
© Copyright 2011 The Authors.
Analysis of Object-Specific Authorization Protocol (OSAP) using Coloured Petri Nets-Version 1.0

Younes Seifi, Suriadi Suriadi, Ernest Foo, and Colin Boyd
Queensland University of Technology (QUT)
Brisbane, Australia
{younes.seifi@student.qut.edu.au, s.suriadi,e.foo,c.boyd}@qut.edu.au

Abstract. Analyzing security protocols is an ongoing research in the last years. Different types of tools are developed to make the analysis process more precise, fast and easy. These tools consider security protocols as black boxes that can not easily be composed. It is difficult or impossible to do a low-level analysis or combine different tools with each other using these tools. This research uses Coloured Petri Nets (CPN) to analyze OSAP trusted computing protocol. The OSAP protocol is modeled in different levels and it is analyzed using state space method. The produced model can be combined with other trusted computing protocols in future works.

Keywords: Coloured Petri Nets, CPN, CPN/Tools, security analysis

1 Introduction

Security protocols have been an ongoing research in the last years. Errors found in famous protocols like Needham-Schroeder Public Key (NSPK)[19], 17 years after it was introduced, have made usage of automated tools for protocol verification more evident. Tools like AVISPA[26], CASPER[20], ProVerif[4], Hermes[5], NRL protocol analyzer[21], Isabelle[23], PRISM[18], Athena[24], Securify[10] and Scyther[11] create a formal model to verify security properties.

In [22] a number of emerging issues in protocol analysis is listed. One of them is high fidelity. Techniques usually behave with algorithms like black boxes that a number of algebraic properties are included in them. However, many security problems arise at lower levels of abstraction. They can come from interactions of the protocol with crypto system, problems with other functions such as hash functions.

The other issue is composability. Most of the analysis’s of cryptographic works concentrate on the analysis of protocols that can be described by a single sequence of messages and there is no loop or choice points. This problem is not just a theoretical concern. The reported attack in [3] on an early version of SSL is based on it.
Both of the problems can be resolved using CPN modeling. The CPN models can be created in different levels of abstraction. Hierarchical mechanism of CPN is an effective method to design top-down or bottom-up models. Moreover, CPN can model a combination of protocols or security properties of a specific protocol.

The CPN model can be extended to involve different levels of defined security properties. They have the advantage of expandability as well. For example, it is possible to combine secrecy definition in a security protocol with the definition of security in communication components of the system. General purpose modeling can help verification of different properties at the same time in a system. This feature in new systems like trusted computing that different components are interconnected to each other over a network, makes the analysis possible.

General purpose verification method such as Coloured Petri Nets (CPN) and a tool like CPN/Tools. General modeling methods like CPN can model verifier-defined system properties including security ones. CPN model can illustrate system operations obviously. Its simulation tools provide trace and simulation functionalities to check the correctness of the model. Finally, state space tools and logics like ASK-CTL make the system verification possible.

Coloured Petri Nets is a graphical language used for design, specification and simulation of variety of systems. It is specially used for the systems that communication, resource sharing and synchronization are important in them. Its typical examples of usage area are automated production systems, embedded systems, communication protocols, VLSI chips and workflow analysis[15]. Whilst Petri Net is used to provide required notations for modeling communication, synchronization, resource sharing and concurrency, Standard ML (SML) is used to provide required primitives of data type definition and data manipulation required for created Petri Net model[16]. CPNs combine functional programming language Standard ML with Petri Net to afford both of their features. Detailed illustration of Standard ML, Petri nets and CPN is beyond this report. The main objects of the CPN are illustrated in the CPN modeling section.

Graphical modeling languages like CPN has the advantage of graphical user-interface. The interface can demonstrate the system clearly. Moreover, the behavior of each CPN can be defined unambiguously using well defined semantics of CPN. The offered hierarchical structure of the system by CPN is a powerful tool to model large systems. The CPN can model time concepts that is not offered by many of security analysis tools. A large number of state space creation, analysis and reduction algorithms are designed for CPN models that makes analyzing large CPN models possible.

This research investigates how CPN can be used to model and analyze a security protocol in different levels. The hierarchical approach of CPN is used to model a session of OSAP protocol in the highest level. Then any substitution transition has been modeled to create models with more detail. To a number of substitution transitions three levels of abstraction has been designed. In the lowest level of abstraction the authentication property is defined for the OSAP protocol. Then state space analysis is used to evaluate this property. The ASK-
CTL logic is used to validate the results and to investigate whether they are valid in all the state space time slots or not.

After the introduction the background section introduces authorization protocol, CPN modeling, state space analysis and ASK-CTL. Modeling OSAP using CPN section illustrates how OSAP is modeled using CPN in CPN/Tools. Section four shows how the model can be verified using state-space, how authentication property is defined and how ASK-CTL can validate the results and authentication property.

