Modeling and Verification of Privacy Enhancing Protocols

Suriadi Suriadi, Chun Ouyang, Jason Smith, and Ernest Foo

Queensland University of Technology, Australia
{s.suriadi,c.ouyang,j4.smith,e.foo}@qut.edu.au

Abstract. Privacy enhancing protocols (PEPs) are a family of protocols that allow secure exchange and management of sensitive user information. They are important in preserving users' privacy in today's open environment. Proof of the correctness of PEPs is necessary before they can be deployed. However, the traditional provable security approach, though well established for verifying cryptographic primitives, is not applicable to PEPs. We apply the formal method of Coloured Petri Nets (CPNs) to construct an executable specification of a representative PEP, namely the Private Information Escrow Bound to Multiple Conditions Protocol (PIEMCP). Formal semantics of the CPN specification allow us to reason about various security properties of PIEMCP using state space analysis techniques. This investigation provides us with preliminary insights for modeling and verification of PEPs in general, demonstrating the benefit of applying the CPN-based formal approach to proving the correctness of PEPs.

1 Introduction

As a response to the increasing number of incidents compromising the privacy of millions of users [1], there has been an increase in the research related to privacy enhancing protocols (PEPs). PEP is a generic term that refers to protocols whose main purpose is to preserve users privacy in an open communication environment (e.g. over the Internet). For example, emulating the off-line anonymity afforded by cash transactions, a PEP ensures that when a user purchases goods on-line, the on-line seller does not learn the identity of the user. A PEP normally *applies* complex cryptographic primitives (such as custodian-hiding group encryption and verifiable encryption) to achieve the privacy-enhancing features. Recently, the Trusted Platform Module (TPM) technology - which provides secure hardware storage of cryptographic keys and implementation of common cryptographic primitives - has also been used in PEPs [2].

An important issue in the design of applied cryptographic protocols, such as PEPs, is to ensure they work correctly and do not contain errors that may weaken the original security protections provided by the cryptographic primitives employed. Formal methods are necessary for the construction of unambiguous and precise models that can be analysed to identify errors and verify correctness before implementation. The application of formal methods has been demonstrated

K. Breitman and A. Cavalcanti (Eds.): ICFEM 2009, LNCS 5885, pp. 127–146, 2009.

to lead to reliable and trustworthy security protocols [3, 4, 5]. However, to the best of our knowledge, no existing work provides a formal verification of PEPs.

In the domain of cryptography, the main method to verify a cryptographic primitive is the provable security approach [6]. This approach aims to prove some standard security properties of cryptographic primitives by reducing the proof of those properties to some hard (normally mathematical) problem within the context of a simplified standard attack model with well-defined boundaries (such as the random oracle model). It is however not suitable for verification of PEPs and the reasons are two-fold. On the one hand, the security properties of a PEP are *behavioral* properties and proof of these properties can hardly be reduced to pure mathematical problems. On the other hand, the simplified assumptions employed in the provable security approach are not applicable to PEPs due to the expanded threat environment in which PEPs operate. In PEPs, one needs to consider attacks introduced by the existence of multi-party entities and attacks targeted at the *design* of a protocol, not directly at the cryptographic primitives employed. The lack of computer-aided tools in the provable security approach also makes such an approach not scalable when modeling and verifying a large system such as PEPs. While provable security has been used to verify certain types of protocols (notably key establishment protocols), we note that it is nevertheless not suitable to verify behavioral properties.

Coloured Petri Nets (CPNs) [7] are a widely-used formal method for system specification, design, simulation and verification. They provide a graphical-oriented modeling language capable of expressing concurrency, synchronisation, non-determinism, and system concepts at different levels of abstraction. CPNs combine Petri nets [8] and the functional programming language Standard ML (SML) [9]. Petri nets are used to model concurrency, synchronisation and resource sharing, and support an abundance of analysis techniques such as the well-known state space techniques. SML is used to capture data manipulation and to create compact and parameterisable models. CPN Tools [10] is a graphical tool supporting the construction, simulation and analysis of CPN models.

In this paper, we propose a CPN-based approach for modeling and verification of PEPs. CPNs are used to construct a formal specification of a representative PEP, namely the Private Information Escrow Bound to Multiple Conditions Protocol (PIEMCP) [11]. PIEMCP involves large multi-party communication and employs complex cryptographic primitives and TPM functionalities. The hierarchical structuring mechanism of CPNs supports a modular and systematic approach in capturing the behavior of PIEMCP at different levels of abstraction. Using SML, a wide variety of cryptographic primitives and the processing of these primitives are captured in meta-models that are embedded in higher levels of the protocol operations. By parameterising the protocol model with different types of attacks, a large number of attack scenarios are captured for analysis. The CPN model of PIEMCP is executable and can be analysed to verify the security behavior of the protocol. The analysis of PIEMCP is performed using the state space generated from the parameterized CPN model and the selective runtime protocol session data stored as external files.

The contributions of this paper are two-fold. First of all, it demonstrates the use of CPN to model and verify the security behavior of PEPs. To the best of our knowledge, this is the first attempt at the formal verification of a PEP. Secondly, the paper proposes several modeling and analysis techniques that have been applied to other PEPs [12, 13]. These techniques may be used as preliminary guidelines for a general CPN-based approach for modeling and verification of PEPs. Also, efficiency is another major concern in PEPs due to the use of resource-intensive cryptographic primitives. The CPN model of PIEMCP developed in this paper can be easily extended in the future to allow a simultaneous analysis of both the protocol performance and security behavior.

The rest of the paper is structured as follows. Sect. 2 provides some background information about PIEMCP. Sect. 3 proposes the modeling approach and describes selected parts of the CPN model of PIEMCP. Based on this CPN model, Sect. 4 details the verification of a set of security behaviors of PIEMCP. Sect. 5 reviews related research efforts. Finally, in Sect. 6 we summarize our contribution and discuss future work.

2 Overview of PIEMCP

The PIEMCP [11] is used in a federated single-sign on (FSSO) environment whereby a user only has to authenticate once to an identity provider (IdP) to access services from multiple service providers (SPs). The entities involved are users, IdPs, SPs, and an anonymity revocation manager (ARM) or some referees. An IdP assures SPs that although users are anonymous, when certain conditions are fulfilled, the users' identities can be revealed. A user's identity refers to a set of personally identifiable information (PII). Although the services that SPs provide can be delivered without the need of PII, they require the PII to be revealed by an ARM *or* some referees when certain conditions are satisfied.

The PIEMCP consists of four stages, namely PII escrow (PE), key escrow (KE), multiple conditions (MC) binding, and revocation. An execution of the protocol involves two distinct sessions: the *escrow session* which consists of a sequential execution of the PE, KE and MC stages, and the *revocation session* which consists of an execution of the revocation stage. A user can run n escrow sessions, during which his/her PII is hidden (anonymous). At least one escrow session has to be completed before a revocation session can start. During the revocation session, the user's PII linked to a specific SP in a specific escrow session is revealed. For n escrow sessions, each with m-number of SPs, up to $n \times m$ revocation sessions can be performed.

The PIEMCP has two variants: the first variant (PIEMCP-T) uses a trusted ARM for anonymity revocation, and the second variant (PIEMCP-NT) uses a group of referees instead of ARM. In both variants, most of the operations, especially those in the PE, KE and MC stages, are performed in a similar way. Therefore, we describe the main operations in one of them, the PIEMCP-NT. Fig. 1 depicts the message exchanges between the different entities within the four stages of this protocol.

The *PE stage* begins when a user requests a service from a service provider SP1. This triggers the agreement of conditions ($Cond1$) whose fulfillment allows the PII to be revealed. SP1 then sends a message `NT-PE-1` containing $Cond1$ to an IdP to escrow the user's PII. The IdP contacts the user to obtain his encrypted PII. The user encrypts the PII using a Verifiable Encryption (VE) scheme under a freshly generated key pair (public and private keys). The user sends to the IdP `NT-PE-2` comprising the VE ciphertext and the public key used for the encryption. The user keeps the private key, which is needed to decrypt the ciphertext. Next, the user and the IdP engage in a cryptographic "proof-of-knowledge" (PK) protocol (`NT-PE-3`). This is to prove to the IdP that the VE ciphertext given correctly hides some *certified* PII without letting the IdP learn the value of the PII itself. We denote this operation as `PKVE`. The output of PKVE is an acceptance or rejection of the VE ciphertext.

