Model-based Confidentiality Analysis under Uncertainty

Sebastian Hahner
Karlsruhe Institute of Technology
Karlsruhe, Germany
sebastian.hahner@kit.edu

Tizian Bitschi
Karlsruhe Institute of Technology
Karlsruhe, Germany
tizian.bitschi@student.kit.edu

Maximilian Walter
Karlsruhe Institute of Technology
Karlsruhe, Germany
maximilian.walter@kit.edu

Tomáš Bureš
Charles University
Prague, Czech Republic
bures@d3s.mff.cuni.cz

Petr Hnětynka
Charles University
Prague, Czech Republic
hnetynka@d3s.mff.cuni.cz

Robert Heinrich
Karlsruhe Institute of Technology
Karlsruhe, Germany
robert.heinrich@kit.edu

***Abstract*—In our connected world, ensuring the confidentiality of the software systems we build becomes increasingly difficult. Model-based design time confidentiality analyses have been proposed to cope with this complexity early. However, the usefulness of such analyses is limited due to uncertainty about the software architecture itself and the software's execution environment. This leads to conclusions about confidentiality violations that lack both precision and comprehensiveness. Although there exist approaches to deal with design time uncertainty, existing research lacks precise statements about the impact of uncertainty on confidentiality. To address this, we include uncertainty as part of our software architectural model. We extend a data flow-based analysis to include the impact of uncertainty on confidentiality violations. The results of the case study-based evaluation show high accuracy with typical design time uncertainty. Also, our analysis yields more precise statements about the impact of uncertainty on confidentiality than the state of the art.**

***Index Terms*—Model-driven Security, Software Architecture, Uncertainty, Confidentiality, Data Flow Analysis, Access Control**

I. INTRODUCTION

Software systems are becoming increasingly complex, e.g., in Industry 4.0 [6] or automotive systems [1]. Thus, ensuring security-related quality properties like confidentiality becomes a major challenge. Confidentiality demands that "information is not made available or disclosed to unauthorized individuals, entities, or processes" [21]. Violations cannot only have legal consequences [20] but also affect user acceptance [46]. As proposed by *Privacy by Design* [35], confidentiality should be considered early to avoid costly repairs [5]. This has been addressed with design time confidentiality analyses. By analyzing data flows [38] or potential attack paths [44] in modeled software architectures, confidentiality requirements [18] can be evaluated early.

However, in early development and complex systems of systems, uncertainty exists about the software architecture and its environment [1]. Uncertainty describes "any departure from the unachievable ideal of complete determinism" [42]. Examples are uncertain component choices and deployment, or system behavior and user behavior. The high degrees of uncertainty in software architecture while making design decisions are also referred to as the *cone of uncertainty* [30]. This affects both the precision and comprehensiveness of the results of confidentiality analysis [16].

The handling of uncertainty can either be included as part of the analysis in a white-box manner or delegated to an uncertainty-aware framework [1]. Although uncertainty-aware analyses of software architectures exist [11], they usually focus on other quality properties like performance [39] or are limited to selected uncertainty types [7]. There is also an approach to combine confidentiality analysis with uncertainty [45], but due to its black-box nature, this approach lacks precision.

In this paper, we extend an existing model-based confidentiality analysis [38] to make it uncertainty-aware, increasing both its precision and comprehensiveness. We model uncertainty using software architecture variation [43] that can express uncertainties as their impact on the software architecture and its environment. Following the white-box approach, we interpret the modeled uncertainty as well as the analysis results for more detailed statements about the impact of uncertainty on confidentiality. This shall simplify phase containment, i.e., fixing defects in the same phase as they appear.

We start by classifying and modeling uncertainty in Section II and present the contributions of this paper thereafter.

C1 First, we discuss in Section III how information available in the modeling and analysis of software architecture models can be used for increasing uncertainty awareness.

C2 Second, we present an uncertainty-aware confidentiality analysis that considers the impact of uncertainty on confidentiality violations by analyzing the data flow of architectural variants in Section IV.

Our evaluation in Section V is based on three case studies already used in other work [38], [45]. The results indicate a high accuracy while only using information that is already available to software architects at design time. Our interpretation of confidentiality analysis results shows a higher precision regarding the impact of uncertainty than state of the art, which is discussed in Section VI. Section VII concludes this paper.

II. Expressing Uncertainty as Model Variation

In this section, we discuss the modeling of uncertainty for analyzing software architecture. Uncertainty can have many different forms, such as lack of knowledge, imperfect or incorrect information, or natural variability [40]. To cope with this complexity, uncertainty should be considered a first-class concern in software architecture [12]. A variety of modeling approaches have been proposed, including Bayesian networks, fuzzy values, automatons, and petri nets [40].

Design uncertainty is “normally represented in software models by variability models” [40]. These are closely related to architectural design using Architectural Design Decisions (ADDs) [22] and design space exploration [28], where decisions are intentionally left open to form degrees of freedom. The underlying uncertainty—the gap between the model and the real world—is sometimes also referred to as *model structural uncertainty* [31]. In previous work, we presented a classification to describe software-architectural uncertainty more precisely regarding its impact on confidentiality [17].

In the following, we demonstrate how to classify and model uncertainty using a running example. This example represents a simplified online shop under uncertainty and is based on other work [15], [17]. It consists of three components and two possible deployment locations, as shown in Figure 1. Users access the *Online Shop* component to list and buy available products. The *Product Database Service* and the *User Database Service* are responsible for storing product and user data, respectively. Both can either be deployed on an *EU Cloud* server or a *Non-EU Cloud* server.

