Analysis of Object-Specific Authorization Protocol (OSAP) using Coloured Petri Nets

Younes Seifi[1,2] **Suriadi Suriadi**[1] **Ernest Foo**[1] **Colin Boyd**[1]

[1]Queensland University of Technology (QUT)
Brisbane, Australia
Email: younes.seifi@isi.qut.edu.au

Email: {s.suriadi,e.foo,c.boyd}@qut.edu.au

[2]Bu-Ali Sina University, Hamadan, Iran

Abstract

The use of Trusted Platform Module (TPM) is becoming increasingly popular in many security systems. To access objects protected by TPM (such as cryptographic keys), several cryptographic protocols, such as the Object Specific Authorization Protocol (OSAP), can be used. Given the sensitivity and the importance of those objects protected by TPM, the security of this protocol is vital. Formal methods allow a precise and complete analysis of cryptographic protocols such that their security properties can be asserted with high assurance. Unfortunately, formal verification of these protocols are limited, despite the abundance of formal tools that one can use. In this paper, we demonstrate the use of Coloured Petri Nets (CPN) - a type of formal technique, to formally model the OSAP. Using this model, we then verify the authentication property of this protocol using the state space analysis technique. The results of analysis demonstrates that as reported by Chen and Ryan the authentication property of OSAP can be violated.

Keywords: Coloured Petri Nets; CPN; CPN/Tools; security analysis; TPM, OSAP; Trusted Computing

1 Introduction

CPN is a type of formal method introduced in (Jensen; 1992, 1994, 1997) as a graphical language to model and analyze systems. The mathematical foundation of CPN provides well-defined semantics which facilitates the unambiguous and precise modeling of a system and its properties. Yet, the graphical modeling interface of CPN makes it a user-friendly and arguably easy to use and understand. CPN models are also executable and to create these executable models, specifications must be complete. Through the process of model creation, execution, and simulation, protocol designers may detect flaws and errors in the protocol design, and may subsequently improve the correctness of the protocol.

Most importantly, however, is that CPN as a general purpose formal modeling tool can generate the state space information of the model. This state space can then be used by standard state space analysis techniques (such as computational tree logic - CTL) to verify various system properties, both standard properties (such as liveliness and boundedness) and verifier-defined properties (such as security-related properties). Formal analysis of a protocol using CPN can be aided with a tool known as the CPN Tools (Jensen et al.; 2007). These tools automate many of the tasks required to model, simulate, and analyze systems and protocols.

CPN has been widely used (Jensen et al.; 2007) as a language to model and validate systems like communication protocols, software and engineering systems. Practical implementations of using CPN in business process modeling, manufacturing systems, agent system and workflow modeling are available now. A number of usages of CPN in modeling and analyzing cryptographic protocols are introduced in previous works section.

The Object Specific Access Protocol (OSAP) is a cryptographic protocol defined as part of the trusted computing (TC) platforms. This protocol governs how one can access the Trusted Platform Module (TPM)'s protected objects (such as cryptographic keys). Given the importance and sensitive of objects protected by TPM, it is important that we analyze it precisely such that we can be confident of its security. As Trusted computing (TC) platforms are expected to be an important component of a secure computing paradigm, any breach in the OSAP protocol will have a major impact on the TC platforms in general.

The use of formal methods to assert, with high assurance, the security of cryptographic protocols has been an active research area in the last few decades. Through formal analysis, errors in famous protocols like the Needham-Schroeder Public Key (NSPK) (Lowe; 1995), have been found 17 years after it was introduced. This has made the use of automated tools for protocol verification more evident. Such an analysis is normally aided with formal tools, including AVISPA (Viganò; 2006), CASPER (Lowe; 2002), ProVerif (Blanchet; 2001), Hermes (Bozga et al.; 2003), NRL protocol analyzer (Meadows; 1996), Isabelle (Paulson; 1994), PRISM (Kwiatkowska et al.; 2004), Athena (Song; 1999), Securify (Cortier; 2003) and Scyther (Cremers; 2008) to verify security properties.

Unfortunately, formal analysis of TPM-related protocols, such as the OSAP, is limited. As more methods, approaches and tools are used to analyze TPM protocols, the more confidence one can gain regarding the security, the reliability, and trustworthiness of these protocols.

The main contribution of this paper is the demonstration of the applicability of Coloured Petri Nets (a

type of formal methods) in the modeling and verification of a representative TPM protocols, namely the OSAP protocol.

In this paper, the hierarchical approach of CPN is used to model one session of the OSAP. From this model, a state space is generated and is then used to analyze the authentication property of OSAP. In particular, a number of states representing the violation of the authentication property are first defined. Then, state space analysis and CTL logics are used to verify whether the violation conditions defined earlier can occur in the state space. Our formal verification of OSAP arrives at the same conclusion as reported by Chen and Ryan (Chen and Ryan; 2009): that the authentication property of OSAP can be violated.

This paper is structured as follows. Section 2 introduces the OSAP protocol, the concept of CPN modeling, state space analysis and ASK-CTL (a dialect of CTL supported by CPN Tools). The modeling of the OSAP using CPN Tools is then described in Section 3. Section 4 shows how the authentication property of OSAP is defined and how ASK-CTL verifies the satisfaction of this property. Section 5 discusses related work, followed by the conclusion in Section 6.

2 Preliminaries

This section discusses trusted computing, authorization protocols of TPM, CPN modeling, state space analysis and ASK-CTL. Trusted Computing (TC) is a new technology that is used in security systems. Trusted computing is defined by Trusted Computing Group (TCG) as a computer system for which an entity inside the system is responsible for supervision of system behavior to be the same as what is predicted for it (TCG; 2007). One of the main creations of TCG's efforts is the Trusted Platform Module(TPM) chip which is to be used for system supervision. TPM chip adds "roots of trust" (TCG; 2007) into computer platform to establish a chain of trust.

Access to roots of trust is governed by the use of authorization protocols. They provide access to the TPM secrets. These protocols are one of the fundamental protocols of trusted computing that are used before other protocols, to check whether the user process is eligible to have access to the TPM secrets or not. These protocols are illustrated in more detail in next section.

2.1 Authorization protocols

Authorization protocols or TPM command validation protocols are one of the most important categories of protocols defined by TCG. TCG enforces all commands to the TPM that affect security, privacy or reveal platform secrets to be authorized. Authorization is based on a secret provided by the caller as a part of the command.

It is possible that different authorization sessions connect to one TPM. For each session a unique session identifier, unique nonce for each end point, a hash digest for messages which have been sent or received and an ephemeral secret, used to tie message exclusively to a specific object or to encrypt message traffic if necessary, will be allocated.

These sessions are established to provide authorized access to the TPM. Any entity which decides to participate in an authorization session must provide a pass-phrase which is used to authorize and authenticate it. The pass-phrase, authorization secret or Attestation Identity Key (AIK) is a 160-bit value which is ideally random and non-guessable. The size of this secret is the same as the size of a SHA-1 operation result. After hashing secrets, salts and any other values, the result will be a fixed sized value, called authorization data (authData) (Chen and Ryan; 2009).

Authorization data can be associated with any TPM object, TPM command, TPM command interface or TPM itself. An authorized session between the caller and TPM before creating the new authData is created. Authorization protocols have been designed in a manner that never relies on the security properties of communication protocols. When TPM is communicating with other user processes it always assumes they are un-trusted in relation to itself (TCG; 2007). There are different authorization protocols. In the next section OSAP which is used in this research is illustrated.

