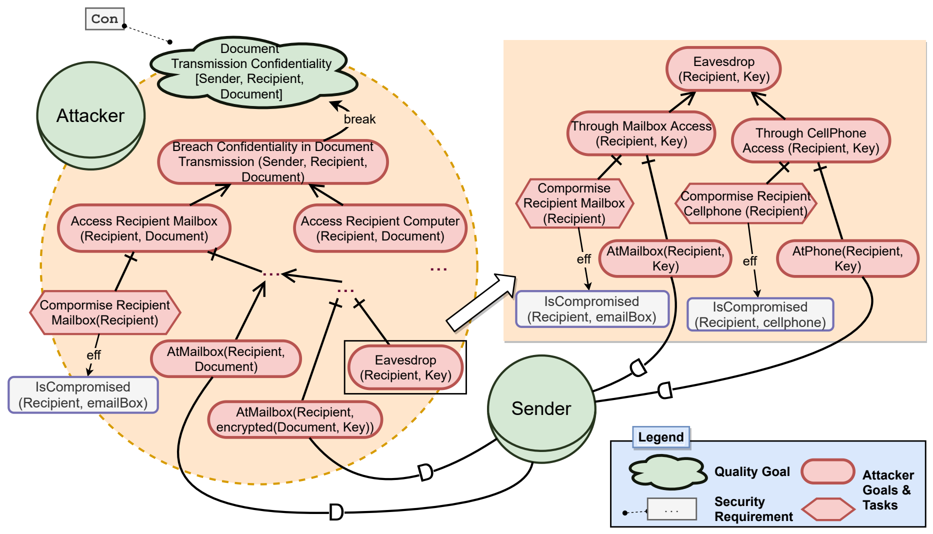
Con
Document Transmission Confidentiality [Sender, Recipient, Document]
Attacker
break
Breach Confidentiality in Document Transmission (Sender, Recipient, Document)
Access Recipient Mailbox (Recipient, Document)
Access Recipient Computer (Recipient, Document)
Compromise Recipient Mailbox(Recipient)
eff
IsCompromised (Recipient, emailBox)
AtMailbox(Recipient, Document)
AtMailbox(Recipient, encrypted(Document, Key))
Eavesdrop (Recipient, Key)
Sender
Eavesdrop (Recipient, Key)
Through Mailbox Access (Recipient, Key)
Through CellPhone Access (Recipient, Key)
Compromise Recipient Mailbox (Recipient)
Compromise Recipient Cellphone (Recipient)
AtMailbox(Recipient, Key)
AtPhone(Recipient, Key)
IsCompromised (Recipient, emailBox)
IsCompromised (Recipient, cellphone)
Legend
Quality Goal
Security Requirement
Attacker Goals & Tasks

**Fig. 4.** An attack tree. Ellipses ("...") signify decomposition parts omitted for brevity.

sented as goals at the leaf level of the attack tree, containing domain predicates that appear in the associated generic decomposition (Figure 2).

From the attacker's standpoint, each attack tree reduces to an AND/OR formula that reflects the decomposition structure and is grounded on domain literals that are either contained in effects of attacker's actions (e.g., *IsCompromised(Recepient, emailBox)*) or are included as the content of the leaf level goals delegated to other actors (e.g., *AtMailbox(Recepient, Document)*). Depending on the truth values of the domain literals, the formula is true or false, signifying that the attack can be successful or not, respectively. The goal, hence, is to identify task trajectories $tH$ in the decomposition pattern such that, given an initialization instantiation of the generic decomposition, the root(s) of one or more attack trees relating to desired security qualities is/are not satisfied.

In addition to attack trees, patterns may contain *satisfaction conditions* that describe necessary conditions that must be true for a specific security requirement to be satisfied. These are also AND/OR decompositions whose leaf level is exclusively goals containing domain predicates from the generic decomposition. For example, the satisfaction condition for *authentication* may consist of the single goal containing predicate $HasAuthenticated(Recepient, Sender, Document)$.

## 2.6 Pattern Selection and Instantiation

With the domain specific models developed and the set of security decomposition patterns available, analysts identify specific patterns from that set that match goals that they have identified in their analysis of the specific problem (Step 3 in Figure 1). In Figure 2, for example, they identify two instances of the pattern of Figure 3, namely the transmission of the order and the transmission of the invoice. The matching is represented using a dashed line connecting the former with the root of the latter.