We assume that the online shop will be used from inside the EU by EU customers. Thus, legal restrictions apply, e.g., as defined by the GDPR [9]. We define two simplified confidentiality requirements for the running example: Personal user data shall only be stored on servers within the EU. Additionally, personal user data must never be stored on a cloud server without encryption. In this example, multiple uncertainties (**U1** – **U3**) exist that can have an impact on these confidentiality requirements. First, the component developers of the *Online Shop* have not yet specified whether user data will be encrypted (**U1**). The allocation of the *Product Database Service* (**U2**) and the *User Database Service* (**U3**) is also not defined. In this simplified example, one can quickly see the potential violation of the confidentiality requirements. While the allocation of the product database (**U2**) is noncritical, the uncertainties **U1** and **U3** can affect the system's confidentiality. With larger systems that are developed by multiple experts or teams, this can quickly become unmanageable without help.

To understand these uncertainties, we first describe them more precisely using a classification [17]. The encryption of user data (**U1**) is located in the *system behavior*, and the allocation (**U2** and **U3**) affects the *system structure*. All uncertainties are directly related to *components* and are at least *partially reducible*. They can be described by a finite set of possibilities, so the type of these uncertainties is *Scenario Uncertainty*. This enables scenario-based mitigation using model variation [17].

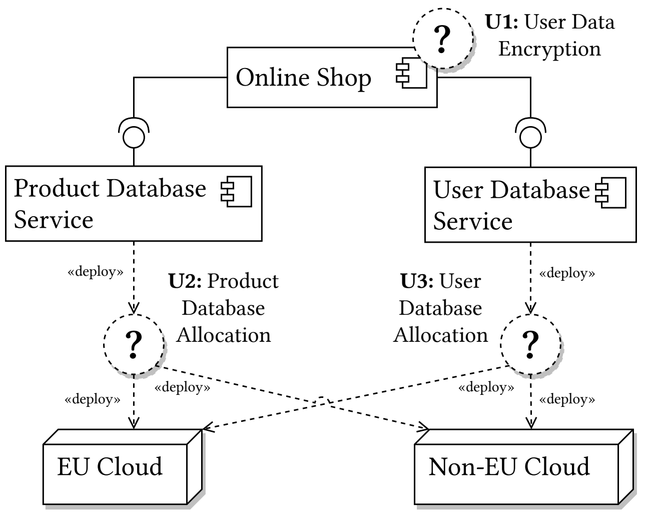

Figure 1. Component and deployment diagram of the Online Shop example

Table I
Modeled uncertainties of the running example

Uncertainties	**Modeled variants**
U1: User Data Encryption	Forward data only, Encrypt data
U2: Product Database Allocation	EU Cloud, Non-EU Cloud
U3: User Database Allocation	EU Cloud, Non-EU Cloud

Model variation can be used to describe possible outcomes of *Scenario Uncertainty*. Each uncertainty is represented by a variation point, e.g., by modeling a selection of alternative architectural elements. This description results in a variation model [43] of the software architecture. Table I shows the three uncertainties (**U1** – **U3**) of the running example and the resulting variants in the variation model. The user data encryption in the *Online Shop* (**U1**) is represented by two different behaviors, i.e., only forwarding or also encrypting the data. The allocation of the *Product Database* (**U2**) and the *User Database* (**U3**) is modeled with the two possible servers inside or outside the EU.

Based on the variation model, architectural variants can be generated by permutation. In our example, three uncertainties exist, with two variants each. The automated generation [43] would yield $2 \times 2 \times 2 = 8$ variants that represent all possible outcomes of the modeled uncertainty. The resulting variants can be analyzed by existing analyzes that do not yet account for uncertainty. This enables software architects to focus on critical variants, e.g., variants that violate confidentiality.

III. Increasing the Uncertainty Awareness

In this section, we discuss which information is available at design time to model and analyze software architectures. This enables the comparison of the uncertainty awareness of different confidentiality analysis approaches. Table II shows information categories regarding confidentiality and uncertainty. Note that this list shall not be comprehensive but rather represent current research in the related domains.

Table II
AVAILABLE INFORMATION CATEGORIES IN MODELING AND ANALYSIS

Domain	Information categories
Confidentiality	**1.** Confidentiality violation occurrence, **2.** Violated confidentiality requirements, **3.** Location within the model, **4.** Analyzed data flow, **5.** Variable state at the violation
Uncertainty	**6.** Uncertainty source, **7.** Uncertainty properties and classification, **8.** Uncertainty impact within the model, **9.** Uncertainty mitigation, **10.** Uncertainty interaction

A. Available Information Categories

The confidentiality information categories are derived from data flow-based analyses of architectural models [37]. These analyses transform modeled software architectures into data flows and search for violations of confidentiality requirements by propagating modeled data variables. The results include knowledge about the existence of confidentiality violations (**1.**) and the related violated requirements (**2.**). Also, such analyses can point to the location within the architectural model (**3.**) and the analyzed data flow (**4.**) where the violation occurred. The state of related data variables (**5.**) at this point is included. In our running example, a violation (**1.**) of the confidentiality requirement concerning user data (**2.**) can be found. It can occur in the data flow (**4.**) to the *User Database Service* component (**3.**) because it contains personal information (**5.**).