Object-Specific Authorization Protocol (OSAP)

OSAP is a challenge and response protocol used by the TPM object caller to demonstrate its knowledge of authorization data. This protocol is used to provide access to just one type of TPM object. A sample usage of this protocol that asks TPM to create a key is illustrated in (Chen and Ryan; 2009). Figure 1 from the same source demonstrates the protocol sequences. In this figure a name for each message exchange is considered. The names of exchanges 1 to 4 are *Exchange#1*, *Exchange#2*, *Exchange#3* and *Exchange#4*. To design the CPN model, the processes *'Process TPM_OSAP'*, *'Process TPM_CreateWrapKey message'* and *'Process TPM_OSAP response'*, *'Process TPM_CreateWrapKey(...) Response'* are added to the TPM and user side. The defined operation for these processes is extracting input parameters and storing them in designated places for future usage.

1. In the first step, the user process sets up an OSAP session. The goal of this step is to request TPM to *create* a key based on a preloaded key in the TPM named the parent key. The handle of the parent key is *pkh* (parent key handle) and *ad(pkh)* is its authorization data. Both *pkh* and *ad(pkh)* are included in the TPM_OSAP command and are sent to the TPM.

2. TPM, after receiving the TPM_OSAP command, generates n_e, n_e^{osap} and assigns a new session authorization handle *ah*. These new items are sent to the user as the response.

3. The TPM and user process calculate the shared secret. Calculation of shared secret is done using hmac algorithm. The input arguments of the hmac algorithm are *ad(pkh)*, n_e^{osap} and n_o^{osap}.

4. The user process calls the TPM_CreateWrapKey function. The *ah*, *pkh*, n_o and *newauth* are sent by this function to the TPM. To protect *newauth*, it is XORed with $SHA1(S, n_o)$.

5. When this command is received, the TPM checks the HMAC and creates the new key. Then private key and new *authData* are put in an encrypted (using s as the key) package. The encrypted package and public key are put in *keyblob*. The keyblob is returned by the TPM and is authenticated with an HMAC. The hmac is created with n_o and $n_o^{'}$ nonces and is keyed on S.

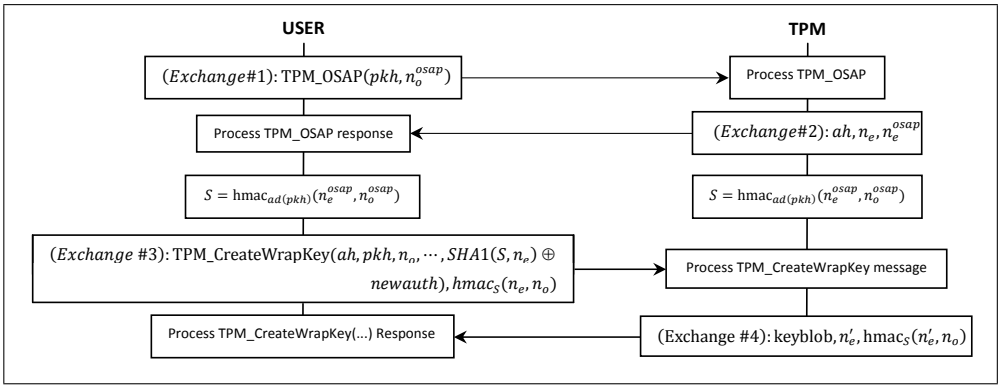

Figure 1: OSAP sequence diagram

6. The encrypted package is decrypted and authdata will be retrieved from it. If the received authdata is not the same as new created authdata by user then the protocol will be terminated. Otherwise protocol will be continued and ends normally.

This protocol can be modeled using different tools. The *'Modeling OSAP using CPN'* section illustrates the usage of CPN for modeling mentioned steps. The next section will describe CPN modeling.

2.2 CPN modeling

The CPN models are depicted as graphical drawings composed of places, transitions, arcs and inscriptions(written text in the CPN ML programming language). *Places* are shown using circles and ellipses. *Transitions* shown by rectangles describe actions. The transitions and places are connected to each other using arrows called *arcs*. For any arc, an *arc inscription* can be written in CPN ML language. Input arc inscriptions are used to define the binding of tokens from input places to transition. The output arc inscriptions are used to define tokens that will be put into the output place of a transition. A place can have zero or more *tokens* of the colour set of the place. For each of these tokens a data value from a given *type* has been considered. The data value of each token is called the *token colour*. The set of all the tokens that can exist in a place is defined as its *colour set*. The colour set of each place is written below the place using an inscription. The value of each variable specifies its *binding*. The number of tokens and their colours in all the individual places specifies the *marking* of the CPN model. The number of tokens in just one place and their colours specify the marking of that place. Most of the times next to each place another inscription except its colour set is written that determines the *initial marking* of place.

For each transition, a pair consisting of transition and binding of all the variables of transition is called the *binding elements*. It is possible to consider special inscriptions named *guards* for transitions. These inscriptions are boolean expressions that when they are evaluated to true the transition can be enabled. Otherwise even if all the input tokens are provided the transition can not be enabled.

Tokens do not move between pages of the CPN model, rather, pages are connected through special places which are marked as either an input, an output, or an input/output socket. The place that constitutes the interface through which one page exchanges tokens with the others is an *input/output port*. The *input sockets* are the input places of substitution transitions, while their output places are *output socket*. The other method of moving tokens between different pages is *fusion set*. Fusion sets glue a number of places in one or more CPN pages together. They all create a compound place across the model.

A sample CPN model is shown in Figure 2. In this model the defined colour set for *Sender* and *Receiver* is *STRING*. This colour set like a variable in programming languages allows tokens with the type of *STRING* to be stored. The initial marking of place *Sender* is 1`"ONE" ++ 1`"SAMPLE". Variable *vs* with string colour set is used to move the token between place and transition. This place is connected to transition *'Send Packet'*. This transition when guard, *[vs="ONE"]* is evaluated to TRUE and required input tokens are provided by input places, can be enabled. In the model of Figure 2, for the first token, *[vs="test"]* guard is TRUE and *'Send Packet'* is enabled. When this transition is enabled its border becomes thicker than when it is disabled. In the next step the token is moved to the *'Receiver'* place. For the second token of place *Sender*, because the guard *[vs="ONE"]* is evaluated to FALSE, the *Send* transition will not be enabled and this token remains in its place. The final marking of this model is shown in Figure 3.

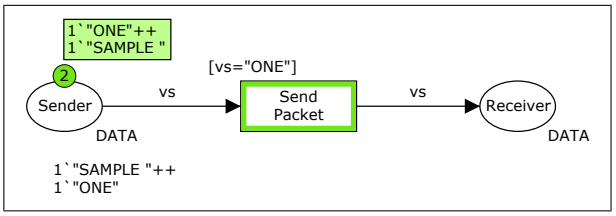

Figure 2: Sample CPN model and its initial marking

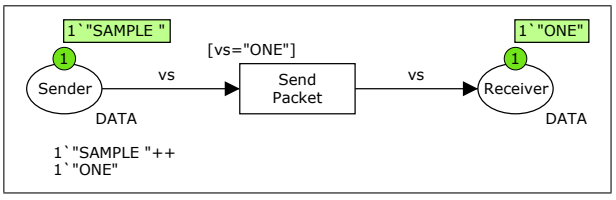

Figure 3: Final marking of sample CPN model

2.3 State Space Analysis

Simulation of a CPN model analyzes a finite number of executions. This helps validate the model by de-

tecting and finding errors in the CPN model. It can demonstrate the model is working correctly. However, it is impossible to guarantee the correctness of a model with 100% certainty because all the possible executions are not covered (Jensen et al.; 2007).

A full state space generation (Occurrence Graph-OG, reachability graph/tree) calculates all possible executions of the model. It calculates all reachable markings and binding elements of the CPN model. The result is represented in a directed graph where its nodes are a set of reachable markings and the arcs correspond to the occurring binding elements.