Instantiation of the pattern to fit the domain specific model takes place in two steps. The first is binding the parameters of the root goal to domain-specific objects. In the invoice transmission example, *Sender* is bound to *supplier*, *Recipient* to *contractor*, and *Document* to *invoice*. The second step involves appropriately instantiating the initialization template of the generic decomposition to encode the desired threat model and specific security requirements. In our example, the initialization template can be instantiated with the threat model *isCompromised(contract,mailbox)* (vuln. assumption) and *CanMeet(contractor,supplier)* (dom. assumption), and concrete security requirements *require(con)*, *require(int)*, and *require(auth)* as per the STS-ml model of Figure 2.

The result of the above process is that domain-specific goals are now linked with generic decompositions, appropriately instantiated to information specific with the problem at hand. This linkage can be seen as supplying authoritative decomposition trees to security-relevant goals of the domain-specific model, instead of developing these decompositions from scratch for every problem instance. However, the adopted authoritative decompositions continue to be generic in a sense that they support all possible ways, secure or otherwise, to fulfill the security-relevant goals. The final step is to identify solutions *tH* of each decomposition that satisfy the threat model and the security requirements, the latter via denying and enabling, respectively, the corresponding attack trees and satisfaction conditions. To do so, we formalize the instantiated patterns (Step 4 of Figure 1) and utilize an automated reasoner to find such solutions (Step 5).

## 3 Generating Secure Workflows

To identify security requirements-satisfying solutions of the instantiated patterns we translate the latter into an AI planning specification, according to translation rules we sketch in this section. We start by presenting the constituents of an AI planning specification and proceed to present the rules.

### 3.1 HTN Planning

AI planners are automated reasoners which receive a *domain specification* and a *problem specification* as input and generate a solution to the planning problem as output. We will here focus on Hierarchical Task Network (HTN) Planners, specifically SHOP2 [37] and its specification language.

A domain specification consists of: a set of domain predicates, a set of *tasks*, a set of *operators*, a set of *methods*, and a set of *axioms* [37]. Domain predicates are first-order predicate symbols used for representing state features, as, e.g., in literal `Encrypted(Document,Key)`. Tasks represent activity to be performed and consist of a task symbol and its parameters, as in `Encrypt(Document,Key)`. Tasks can be *primitive*, in which case they can be accomplished through an operator, or they can be *compound*, in which case they are broken down into smaller tasks according to one or more alternative method specifications. Operators describe how primitive tasks can be performed and consist of a head $\texttt{t}$, which is the primitive task that the operator specifies and its parameters, a *precondition* $\texttt{t}^{pre}$ containing a formula specifying what needs to be true in order for the operator to be applicable, a *delete list* $\texttt{t}^{del}$, and an *add list* $\texttt{t}^{add}$, listing predicates that turn, respectively, false or true, upon application of the operator. Operators can also carry a *cost* value $\texttt{t}^{c}$. In a well-formed operator, the parameters of the predicates of $\texttt{t}^{del}$ and $\texttt{t}^{add}$ are the subset of the union of parameters of $\texttt{t}$ and $\texttt{t}^{pre}$. Methods consist of a compound task $\texttt{m}$ that the method decomposes, a precondition $\texttt{m}^{pre}$ which needs to be true before the method can be considered, and a list of tasks $\texttt{m}^{t} = \{\texttt{m}_1, \texttt{m}_2, \ldots, \texttt{t}_1, \texttt{t}_2\}$ (marked *ordered* or *unordered*), to which the method decomposes the head task. Finally, axioms are clauses of the form $\texttt{h} \leftarrow \texttt{l}$, signifying that predicate $\texttt{h}$ is true if formula $\texttt{l}$ is true.

The planning problem specification includes an initial state, i.e., a list of literals initially true, and a set of tasks to be accomplished. Given a domain and a problem specification, an HTN planner recursively attempts different method applications, aimed at finding sequences of primitive tasks $\texttt{t}_1, \texttt{t}_2, \ldots, \texttt{t}_n$ (a *plan*) that can be performed using operators, such that $\texttt{t}_1$ is possible given the initial state and $\texttt{t}_{i+1}$ is possible given the state after $t_i$. In searching for such plans, the planner will also bias towards plans that minimize cost $\sum_{i=1}^{n} \texttt{t}_i^c$.

## 3.2 Transforming to Planning Specifications

We now sketch how the instantiated decomposition patterns, which are iStar T models, translate into HTN domain and problem specifications to allow reasoning. The domain specification includes translations of: (a) the generic decomposition model of the matched pattern (Figure 3), (b) attack trees related to the matched pattern (e.g., Figure 4), and (c) security requirements as initialization instances of the generic decomposition. The following are the key components of the translation of the generic decomposition model to the domain specification.