The uncertainty information categories are derived from the uncertainty management in self-adaptive systems [19] and cyber-physical systems [1] and are based on the handling of uncertainty regarding confidentiality [17]. By investigating an uncertainty source (**6.**) in the model or its environment [1], it can be described more precisely, e.g., based on a classification (**7.**). Also, the uncertainty's impact (**8.**) on the software architecture can be derived and modeled. By analyzing the architectural model, mitigation techniques (**9.**) can be chosen already at design time [19]. This process can be applied to one uncertainty source or by also considering the interaction (**10.**) of multiple uncertainties [8]. In the running example, a source of uncertainty (**6.**) is the encryption of user data (**U1**) that has been classified (**7.**) in Section II. The uncertainty impact (**8.**) is modeled as a variation of the behavior of the *Online Shop* component. This uncertainty can be mitigated (**9.**), e.g., by enforcing an ADD regarding the component's realization. However, there could be uncertainty interactions (**10.**), e.g., with the allocation of the *User Database Service*.

B. Confidentiality Analysis and Uncertainty Awareness

Based on these categories (**1.** – **10.**), we discuss existing and potentially possible uncertainty-aware confidentiality analyses. Regarding model-based confidentiality analysis, showing violations (**1.**) with the related reason (**2.**) and (**3.**) location to software architects is considered to be the minimum viable information [18]. The analysis results become more expressive by also showing the analyzed data flow (**4.**) and variable state (**5.**). Still, they do not consider uncertainty and thus lack precision and comprehensiveness [15]. This becomes visible in over-estimations and also in missing violations [36].

The *naive* approach to handling uncertainty is to model its impacts (**8.**)—e.g., as model variation, see Section II—and to reject a software architecture if there exist confidentiality violations in at least one variant. While this yields more comprehensive results, it still highly lacks precision [4]. *Scenario-aware* analyses build on this approach and reject only such architecture variants that violate confidentiality. An example is the black-box combination of the software architecture optimizer PerOpteryx [28] with model-based confidentiality analysis [45]. However, this approach still lacks precision because it only considers the existence of violations (**1.**). By also considering the location where the violation occurred (**3.**) and the affected data flow (**4.**), the confidentiality violation can be related to the modeled uncertainty impacts (**8.**). This also enables filtering modeled uncertainty impacts to those that actually affect confidentiality. The *data flow-aware* approach presented in this paper realizes this concept and thus uses the available information more efficiently. In the running example, the *naive* approach would directly reject the model because confidentiality violations occur. The *scenario-aware* analysis yields a number of architecture variants that maintain confidentiality but without further explanation. The *data flow-aware* analysis additionally relates identified violations to their originating uncertainty, e.g., the encryption of user data (**U1**).

This approach can be further enhanced by mapping the uncertainty impacts (**8.**) to their originating sources (**6.**), e.g., an ADD that has multiple impacts on the software architecture [1]. This enables the use of uncertainty classifications (**7.**) and also simplifies mitigation (**9.**). Here, considering the impact within the model is required to understand the consequences while also considering the source helps in understanding the underlying problem of uncertainty. This can help in prioritizing ADDs [17] to resolve issues, recommending mitigation approaches, and enabling self-adaption. Last, uncertainty interaction (**10.**) can be considered in the analysis. Here, we can reuse the information about data flows (**4.**) and the mapping of uncertainty sources and impacts (**6.**). Still, there could be complex relationships between multiple uncertainties that go beyond our current analysis capabilities [8].

IV. UNCERTAINTY-AWARE CONFIDENTIALITY ANALYSIS

This section presents our *data flow-aware* approach for model-based confidentiality analysis under uncertainty. With this approach, identified confidentiality violations are associated with potential influencing uncertainty impacts and are displayed to the software architects. Additionally, uncertainties that are not responsible for a specific violation can be filtered.

Our approach is based on an existing data flow-based confidentiality analysis [38]. The analysis receives an architectural model described in the Architectural Description Language (ADL) Palladio [33] and automatically transforms it into a data flow diagram [10] by extracting possible data flows of the modeled behavior. Figure 2 shows the data flow diagram that corresponds to the running example presented in Section II.

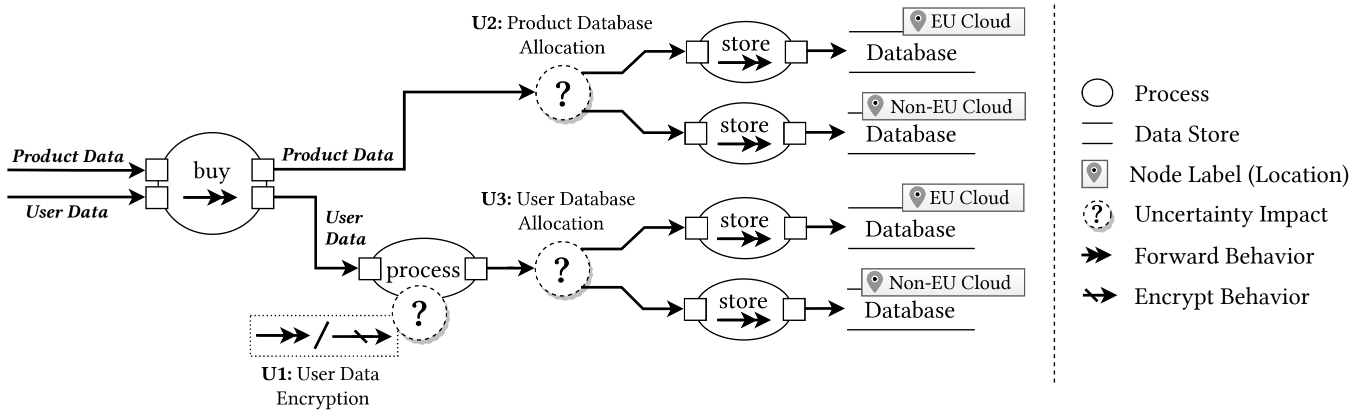

Figure 2. Data flow diagram of the running example with multiple uncertainty impacts that are annotated as question marks