The *KE stage* is started when the `PK-VE` outputs an acceptance of the ciphertext. The IdP and the user then engage in another PK protocol - the Direct Anonymous Attestation (DAA) (`NT-KE-1`). This is to convince the IdP that the user is using a valid TPM device while concealing the identity of the TPM device. A successful DAA prompts the user's TPM to generate (1) a universal custodian-hiding verifiable group encryption (UCHVE) of the VE private key under $Cond1$ and (2) a TPM proof of a correct UCHVE execution. A UCHVE produces n ciphertext pieces for a group of n referees among whom there are t ($t \leq$n) *designated* referees, and only designated referees can decrypt these ciphertext pieces. At least k ($k \leq t$) decrypted pieces are required to recover the VE private key (i.e. k is the *threshold* value). Both the n ciphertext pieces and the TPM proof are sent to the IdP in `NT-KE-2`. The IdP then verifies the proof and if correct, prepares a response `NT-KE-3` to SP2 which includes the VE of PII (from the PE stage) and the UCHVE of the VE private key. SP1 now has the ciphertext of the PII (from the PE stage) and the ciphertext of the corresponding private key. With the help of referees, SP1 can recover the user's PII when $Cond1$ is fulfilled, but *cannot* decrypt these ciphertexts until that time.

In the *MC stage*, the user goes to another service provider SP2. This time SP2 (instead of SP1 in the PE stage) needs the IdP to escrow the VE private key in `NT-MC-1` under different conditions $Cond2$ ($Cond1 \neq Cond2$). The IdP requests the user's TPM to produce a new UCHVE ciphertext of the VE private key and the associated TPM proof in `NT-MC-2`. The user replies with the requested encryption and proof in `NT-MC-3`. The IdP verifies the proof and if correct, prepares a response `NT-MC-4` to SP2 which includes the VE of PII (from the PE stage) and the UCHVE of the VE private key (bound to $Cond2$). SP2 now has the data that, with referees' help, can reveal the PII when $Cond2$ are satisfied, but yet *cannot* decrypt these ciphertexts at this point. Note that the user may go to a third provider SP3, in which case, only the MC stage needs to be executed.

The *revocation stage* is executed when the agreed conditions are satisfied and when a user has completed at least one escrow session. Assuming that $Cond1$ is satisfied, SP1 sends a revocation request `NT-REV-1` comprising n ciphertext pieces to the n referees with $Cond1$. Each referee checks if $Cond1$ is fulfilled,

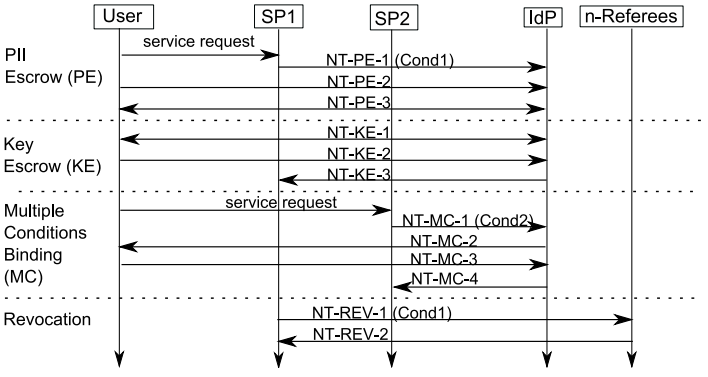

Fig. 1. Message exchanges within the four stages of PIEMCP-NT

and if so, the referee tries to decrypt the given ciphertext piece. Only the designated referees can decrypt the ciphertext pieces. If decryption is successful, each designated referee sends the decrypted data `NT-REV-2` to SP1. When k or more decrypted data are received, SP1 can recover the VE private key, and subsequently decrypt the VE ciphertext to recover the PII.

In the above we described the normal execution of PIEMCP-NT (i.e. without attacks). However, each of the parties involved in PIEMCP (both variants) may behave maliciously resulting in different attack scenarios. The design goal of PIEMCP is to achieve the expected security behavior with and without considering the attacks. In the next section, a CPN model of PIEMCP is presented which can be configured to capture possible attack and non-attack senarios. The model is then used as a basis for the verification of PIEMCP in Sect. 4.

3 CPN Model of PIEMCP

CPN Preliminaries. CPNs are a class of high-level nets that enhance Petri nets with data types. A CPN consists of two types of nodes, *places* (drawn as ellipses) and *transitions* (rectangles), and directed edges known as *arcs*. A place is typed by a *color set* and contains collections (*multi-sets*) of data items called *tokens* of the same type as the place. A transition represents an event and may have a *guard* associated with it. The guard is a boolean expression enclosed in square brackets. Arcs connect places to transitions and transitions to places, and are inscribed by expressions comprising variables, constants and functions. Variables are typed and can be assigned values known as *binding*. CPNs use a variant of SML for net inscriptions and declarations of variables and types.

A transition's *input places* have arcs going to the transition, while its *output places* have arcs coming from the transition. A transition is *enabled* if: 1) sufficient tokens exist in each input place to match each respective input arc inscription when evaluated for a particular binding of its variables, and 2) the transition guard evaluates to true for the same binding. If a transition is enabled, it can *occur* (or be *fired*). The occurrence of a transition removes tokens specified by the respective arc inscriptions from input places, and deposits tokens

specified by inscriptions on the output arcs into output places. The state of a CPN is called a *marking*. It consists of tokens distributed on each place of the CPN. The occurrence of transitions represent stage changes.

CPNs support hierarchical modeling which facilitates the construction of large models by using a number of CPN modules called *pages*. Each page is linked to a *substitution transition* (sub-transition) at a higher level of the model. By means of the hierarchical structuring mechanism it is possible to capture different abstraction levels of the modeled system in the same CPN model.

3.1 Modeling Approach

The PIEMCP (both variants) is modeled using hierarchical CPNs. There is one top-level (main) page and four sub-pages capturing the four stages of PIEMCP. Each of these sub-pages is named according to the stage it models. The PE page has one further sub-page. The PE page, KE page, and MC page can be executed in a loop to form an escrow session. The number of escrow sessions to be executed is parameterized. The revocation page can be executed after the completion of at least *one* escrow session. Below, we introduce three modeling approaches that are specific to PEPs. These approaches are demonstrated in Sect. 3.2.

Cryptographic primitive abstraction. To capture complex cryptographic behaviors, we firstly model the representation of a ciphertext as a CPN colour set, and then capture its operations by describing them as SML functions. This approach is flexible and inclusive as virtually any type of cryptographic primitives can be captured. The CPN `record` type can encode the necessary information to represent a primitive properly, and the SML can be used to simulate the operations. Expressing cryptographic operations as functions promotes reuse which leads to a cleaner and more concise model. In Sect. 3.2, we demonstrate this approach by modeling a VE ciphertext and a zero-knowledge operation (PK-VE). The complexity of UCHVE ciphertext prevents use from describing it due to the space constraint. However, it is available in the full-version of this paper [14].

We also propose a technique to capture the commonly-used message signing and verification operations. We define a CPN colour set for the message to be signed, followed by a definition of its signature. A signed message is a pair consisting of the message and its signature. The verification of a signed message upon the receipt of the message is *enforced* within a transition guard. If the signature verification fails, the message integrity and/or authenticity are compromised. As a result, the guard returns a false value, thus halting any further processing on the message - an expected fail-stop behavior.