Occurrence sequence describes different occurring steps and the reached *intermediate markings* to execute a CPN model. If a marking via an occurrence sequence is reachable and it starts from the initial marking then it is called a *reachable marking* (Jensen et al.; 2007).

In most cases after producing all states the *Strongly Connected Component Graph (SCC-graph)* is generated. The SCC-graph nodes are subgraphs called *Strongly Connected Components(SCC)*. Disjoint division of the nodes in the state space creates the SCC. This division is in a manner that two state space nodes are in the same SCC if and only if they are mutually reachable. This means that a path exists in the state space from the first node to the second node and vice versa. The structure of the SCC-graph can provide information about the behavior of the model (Jensen et al.; 2007).

State space analysis or model checking is mainly used for model based verification of concurrent systems. It is applied successfully in many formal models as the analysis method. *State space explosion* is its main limitation. The CPN models should be designed carefully to prevent state space explosion. This research uses the CPN/Tools to create and analyze the model.

2.4 The CPN/Tools ASK-CTL

State spaces analysis tools usually provide a number of standard properties such as liveliness that can be evaluated. However, all the required verification properties are not included in these tools. Temporal logics like CTL are able to reason about certain facts based on model's state (Cheng et al.; 1997). CTL provides a model of time such that its structure is like a tree. In this structure the future is not determined and different paths can occur in the future. Any of the branches might be an actual path that is realized. Software applications like model checkers use CTL in formal verification of hardware or software artifacts.

ASK-CTL is an extension of CTL (Clarke et al.; 1986) temporal logic implemented in CPN/Tools. This extension takes into account both the state information and arc information. The ASK-CTL statement is interpreted over the state space of the Coloured Petri net model. Then the model checker of CPN/Tools checks the formula over the state space and defines whether it is true or false. Complete information about the ASK-CTL can be studied in (Christensen and Mortensen; 1996).

This research uses ASK-CTL to verify the authentication property of the OSAP protocol. Using ASK-CTL formula to verify the CPN model, will ensure that all the specific verified properties are valid in a specific marking of the model. Otherwise, any part of the criteria of the verified property can be valid in different markings that other conditions of property may be incorrect.

3 Modeling OSAP using CPN

To create the CPN model of the OSAP protocol and verify its authentication property the following steps are considered:

1. A CPN model for the protocol and intruder is designed and implemented. This stage consists of a number of steps including:
   (a) Identifying all the participating entities of the protocol and modeling them in the CPN modeling tool (for this research CPN/Tools)
   (b) Designing and implementing required colour sets, variables, ML functions and CPN pages.
2. Validating the CPN model using simulation to ensure that the model behaves as specified in the standard.
3. Calculate the state space
4. Validate the authentication property using ASK-CTL.

In the first step a CPN model for the OSAP protocol and intruder is designed and implemented. To design the CPN model at first three different entities, user, TPM and intruder are identified. The designed colour set for the entities are shown in the Appendix A.

To store all the knowledge of intruder a special database, using *csINTDB* colour set, is designed. The detailed information about this DB is illustrated in 3.2 sub-section.

To prevent state space explosion problem *csSEQ* colour set is designed. This colour set is used to write specific guards for transitions. More information about this mechanism is provided in 3.1.2 subsection. Variables that are used in the CPN model are shown in the Appendix A.

Modeling an OSAP protocol and verifying its authentication property needs a special intruder model. This model is illustrated in section 3.3. The CPN model of intruder in *Exchange#1* and *Exchange#2* is similar to intruder in *Exchange#3* and *Exchange#4* . Thus in this paper just the first two intruder models are described.

After illustrating the necessary colour sets and intruder model the main page of the CPN model is illustrated in *'OSAP CPN model'* section. This section describes how different substitution transitions are enabled and run. More detailed information about all the modules, substitution transitions, variables, ML-functions and other parts of CPN model are available.

CPN model verification using simulation is conducted in CPN/Tools. It is shown that the model is operating based on its definitions. When the state space calculation finishes within a reasonable time we knew that state space explosion has not occurred. Running the ASK-CTL formula verifies the authentication property.

3.1 Managing state space

The state space of the CPN model takes a long time to be created and the model can suffer from state space explosion problem in the absence of optimization techniques. To prevent these problems, we include two techniques to mitigate the state space explosion problem: model parameterization and sequence-token mechanism.

3.1.1 Model Parameterization

To prevent the state space explosion problem, we parameterize the model via two boolean parameters: the *vinc_int* and the *vexcl_tpms*. Assigning a true/false value to *vinc_int* causes the OSAP model to include/exclude the intruder model. In the former case, it represents the original protocol without intruder consideration. Assigning a true/false value to *vexcl_tpms* will cause the model to bypass/include the TPM in the OSAP session.

These two variables can split the state space SCC to smaller SCCs. The smaller SCC can be calculated faster. The state space explosion problem does not happen during the computation of this model. The smaller SCC is a subset of complete SCC. Therefore if any authentication violation condition be found in it, the whole model SCC contains that violation condition.

3.1.2 Sequence token mechanism

The OSAP CPN model without any optimization has the problem of state space explosion. When the number of occurrence graph nodes of state space increases significantly this problem occurs. To prevent this issue the number of occurrence graph states should be reduced. This reduction can be done by using other tools that implement more optimized state space analysis algorithms such as (Westergaard et al.; 2009) or by improving the CPN model to make it more efficient preventing its useless states to be enabled.

In this research the latter approach is chosen to reduce the number of states and prevent state space explosion. Al-Azzoni in (Al-Azzoni; 2004) proposes a token passing mechanism to prevent concurrent runs of different transitions. To implement this method a token moves from one transition to the next transition. This approach in complicated models with various pages creates difficulties in managing tokens and connecting arcs, because additional arcs and places are required to move tokens.

To make this approach more manageable, in our model, a specific colour set named *csSEQ* and a special fusion set named *GF_seq*, that is accessible by all pages, is defined. This colour set determines which CPN page is or will be the active page at any given time.

The current active page is always stored in the *GF_seq* fusion set and the *CSI* place (a place located in all the model pages) becomes accessible for the transitions in the page. When the transition fetches the sequence token from the *CSI* place it can either change it to determine the next active page or does not change it to stay in the current page. For example in page *U2*, Figure 4, the current page of the model is fetched from the *GF_seq* fusion set using the *CSI* place and the *vseq* variable. When the value of *vseq* is equal to *'user2'* the process *'Process TPM_OSAP Response'* transition can be enabled.

The next page that should be run after the *U2* page, based on OSAP session main page, Figure 7, is *U3*. Thus, the *'Process TPM_OSAP Response'* transition changes the current sequence value to *'user3'* and stores its token in *GF_seq* fusion set using fusion place *CSO*. This approach is followed for all pages.

This method prevents concurrent runs of protocol in the current model. However, if the analyzed property is violated in one session it will be violated in a number of concurrent sessions. To analyze parallel sessions this approach can be extended by adding the identifier of other session(s) transitions to the *csSEQ* colour set. Moreover, at the end of running one CPN page, the CPN model randomly assigns one of the identifiers of all the pages that can be run in parallel to the sequence token.

3.2 Intruder's database

The intruder model contains a database (DB) which stores all the intruder knowledge. This database is a location to accumulate all the sent and received messages through the intruder. It also stores the initial knowledge of the intruder. The colour set *csINTDB* is designed for this purpose.

The initial values of the intruder's database only contain those values that are assumed to be publicly known, which include the *parent key handle - pkh* and the corresponding authorization data *ad(pkh)*.[1]

3.3 Intruder model

The intruder model of OSAP is based on the Dolev-Yao approach (Yao; 1983). The Dolev-Yao model considers the intruder as the medium that transfers messages. It can edit, remove, forward, duplicate and create new messages. In other words, it acts as a man in the middle who can modify messages between the user and the TPM, or it can bypass the TPM altogether (see Figure 5 and Figure 6 for illustration).