Data flow diagrams consist of processes (e.g., *buy* or *store*), data stores (e.g., *Database*), and data flows (e.g., the arrows annotated with user data). We use the data flow meta model by Seifermann et al. [37]. Input and output pins represent incoming, and outgoing data flows of one specific type (e.g., the pins of the *store* process). They are used to decouple processes from data flows. Last, labels can be assigned to processes and data stores to represent characteristics that can affect the system's confidentiality. All the required information is already available in the architectural model and can automatically be extracted. In the running example, one important node characteristic type is the location (e.g., *EU Cloud*). Processes can alter the characteristics of incoming data, which represents their behavior (e.g., *Encrypt Behavior*).

The extended confidentiality analysis [38] is based on *label propagation*. For each transformed data flow, labels are propagated and updated along the sequence of processes and data stores. In each node, the assignment of labels is compared against data flow constraints that represent confidentiality requirements [18]. A confidentiality violation occurs if at least one data flow constraint cannot be satisfied. In the running example, a violation occurs if user data that is labeled as personal information flows to a *Database* that is labeled as being allocated in the *Non-EU Cloud*. This constraint originates from one GDPR confidentiality requirement presented in Section II.

When users buy products using the *Online Shop* component, product data and user data flow through the system. The data is stored by the *store* process of the *Database Services*. The diagram also demonstrates the uncertainty impacts (**U1** – **U3**) as part of the data flow. This is achieved by mapping the uncertainty impacts from the annotated architectural elements to the corresponding elements of the data flow [17]. In the case of both data types, it is uncertain which allocation is chosen (**U2** and **U3**). This is represented as uncertainty in the data flow toward the *store* process of all possible architectural variants. In the case of user data, it is additionally unclear whether the data is encrypted (**U1**) which is represented as two alternative process behaviors.

Our approach combines variation modeling with data flow-based confidentiality analysis [38]. By generating variants of the software architecture, as explained in Section II, we get a set of data flows without uncertainty for each variant. These variants represent all possible outcomes of the uncertainty impacts, which can be traced back from the variant to the variation model. For each variant, we analyze which uncertainties could be responsible for confidentiality violations. The confidentiality analysis [38] automatically yields places where confidentiality violations occur. By starting at the violation and following the data flow in the reverse direction, we identify uncertainty that affects elements that are part of the data flow. This information is part of the variation model and can be traced for every architectural element. We filter uncertainty impacts that are not responsible for confidentiality violations, i.e., uncertainties that do not affect elements contained in data flows with violations. Compared to existing black-box approaches [45], our approach uses information that emerges during the analysis in a white-box manner, see Section III.

In the following, we explain the analysis with one of the eight possible variants generated using the variation model.

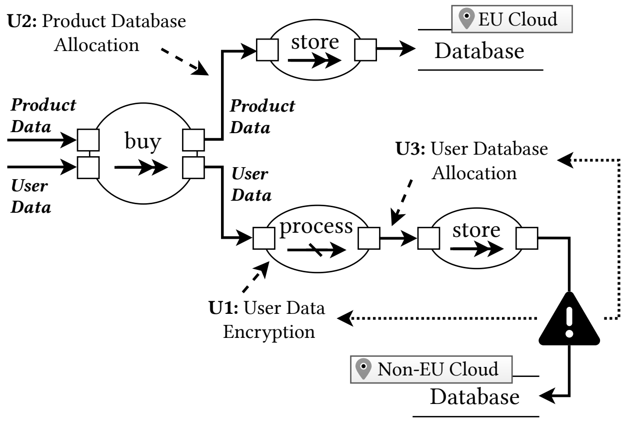

Figure 3. Data flow diagram of one variant of the running example

In this variant, user data is encrypted in the *Online Shop* component (**U1**) before storing it. The *Product Database Service* is allocated on an *EU Cloud* server (**U2**), and the *User Database Service* is allocated in the *Non-EU Cloud* (**U3**). The resulting data flow diagram is shown in Figure 3. We also display which elements of the data flow are related to the uncertainty impacts (**U1** – **U3**). In this variant, a violation occurs in the data flow toward the *Database*. The violation is caused by user data flowing to the *Non-EU Cloud*, which has been forbidden by a confidentiality requirement. It is marked in Figure 3 with an exclamation mark. A *scenario-aware* analysis would thus simply reject this variant. Our *data flow-aware* approach additionally analyzes the impact of uncertainty on critical data flows.

Starting from the point of violation at the *Database*, we traverse the data flow of the variant in reverse order of nodes and edges. In the example variant, this means going back from the *Database* to the *store*, *process*, and *buy* processes. We save which elements have been traversed and also which are the corresponding elements of the modeled software architecture. For each element, we check whether it has been varied by a variation point from the variation model and thus is related to an uncertainty impact. In Figure 3, this is demonstrated with dotted arrows. Any uncertainty impact that varies elements traversed on the data flow is flagged as potentially influencing. In the example variant, this is the case for the *User Database Allocation* (**U3**) that affects the flow between the *process* and *store* processes. We continue to traverse the data flow and also identify the *User Data Encryption* (**U1**) as potentially influencing. This second impact cannot be safely excluded as we cannot make any statements on the role of single impacts in the confidentiality violation. Additionally, there could be potential uncertainty interaction effects [8].