TPM provable execution. We propose an approach to model a TPM's provable execution behavior [15]. Our model depicts how an entity can generate the *expected* TPM proof based on some known information, and compare it with the received TPM-generated output and its corresponding proof. In this way an incorrect TPM-generated output can be detected. The demonstration of this approach is available in the full-version of this paper [14].

parameterized attack. We propose a parameterisation approach to modeling attacks such that one or more attacks can be switched on or off depending on the environmental assumptions. In general, attacks can come from both external intruders (i.e. external entities attempt to access and break the protocol) and malicious insiders (i.e. protocol entities attempt to compromise a users PII to achieve some personal advantage). At this point, we scope our work to only consider malicious insiders - which we think is of a greater concern in PEPs.[1] To this end, the attack models specifying those from external intruders, such as the Dolev-Yao intruder model [16], are not used. There are many attacks that a malicious insider could launch. Creating a new model to capture each type of attack (existing or new) scales poorly as the number of attacks grows. Parameterisation allows the re-use of the existing model while allowing it to behave differently according to the attacks being set - virtually allowing thousands of possible attack scenarios to be captured. We have modeled 17 types of attacks in our model, each with a possible value of 'true' or 'false', thus capturing $2^{17} = 131072$ possible attack scenarios. The attack parameters can be encoded in the arc-inscriptions, transition guards, or transition code-regions (attached to a transition where one can specify side-operations upon execution of the transition, e.g. writing data to an external file). The advantage of this approach is that we do not have to change the structure of the model at all to obtain different behaviors.

In addition, we introduce two general modeling approaches. First, *session-data capture* is applied to capture runtime protocol data generated and received by entities for analysis. We take advantage of the executable CPN model by interfacing it with a set of output text files which store the session data during the execution of the model. Session data are firstly represented as CPN colour sets. Then, functions are written to read session data from text files into the appropriate CPN variables, and to write back the updated variables into text files. This allows easy reading, storing, and updating of session data during the model execution without having to maintain tokens in various places across multiple CPN pages, thus avoiding the application of the 'vacuum cleaner' functionality [17] to remove tokens at the end of each session. Next, we generate *one-time* random data which improves on the simple random (possibly repeated) number generation function supported in the current CPN Tools.

3.2 Model Description

Selected parts of the PIEMCP-NT CPN model (the main page, the PE page, and the revocation page) are described to demonstrate the above modeling approaches. Relevant CPN colour sets definitions are provided in Table 1. The entire model consists of 6 pages, 108 places, 79 transitions, 77 colour sets, 38 functions, 29 code-regions, and 21 parameters.

[1] While many types of attacks from external intruders (e.g. eavesdropping, message modification) can be mitigated through the use of secure communication channels (e.g. Secure Sockets Layer (SSL)), attacks from malicious insider could result in a misuse of PII without having to break the security of the communication channel.

Table 1. Colour Sets Definition

```
colset K_PUB_VE = INT;
colset K_PRIV_VE = INT;
colset K_SIGN_GEN = INT;
colset PII = STRING;
colset LABEL = STRING;
colset PROVABILITY = BOOL;
colset SP_REQ = record genCond:STRING * conditions1:STRING * <other fields omitted>
colset SP_REQ_SIG = record message:SP_REQ * key:K_SIGN_GEN;
colset SIGNED_SP_REQ = record message:SP_REQ * signat:SP_REQ_SIG;
colset COMMITMENT_PII = record message:PII * random:RANDOM;
colset SIGNATURE_GEN = record message:MSG * key:K_SIGN_GEN * provable: PROVABILITY;
colset SIGNED_MSG = record message:MSG * signat:SIGNATURE_GEN;
colset CIPHER_VE_PII = record message:PII * key:K_PUB_VE * label:LABEL * provable:PROVABILITY;
colset DEC_REQ = record conditions:LABEL * uchvePiece:CIPHER_UCHVE_KVE_PIECE;
colset DEC_REQ_SIGNATURE = record message:DEC_REQ * key:K_SIGN_GEN * provable:BOOL;
colset SIGNED_DEC_REQ = record message:DEC_REQ * signat:DEC_REQ_SIGNATURE;
```

Main page. Fig. 2 shows the main page of PIEMCP-NT. The protocol starts with a user and a service provide SP1 agreeing on a set of conditions (transition `U_SP1_GENERATE_ CONDITIONS`) before proceeding to execute the PE stage (sub-transition `PII_ Escrow`) and then the KE stage (sub-transition `Key_Escrow`). Upon completion of the KE stage, the user goes to another service provider SP2. Similarly, they need to agree on a set of conditions (transition `U_SP2_GENERATE_ CONDITIONS`) before starting the MC stage (sub-transition `Multiple_Conditions`). The completion of the MC stage marks the completion of one session which triggers the storage of the session data accumulated by all entities. The number of sessions executed is parameterized by the value of

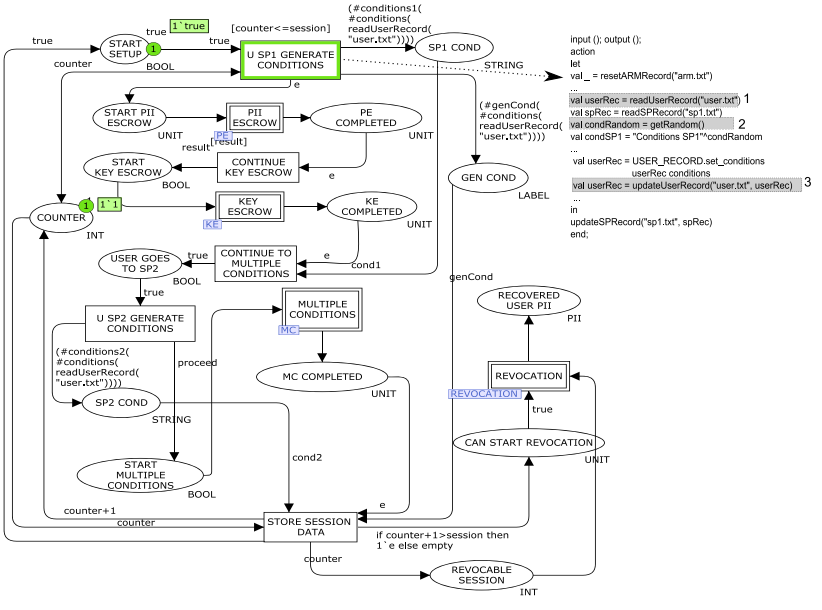

Fig. 2. The PIEMCP-NT CPN – Main page

`session`. Thus, if value of `counter` is less than or equal to `session` (note the guard for the transition `U SP1 GENERATE CONDITIONS`), the model will execute another session. Otherwise, the guard will disable the transition, and a token will be placed at the place `SP1_REVOCATION_CONDITIONS_FULFILLED` which triggers the start of a revocation stage which, if successful, results in the revelation of the user's PII represented by a token in the place `RECOVERED_USER_PII`.

This page also demonstrates the session data capture approach. The shaded text number 1 in Fig. 2 shows a code region which calls the function to read the user session data from a text file to a variable of type `USER_RECORD`. After performing some update operations on the variable (the one-time random number generator function is called in shaded text number 2), the update function is called to store the updated user session data into the text file again (shaded text number 3).

PE page. This page models the PE stage of PIEMCP-NT (Fig. 3). Here, we demonstrate the message signing and verification approach. The place `SP1_PII_REQ_SIGNATURE`, of type `SIGNED_SP_REQ`, represents the `NT-PE-1` message. From Table 1, this colour set represents a SP1-signed message whose content is *Cond*1. Other messages are omitted here for simplicity. As the IdP receives this message, the IdP first verifies the signature validity. As explained in Sect. 3.1, such a validation is captured in a transition guard. In this case, the transition guard at the `IDP_VERIFIES_SP1_REQ_AND_STARTS_PII_ESCROW` transition captures the signature validation process. If it returns true, the signature is valid and the transition is enabled, allowing PE stage to progress normally.