The OSAP intruder can store any sent and received message in its database. For each whole message, the message and each of its fields are stored in the database. After that either a whole message is fetched from database and is sent to TPM or user, or a new message is created by fetching its fields from the intruder's database. When a new message is created, the retrieved fields from the DB can be created by the intruder or they can be stored in the DB, as parts of previous messages, during previous exchanges.

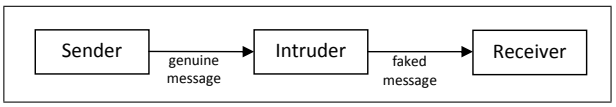

Figure 5: Sent message is changed by the intruder

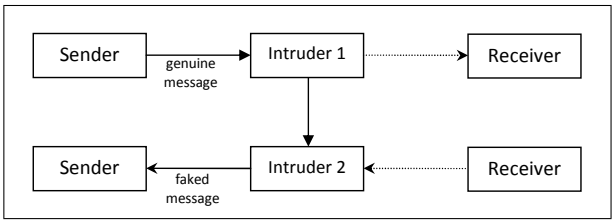

Figure 6: Intruder has bypassed receiver

A more detailed explanation of our intruder model and our modeling approach is provided in Appendix B.

3.4 OSAP CPN model

After designing colour sets, variables and required functions, based on the shown protocol in Figure 1 the main page of the CPN model is designed, Figure 7. The OSAP protocol is composed of four different exchanges. In any exchange, TPM and the

[1] This assumption is consistent with the formal model shown previously in (Chen and Ryan; 2009).

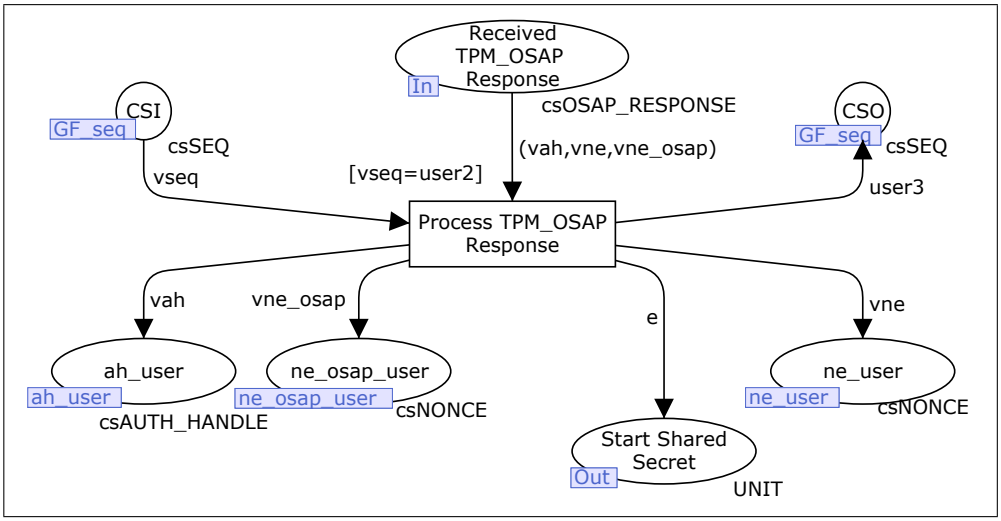

Figure 4: Page U2 module of the OSAP CPN model

user are either the sender or receiver, whilst the intruder acts as both the sender and receiver. The protocol is started from the user and it finally ends with the user. To make the model more readable and to simplify the modeling process, a hierarchical CPN model is proposed in Figure 7. The first substitution transition of this model is used to create the $TPM\_OSAP(pkh, n_o^{osap})$ message. This message is sent to the TPM. However, the intruder can intercept this message. The intruder, using the *Intruder_1* substitution transition, is able to send the original or faked message toward TPM or user. If it sends the message to the TPM, because of the specific format of the message in Figure 7, it can be received only by the *'Process TPM_OSAP'* substitution transition. If the message is sent to the user again, this new message should be created by intruder. The *Intruder_2* substitution transition is the only transition that can do this, thus the method of message movement is changed from the Figure 5 approach to the Figure 6 approach.

When the Figure 5 approach is chosen the message is processed by the *'Process TPM_OSAP'* substitution transition. Then the message *Exchange#2* and shared secret *S* are created by *'Send TPM_OSAP Response'* and *'Create Shared Secret TPM'* substitution transitions. The result will be sent toward user. *Intruder_2* is able to intercept the message *Exchange#2*. It can send the faked message to the user or TPM again. However, because sending new message directly from *Intruder_2* to *Intruder_3* and then to the TPM does not affect the analysis of authentication property no path between *Intruder_2* and *Intruder_3* is created.

The *'Process TPM_OSAP Response'* after processing the message, creates the shared secret. Then *TPM_CreateWrapKey(...)* generates the *Exchange#3* and sends it to the TPM. What happens for this message is the same as *Exchange#1*. It is intercepted by *Intruder_3*. Then will be forwarded to either TPM and will be processed by *'Process TPM_CreateWrapKey message'*, or the *Intruder_4* and will be replaced by a faked message. In the former case *'Send TPM_CreateWrapKey Response'* will be executed as the next step. In the latter case after the *Intruder_4*, *'Process TPM_CreateWrapKey(...) Response'* is executed and the protocol will be ended.

In *Exchange#1* the role of *Exclude Intruder 1* and *Include Intruder 1* transitions is to produce planned configuration for OSAP. When *Intruder_1* is excluded, the *Exclude Intruder 1* transition is enabled, intruder is bypassed and *TPM_OSAP* message moves from *Sent TPM_OSAP message 1* to the *Received TPM_OSAP message* place. Including *Intruder_1* in the model enables *Include Intruder 1* transition and moves the *TPM_OSAP* message toward *Intruder_1*. To implement the required configuration of CPN model equivalent places and transitions are considered in *Exchange#2*, *Exchange#3* and *Exchange#4*.

At the start of a protocol a token with a colour set of *csSEQ* and the colour of *user1*, is stored in the place *'Start Session 1'*. This colour determines that *TPM_OSAP(pkh, no_osap)* is the first substitution transition that is enabled. This token during the simulation and analysis moves from one transition to the other and specifies the sequence of the protocol run. It is the token that is used to implement sequence token mechanism.

4 Verification of the model using state-space

In this research the CPN/Tools state space is used to evaluate the authentication property of OSAP protocol. To evaluate the authentication property a CPN model is created for OSAP. To validate this model it is simulated without intruder. The completion of the model during the simulation with correct results demonstrates the validity of the model. To verify the authentication property, we firstly define several formal notations, predicates, and operator that we will need to use. Then, we formalize a condition (in an ASK-CTL statement) whose fulfillment will violate the authentication property of the OSAP protocol. We then execute the statement to verify if the authentication property can be violated.

The designed CPN model in this research checks the authentication property of OSAP protocol. To verify this property the violation conditions are defined, then the CPN model investigates whether they are fulfilled or not.

A simple way to demonstrate the violation of the authentication property is by demonstrating the ability of an intruder to complete the OSAP protocol successfully (that is, with the user accepting the new session authorization data at the end of the protocol without even involving the TPM whatsoever in the process.