**Primitives.** Primitive elements of the iStar T models are translated as follows: (a) every iStar T domain predicate into an HTN domain predicate, (b) every iStar T goal $g$ into (b-i) a compound task $\texttt{m}_g$, and (b-ii) a predicate $\texttt{sat}_g$ (c) every iStar T task $a$ into (c-i) an HTN primitive task $\texttt{t}_a$, (c-ii) an operator with the same head, (c-iii) an HTN predicate $\texttt{sat}_a$, (d) every iStar T precondition into an HTN formula representing the corresponding precondition formula, and (e) every iStar T effect into a list of HTN literals corresponding to the iStar T literals contained in the effect.

**AND/OR decompositions.** For each AND-decomposition of a goal $g$ into subgoals $g_1, g_2, \ldots$, and subtasks $a_1, a_2, \ldots$, introduce a method whose head $\texttt{m}_g$ is the compound task corresponding to $g$, and its task list $\texttt{m}_g^t$ contains an unordered set $\{\texttt{m}_1, \texttt{m}_2, \ldots, \texttt{t}_1, \texttt{t}_2\}$ of compound and primitive tasks corresponding to the subgoals and the subtasks, respectively. For each OR-decomposition of a goal $g$, introduce a method for *each* subgoal or subtask to which $g$ is decomposed. Each method's head is $\texttt{m}_g$ is the compound task corresponding to goal $g$ and the task list contains exactly one element: the compound task $\texttt{m}$ or primitive task $\mathtt{t}$ corresponding to the subgoal or subtask, respectively. In addition, create an axiom of the form $\mathtt{sat}_g \leftarrow \mathtt{sat}_{g_1}$ $\mathtt{[op]}$ $\mathtt{sat}_{g_2}$ $\mathtt{[op]}$ $\ldots$ $\mathtt{[op]}$ $\mathtt{sat}_{a_1}$ $\mathtt{[op]}$ $\mathtt{sat}_{a_2}$ $\mathtt{[op]}\ldots$ where $\mathtt{[op]}$ is $\wedge$ or $\vee$ for AND and OR decompositions, respectively.

**Effect Links.** For each effect link, identify the operator whose head $\mathtt{t}_a$ is the primitive task corresponding to the iStar T task $a$ from where the effect link originates. Set $\mathtt{t}_a^{del}$ to be the list of predicates (translated to their HTN counterparts) that are negated in the effect (if any), and $\mathtt{t}_a^{add}$ to the remaining predicates (if any). Further, add predicate $\mathtt{sat}_a$ to the $\mathtt{t}_a^{add}$.

**Precedence Links.** Firstly, for every $\xrightarrow{pre}$ and $\xrightarrow{npr}$ link that targets a goal, replace the link with links of the same origin that target each leaf level task that is a descendant of the goal. As a result, only tasks are now recipients of such links. Subsequently, for each task $a$, first, collect predicates $\mathtt{sat}_{g_i}$, $\mathtt{sat}_{a_i}$, and formulae $\phi_{p_i}$ corresponding to goals $g_i$, tasks $a_i$ and preconditions $p_i$ that are origins of a $\xrightarrow{pre}$ link targeting the task. Create formula $\phi^{pre} = \mathtt{sat}_{g_1} \wedge \mathtt{sat}_{g_2} \ldots \wedge \mathtt{sat}_{a_1} \wedge \mathtt{sat}_{a_2} \ldots \wedge \phi_{p_1} \wedge \phi_{p_2} \ldots$. Next, for the same task $a$, collect the corresponding elements that are origins of a $\xrightarrow{npr}$ link that targets the task, and create formula $\phi^{npr} = \mathtt{sat}_{g_1} \vee \ldots \vee \mathtt{sat}_{a_1} \ldots \vee \phi_{p_1} \ldots$. Set the precondition $\mathtt{t}_a^{pre}$ of the operator corresponding to task $a$ to $\phi^{pre} \wedge \neg\phi^{npr}$.

**Costs.** As a final step, the costs $\mathtt{t}^c$ of primitive tasks are set in a way that reflects the delay, cognitive load, or actor dissatisfaction that performing the task entails. Tasks such as *No encryption* have a cost of zero, and tasks such as *Generate Keys* (not seen in the diagrams) have a cost greater than zero.

The above rules concern the translation of the generic decomposition trees of a pattern. A different approach is followed for attack trees.