This procedure is repeated for each variant and all identified confidentiality violations. This gives us, for each variant, a set of uncertainties potentially involved in violations and a set of uncertainties not involved in any violations. An uncertainty impact is not involved *iff* there is no violation after the varied architectural element on any data flow. In our example, the uncertainty regarding the allocation of the *Product Database Service* (**U2**) is filtered out as being unproblematic.

The individual results are combined to form an overall statement about the architecture's confidentiality under uncertainty. This enables us to check which modeled uncertainty impacts have no impact on confidentiality at all and need not be considered. Also, we get architecture variants without any confidentiality violation. In the running example, this would be the variant with the encryption of user data (**U1**) and the allocation in the *EU Cloud* (**U3**). The allocation of the *Product Database* (**U2**) does not affect confidentiality in any variant. The information obtained about the uncertainty impacts can save valuable time for software architects by giving them a direct indication of which uncertainties should be prioritized [17]. This is achieved by adding traceability between modeled uncertainty impact, architecture variation, data flow, and confidentiality violations.

V. Case Study-based Evaluation

In this section, we show the evaluation of our approach. We present the evaluation plan, evaluation design, and evaluation results. Afterward, we discuss potential threats to validity and limitations of our approach.

A. Evaluation Goals, Questions, and Metrics

We use a *Goal Question Metric* plan [2] to evaluate the presented approach. We define the following goals:

G1 Evaluate the accuracy of the analysis results regarding uncertainty, also compared to the state of the art.

G2 Evaluate the usability of the analysis by software architects, e.g., by reducing effort and complexity.

We motivate our work with the lack of precision and comprehensiveness of existing and closely related analyses [38], [45]. Goal **G1** shall evaluate the accuracy of the analysis results of our approach, also compared with the different analysis types discussed in Section III. This represents a typical approach to design time analysis evaluation [26]. This includes evaluating the behavior of the analysis in different variants with and without uncertainty and confidentiality violations. We ask:

Q1.1 Does our analysis at least have the same accuracy as the *naive* approach?

Q1.2 Does our analysis at least have the same accuracy as the *scenario-aware* analysis?

Q1.3 Does our analysis accurately identify uncertainties that have an impact on confidentiality violations?

The evaluation is based on three case studies with an existing gold standard. We measure precision $P = \frac{TP}{TP+FP}$ (**M1.1**) and recall $R = \frac{TP}{TP+FN}$ (**M1.2**) [32].

Goal **G2** considers the usability of our analysis. This includes the feasibility of uncertainty-aware confidentiality analysis and whether software architects already have enough knowledge at design time. Also, the modeling complexity and required effort should be reduced in comparison to a manual analysis conducted by an expert. We ask:

Q2.1 Is the required information to use our analysis available to software architects at design time?

Q2.2 Does our analysis reduce the effort of confidentiality analysis under uncertainty compared to manual analysis and the state of the art?

To answer these questions, we consider the availability of the information categories presented in Section III at design time. Also, we discuss the required steps in the evaluated approaches to conducting uncertainty-aware confidentiality analysis.

B. Evaluation Design

Our evaluation of Goal **G1** is using three case studies that are summarized in the following. Afterward, we describe how we use these case studies to answer our evaluation questions.

All case studies originate from other work [15], [24], [38]. They have been previously used to evaluate confidentiality analysis under uncertainty [45]. In total, the three case studies consist of 36 architectural variants. All variants contain data flows that can lead to confidentiality violations.

Table III
PROPERTIES OF THE CASE STUDIES USED IN THE EVALUATION

Confidentiality requirements	**Variation points**
Travel Planner [24], [38], [45]	
Credit card information may only be used as the user agrees to	Credit card information can be used with or without user consent
Distance Tracker [24], [38], [45]	
GPS-Data may only be processed on the user's smartphone	GPS-Data can be processed on the smartphone or a server
Online Shop [15]	
Personal user data may only be stored encrypted and inside the EU	Users can enter personal data, this data can be encrypted, and both the data and its backup database services can be allocated on different servers which are deployed either in the EU or non-EU cloud

Table IV
RESULTS OF THE ACCURACY EVALUATION FOR Q1.1 AND Q1.2

Case study	**#Variant**	**#Violation**	**Naive**	**Scenario aware**	**Dataflow aware**
			P, *R*	*P*, *R*	*P*, *R*
Travel Planner	2	1	0.5, 1.0	1.0, 1.0	1.0, 1.0
Distance Tracker	2	1	0.5, 1.0	1.0, 1.0	1.0, 1.0
Online Shop	32	18	0.6, 1.0	1.0, 1.0	1.0, 1.0

Table III summarizes the case studies' confidentiality requirements and variation points. The *Travel Planner* offers users a tool to find flight connections for an airline that are paid with a credit card. When booking, users give the airline their credit card information and permission to use this data for billing. The possibility that the airline may handle the user's credit card information differently adds an element of uncertainty. The *Distance Tracker* is an app to keep track of the covered distance while jogging. As the user's GPS data is sensitive, the user's smartphone should be the only location where the GPS data is processed. In some cases, this data may be processed on external servers which adds uncertainty. With the *Online Shop*, users can browse available offers and buy selected products. Our running example represents a simplified version of this case study. The original [15] and the adapted version [4] consist of more uncertainty impacts, e.g., in the user behavior.