In other words, in our model, the authentication

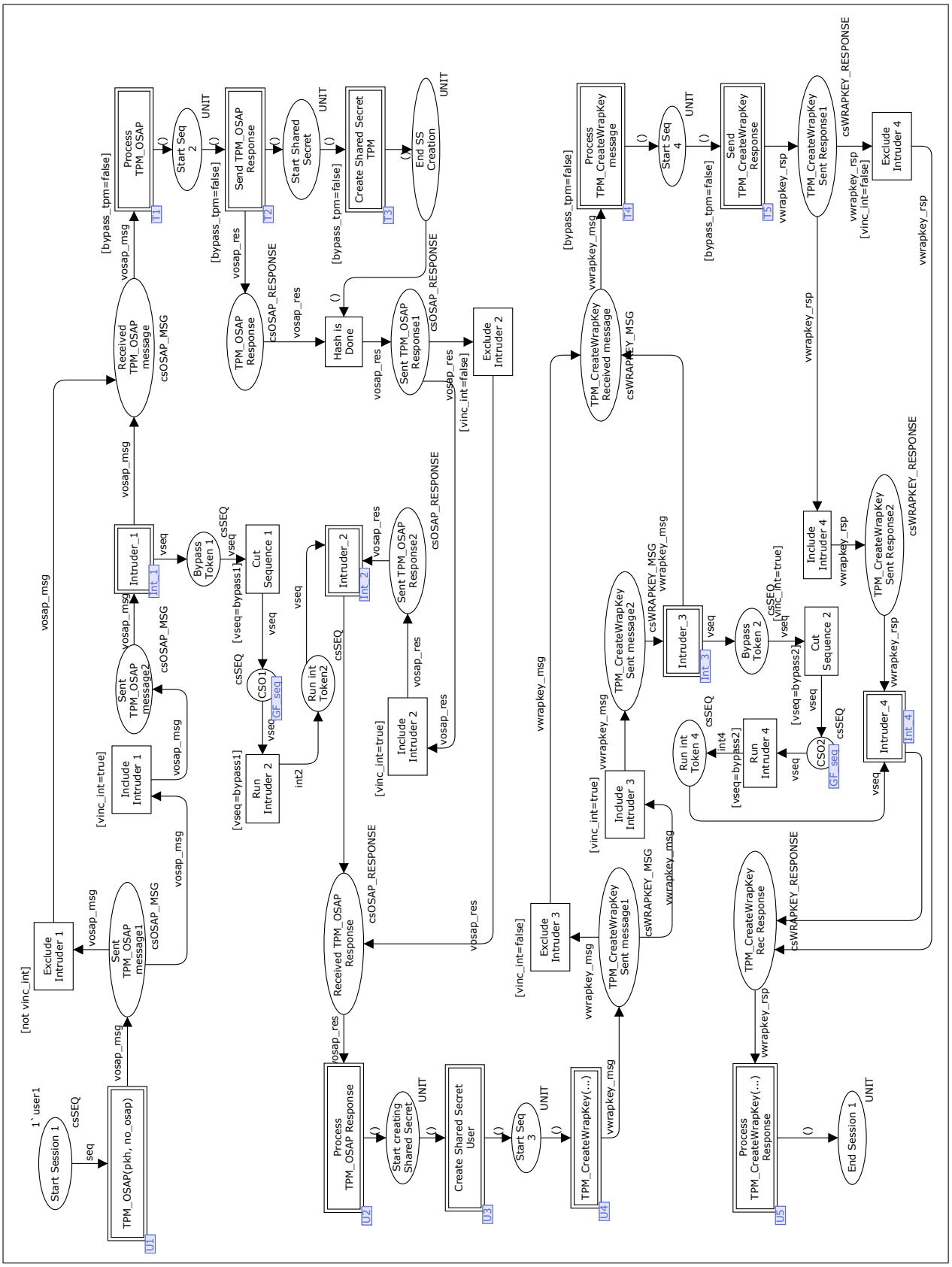

Figure 7: main page of OSAP protocol CPN model

property of the OSAP protocol is violated if the intruder intercepts the message during during *Exchange#1* and *Exchange#3* and does not forward the message to the TPM; instead, the *Intruder_2* and *Intruder_4* modules are executed following the interception of the messages (from the user) during *Exchange#1* and *Exchange#3* respectively.

We can formalize the authentication violation condition by using the ASK-CTL statement. To do so, we define the following notations and predicates:

- let $\mathbb{M}$ be the set of all reachable marking of the OSAP CPN model,
- M_0 be the initial marking of the OSAP CPN model,
- $[M_0\rangle$ be the set of all reachable markings from M_0,
- $P^{OSAP\_Session}_{Received\_TPM\_OSAP\_message}$ be a CPN *place* with the name of *Received_TPM_OSAP_message* on the CPN page called *OSAP_Session*,
- $\mathsf{Marking}(M_i, P^{OSAP\_Session}_{Received\_TPM\_OSAP\_message})$ represents the set of tokens at the CPN place $P^{OSAP\_Session}_{Received\_TPM\_OSAP\_message}$ at a marking M_i where $M_i \in M_0\rangle$,
- $\mathsf{M}_{NoOSAPMsg} = \{M_i | M_i \in [M_0\rangle \wedge |\mathsf{Marking}(M_i, P^{OSAP\_Session}_{Received\_TPM\_OSAP\_message})| == \emptyset$ be a set of markings representing the situation whereby no initial OSAP message (that is, message from user to the TPM in *Exchange#1*) is received by the TPM,
- $\mathsf{M}_{NoCreateWrapKeyMsg} = \{M_i | M_i \in [M_0\rangle \wedge |\mathsf{Marking}(M_i, P^{OSAP\_Session}_{TPM\_CreateWrapKey\_Received\_message})| == \emptyset$ be a set of markings representing the situation whereby no create wrap key message (that is, message from user to the TPM in *Exchange#3*) is received by the TPM, and
- $\mathsf{M}_{EndSuccess} = \{M_i | M_i \in [M_0\rangle \wedge |\mathsf{Marking}(M_i, P^{OSAP\_Session}_{End\_Session\_1})| > 0$ be a set of markings representing the situation whereby the OSAP session was completed and *accepted by the user* as successful.

The main ASK-CTL operator we use to formalize the violation condition of the authentication property is the `EXIST_UNTIL(F1,F2)` operator (`F1` and `F2` are boolean formula). This operator returns *true* if there exists *a path* whereby `F1` holds in *every marking* along the path from a given marking (e.g. M_0) until it reaches another marking whereby `F2` holds.

Having described the above notations, predicates, and operator, we can now formally assert that the authentication property of the OSAP protocol is violated if, from M_0, the following ASK-CTL statement

- `EXIST_UNTIL`$[(\mathsf{M}_{NoOSAPMsg} \wedge \mathsf{M}_{NoCreateWrapKeyMsg}), \mathsf{M}_{EndSuccess}]$

returns true.

Results: we have generated the state space of our OSAP model and we have executed the above ASK-CTL statement. Our model shows that the above ASK-CTL statement is true which means that the authentication property of the OSAP protocol *does not hold*.

5 Previous works

Coloured Petri Nets have been used by (Doyle et al.; 1997) for analyzing cryptographic protocols. They have modeled each legitimate protocol entity and intruder using Petri Net Objects(PNO). Intruder can do a variety of actions. Ultimate goal of the analysis is to determine whether protocol can withstand intruder attacks and actions or not. The large number of attacks that intruder may pursue makes hand analysis impossible. The Prolog is used for analysis in this research. This research provides a model for handset authentication protocol used in CT2 and CT2Plus wireless communication protocols and analyzes them.

The Station-to-Station (STS) security protocol is analyzed in (Aly and Mustafa; 2003) using CPN. Aly and Mustafa use CPN to model all the protocol objects and intruder. They deduce describing protocol entities and its attacker using CPN provides a solid foundation for protocol analysis. However, other analysis approaches do not offer these features.