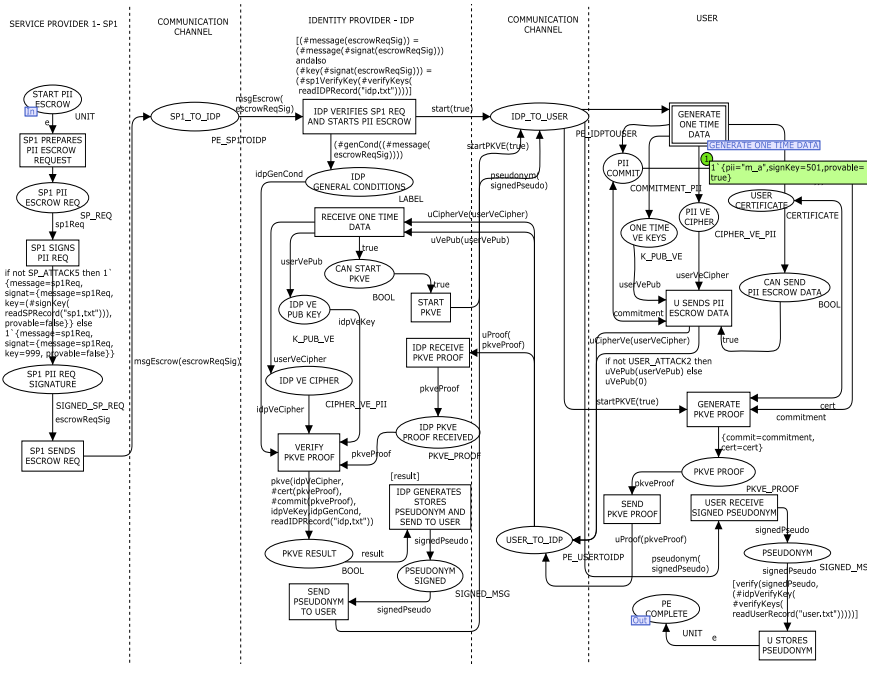

Fig. 3. The PIEMCP-NT CPN – PE page

The user then encrypts the PII. Here, we demonstrate how complex cryptographic primitive behaviors can be modeled. The VE ciphertext is defined as a CPN colour set of type `CIPHER_VE_PII` (see Table 1) which is a record consisting of four fields: the message itself, the public encryption key, the label under which the message is encrypted, and the provability property. A provable ciphertext means that the recipient of the ciphertext can be convinced that the received ciphertext correctly encrypts some claimed value (in this case the user's PII) without the recipient learning the value of either the PII itself or the decryption key. We consider the `message` field inside a CPN colour set that represents a ciphertext to be *unreadable*. The VE operations, including the encryption and decryption operations, are captured as functions. The VE ciphertext of PII is represented by a token in the place `PII VE CIPHER`.

Next, the user sends the `NT-PE-2` message (containing the VE ciphertext of PII, and the public VE key) - represented by the transition `U_SENDS_PII_ESCROW_DATA`. Upon receiving `NT-PE-2`, the PK-VE operation is triggered (NT-PE-3). Here, we demonstrate how a complex zero-knowledge proof protocol, such as PKVE is modeled in CPN. We break this operation into three transitions: `START_PKVE` (triggered by IdP to signal user the start of such a protocol), the `GENERATE_PKVE_PROOF` transition, executed on the user side to generate the required PKVE proof data, and the `VERIFY_PKVE_PROOF` executed by the IdP to verify the given PKVE proof data. The result of PKVE is represented by the place `PKVE_RESULT`. The essential processing required on the IdP to verify the correctness of the proof is captured by the function `pkve` called as arc inscription from the transition `VERIFY_PKVE_PROOF` to the place `PKVE_RESULT`.

There are two parameterized attacks: `SP_ATTACK5` (arc inscription from transition `SPI_SIGNS_PII_REQ` to place `SPI_PII_REQ_SIGNATURE`), and `USER_ATTACK2` (from transition `U_SENDS_PII_ESCROW_DATA` to place `USER_TO_IDP`). `USER_ATTACK2` depicts the behavior of a malicious user who gives an incorrect VE public key to the IdP in the `NT-PE-2` message. Thus, when `USER_ATTACK2` is set to 'true', the user will send an incorrect VE public key value represented by a value of '0', otherwise, a correct value is sent. `SP_ATTACK5` depicts the behavior of a malicious SP1 who uses an invalid signature key to sign the SP1 request message.

Revocation page. This page captures the UCHVE threshold decryption process (Fig. 4). Due to space limitation, it is impossible to go into the detail how we model such a threshold decryption process. Nevertheless, note the place `UCHVE_PIECE_DECRYPT_SUCCESS` and its corresponding output arc. The arc inscription requires t (representing the threshold value) successful decrypted pieces of the UCHVE group encryption by referees before the message (that is, the VE private key) can happen. Also note the parameterized malicious referees' behavior (`REF_ATTACK2`) who attempt to pool all decrypted UCHVE pieces amongst themselves with the hope of being able to recover the VE private key. Since our protocol assume that there is at least one honest designated referee, we assign such role to referee 2 (hence, we do not model referee 2 participating in the attack). This page also demonstrates how CPN can be used to capture concurrent processing required during the threshold decryption process.

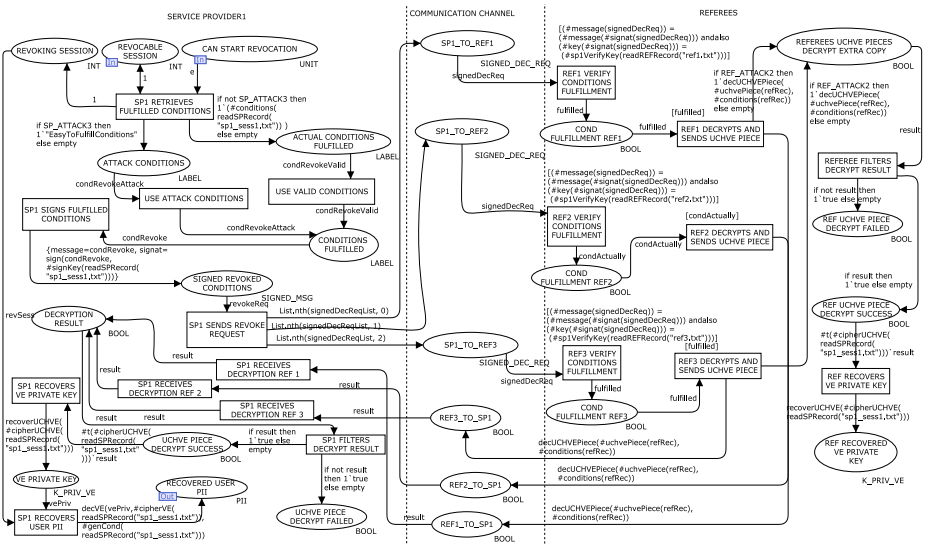

Fig. 4. Revocation page

4 Verification of the PIEMCP

We verify the correctness of PIEMCP using state space analysis. The basic idea behind the state space method of CPNs is to compute all reachable states and state changes of the system based on the CPN model. The verification of PIEMCP is carried out in two stages: the basic behavior verification and security behavior verification. The basic behavior verification is performed through standard state space analysis. It includes the analysis of proper session termination, deadlock freedom, livelock freedom, and absence of unexpected dead transitions. The security behavior verification is the focus of the paper.

Verifying the security behavior of PIEMCP is complicated due to the numerous avenues by which attackers could break the security protection provided by the protocol. We propose to scope the verification of the security behavior of PIEMCP within a set of plausible known attack scenarios. The result of such a verification is the assurance that the desired security behavior is achieved within the set of attack scenarios. As attacks are parameterized in the model, new types of attack scenarios can be added to the existing model without requiring major changes or a new model to be developed. A protocol is proved to be secure if the set of security properties hold in both the presence and absence of attacks. This is especially true in the case of PEPs whose main service (privacy) is in itself already a security behavior. When no attacks are modeled, we expect the security behavior to be fulfilled; when attacks are included, we expect the protocol to either detect it (and therefore stop), or be immune from those attacks.