**Attack Trees.** Recall that an attack tree is an AND/OR formula grounded on iStar T domain literals, originating from effects or goals that reside at the leaf of the tree. Construct first the corresponding HTN formula $\phi^{atk}$, in which the list of domain literals in the effects translate to a conjunction of the corresponding HTN literals, and leaf level goals $g$ translate to the corresponding HTN predicates $\mathtt{sat}_g$. Then, for each attack tree, create an axiom of the form $\mathtt{[SR]Breached} \leftarrow \phi^{atk}_{[SR]}$, where $\mathtt{[SR]}$ is the security requirement the tree attempts to breach. The tree of Figure 4 for example is translated into an *attack axiom* $\mathtt{ConBreached} \leftarrow \phi^{atk}_{Con}$, where $\phi^{atk}_{Con}$ corresponds to the formula representing the specific tree. If satisfaction conditions are defined for a security requirement $\mathtt{[SR]}$, a *success axiom* of the form $\mathtt{[SR]Successful} \leftarrow \phi^{succ}_{[SR]}$ is also introduced, where formula $\phi^{succ}_{[SR]}$ corresponds to $\mathtt{[SR]}$'s satisfaction condition.

The above constitutes a translation of the decomposition pattern that is reusable in any analysis instance. The security requirements and the threat model need to be generated separately for every problem instance, as follows.

**Security Requirements.** Introduce a dummy primitive task $\mathtt{t}_d$ and a dummy compound task $\mathtt{m}_d$. The former will play the role of a necessary final task and the latter will be the new root task. To accomplish that, introduce a method with head $\mathtt{m}_d$ and ordered task list $\mathtt{m}_d^t = \{\mathtt{m}_{root}, \mathtt{t}_d\}$, where $\mathtt{m}_{root}$ is the task that corresponds to the root goal of the decomposition pattern. Notice that task $\mathtt{m}_d^t$ is not solvable unless $\mathtt{t}_d$ is feasible after $\mathtt{m}_{root}$ is attained. For each security requirement

$SR$ mentioned in the pattern instantiation (through *require(SR)*), introduce a conjunct to the $\mathtt{t}_d^{pre}$ that is itself a conjunction of the head of the success axiom with the negation of the head of the attack axiom. For example, if `Auth` is required, $\neg$`AuthBreached`$\wedge$`AuthSuccessful` is added as a conjunct in $\mathtt{t}_d^{pre}$. The intuition is that the solver will ignore plans which produce effects that satisfy the attack model or deny the satisfaction conditions, as the precondition of the last mandatory action $\mathtt{t}_d$ would not be satisfied in that case.

**Threat Model.** The threat model informs the construction of the HTN problem specification. Specifically, recall that a pattern instantiation includes an instantiation of the initalization associated with the pattern's generic decomposition with literals grounded on objects from the specific domain. Of these, collect the domain and vulnerability assumptions and form formulae $\phi_{dom}$ and $\phi_{vln}$, respectively, which are conjunctions of corresponding predicates. Set the initial state of the problem specification to be $\phi_{init} = \phi_{dom} \wedge \phi_{vuln}$.

## 3.3 Identifying Secure Workflows

With the instantiated patterns translated we can now use the planner to identify secure workflows. The process involves selecting a domain-specific goal of interest, and use the HTN specification of the associated decomposition instance to generate plans. Each generated plan corresponds to a trajectory $tH$ in the corresponding iStar T decomposition.

Let us return to our running example and assume we are interested in a workflow for goal *Invoice Received*. The goal is associated with an instantiated decomposition pattern *Have Document Transmitted(supplier, customer, invoice)*. We can calculate different workflows for accomplishing invoice transmission for different threat models and security requirements, effectively cycling through Steps 2-6 (noting that Step 3 is performed once) in an exploratory mood. Let us consider a scenario in which no security requirements are given. This means that, in the translation, the dummy final task $\mathtt{t}_d$ has no preconditions. In such case, the planner will not be constrained in any way to produce the cheapest possible plan, which involves a simple plaintext transmission of the invoice:

```
{0: email(supplier, contractor, invoice)}
```

Assume now confidentiality requirements for invoice transmission, as per the STS-ml model of Figure 2. Then, plans will depend on the threat model. If no vulnerability assumptions are made, meaning that attackers are not assumed to be capable/motivated to perform any of the tasks in their attack tree, emailing plaintext is again the best plan: none of the workflows satisfies the attack tree.

Assume subsequently that `IsCompromised (mailbox, contractor)` is true, i.e., we assume that, in the worst scenario, the attackers can actually gain access to the mailbox of the contractor. This can make the attack tree formula satisfied if the domain actors perform certain tasks. Indeed the emailing of plaintext no longer returns as part of a recommended plan. The planner will propose sending the plaintext invoice via SMS attachment. At this point the analysts may decide that either SMS is an inappropriate means of exchange of the document itself or that that cellphones are also assumed to be compromised. These are expressed as an additional domain or vulnerability assumption, respectively. Under the new assumption, the planner will now output plans that involve encryption, e.g.:

```
{1: genSharedKey(supplier,contractor)
 2: exchangeViaPhoneCall(supplier,contractor,key),
 3: symmetricEncrypt(supplier,invoice,key),
 4: email(supplier,contractor,invoice),
 5: symmetricDecrypt(contractor,invoice,key)}
```

Thus, the plan involves symmetric encryption of the invoice preceded by key generation and exchange through a channel not assumed to be compromised according to the vulnerability assumptions, in our case via a voice call.