Al-Azzoni in (Al-Azzoni et al.; 2005) has used a hierarchical CPN model to analyze TMN key exchange protocol. The proposed approach at first models TMN entities. The intruder CPN model is designed and added to the protocol model in the next step. The Design/CPN tool is used to analyze the created model. Concept of DB-place is introduced to simplify representation of the intruder's knowledge. This concept is used in this research to design the DB of intruder. Al-Azzoni uses the application of the token passing scheme to resolve the problem of state space explosion that during the simulation in Design/CPN occurs. This research is based on Al-Azzoni's approach. Moreover, a current state token mechanism is used to determine current page of the model that should be run using fusion sets. In this mechanism a guard is added to transitions of a nominated page. This guard enables a transition just when container page of transition is the active page of model.

6 Conclusion and Future Works

The goal of this research is to analyze the OSAP protocol using CPN. The results of the analysis show that authentication property of this protocol can be violated. This model is designed based on assumptions from (Chen and Ryan; 2009). The analysis can be completed by different assumptions to study the protocol in more detail.

The approach used can be applied to other security properties such as secrecy. Analyzing other properties would require some refinements in the model to add the required places, transitions and colour sets. It is necessary to write new ASK-CTL formulas to validate results. It is even possible to use CPN for defining new security properties and analyze them to investigate new problems. This goal can not be easily achieved using specific purpose security analysis tools. However, the process of analyzing the same property for general purpose tools such as CPN is more time consuming than specific purpose tools. To make the modeling time as fast as possible new modules, libraries and constructs should be added to the CPN.

The designed intruder model based on Dolev-Yao approach can be replaced by other models. However, this replacement needs significant changes. Such changes would require effort and time. Because the Dolev-Yao attacker model is a powerful and popular

attacker model used in analyzing protocols, change is not recommended.

The OSAP protocol is a part of trusted computing protocols. As mentioned earlier, one of the advantages of using CPN for modeling is its ability to compose different models. This makes CPN a solution for combining OSAP with other trusted computing protocols. The combined model can then be analyzed.

The main disadvantage of using CPN in modeling is its firm connection with protocol structure. In the created model, any inconsiderable change in protocol and its message structure can cause significant changes in the CPN model. This leads to inevitable cascaded changes in the CPN model. However, this firm connection helps designers to be more familiar with the protocol specifications. Specifications can be compared with their implementations, to investigate whether they are compliant with each other or not. The other drawback of using CPN is state space explosion during state space analysis. Unfortunately, this issue can not be predicted before ending the protocol design.

As future work CPN can be used for modeling Session Key Authorization Protocol (SKAP), proposed in (Chen and Ryan; 2009), Digital Rights Management (DRM) and other protocols that their specification and prospected analyzed security property is compatible with CPN capabilities.

References

Al-Azzoni, I. (2004). *The verification of cryptographic protocols using coloured petri nets*, Master's thesis, Department of Software Engineering.

Al-Azzoni, I., Down, D. G. and Khédri, R. (2005). Modeling and verification of cryptographic protocols using coloured petri nets and *esign/cpn*, *Nord. J. Comput.* **12**(3): 200–228.

Aly, S. and Mustafa, K. (2003). Protocol verification and analysis using colored petri nets, *Depaul University. July* .

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules, *CSFW*, number 0-7695-1146-5, IEEE Computer Society, pp. 82–96.

Bozga, L., Lakhnech, Y. and Périn, M. (2003). Hermes: An automatic tool for verification of secrecy in security protocols, *Computer Aided Verification*, Springer, pp. 219–222.

Chen, L. and Ryan, M. (2009). Attack, solution and verification for shared authorisation data in tcg tpm, *Formal Aspects in Security and Trust*, pp. 201–216.

Cheng, A., Christensen, S. and Mortensen, K. (1997). *Model checking coloured petri nets exploiting strongly connected components*, Citeseer.

Christensen, S. and Mortensen, K. (1996). *Design/CPN ASK-CTL Manual*.

Clarke, E., Emerson, E. and Sistla, A. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **8**(2): 244–263.

Cortier, V. (2003). A guide for securify.

Cremers, C. (2008). The scyther tool: Verification, falsification, and analysis of security protocols, *Computer Aided Verification*, Springer, pp. 414–418.

Doyle, E., Tavares, S. and Meijer, H. (1997). *Automated security analysis of cryptographic protocols using coloured petri net specifications.*, Master's thesis, Queen' s University at Kingston.

Jensen, K. (1992). Coloured petri nets. basic concepts, analysis methods and practical use. basic concepts, *Springer, Berlin* **vol. 1.**

Jensen, K. (1994). Coloured petri nets. basic concepts, analysis methods and practical use. analysis methods, *Springer, Berlin* **vol. 2**.

Jensen, K. (1997). A brief introduction to coloured petri nets, *in* E. Brinksma (ed.), *TACAS*, number 3-540-62790-1 in *Lecture Notes in Computer Science*, Springer, Enschede, The Netherlands, pp. 203–208.

Jensen, K., Kristensen, L. and Wells, L. (2007). Coloured petri nets and cpn tools for modelling and validation of concurrent systems, *International Journal on Software Tools for Technology Transfer (STTT)* **9**(3): 213–254.

Kwiatkowska, M., Norman, G. and Parker, D. (2004). Prism 2.0: A tool for probabilistic model checking, *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, IEEE, pp. 322–323.

Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol, *Information processing letters* **56**(3): 131–133.

Lowe, G. (2002). Casper: A compiler for the analysis of security protocols, *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, IEEE, pp. 18–30.

Meadows, C. (1996). The nrl protocol analyzer: An overview, *J. Log. Program.* **26**(2): 113–131.

Paulson, L. (1994). *Isabelle: A generic theorem prover*, Springer.

Song, D. X. (1999). Athena: A new efficient automatic checker for security protocol analysis, *CSFW*, pp. 192–202.

TCG (2007). Tcg specification architecture overview revision 1.4.

Viganò, L. (2006). Automated security protocol analysis with the avispa tool, *Electr. Notes Theor. Comput. Sci.* **155**: 61–86.

Westergaard, M., Evangelista, S. and Kristensen, L. M. (2009). Asap: An extensible platform for state space analysis, *in* G. Franceschinis and K. Wolf (eds), *Petri Nets*, number 978-3-642-02423-8, Springer, Paris, France, pp. 303–312.

Yao, A. (1983). On the security of public key protocols, *IEEE Transactions on Information Theory* **29**(2): 198–208.

```
01 colset csTERMS = with null | ah | ahi |
                no_osap | ne | ne_osap |
                ne_osap1 | no | ne1 | ni1 |
                pkh_pub |  pkhi | keyblob |
                keyblobi | ad_pkh_pub |
                newauth | authdatai;
02 colset csATTACK = with posattack | negattack;
03 colset csSEQ = with user1 | user2 |
                user3 | user4 | user41 | user42 |
                user43 | user5 | int1 | int2 | int3 |
                int4 | intx3 | tpm1 |tpm2 | tpm3 |
                tpm4 | tpm5 | bypass1 |
                bypass2 | endses | terminateses;
04 colset csAUTH_HANDLE = subset csTERMS with [ah,ahi];

05 colset csNONCE = subset csTERMS with [no_osap,
                ne, ne_osap, no, ne1, ni1];

06 colset csPUBKH = subset csTERMS with
                [pkh_i, pkh_pub];
07 colset csPubKey = subset csTERMS with [pub_key];
08 colset csAUTH_DATA = subset csTERMS with
                [ad_pkh_pub,newauth,authdatai];
09 colset csOSAP_MSG = product
                csPUBKH * csNONCE;
10 colset csOSAP_RESPONSE = product
                csAUTH_HANDLE * csNONCE * csNONCE;
11 colset csSHARED_SECRET = product
                csAUTH_DATA * csNONCE * csNONCE;
12 colset csKEYBLOB = product
                csSHARED_SECRET * csAUTH_DATA;
13 colset csXOR_OUTPUT = product
                csSHARED_SECRET * csNONCE *
                csAUTH_DATA;
14 colset csHMAC_OUTPUT = product
                csSHARED_SECRET * csNONCE * csNONCE;
15 colset csWRAPKEY_INPUT = product
                csAUTH_HANDLE * csPUBKH *
                csNONCE * csXOR_OUTPUT;
16 colset csWRAPKEY_MSG = product
                csWRAPKEY_INPUT * csHMAC_OUTPUT;
17 colset csWRAPKEY_RESPONSE = product
                csKEYBLOB * csNONCE *
                csHMAC_OUTPUT;
18 colset csINTDB = union
                fipkh : csPUBKH +
                finonce : csNONCE +
                fiah : csAUTH_HANDLE +
                fixor_output : csXOR_OUTPUT +
                fihmac_output : csHMAC_OUTPUT +
                fikeyblob : csKEYBLOB +
                fiosap_msg : csOSAP_MSG +
                fiosap_res : csOSAP_RESPONSE +
                fiwrapkey_msg : csWRAPKEY_MSG +
                fiwrapkey_rsp : csWRAPKEY_RESPONSE +
                fiwrapkey_input : csWRAPKEY_INPUT +
                fiss : csSHARED_SECRET +
                fiauthdata : csAUTH_DATA;
```