The verification of the security behavior of the PIEMCP is performed as follows: firstly, the security behaviors of PIEMCP are formalized as Computational

Tree Logic (CTL) and/or standard state space statements; next, the formalized statements are used as queries for model-checking the state space generated from the PIEMCP CPN. Session data analysis is used when appropriate.

CPN Tools support state space analysis and model-checking the state space via ASK-CTL [18]. ASK-CTL is an implementation of a subset of CTL (mainly the "until" operator). It implements two basic operators to capture this logic: `EXIST_UNTIL`(A_1, A_2) and `FORALL_UNTIL`(A_1, A_2). The `EXIST_UNTIL` operator means that there must be at least *one* path, from a given state, whereby A_1 is true for every state in the path until the last state where A_2 is true. The `FORALL_UNTIL` operator is similar, except that it requires *all* paths to fulfill A_1 until A_2 is true. Based on these two operators, there are also `POS` and `EV` operators, where `POS`(A) = `EXIST_UNTIL`(TT, A), and `EV`(A) = `FORALL_UNTIL`(TT, A) (TT refers to a true value). These operators check the reachability of a state where A is true. `POS` checks if there is at least one path that leads to A, while `EV` checks if all paths lead to A. The `NF` operator contains a state formula function which returns a boolean value. There are many other ASK-CTL operators which we do not use, thus, not elaborated. CPN Tools contain a model checker which takes an ASK-CTL formula as an argument, checks the formula against the current state space of the CPN model, and returns the truth value of the given formula. Both the ASK-CTL logic and model checker are implemented in SML and thereby queries are formulated directly in SML syntax.

For simplicity, we consider a minimum full protocol execution. The PIEMCP CPN model is parameterized to execute two escrow sessions sequentially, followed by one revocation session. Note that it is possible for both the escrow and revocation session to run in parallel, however, modeling such concurrency does not capture any additional behaviors of the protocol as these two sessions are distinct, i.e. they do not interfere with each other. The state space generated from the above in the absence of attack behavior contains 147 nodes and 226 arcs. Next, the CPN model is parameterized to include a number of known attacks, resulting in a set of parameterized CPN models. Each of these models is executed to generate the state space for analysis of certain security properties.

Below, we define four security behaviors for PIEMCP and discuss in detail how we implement the first two properties in ASK-CTL queries in CPN Tools (the details of the queries for the other properties are available in the full version of this paper [14]). Fig 5 includes a set of notations to be used in the definition of these properties.

Let $T = \{t_s | s \in \{1, 2, ..., n\}\}$ be the set of (executed) escrow sessions, $P = \{SP_i | i \in \{1, 2, ..., n\}\}$ be the set of SPs. $\forall t_s \in T$, $\forall SP_i \in P$:

- v_i^s represents a VE ciphertext that SP_i holds for session t_s;
- u_i^s represents a UCHVE ciphertext that SP_i holds for t_s;
- v_{usr}^s represents a user-generated VE ciphertext for t_s;
- $\overline{u}_i^s$ represents the user-generated UCHVE ciphertext for SP_i in t_s;
- C_i^s represents the set of agreed conditions between a user and SP_i in t_s;
- G^s is the set of the general conditions in t_s;
- k^s represents the one-time VE public key that an IDP receives in t_s.

Fig. 5. List of notations to be used in the definition of security properties

4.1 Multiple Conditions

When PIEMCP runs without attacks, it is expected to reach the end of every escrow session, and also each SP should receive an escrowed PII that is cryptographically bound to conditions which are different from one SP to another. However, an attack may occur during an escrow session and as a result it is not possible for the PIEMCP to reach the end of that session. In the PIEMCP CPN, when the protocol reaches the end of an escrow session s, the place `MC_COMPLETE` on the `Main` page ($P^{\texttt{Main}}_{\texttt{MC\_COMPLETE}}$) is marked by 1`e and the place `COUNTER` on the same page ($P^{\texttt{Main}}_{\texttt{COUNTER}}$) is marked by a token of integer carrying the value of 1`s. This can be specified by the following predicate:

$$\mathsf{SessionEnd}^s(M_i) = (\mathsf{Marking}(M_i, P^{\texttt{Main}}_{\texttt{MC\_COMPLETE}}) = 1\text{`}\texttt{e})\ \mathsf{AND}\ (\mathsf{Marking}(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}}) = 1\text{`}s)$$

where $M_i \in \mathbb{M}$ i.e. the set of reachable markings (states) of the PIEMCP CPN.

*Property 1 (**Multiple Conditions**).* When there is no attack:

- ϕ_1^{mc}: $\forall t_s \in T$, $\texttt{EV}(\mathsf{SessionEnd}^s)$;[2] and
- ϕ_2^{mc}: $\forall t_s \in T$, $\forall M$ such that $\mathsf{SessionEnd}^s(M)$=true, $\forall SP_i, SP_j \in P$, if $i \neq j$, then $\mathsf{Cond}(u_i^s) \neq \mathsf{Cond}(u_j^s)$.[3]

When an attack occurs:

- ϕ_3^{mc}: $\exists t_s \in T$, $\texttt{NOT}(\texttt{POS}(\mathsf{SessionEnd}^s))$.[4]

To verify this property, we use both ASK-CTL and session data analysis. In a normal environment (i.e. without an attack), ϕ_1^{mc} states that the end of the session is reachable. It can be directly queried using ASK-CTL formulas (see

Table 2. ASK-CTL and session-data queries for Multiple Conditions property

```
1  fun SessionEnd_1 n = Mark.Main'MC_COMPLETED 1 n = 1`() andalso Mark.Main'COUNTER 1 n = 1`1;
2  fun SessionEnd_2 n = Mark.Main'MC_COMPLETED 1 n = 1`() andalso Mark.Main'COUNTER 1 n = 1`2;
3  val MC_Phi1_1 = EV(NF("", SessionEnd_1));
4  val MC_Phi1_2 = EV(NF("", SessionEnd_2));
5  val sp1Rec1 = readSPRecord("sp1_sess1.txt");
6  val sp1Rec2 = readSPRecord("sp1_sess2.txt");
7  val sp2Rec1 = readSPRecord("sp2_sess1.txt");
8  val sp2Rec2 = readSPRecord("sp2_sess2.txt");
9  val cipherUCHVE11 = #cipherUCHVE(sp1Rec1);
10 val cipherUCHVE21 = #cipherUCHVE(sp2Rec1);
11 val cipherUCHVE12 = #cipherUCHVE(sp1Rec2);
12 val cipherUCHVE22 = #cipherUCHVE(sp2Rec2);
13 val MC_Phi2 = #label(cipherUCHVE11) <> #label(cipherUCHVE21) andalso
14              #label(cipherUCHVE12) <> #label(cipherUCHVE22);
15 val MC_Phi3_1 = NOT(POS(NF("", SessionEnd_1)));
16 val MC_Phi3_2 = NOT(POS(NF("", SessionEnd_2)));
17 val multipleConditions =
18 if not SP_ATTACK7 then (eval_node MC_Phi1_1 InitNode andalso
19 eval_node MC_Phi1_2 InitNode andalso MC_Phi2) else
20 (eval_node MC_Phi3_1 InitNode andalso eval_node MC_Phi3_2 InitNode);
```

[2] $\mathsf{SessionEnd}^s$ must eventually become true.
[3] Each SP holds a UCHVE ciphertext bound to a unique set of conditions.
[4] It is not possible to reach the end of an existing session.

line 1-4 in Table 2). By running two sessions, we have $s \in \{1, 2\}$. From the main page of the PIEMCP CPN, it is obvious that the session data is stored when `SessionEnd`^{s} becomes true in each session. Therefore, ϕ_2^{mc} (which formalizes the 'different conditions within a session clause') can be directly verified using the saved session data as shown in line 5-14 of Table 2. We have parameterized the model with one attack parameter that may compromise this property: `SP_ATTACK7` which depicts the scenario of SPs colluding to use the same condition string with the same user in a session. In such a scenario, we expect the protocol to behave in a fail-stop manner - therefore, ϕ_3^{mc} states that the protocol cannot reach the end of both sessions. ϕ_3^{mc} is directly translated into ASK-CTL queries as shown in line 15-16 of Table 2. Finally, after formulating the queries, we execute the formulas to check if all of the predicates hold (line 17-20).