Assume, finally, that confidentiality is not an issue but invoice authenticity is, i.e., the contractor must verify that the invoice was prepared by the supplier (`Auth`) and has not been tampered with along the way (`Int`). Thus initialization will instantiate with *require(auth)* and *require(int)* which will be translated into the appropriate preconditions to $\mathtt{t}_d$. For the same vulnerability assumptions (compromised email accounts), the solution includes asymmetric public key exchange (`pKey`) and digitally signing the document, but excludes encryption:

```
{1: installEncSoftware(supplier),
 2: installEncSoftware(contractor),
 3: genAsymKeys(supplier),
 4: email(supplier,contractor,pKey),
 5: sign(supplier,invoice,sKey),
 6: email(supplier,contractor,invoice),
 7: verify(contractor,supplier,invoice)}
```

Note that the above plans have been simplified for the interest of space; readers can find full examples and commentary in our accompanying page [30]. Analysts can repeat this exploratory process for all security-relevant information exchange tasks of the initial domain-specific model. With relatively minor adaptations to the translation rules, omitted here for simplicity, global analysis can be performed on the basis of a higher level domain-specific goal, such as *Have Building Framing Completed*, following fusion of pattern instances into a now expanded domain-specific model. The generated workflows then encompass activities that aim to satisfy all relevant goals including order exchange, receipt of invoice, and submission of payment.

## 4 Related Work

Several approaches to model security requirements have been proposed. The SI* security language and the Secure Tropos methodology, for example, allow modeling and reasoning with permission delegations in the presence or absence of trust among agents [18,33,36]. Reasoners such as ASP [33], or, closer to our approach, planners [6] have been used for reasoning about such models. These early proposals appear to have inspired the Socio-Technical Security (STS) approach and modeling language we utilize here [9,39]. However, these approaches largely stay at the requirements level, naturally leaving implementation details, such as use of cryptographic primitives, outside of their scope. Work that does address utilization of cryptographic primitives has been introduced within the UMLSec context [24,25,42] where architectural security patterns are used for informing the design problem, supported by theorem proving techniques.

Many threat modeling frameworks have also been proposed – [44,49] for surveys. Our approach is based on attack trees [43,45] implemented here as goal decomposition models; an idea introduced and explored within the KAOS [10] goal modeling framework [46]. Among the differences of our work to the KAOS-based approach is the focus on run-time agent decisions (vs. design-time analysis) and on the generation of concrete secure sequences of actions.

Pattern systems have also been introduced [13,17,21,38,48] including for modeling attack types and using them to test an STS specification [29]. Rather than patterns of attack, we here envision design patterns (e.g., as in [47]) that specifically describe workflows for correctly utilizing cryptographic and other security primitives. Li et al. do adopt such a pattern system [27,28] based on contextual goal models and a matching, adoption, and instantiation approach similar to the one we propose. Among several differences, however, our proposal exploits iStar T's ability to model the temporal and causal aspect, allowing the generation of concrete solution workflows. Our approach is, further, distinct in scope from the broad literature on formalizing and verifying security protocols, (e.g., [3,4,7,22,31]) – [34] and [40] for surveys. This also applies vis-à-vis efforts for using AI planners specifically [5,20], which, in addition, focus on generating attack plans rather than security designs. Rather than verifying the nuts-and-bolts of security protocols against elaborate attacks [12], our goal is to abstract away the details of both protocols and attacks and recommend high-level designs that take into account practical aspects of the underlying socio-technical processes while remaining aligned with the specific security requirements of stakeholders.

## 5 Concluding Remarks

We presented a model-driven technique for automated generation of secure workflow designs based on identified security requirements. The technique is based on the availability of patterns, in the form of high-variability goal decomposition models, that encode authoritative best practices for performing common business tasks. The patterns also include attack trees that describe alternative attacks against these authoritative models with respect to different security properties, as well as templates for modeling security goals and threat assumptions of specific problems. When solving a specific problem, analysts adopt and instantiate the patterns within their domain-specific models, and through automated reasoning identify workflows whereby domain actors can fulfill business tasks in compliance with security requirements. The benefits of the approach include: (i) assisting analysts with no deep security expertise to produce high-level designs that make coherent use of security primitives, (ii) ensuring that the necessary and only the necessary security-related steps are included in the resulting designs, and (iii) offering a mechanism for exploring solutions under alternative security requirements and threat assumptions, informing thereby the risk management process at an early requirements stage. Our contribution is centered in the modeling techniques introduced to accomplish the above: the use of high-variability goal models to concisely represent multiple workflows involving different levels of security-related actions (including none), the use of attack models to test the appropriateness of candidate workflows, and the automated generation of compliant workflows using an off-the-shelf reasoning tool.