To evaluate Goal **G1**, we used existing architectural models of all three case studies together with an implementation of our *data flow-aware* analysis. For the comparison, we additionally implemented a *naive* and *scenario-aware* approach, according to Walter et al. [45]. All used models and implementations are available online [3]. To evaluate Goal **G2**, we discuss the availability of modeling information and knowledge based on other work [17], [38]. Then, we compare the effort to use our approach to manually modeling architectural variants for confidentiality analysis [38] and *scenario-aware* analysis [45].

C. Evaluation Results and Discussion

We present and discuss our evaluation results. For Question **Q1.1** and Question **Q1.2**, we compared the analysis results of our *data flow-aware* approach with the *naive* and *scenario-aware* approaches discussed in Section III. Our hypothesis was that all three analysis approaches should have the same recall *R* but differ in precision *P*. This has also been shown by the analysis results that are presented in Table IV.

The *naive* approach can analyze confidentiality under uncertainty with high recall. However, as an architectural model is directly rejected in the case of a confidentiality violation, the precision is inferior. For example, the *Online Shop* case study only has 18 of 32 variants that violate confidentiality. The naive approach only reaches a precision of $\frac{18}{32} = 0.56$. In cases with more infrequent violations, the precision can become much worse. Both the *scenario-aware* and the *data flow-aware* analysis have the expected high recall and precision. The values of 1.0 indicate a perfect match to the gold standard.

The difference between these approaches becomes visible in answering Question **Q1.3**. Here, only the results of the *Online Shop* case study are relevant, as the others only contain one uncertainty impact. While the recall is equal at 1.0, the precision between both approaches differs. Here, *scenario-aware* analysis cannot distinguish between the five uncertainty impacts shown in Table III and thus only reaches a precision of $\frac{4}{5} = 0.8$. Our approach accurately detects the product data as not critical and reaches a precision of 1.0.

To answer Question **Q2.1**, we discuss whether the information required for uncertainty-aware confidentiality analysis is available at design time. The availability of architectural models and confidentiality-related information (**1.** – **5.**) can be assumed as this information belongs to the requirements and the design phase of software systems [36], [38]. Uncertainty sources (**6.**), their classification (**7.**) and mitigation (**9.**) can be derived from existing collections [17] already at design time without special knowledge. Only annotating the uncertainty impact (**8.**) could require additional expertise. However, this is also the case for the other considered approaches. The interaction of uncertainties (**10.**) also represents a new challenge and requires additional insights beyond our current research. Regarding Question **Q2.2**, directly using a confidentiality analysis [38] would require software architects to manually model each variant. With a growing number of uncertainties to model and analyze, this quickly becomes unpractical. Also, changes in the software architecture have to be manually applied to each variant which is not feasible. Compared to the state of the art [45], which uses degrees of freedom to model uncertainty [27], we assume that our variation model [43] requires similar effort due to an automated generation and analysis. This shall be improved in the future by mapping uncertainty sources to impacts in the architectural model [17].

D. Threats to Validity

We briefly discuss the threats to validity based on the scheme proposed by Runeson and Höst [34]. Regarding *internal validity*, a threat is the reimplementation of the *scenario-aware* analysis using the variation model [43]. Here, we minimized the risk by comparing our results to the original analysis [45] and by using the same case studies and the same gold standards for the evaluation. Regarding *external validity*, the biggest threat originates from the selection of case studies, which limits generalizability. To that end, we included differently classified uncertainties [17], e.g., uncertainties that affect the system's structure, behavior, and deployment. To maximize *construct validity*, we used a *Goal Question Metric* plan [2] and oriented our evaluation plan to similar approaches [7], [45]. Konersmann et al. [26] state the lack of replication packages and the availability of tools used for the evaluation. To overcome this limitation and to increase *reliability*, we published a data set containing all evaluation data [3].

E. Limitations

We are aware of two limitations of our approach. First, our analysis depends on a model-based confidentiality analysis [38] and an architectural variation model [43]. Our analysis capabilities are restricted by the performance and the expressiveness of these approaches, i.e., by the analyzable confidentiality requirements [18] and uncertainty types. In particular, our approach can only model and analyze *Scenario Uncertainty* where a finite set of possible outcomes exist [17]. In future work, we plan to expand the modeling capabilities, e.g., by mapping *Statistical Uncertainty* to distinct scenarios. Second, our approach requires the annotation of uncertainty impacts in the architectural model instead of relating to the uncertainty sources. While this does not affect our analysis capabilities, we assume that this requires experienced software architects. We plan to automate the mapping of uncertainty sources to potential impact locations in future work.

VI. Related Work

In this section, we give an overview of related work and compare it to our approach based on the categories presented in Section III. We divide the work into three parts: Design time confidentiality analysis, architecture-based uncertainty analysis, and uncertainty-aware confidentiality analysis.

Design time confidentiality analysis: Confidentiality is included in the system architecture design processes using methods like *Privacy by Design* [35]. A wide range of model-based confidentiality analyses exist. Examples are the analysis of access control and information flow policies using data flow diagrams [38] or the pattern-based detection of design flaws [41]. Model-checking has also been proposed to evaluate information flow security [13]. Other approaches use theorem proving based on annotating UML diagrams [25] or by utilizing UML profiles [23]. However, none of these approaches considers uncertainty in the architectural modeling.