4.2 Zero-Knowledge

When there are no attacks, before the revocation of a user's PII for an escrow session, IdP, SPs and referees must not learn the value of the user's PII but at the same time be convinced that its encryption is correct. When the attacks occur, it is expected that *at least* one of the encryptions is corrupted. For example, if a user-generated VE ciphertext is correct, the place `PKVE_RESULT` on the `PE` page is marked by 1`true`; otherwise, the place is marked by 1'`false`. Fig. 6 lists the predicates specifying the acceptance (i.e. conviction) or rejection of the user's encryption data by IdP and SPs while the user's PII is not revealed. Finally, an escrow session cannot be revoked until the user's encryption data has been all accepted. The place `REVOCABLE_SESSION` on the `Main` page records the revocable sessions in terms of session numbers. Thus, we define the predicate RevocableSession$^s(M_i) = 1`s \in \mathsf{Marking}(M_i, P^{\texttt{Main}}_{\texttt{REVOCABLE_SESSIONs}})$ over $\mathbb{M}$ which indicates if a session t_s has been revoked.

$\forall t_s \in T$:

- TrueUsrVE$^s(M_i)$ = (Marking$(M_i, P^{\texttt{PE}}_{\texttt{PKVE_RESULT}})$=1`true) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- FalseUsrVE$^s(M_i)$ = (Marking$(M_i, P^{\texttt{PE}}_{\texttt{PKVE_RESULT}})$=1`false) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- TrueUsrTPM$^s(M_i)$ = (Marking$(M_i, P^{\texttt{KE}}_{\texttt{PK_DAA_RESULT}})$=1`true) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- FalseUsrTPM$^s(M_i)$ = (Marking$(M_i, P^{\texttt{KE}}_{\texttt{PK_DAA_RESULT}})$=1`false) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- TrueUCHVEinKE$^s(M_i)$ = (Marking$(M_i, P^{\texttt{KE}}_{\texttt{TPM_PROOF_RESULT}})$=1`true) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- FalseUCHVEinKE$^s(M_i)$ = (Marking$(M_i, P^{\texttt{KE}}_{\texttt{TPM_PROOF_RESULT}})$=1`false) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- TrueUCHVEinMC$^s(M_i)$ = (Marking$(M_i, P^{\texttt{MC}}_{\texttt{TPM_PROOF_RESULT}})$=1`true) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)
- FalseUCHVEinMC$^s(M_i)$ = (Marking$(M_i, P^{\texttt{MC}}_{\texttt{TPM_PROOF_RESULT}})$=1`false) AND (Marking$(M_i, P^{\texttt{Main}}_{\texttt{COUNTER}})$=1`$s$)

where $M_i \in \mathbb{M}$ i.e. the set of reachable markings (states) of the PIEMCP CPN.

Fig. 6. List of predicates specifying acceptance or rejection of user's encryption data

*Property 2 (**Zero-knowledge**).* Without attacks:

- ϕ_1^{zk}: $\forall t_s \in T$, `EV`(TrueUsrVE^s) $\wedge$ `EV`(TrueUsrTPM^s) $\wedge$ `EV`(TrueUCHVEinKE^s) $\wedge$ `EV`(TrueUCHVEinMC^s);

- ϕ_2^{zk}: $\forall t_s \in T$, NOT(POS(FalseUsrVE^s)) $\wedge$ NOT(POS(FalseUsrTPM^s)) $\wedge$ NOT(POS(FalseUCHVEinKE^s)) $\wedge$ NOT(POS(FalseUCHVEinMC^s)); and
- ϕ_3^{zk}: $\forall t_s \in T$, FORALL_UNTIL(NOT(RevocableSession^s), ϕ_1^{zk}).

With attacks:

- ϕ_4^{zk}: $\forall t_s \in T$, NOT(POS(TrueUsrVE^s)) $\wedge$ NOT(POS(TrueUsrTPM^s)) $\wedge$ NOT(POS(TrueUCHVEinKE^s)) $\wedge$ NOT(POS(TrueUCHVEinMC^s)); and
- ϕ_5^{zk}: $\forall t_s \in T$, POS(FalseUsrVE^s) $\vee$ POS(FalseUsrTPM^s) $\vee$ POS(FalseUCHVEinKE^s) $\vee$ POS(FalseUCHVEinMC^s);
- ϕ_6^{zk}: $\forall t_s \in T$, NOT(POS(RevocableSession^s)).

For brevity, Table 3 only show queries related to TrueUsrVE^s, FalseUsrVE^s and RevocableSession^s (queries related to other predicates are performed in the same manner as for the first two predicates). ϕ_1^{zk}, ϕ_2^{zk} and ϕ_3^{zk} can be directly translated into ASK-CTL queries as shown in line 12-17 of Table 3. These three predicates are finally executed at line 18-19, where the zero-knowledge property of no attacks holds if all three predicates return true.

We have modeled six attacks that may compromise this property, which are parameterized as USER_ATTACK1, USER_ATTACK2, USER_ATTACK3, USER_ATTACK4, SP_ATTACK12, SP_ATTACK22. The formulas ϕ_4^{zk}, ϕ_5^{zk} and ϕ_6^{zk} are directly translated into ASK-CTL queries as shown in line 23-28 of Table 3. These predicates are executed at line 29-30 and all must return true if the zero-knowledge property with attacks is to hold.

```
1  fun TrueUsrVE_1 n = Mark.PE'PKVE_RESULT 1 n = 1`true andalso Mark.Main'COUNTER 1 n = 1`1;
2  fun TrueUsrVE_2 n = Mark.PE'PKVE_RESULT 1 n = 1`true andalso Mark.Main'COUNTER 1 n = 1`2;
3  ...
4  fun FalseUsrVE_1' n = Mark.PE'PKVE_RESULT 1 n = 1`false andalso Mark.Main'COUNTER 1 n = 1`1;
5  fun FalseUsrVE_2' n = Mark.PE'PKVE_RESULT 1 n = 1`false andalso Mark.Main'COUNTER 1 n = 1`2;
6  ...
7  fun RevocableSession_1 n = List.exists (fn y => y=1) (Mark.Main'REVOCABLE_SESSION 1 n);
8  fun RevocableSession_2 n = List.exists (fn y => y=2) (Mark.Main'REVOCABLE_SESSION 1 n);
9
10 NO ATTACKS (NA)
11 ==================
12 val ZK_Phi1 = eval_node EV(NF("",TrueUsrVE_1)) InitNode andalso ...
13              eval_node EV(NF("",TrueUsrVE_2)) InitNode andalso ...;
14 val ZK_Phi2 = eval_node NOT(POS(NF("",FalseUsrVE_1))) InitNode andalso ...
15              eval_node NOT(POS(NF("",FalseUsrVE_2))) InitNode andalso ...;
16 val ZK_Phi3 = FORALL_UNTIL(NOT(NF("",RevocableSession_1)), ZK_Phi1_1 initNode) andalso
17              FORALL_UNTIL(NOT(NF("",RevocableSession_2)), ZK_Phi1_2 initNode);
18 val zeroKnowledgeNA = eval_node ZK_Phi1 InitNode andalso eval_node ZK_Phi2 InitNode andalso
19                      eval_node ZK_Phi3 InitNode;
20
21 WITH ATTACKS (WA)
22 ===================
23 val ZK_Phi4 = eval_node NOT(POS(NF("",TrueUsrVE_1))) InitNode andalso ...
24              eval_node NOT(POS(NF("",TrueUsrVE_2))) InitNode andalso ...;
25 val ZK_Phi5 = eval_node POS(NF("",FalseUsrVE_1)) InitNode orelse ...
26              eval_node POS(NF("",FalseUsrVE_2)) InitNode orelse ...;
27 val ZK_Phi6 = eval_node NOT(POS(NF("",RevocableSession_1))) InitNode andalso
28              eval_node NOT(POS(NF("",RevocableSession_2))) InitNode;
29 val zeroKnowledgeWA = eval_node ZK_Phi4 InitNode andalso eval_node ZK_Phi5 InitNode andalso
30                      eval_node ZK_Phi6 InitNode;
```