Figure 8: List of CPN model colour sets

A Colour Set Definition

The colour sets and variables definition used in the OSAP model are detailed in Figure 8 and Figure 9 respectively. Understanding these colour sets and variables is preliminary in learning how Figures 4, 7, 10 and Figure 11 models work. The design method of CPN models can be used to model and analyze other protocols.

B Details of the Intruder Model

In the OSAP CPN model, Figure 7, the behavior of the intruder varies in different exchanges. For the first exchange the intruder model operation is based on what was illustrated for Figure 6. In the second message exchange the intruder's role is like Figure 5, but *Intruder_2* is not always enabled by the TPM. Sometimes, when *Intruder_1* has bypassed the TPM, *Intruder_2* can start its operation from the transition that should create a new message. In the third sequence, the *Intruder_3* acts exactly the same as *Intruder_1*. It can either send a faked message directly to the TPM or bypass the TPM and ask *Intruder_4* to create a faked message and send it to the user. *Intruder_4* accomplishes the same operations of *Intruder_2* for different input message colour sets.

```
val vinc_int = true;
val vexcl_tpms = true;
var e : UNIT;
var vseq, vseq1, vseq2 : csSEQ;
var tmpstr : STRING;
var vosap_res : csOSAP_RESPONSE;
var vne, vne1, vnonce1,
    vnonce2 , vne_osap, vno,
    vne_osap1, vno_osap : csNONCE;
var vah : csAUTH_HANDLE;
var vosap_msg : csOSAP_MSG;
var vauthdata, vnewauth : csAUTH_DATA;
var vss: csSHARED_SECRET;
var vxor_output : csXOR_OUTPUT;
var vwrapkey_input,
    vwrapkey_output : csWRAPKEY_INPUT;
var vwrapkey_msg : csWRAPKEY_MSG;
var vwrapkey_rsp : csWRAPKEY_RESPONSE;
var vhmac_output, vhmac_user,
    vhmac_tpm : csHMAC_OUTPUT;
var vkeyblob : csKEYBLOB;
var vpkh, vpkhu, vkh : csPUBKH;
```

Figure 9: List of model variables

The main goal of the intruder's CPN model is to verify the authentication property. When this property is violated the intruder can bypass the TPM or it can fake messages that are sent from TPM to the user. Because of the first situation (intruder can bypass the TPM) a connection between *Intruder_1* and *Intruder_2* and another connection between *Intruder_3* and *Intruder_4* is created.

To introduce functionalities of the intruder, the next two sections illustrate the CPN model of *'Intruder_1'* and *'Intruder_2'* in more detail.

The *Intruder_1* functionality

The input token of *Intruder_1* (Figure 10, page *Int_1* of OSAP CPN model) substitution transition is stored in the *tmp_storage* place. Then the *'store message parts in DB'* transition stores each of its fields, *pkh* and *no_osap*, in the intruder's database, *fipkh* and *finonce*, fields respectively. This transition is enabled when the current sequence token, coming from the *CSI* place and always stored in *GF_seq* global fusion set, is equal to *int1*. The considered guard for transition, *[vseq=int1]*, is used to enable the transition. To prevent this transition from being enabled more than once, the *'JO1'* place, holding just one token, is connected to the transition. At the end of this transition the sequence token is moved to the *'ST1'* place and will enable the *'Store Whole message in DB'* transition.

The *'Store Whole message in DB'* transition stores the token of *'Sent TPM_OSAP'* place in *fiosap_msg* field of intruder's database. This token at the start of the *Intruder_1* page was stored in *'Tmp_echg1'* fusion set. The functionalities of *'JO2'* and the *[vseq=int1]* guard are the same as the equivalent place and guard for the *'store message parts in DB'* transition. At the end of this transition the sequence token will move to the *'ST2'* place. At *ST2* based on the model configuration the next steps of the model will be determined. If TPM is excluded([vexcl_tpm andalso vinc_int] is evaluated to TRUE) then intruder does