Architecture-based uncertainty analysis: Current research on uncertainty in software architectures often analyzes uncertainty with detailed inputs to the uncertainty model. Troya et al. [40] give an overview of the modeling of uncertainty. Sobhy et al. [39] provide an overview of the handling of uncertainty in the analysis of software architecture. Values such as the probability that a particular component is used, or the reliability of a component contribute to the analysis of uncertainty [14]. Other approaches use reusable architecture design decisions and fuzzy logic to derive the quality of software architecture under uncertainty [29]. GuideArch [11] also uses fuzzy values for design space exploration under uncertainty. PerOpteryx [28] represents another approach for design space exploration based on defining degrees of freedom in software architecture models. While these approaches can analyze a wide variety of quality properties, they are not appropriate to consider confidentiality as they lack the required expressiveness to consider data processing and constraints.

Uncertainty-aware confidentiality analysis: By considering both uncertainty and confidentiality in the analysis, expressiveness can be enhanced. Walter et al. [45] present an approach that combines PerOpteryx [28] with a model-based confidentiality analysis [38]. This enables the rejection of architecture variants that violate confidentiality, i.e., *scenario-aware* analysis. Boltz et al. [7] represent environmental uncertainty by fuzzy values and incorporate them in design time confidentiality analysis [38]. These approaches are closest to our work. However, they either lack precise analysis results or expressiveness in representing different types of uncertainty.

VII. Conclusion

In this paper, we presented an approach for confidentiality analysis under uncertainty. We described which information categories are available in the design regarding confidentiality and uncertainty. This enables the identification and categorization of existing and possible uncertainty-aware confidentiality analyses. Afterward, we presented a *data flow-aware* analysis that combines aspects both of uncertainty and confidentiality analysis. Based on modeling uncertainty impacts as variation in the software architecture, model variants are generated. After analyzing each variant, identified confidentiality violations are related to potential uncertainty impacts. Our approach is not only able to yield variants with and without violations but also filters uncertainties by their impact on the system's confidentiality. The evaluation of our analysis showed an increased precision compared to the state of the art while maintaining or even reducing the required effort and knowledge.

The results of our analysis utilize the available information within architectural models more efficiently and shall enable an easier interpretation by software architects. This is especially true for large models with specific confidentiality violations that have to be traced through the complete system to become manageable. Here, our analysis' precision becomes a major advantage compared to the also considered *naive* and *scenario-aware* approaches. Additionally, the variation model was designed to be extended with uncertainty types.

In future work, we want to tackle several limitations of the presented analysis. First, we want to connect the presented *data flow-aware* approach to an *Uncertainty Impact Analysis* [17]. This shall bridge the gap between uncertainty sources in the software architecture and its environment and the impact on the architectural model. This would enable us to connect our analysis to existing classifications [1], [17] of uncertainty to simplify prioritization and mitigation. Second, we imagine moving the variability modeling from the architectural abstraction to data flow diagrams. Due to the simplified meta model of data flow diagrams, this would decrease the complexity while maintaining expressiveness. Third, we aim to support more uncertainty types. Starting with the modeling of more uncertainty locations [17], we also want to support the analysis of *Statistical Uncertainty*. Here, one approach is the mapping of uncertainty characteristics to a finite number of classes [7].

Acknowledgment

This work was supported by the German Research Foundation (DFG) under project number 432576552, HE8596/1-1 (FluidTrust), as well as by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. It was also partially supported by the Czech Science Foundation project 20-24814J.

References

[1] M. Acosta, S. Hahner, A. Koziolek, T. Kühn, R. Mirandola, and R. Reussner, "Uncertainty in coupled models of cyber-physical systems," in *MODELS-C*, 9, 2022, pp. 569–578.

[2] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, p. 10, 1994.

[3] T. Bitschi, *Prototype Implementation: Uncertainty-aware Confidentiality Analysis Using Architectural Variations*, 2022. DOI: 10.5281/zenodo.7236107.

[4] T. Bitschi, "Uncertainty-aware confidentiality analysis using architectural variations," Bachelor's Thesis, Karlsruhe Institute of Technology (KIT), 2022. DOI: 10.5445/IR/1000153079.

[5] B. Boehm and V. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[6] N. Boltz, M. Walter, and R. Heinrich, "Context-based confidentiality analysis for industrial IoT," in *SEAA*, 2020, pp. 589–596.

[7] N. Boltz *et al.*, "Handling environmental uncertainty in design time access control analysis," in *SEAA*, 2022.

[8] J. Cámara *et al.*, "Addressing the uncertainty interaction problem in software-intensive systems: Challenges and desiderata," in *MODELS*, 23, 2022, pp. 24–30.

[9] Council of European Union. "REGULATION (EU) 2016/679 (general data protection regulation)." (2016), [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04 (visited on 12/01/2022).

[10] T. DeMarco, "Structure analysis and system specification," in *Pioneers and Their Contributions to Software Engineering*, 1979, pp. 255–288.

[11] N. Esfahani, S. Malek, and K. Razavi, "GuideArch: Guiding the exploration of architectural solution space under uncertainty," in *ICSE*, ISSN: 1558-1225, 2013, pp. 43–52.

[12] D. Garlan, "Software engineering in an uncertain world," in *FoSER*, 2010, p. 125.