The patterns we have developed so far employ basic cryptographic primitives for information exchange using common channels, such as email, text messaging, or in-person exchange [30]. A set of proof-of-concept attack trees that we have built encode examples of compromising these exchange media as well as the physical contexts in which they are utilized. For the framework to be directly applicable in practice vis-à-vis information exchange, more comprehensive attack trees and digital communication process models will be needed. Security pattern development is generally known to be non-trivial [41]. Development of a useful and reliable pattern base for our framework would require consultation with experts and the literature, followed by extensive testing of the reasoning outputs.

While our current application scope, secure information exchange, by no means exhausts the range of advanced threats and sophisticated attacks against organizations and citizens, seemingly solvable problems within that scope such as email compromise due to user oversight continue to be a major societal problem [26]. Our modeling and reasoning architecture is aimed at being applicable for different business activities, such as storing and archiving data, accessing system functions and resources, handling payments (see Figure 2), or even handling physical security and privacy in, e.g., transporting goods and people, or storing materials. These can, moreover, be analyzed with respect to a variety of security or privacy concerns via development of the corresponding attack trees. In general, the approach is particularly applicable when complex sequences of steps are required by both the domain actors to perform a business task and the attackers for compromising said task with respect to a specific security requirement.

Moreover, substantial empirical work will be needed to evaluate the usability and utility of the proposed modeling process, and to identify ways to optimize it. For example, our so far experimentation with building patterns has revealed that identifying the right level of model detail while keeping automated reasoning tractable can be challenging. Examples are deciding the level of detail to include when modeling primitives, and choosing whether to explicate exemplar workflows or leave the reasoner to construct such. Further, case studies in different domains will help us to better understand the scope of applicability of our technique.

Finally, tool support is essential for performing meaningful user and case studies. Thanks to the translation rules (Section 3.2), the core of the formalization process is automatable. Nevertheless, an interactive visual tool in support of the end-to-end modeling and analysis process requires substantial interaction design and evaluation. Alternatively, a textual representations (e.g., [1]) could be explored for performance of said tasks through scripting.

## References

1. Abdelzad, V., Amyot, D., Alwidian, S., Lethbridge, T.: A Textual Syntax with Tool Support for the Goal-Oriented Requirement Language. In: Proceedings of The 8th International i* Workshop (2015), https://ceur-ws.org/Vol-1402/paper6.pdf
2. Adobe: Choosing a security method for PDFs (2024), https://helpx.adobe.com/ca/acrobat/using/choosing-security-method-pdfs.html, accessed: 2024-08-19
3. Armando, A., Carbone, R., Compagna, L.: LTL model checking for security protocols. Journal of Applied Non-Classical Logics **19**(4), 403–429 (2009). https://doi.org/10.3166/jancl.19.403-429
4. Blanchet, B.: A Computationally Sound Mechanized Prover for Security Protocols. IEEE Transactions on Dependable and Secure Computing **5**(4), 193–207 (2008). https://doi.org/10.1109/TDSC.2007.1005
5. Boddy, M.S., Gohde, J., Haigh, T., Harp, S.A.: Course of action generation for cyber security using classical planning. In: In Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 2005). pp. 12–21. AAAI, Monterey, California, USA (2005), https://dl.acm.org/doi/10.5555/3037062.3037065
6. Bryl, V., Massacci, F., Mylopoulos, J., Zannone, N.: Designing security requirements models through planning. In: Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06). Luxembourg, Luxembourg (2006). https://doi.org/10.1007/11767138_4
7. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Transactions on Computer Systems **8**(1), 18–36 (feb 1990). https://doi.org/10.1145/77648.77649, https://doi.org/10.1145/77648.77649
8. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. The Computing Research Repository (CoRR) (2016), http://arxiv.org/abs/1605.07767
9. Dalpiaz, F., Paja, E., Giorgini, P.: Security Requirements Engineering: Designing Secure Socio-Technical Systems. MIT Press (2016)
10. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming **20**(1-2), 3–50 (1993), https://doi.org/10.1016/0167-6423(93)90021-G
11. Devanbu, P.T., Stubblebine, S.: Software engineering for security: a roadmap. In: Proceedings of the 22nd International Conference on on Software Engineering, Future of Software Engineering Track (ICSE'00). pp. 227–239. Association for Computing Machinery, New York, NY, USA (2000), https://doi.org/10.1145/336512.336559
12. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650
13. Dougherty, C.R., Sayre, K., Seacord, R., Svoboda, D., Togashi, K.: Secure Design Patterns. Tech. rep., Carnegie Mellon University (2018). https://doi.org/10.1184/R1/6583640.v1
14. Dubois, E., Mouratidis, H.: Guest editorial: security requirements engineering: past, present and future. Requirements Engineering **15**(1), 1–5 (2010), https://doi.org/10.1007/s00766-009-0094-8
15. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An empirical study of cryptographic misuse in Android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 73–84. CCS '13, Association for Computing Machinery, New York, NY, USA (2013), https://doi.org/10.1145/2508859.2516693