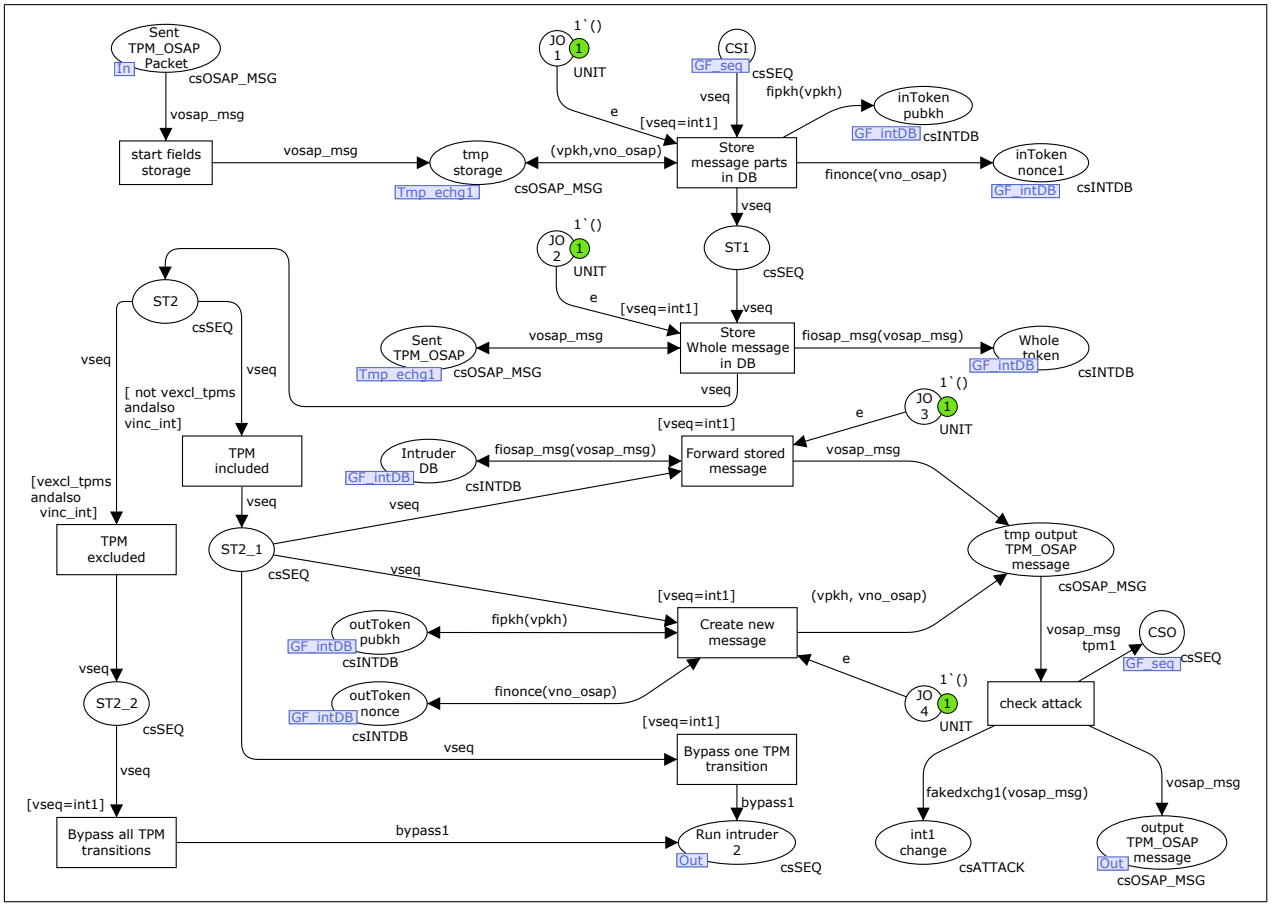

Figure 10: CPN model of *Intruder_1* module

not create any message and after bypassing TPM enables *Intruder_2*. Including TPM in the model moves sequence token to the *ST2_1* place. At this time three different transitions can be enabled.

First, the TPM can be bypassed by enabling the *'Bypass TPM'* transition. This transition moves the sequence token to the *'Run intruder 2'* place. This makes *intruder_2* enabled and none of the TPM transitions in the TPM-related pages will be enabled.

Second, the *'Forward stored message'* transition can be enabled. A token from the intruder's database is fetched by *'Intruder DB'* place and then it is stored in the *'tmp output TPM_OSAP message'* place. This token after been checked by *'check attack'* transition will be sent to the receiver.

Third, the *'Create new message'* transition can be enabled. After that all the required tokens are fetched from the intruder's database to compose a new message. The new token is again checked by the *'check attack'* transition and then will be sent to the TPM.

The *Intruder_2* functionality

The CPN model of *Intruder_2* substitution transition is shown in figure 11. The name of CPN page of *Intruder_2* in figure 7 is *Int_2*. The input token moves from *OSAP_Session* to the *Int_2* page using *'Sent TPM_OSAP Response'* input port. To make the model simpler the *Tmp_xch* fusion set is created. This fusion set easily provides access to the OSAP response message. The *'Store Whole message in DB'* and *'Store message parts in DB'* transitions functionality is the same as the corresponding transitions in *Int_1* page.

The difference between *Intruder_1* and *Intruder_2* is in the method of getting the token by *ST2*. In *Intruder_1* the stored sequence token in this place only comes from the previous transition of the *Int_1* page. However, in *Intruder_2* this token can come either from the *OSAP_Session* page or from the *'Store message parts in DB'* transition in page *Int_2* . When the input token of *ST2* comes from *OSAP_session* page the TPM is bypassed by *Intruder_1* and then the *Intruder_2* substitution transition is enabled immediately after *Intruder_1*. Bypassing TPM means that there is no OSAP message sent from TPM to the user. Thus *Intruder_2* does not need to store any message or its parts in the intruder's database. The functionality of *Intruder_2* starts from the *ST2* place. In this place based on the model configuration if TPM is excluded from the model an intruder required token is inserted into the intruder database. Otherwise the sequence token moves to the *ST3* place. In *ST3* place either the *'Forward stored message'* or *'Create new message'* substitution transition will be enabled. Whether the former transition is enabled or the latter, their operation is the same as the corresponding transitions in the *Int_1* page. The transitions and places located between *ST3_1* to *ST3_3* provide the sequential access of intruder to the *finonce* field of intruder database to prevent racing condition deadlock.

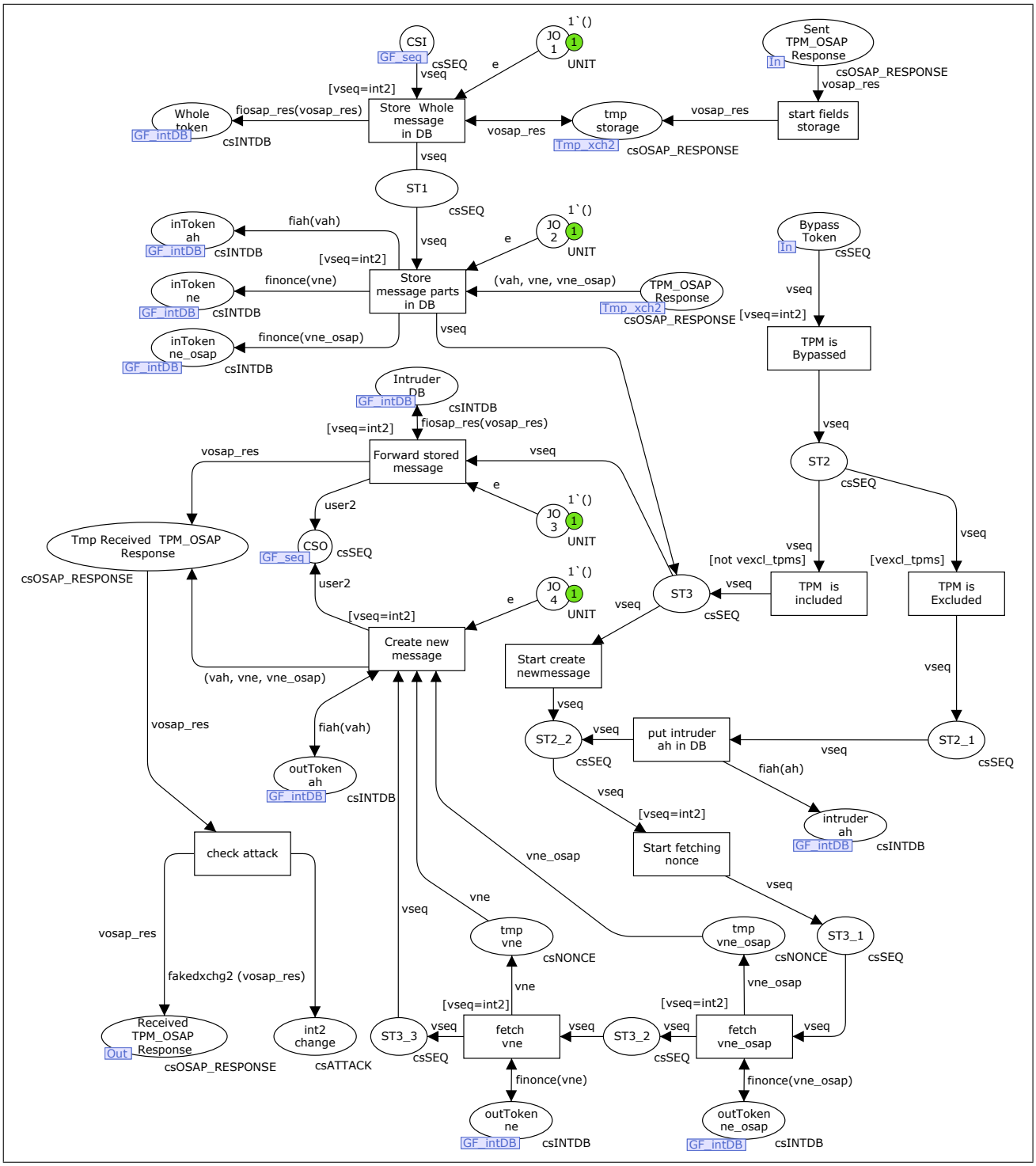

Figure 11: CPN model of *Intruder_2*