[13] C. Gerking, D. Schubert, and E. Bodden, "Model checking the information flow security of real-time systems," in *Engineering Secure Software and Systems*, 2018, pp. 27–43.

[14] K. Goseva-Popstojanova and S. Kamavaram, "Assessing uncertainty in reliability of component-based software systems," in *ISSRE*, 2003, pp. 307–320.

[15] S. Hahner, "Architectural access control policy refinement and verification under uncertainty," in *ECSA-C*, 2021.

[16] S. Hahner, "Dealing with uncertainty in architectural confidentiality analysis," in *Proceedings of the Software Engineering 2021 Satellite Events*, 2021, pp. 1–6.

[17] S. Hahner, S. Seifermann, R. Heinrich, and R. Reussner, "A classification of software-architectural uncertainty regarding confidentiality," in *ICETE*, To appear, 2023.

[18] S. Hahner, S. Seifermann, R. Heinrich, M. Walter, T. Bureš, and P. Hnětynka, "Modeling data flow constraints for design-time confidentiality analyses," in *ICSA-C*, 2021, pp. 15–21.

[19] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, "Uncertainty in self-adaptive systems: A research community perspective," *ACM TAAS*, vol. 15, no. 4, 10:1–10:36, 2021.

[20] J. Isaak and M. J. Hanna, "User data privacy: Facebook, cambridge analytica, and privacy protection," *Computer*, vol. 51, no. 8, pp. 56–59, 2018.

[21] ISO, "ISO/IEC 27000:2018(e) information technology – security techniques – information security management systems – overview and vocabulary," Standard, 2018.

[22] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 2005, pp. 109–120.

[23] J. Jürjens, "Towards development of secure systems using UMLsec," in *Fundamental Approaches to Software Engineering*, 2001, pp. 187–200.

[24] K. Katkalov, "Ein modellgetriebener Ansatz zur Entwicklung informationsflusssicherer Systeme," Ph.D. dissertation, University of Augsburg, 2017.

[25] K. Katkalov, K. Stenzel, M. Borek, and W. Reif, "Model-driven development of information flow-secure systems with IFlow," in *2013 International Conference on Social Computing*, 2013, pp. 51–56.

[26] M. Konersmann *et al.*, "Evaluation methods and replicability of software architecture research objects," in *ICSA*, 2022, pp. 157–168.

[27] A. Koziolek, "Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes," Ph.D. dissertation, 2011. DOI: 10.5445/IR/1000024955.

[28] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated application of tactics in multi-objective software architecture optimization," in *ISARCS*, 20, 2011, pp. 33–42.

[29] I. Lytra and U. Zdun, "Supporting architectural decision making for systems-of-systems design under uncertainty," in *SESoS*, 2, 2013, pp. 43–46.

[30] S. McConnell, *Software project survival guide*. Microsoft Press, 1998.

[31] D. Perez-Palacin and R. Mirandola, "Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation," in *ICPE*, 2014, pp. 3–14.

[32] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to ROC, informedness, markedness and correlation," *CoRR*, vol. abs/2010.16061, 2011.

[33] R. H. Reussner *et al.*, *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press, 2016.

[34] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.

[35] P. Schaar, "Privacy by design," *Identity in the Information Society*, vol. 3, no. 2, pp. 267–274, 2010, Springer.

[36] S. Seifermann, "Architectural Data Flow Analysis for Detecting Violations of Confidentiality Requirements," Dissertation, Karlsruhe Institute of Technology (KIT), 2022.

[37] S. Seifermann, R. Heinrich, D. Werle, and R. Reussner, "A unified model to detect information flow and access control violations in software architectures:" in *SECRYPT*, 2021, pp. 26–37.

[38] S. Seifermann, R. Heinrich, D. Werle, and R. Reussner, "Detecting violations of access control and information flow policies in data flow diagrams," *JSS*, vol. 184, p. 111 138, 1, 2022.

[39] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman, "Evaluation of software architectures under uncertainty: A systematic literature review," *ACM TOSEM*, p. 50, 2021.

[40] J. Troya, N. Moreno, M. F. Bertoa, and A. Vallecillo, "Uncertainty representation in software models: A survey," *SoSyM*, 8, 2021.

[41] K. Tuma, L. Sion, R. Scandariato, and K. Yskout, "Automating the early detection of security design flaws," p. 11, 2020.

[42] W. E. Walker *et al.*, "Defining uncertainty: A conceptual basis for uncertainty management in model-based decision support," *Integrated assessment*, vol. 4, no. 1, pp. 5–17, 2003, Publisher: Taylor & Francis.

[43] M. Walter, S. Hahner, T. Bures, P. Hnetynka, R. Heinrich, and R. Reussner, "Architectural attack propagation in industry 4.0," *at - Automatisierungstechnik*, accepted, to appear.

[44] M. Walter, R. Heinrich, and R. Reussner, "Architectural attack propagation analysis for identifying confidentiality issues," in *ICSA*, 2022, 12 S.

[45] M. Walter *et al.*, "Architectural optimization for confidentiality under structural uncertainty," in *Software Architecture*, 2022, pp. 309–332.

[46] H. Weisbaum, *Trust in facebook has dropped by 66 percent since the cambridge analytica scandal*, accessed 11/28/2022, 2018. [Online]. Available: https://nbcnews.com/business/consumer/trust-facebook-has-dropped-51-percent-cambridge-analytica-scandal-n867011.