16. Fabian, B., Gürses, S., Heisel, M., Santen, T., Schmidt, H.: A comparison of security requirements engineering methods. Requirements Engineering **15**(1), 7–40 (2010), https://doi.org/10.1007/s00766-009-0092-x
17. Fernandez, E.B., Yoshioka, N., Washizaki, H., Yoder, J.: Abstract security patterns and the design of secure systems. Cybersecurity **5**(1), 7 (2022), https://doi.org/10.1186/s42400-022-00109-w
18. Giorgini, P., Massacci, F., Zannone, N.: Security and trust requirements engineering. International School on Foundations of Security Analysis and Design III (FOSAD 2005, FOSAD 2004) **LNCS 3655**, 237–272 (2005). https://doi.org/10.1007/11554578_8
19. Haley, C., Laney, R., Moffett, J., Nuseibeh, B.: Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Transactions on Software Engineering **34**(1), 133–153 (2008). https://doi.org/10.1109/TSE.2007.70754
20. Harp, S.A., Gohde, J., Haigh, T., Boddy, M.S.: Automated vulnerability analysis using AI planning. In: AI Technologies for Homeland Security, Papers from the 2005 AAAI, Spring Symposium. pp. 46–53. AAAI (2005)
21. Hatebur, D., Heisel, M., Schmidt, H.: A pattern system for security requirements engineering. Proceedings of the Second International Conference on Availability, Reliability and Security (ARES 2007) pp. 356–365 (2007). https://doi.org/10.1109/ARES.2007.12
22. Hommersom, A., Meyer, J.j., De vink, E.: Update Semantics of Security Protocols. Synthese **142**(2), 229–267 (2004), https://doi.org/10.1007/s11229-004-2247-0
23. Howard, M., Lipner, S.: The Security Development Lifecycle. Microsoft Press (2006)
24. Jürjens, J.: Sound methods and effective tools for model-based security engineering with UML. Proceedings of the 27th International Conference on Software Engineering (ICSE'05) pp. 322–331 (2005). https://doi.org/10.1145/1062455.1062519
25. Jürjens, J., Yu, Y.: Tools for model-based security engineering: Models vs. code. In: Proceedings of the 22nd ACM/IEEE International Conference on Automated Software Engineering (ASE'07). pp. 545–546. Atlanta, Georgia (2007). https://doi.org/10.1145/1321631.1321736
26. Langan, T.: Internet Crime Report 2023. Tech. rep., Federal Bureau of Investigation (2023), https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf
27. Li, T., Horkoff, J., Mylopoulos, J.: Integrating Security Patterns with Security Requirements Analysis Using Contextual Goal Models. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) Proceedings of the 7th IFIP Working Conference on The Practice of Enterprise Modeling (PoEM 2014). pp. 208–223. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), https://doi.org/10.1007/978-3-662-45501-2_15
28. Li, T., Horkoff, J., Mylopoulos, J.: Analyzing and Enforcing Security Mechanisms on Requirements Specifications. In: Fricker, S.A., Schneider, K. (eds.) Proceedings of the 21nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'15). pp. 115–131. Springer International Publishing, Cham (2015), https://doi.org/10.1007/978-3-319-16101-3_8
29. Li, T., Paja, E., Mylopoulos, J., Horkoff, J., Beckers, K.: Security attack analysis using attack patterns. In: Proceedings of the 10th IEEE International Conference on Research Challenges in Information Science (RCIS'16). pp. 1–13. Grenoble, France (2016). https://doi.org/10.1109/RCIS.2016.7549303
30. Liaskos, S.: Model-driven implementation of security requirements using patterns - Supplementary Material. https://github.com/cmg-york/sFuse (2024)