Table 3. ASK-CTL and session-data queries for Zero-knowledge property

4.3 Enforceable Conditions

When PIEMCP is executed, a user's PII should never be revealed unless all designated referees agree that the cryptographically bound conditions are satisfied. This property should hold regardless of whether there is an attack or not. We define the following: $\forall t_s \in T$, $\forall M_i \in \mathbb{M}$,

- $\mathsf{HasRefPKVE}^s(M_i) = \mathsf{Marking}(M_i, P^{\texttt{Revocation}}_{\texttt{REF\_RECOVERED\_VE\_PRIVATE\_KEY}}) \neq \texttt{empty}$,
- $\mathsf{HasRecUsrPII}^s(M_i) = \mathsf{Marking}(M_i, P^{\texttt{Revocation}}_{\texttt{RECOVERED\_USER\_PII}}) \neq \texttt{empty}$,
- $threshold \in \{2..n\}$ specifies the minimum referees needed for a successful PII revocation, and
- $revCondition$ denotes the actual status of a revocation condition which is either true or false.

*Property 3 (**Enforceable Conditions**:)*

- ϕ^{ec}_1: $\forall t_s \in T$, $\texttt{NOT}(\texttt{POS}(\mathsf{HasRefPKVE}^s))$;
- ϕ^{ec}_2: if $revCondition$=`true` then $\forall t_s \in T$ where t_s is being revoked,
  - ϕ^{ec}_{2a}: $|P^{\texttt{Revocation}}_{\texttt{UCHVE\_PIECE\_DECRYPT\_SUCCESS}}| < threshold$ and
  - ϕ^{ec}_{2b}: $\texttt{NOT}(\texttt{POS}(\mathsf{HasRecUsrPII}^s))$;
- ϕ^{ec}_3: if $revCondition$=`false` then $\forall t_s \in T$ where t_s is being revoked,
  - ϕ^{ec}_{3a}: $|P^{\texttt{Revocation}}_{\texttt{UCHVE\_PIECE\_DECRYPT\_SUCCESS}}| \geq threshold$ and
  - ϕ^{ec}_{3b}: $\texttt{EV}(\texttt{POS}(\mathsf{HasRecUsrPII}^s))$.

While there are attacks that can be launched to compromise this property (parameterized by `SP_ATTACK4, REF_ATTACK1, REF_ATTACK2`, the above definition remains the same. Standard state space queries, and ASK-CTL queries are used to verify this property. ϕ^{ec}_1 states that the marking indicating illegal recovery of private VE key by the referees must not be reached *at any time*. When some conditions for session t_s are not fulfilled, the number of decrypted UCHVE pieces must be fewer than the threshold value required (ϕ^{ec}_{2a}), and that the marking which indicates the revelation of the user PII must not be reached too (ϕ^{ec}_{2b}). When conditions are fulfilled, we expect the number of decrypted UCHVE pieces to be greater or equal to the threshold value (ϕ^{ec}_{3a}), and that the user PII must eventually be revealed (ϕ^{ec}_{3b}).

In summary, the predicates ϕ^{ec}_1, ϕ^{ec}_{2b} and ϕ^{ec}_{3b} can be directly translated into ASK-CTL queries, while ϕ^{ec}_{2a} and ϕ^{ec}_{3a} are verified using the standard state space query `UpperIntegerBound` (i.e. the maximum number of tokens that can reside in a place). See [14] for details.

4.4 Conditions Abuse Resistant

During the execution of PIEMCP, an SP and an IdP must not be able to make the user encrypt the PII, or the VE private key, under a set of conditions different from those originally agreed. Similarly, an SP or IdP must not be able to successfully revoke the user's PII using conditions different from those originally agreed. In the following definition, we have used the notations shown in Fig. 5.

*Property 4 (**Conditions Abuse Resistant**).* No attack:

- ϕ_{1a}^{car}: $\forall t_s \in T$, $G^s = \mathsf{Cond}(v_{usr}^s)$ and
- ϕ_{1b}^{car}: $\forall t_s \in T$, $\forall SP_i \in P$, $C_i^s = \mathsf{Cond}(\overline{u}_i^s)$

With attacks: $\forall t_s \in T$, $\forall M_i \in \mathbb{M}$, let:

- $\mathsf{PE\_End}^s(M_i) = (\mathsf{Marking}(M_i, P^{\mathtt{Main}}_{\mathtt{PE\_COMPLETE}})=1\text{`}\mathsf{e})$ AND $(\mathsf{Marking}(M_i, P^{\mathtt{Main}}_{\mathtt{COUNTER}})=1\text{`}s)$
- $\mathsf{KE\_End}^s(M_i) = (\mathsf{Marking}(M_i, P^{\mathtt{Main}}_{\mathtt{KE\_COMPLETE}})=1\text{`}\mathsf{e})$ AND $(\mathsf{Marking}(M_i, P^{\mathtt{Main}}_{\mathtt{COUNTER}})=1\text{`}s)$

then:

- for attacks that manipulate the general conditions, ϕ_{2a}^{car}: $\mathtt{NOT}(\mathtt{POS}(\mathsf{PE\_End}^s))$;
- for attacks that manipulate conditions with SP_1, ϕ_{2b}^{car}: $\mathtt{NOT}(\mathtt{POS}(\mathsf{KE\_End}^s))$;
- for attacks that manipulate conditions with $\mathrm{SP}_{2,3,\ldots,y}$, ϕ_{2c}^{car}: $\mathtt{NOT}(\mathtt{POS}(\mathsf{SessionEnd}^s))$;
- for attacks that use wrong conditions for revocation,
  - ϕ_{2d}^{car}: Transition $T^{\mathtt{Revocation}}_{\mathtt{USE\_ATTACK\_CONDITIONS}}$ is not dead $\wedge$
  - ϕ_{2e}^{car}: $\mathtt{NOT}(\mathtt{POS}(\mathsf{HasRecUsrPII}^s)) \wedge \mathtt{NOT}(\mathtt{POS}(\mathsf{HasRefPKVE}^s))$.

In a normal environment (no attacks), ϕ_{1a}^{car} states that the cryptographically bound conditions (or label) used to produce a VE ciphertext must be the same as the one originally agreed. Similar explanation applies to ϕ_{1b}^{car}. When there are attacks targeting the general conditions used in the PE stage (parameterized by `USER_ATTACK1, SP_ATTACK1`, we expect the PE stage to fail stop (hence ϕ_{2a}). For attacks targeting the conditions used during the KE stage (with SP1 - parameterized by `USER_ATTACK4, SP_ATTACK11`), we expect the KE stage to fail stop (hence ϕ_{2b}^{car}). For attacks targeting the conditions used during the MC stage (for subsequent SPs - parameterized by `SP_ATTACK2`), we expect the MC stage to fail stop (hence ϕ_{2c}^{car}). For attacks targeting the use of invalid conditions during the revocation stage (parameterized by `SP_ATTACK3`, we expect that $T^{\mathtt{Revocation}}_{\mathtt{USE\_ATTACK\_CONDITIONS}}$ is not a dead transition (i.e. a transition that can never fire), and that the marking which indicate the revelation of user PII, or the illegal revelation of VE private key to *not* be reached (hence ϕ_{2d}^{car} and ϕ_{2e}^{car}).

In summary, ϕ_{1a}^{car}, ϕ_{1b}^{car}, and ϕ_{2d}^{car} can be verified using state space queries (notably the search nodes and token value comparisons queries). ϕ_{2a}^{car}, ϕ_{2b}^{car}, ϕ_{2c}^{car}, and ϕ_{2e}^{car} can be directly translated into ASK-CTL queries. See [14] for details.

5 Related Work

We briefly review several formal methods that have been used to verify security protocols. Earlier work, such as Burrows, Abadi, and Needham (BAN) logic [19], use the modal logic approach whereby the security of a protocol is assessed by studying the evolution of beliefs and/or knowledge over the course of the protocol to evaluate their adequacy for some pre-defined protocol objectives. We do not use this method because it is not evident if this approach is able to capture

and verify behavioral properties. Besides, the modal logic approach is generally considered a weaker approach in comparison to other formal methods [20].

Formal methods based on process algebra have also been used to model and verify security protocols (such as LySa [5] and CSP [21]). Process algebra allows the modeling of a system's behavior (including concurrency) as a set of algebraic statements. Common verification techniques used with process algebra include equational reasoning and model checking [22]. For example, Pi-Calculus [23] supports *labeled transition* semantics in modeling a system. This allows the verification of protocols through state exploration techniques such as model checking. However, we choose not to use process algebra approach due to its complexity which tends to (unnecessarily) complicate even simple things [17]. In comparison to the graphical-based modeling approach in CPN, this is a less intuitive approach to modeling large distributed systems such as PEPs. Model validation can only be performed by users who are experts in both the protocol itself *and* the process algebra syntax. While one still needs to understand the concept of CPN, the intuitive graphical-based modeling approach is a more human-friendly approach and thus easier to learn. The interactive and simulatable CPN model help modelers in detecting inconsistencies between a model and its original protocol specification, thus facilitating effective model validation.

State exploration techniques (such as state space analysis and model checking) have also been widely used for security protocol analysis. Examples of formal methods belonging to this category are the Automated Validation of Internet Security Protocols and Applications (AVISPA) tool [3], Scyther [24], and ProVerif [25]. These are state-of-the-art tools capable of automatically detecting attacks in many security protocols. Nevertheless, the main reason we do not use these tools is because the types of security properties verifiable by these tools are not relevant to PEPs. Instead, they are mostly relevant to authentication and key agreement protocols, i.e. *secrecy*, *authenticity*, and their variants. When protocols related to privacy are verified using these tools, the privacy property is reduced to confidentiality and authenticity. We argue that this is a simplistic approach to verifying privacy and that privacy does not simply equate to confidentiality and/or authenticity. The *behavior* of a protocol in preserving/violating a user's privacy is just as important. As stated in the introduction, it is the behavior of the protocol that we are interested to verify.

Similar to process algebra, these tools also lack the rich graphical and simulation support of CPN.[5] Finally, it is not evident if concurrent processing (as oppose to concurrent protocol execution supported in Scyther) is supported in these tools. Therefore, we do not find these tools to be suitable for our purpose. Although CPN has been widely used to analyze industrial communication protocols (such as Transmission Control Protocol (TCP)), its use in the area of security protocols is still very new with limited documented cases. For example, Al-Azzoni et al [4] used CPN to model and verify the Tatebayashi, Matsuzaki, and Newman (TMN) key exchange protocol [26]. The main difference between

[5] Scyther provides some static graphical support. However, it falls short of interactive protocol simulation and graphically-driven protocol specification.

our work and theirs is that they focus on verifying the *secrecy* property of the TMN protocol, while our work focus on verifying the security behavior of PEPs.

Our work has not reached the maturity of the other methods discussed. However, we see its potential. By exploiting the intuitiveness of CPN's graphical-based modeling approach (which is also based upon a solid underlying mathematical foundation) in combination with its rich modeling capability (hierarchical modeling, concurrent processing, flexible colour sets definition, model parameterization, etc), performance analysis capability, and its powerful verification techniques, CPN has the potential to be an easy-to-use yet powerful formal method for modeling and verifying large multi-party PEPs.

6 Conclusion

We have shown that CPNs can be used to model complex PEPs using CPN and verify its behavioral properties using state space analysis, ASK-CTL (model checking language), and session data analysis. We have also proposed several modeling techniques, notably the cryptographic primitive abstraction (capturing complex primitives and zero-knowledge proof protocol), TPM provable execution, parameterized attacks, and session data capture. We have also shown how a set of behavioral properties can be formalized which can be directly verified using the existing state space, ASK-CTL, and session data queries.

Future work involves using the model to conduct a performance analysis of the protocol and to assess its efficiency in deployment. We will also be looking at refining and generalizing the modeling techniques proposed in this paper such that they can be applied to other PEPs. CPN Tools can be improved by providing a better user front-end to simplify and automate the tasks required in the modeling and verification of PEPs. The function of such a front-end could be as simple as aiding users with the configuration of the model parameters, to a full-blown automation whereby a user without any knowledge of CPN can generate the required back-end CPN model with only a PEP specification. As to the issue of attack behavior, so far we have considered in the protocol verification a set of known attack scenarios, while ultimately the goal is to achieve an automated attack detections for PEPs using the CPN-based approach.

References

1. Holtzman, D.H.: Privacy Lost. Jossey-Bass (2006)
2. WP 14.1: PRIME (Privacy and Identity Management for Europe) - Framework V3 (March 2008)
3. Team, T.A.: AVISPA v1.1 User Manual. Information Society Technologies Programme (June 2006), `http://avispa-project.org/`
4. Al-Azzoni, I., Down, D.G., Khedri, R.: Modeling and verification of cryptographic protocols using coloured petri nets and design/cpn. Nordic Journal of Computing 12(3), 201–228 (2005)
5. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. J. Comput. Secur. 13(3), 347–390 (2005)

6. Koblitz, N., Menezes, A.: Another look at provable security. J. Cryptology 20(1), 3–37 (2007)
7. Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, 2nd edn. Monographs in Theoretical Computer Science, vol. 1. Springer, Berlin (1997)
8. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
9. Gilmore, S.: Programming in standard ml 1997: A tutorial introduction. Technical report, The University of Edinburgh (1997)
10. Jensen, K., Kristensen, L.M., Wells, L.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. STTT 9(3-4), 213–254 (2007)
11. Suriadi, S., Foo, E., Smith, J.: Private information escrow bound to multiple conditions. Technical report, Information Security Institute - Queensland University of Technology (2008), http://eprints.qut.edu.au/17763/1/c17763.pdf
12. Suriadi, S., Foo, E., Josang, A.: A user-centric federated single sign-on system. Journal of Network and Computer Applications 32(2), 388–401 (2009)
13. Suriadi, S., Foo, E., Smith, J.: A user-centric protocol for conditional anonymity revocation. In: Furnell, S.M., Katsikas, S.K., Lioy, A. (eds.) TrustBus 2008. LNCS, vol. 5185, pp. 185–194. Springer, Heidelberg (2008)
14. Suriadi, S., Ouyang, C., Smith, J., Foo, E.: Modeling and verification of privacy enhancing security protocols. Technical report, ISI, Queensland University of Technology (April 2009) http://eprints.qut.edu.au/20088/
15. McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: an execution infrastructure for TCB minimization. In: Sventek, J.S., Hand, S. (eds.) EuroSys, pp. 315–328. ACM, New York (2008)
16. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–207 (1983)
17. van der Aalst, W.: Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype. In: BPTrends, pp. 1–11 (May 2005)
18. Christensen, S., Mortensen, K.H.: Design/CPN ASK-CTL Manual - Version 0.9. University of Aarhus, Aarhus C, Denmark (1996)
19. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. 8(1), 18–36 (1990)
20. Meadows, C.: Open issues in formal methods for cryptographic protocol analysis. In: DISCEX 2000, pp. 237–250. IEEE Computer Society Press, Los Alamitos (2000)
21. Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
22. Baeten, J.C.M.: A brief history of process algebra. Theor. Comput. Sci. 335(2-3), 131–146 (2005)
23. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, Cambridge (1999)
24. Cremers, C.J.: The scyther tool: Verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
25. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, pp. 82–96 (June 2001)
26. Tatebayashi, M., Matsuzaki, N., Newman Jr., D.B.: Key distribution protocol for digital mobile communication systems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 324–334. Springer, Heidelberg (1989)