31. Lomuscio, A., Woźna, B.: A complete and decidable security-specialised logic and its application to the TESLA protocol. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06). pp. 145–152. Association for Computing Machinery, New York, NY, USA (2006), https://doi.org/10.1145/1160633.1160658
32. Martin, K.M.: Everyday Cryptography: Fundamental Principles and Applications. Oxford University Press (2012), https://doi.org/10.1093/acprof:oso/9780199695591.001.0001
33. Massacci, F., Mylopoulos, J., Zannone, N.: Security requirements engineering: The SI* modeling language and the Secure Tropos methodology. Studies in Computational Intelligence **265**, 147–174 (2010). https://doi.org/10.1007/978-3-642-05183-8_6
34. Meadows, C.: Formal methods for cryptographic protocol analysis: Emerging issues and trends. IEEE Journal on Selected Areas in Communications **21**(1), 44–54 (2003). https://doi.org/10.1109/JSAC.2002.806125
35. Microsoft: How to password protect your Word documents (2024), https://www.microsoft.com/en-us/microsoft-365-life-hacks/privacy-and-safety/how-to-password-protect-word-documents, retrieved: 2024-08-19
36. Mouratidis, H., Giorgini, P.: Secure Tropos: A Security-Oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering **17**(02), 285–309 (2007). https://doi.org/10.1142/S0218194007003240
37. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research (JAIR) **20**, 379–404 (2003), https://dl.acm.org/doi/abs/10.5555/1622452.1622465
38. Nguyen, P.H., Yskout, K., Heyman, T., Klein, J., Scandariato, R., Traon, Y.L.: SoSPa: A system of Security design Patterns for systematically engineering secure systems. In: Proceedings of the 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2015). pp. 246–255 (2015). https://doi.org/10.1109/MODELS.2015.7338255
39. Paja, E., Dalpiaz, F., Giorgini, P.: Modelling and reasoning about security requirements in socio-technical systems. Data and Knowledge Engineering **98**, 123–143 (2015). https://doi.org/10.1016/j.datak.2015.07.007
40. Pucella, R.: Knowledge and Security. In: van Ditmarsch, H., Halpern, J.Y., van der Hoek, W., Kooi, B. (eds.) Handbook of Epistemic Logic, chap. 12, pp. 591–565. College Publications (2015)
41. Riaz, M., Williams, L.: Security requirements patterns: Understanding the science behind the art of pattern writing. Proceedings of the 2nd IEEE International Workshop on Requirements Patterns (RePa 2012) pp. 29–34 (2012). https://doi.org/10.1109/RePa.2012.6359977
42. Schmidt, H., Jürjens, J.: Connecting security requirements analysis and secure design using patterns and UMLsec. In: Proceedings of the 23rd International Conference on Advanced Information Systems Engineering (CAiSE'11). vol. LNCS 6741, pp. 367–382. London, UK (2011). https://doi.org/10.1007/978-3-642-21640-4_28
43. Schneier, B.: Attack Trees: Modelling Security Threats. Dr. Dobb's Journal (December 1999) (1999), https://www.schneier.com/academic/archives/1999/12/attack_trees.html

44. Shevchenko, N., Chick, T.A., O'riordan, P., Scanlon, T.P., Woody, C.: Threat Modeling: A Summary Of Available Methods, Carnegie Mellon University: Software Engineering. Tech. Rep. July (2018), https://insights.sei.cmu.edu/documents/569/2018_019_001_524597.pdf
45. Tøndel, I.A., Jensen, J., Røstad, L.: Combining misuse cases with attack trees and security activity models. In: Proceedings of the 5th International Conference on Availability, Reliability, and Security (ARES 2010). pp. 438–445. Krakow, Poland (2010). https://doi.org/10.1109/ARES.2010.101
46. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proceedings of the 26th International Conference on Software Engineering (ICSE'04). pp. 148–157 (2004). https://doi.org/10.1109/ICSE.2004.1317437
47. Weiss, M., Mouratidis, H.: Selecting security patterns that fulfill security requirements. In: Proceedings of the 16th IEEE International Requirements Engineering Conference (RE'08). pp. 169–172. Barcelona, Spain (2008)
48. Wen, Y., Haihong Zhao, Liu, L.: Analysing security requirements patterns based on problems decomposition and composition. In: Proceedings of the 1st International Workshop on Requirements Patterns, RePa'11. pp. 11–20. IEEE (2011). https://doi.org/10.1109/RePa.2011.6046726
49. Xiong, W., Lagerström, R.: Threat modeling – A systematic literature review. Computers and Security **84**, 53–69 (2019), https://doi.org/10.1016/j.cose.2019.03.010