2 Background

Trusted Computing (TC) is a new technology that will be the core of most of the security systems in the next years. Trusted computing is defined by Trusted Computing Group (TCG) as a computer system for which any entity inside the system is responsible for supervision of system behavior to be the same as what is predicted for it. The Trusted Platform Module (TPM) chip is invented by TCG to this aim. Trusted computing is considered by [17] as a technology trying to answer two main questions:

1. Which software is running on a remote computer (remote attestation): For example Microsoft can check which Microsoft programs have been installed on a computer running in Australia.
2. How insurance can be provided that just using a specific software stack access to stored secret is possible? (sealed memory)

Trusted Computing Group (TCG) has been working from spring of 2003 to create the building blocks of trusted platform. The result of these efforts is the creation of a Trusted Platform Module(TPM) chip and its related standards. TPM chip as the main outcome of the TCG adds “roots of trust” into computer platform to establish a chain of trust (discussed in the next section). There are currently three different roots of trust considered by TCG:

1. Root of Trust for Measurement (RTM): a computing engine that can compute a measurement for all software on a system by hashing them.
2. Root of Trust for Storage (RTS): a secure storage that can store RTM values.
3. Root of trust for Reporting (RTR): a reliable mechanism to report values stored by RTS to other entities.

To provide access to these roots of trust different protocols are designed. One of the most important type of them are authorization protocols. They provide access to the TPM secrets. They will be illustrated in more detail in the next section. 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. This makes the analysis of them more important than the other types of protocols.
coloured Petri Nets is introduced in [13–15] as a graphical language to model and analyze concurrent systems. In [16] CPN has been considered 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. The complexity of modern system makes using CPN for system modeling important. The CPN provides a visual model of system behavior which makes formal analysis of it possible.

Kurt Jensen in [16] introduces insight, completeness and correctness as benefits of creating a model. Creating a model for a system helps developers to be more familiar with the system and leads to new insights into various aspects of the system. CPN models are executable and to create these executable models, specifications must be completely understood and they must be complete as well. Creating this model helps designers to find gaps in definitions.

Executing a model and simulating it several times helps detect flaws and errors by designers and to use results to improve correctness of the model. CPN is used by variety of researchers in different fields of research. This popularity provides more support for CPN and makes the modeling difficulties less. To implement CPN models different tools can be used. The CPN/Tools has been used to implement CPN models of this research, previous model in this report.

CPN/Tools is a free tool that its license can be obtained free of charge for academic and commercial usage. This tool can simulate behavior of modeled system. It is suitable for evaluating different properties of model using state-space method or simulation-based performance analysis. A quick survey of using this tool can be studied in [16].

2.1 Authorization protocols

Authorization protocols or TPM command validation protocols are one of the most important categories of protocols defined by TCG in [25]. 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. There are a number of commands that do not need to be authorized.

The entities in the computer platform can submit TPM specific API functions to the TPM to be executed. These entities are processes, threads and embedded controllers. To send a command a secure channel between the TPM and entity will be created [25]. 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 has 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 as 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).

Authorization data can be associated with any TPM object, TPM command, TPM command interface or TPM itself. Before creating authData an authorized session between the caller and TPM is created. Any message in an authorized session consists of three different parts: message container, TPM command and session state. Message container identifies message type, size and its format. TPM command contains command name, I/O parameters and return code of command. The last part, session state, is storing session ID, control flag and digest values of messages in the session.

TPM and caller before moving to any next step of protocol confirm validity of message. To prevent replay attack a nonce is sent with each message. The number of concurrent sessions is left as an implementation decision. However, it is mandated by TCG Core Services (TCG), that this number must not be less than three sessions. Moreover, any exchanged message between TPM and caller must be atomic and before processing any request accepting other requests by TPM is impossible. Authorization protocols have been designed in a manner that never relies on security properties of communication protocols. When TPM is communicating with other user process it always assumes them as un-trusted in relation to itself[25].

Object-Specific Authorization Protocol (OSAP) OSAP is a challenge and response protocol used by TPM object caller to demonstrate its knowledge about authorization data. This protocol is used to provide access to just one type of TPM object whilst Object-Independent Authorization Protocol (OIAP), another TPM authorization protocol, can be used to admit requests for different type of objects. A sample usage of this protocol that asks TPM to create a key is illustrated in [6]. Figure 1 from the same source demonstrates the protocol sequences.

**Fig. 1. OSAP sequence diagram**
1. In the first step, the user process sets up an OSAP session. The goal of this step is asking TPM to create a key based on a preloaded key in the TPM named parent key. The handle of 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 and are sent to the TPM.

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

3. The TPM and user process will calculate shared secret using hmac algorithm and based on $ad(pkh)$, $n_{osap}$ and $n_{osap}$.

4. The user process calls 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 $SHA1(S, n_o)$. The keyed HMAC on $S$ demonstrates knowledge of storage root key authData.

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 package. The encrypted package and public key are put in keyblob. The keyblob is returned by the TPM and is authenticated with a HMAC. The hmac is created with $n_o$ and $n_o'$ nonces and is keyed on $S$.

### 2.2 CPN modeling

The CPN models are depicted as graphical drawings composed of places, transitions, arcs and inscriptions. Places are shown using circles and ellipses. They demonstrate different states of the system. Transitions shown by rectangles describe actions. For example, sending a packet in a network is a transition that changes state of network (shown by a named place in CPN) from place 1 to place 2. The transitions and places are connected to each other using arrows called arcs. For any arc, an arc expression can be written in CPN ML language to define how the state of net changes after occurring the transition. Places can be marked with a set of markers called tokens. 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 color. The set of all the tokens that can exist in a place is defined as its colour set and is written below the place using an inscription (written text in the CPN ML programming language). The value of each variable specifies its binding. The number of tokens and their colours in all the individual places specifies marking of the CPN model. The number of tokens in just one place and their colours specify 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 binding elements. For each transition it is possible to define a Boolean expression named guard that when evaluated to true the binding elements will be enabled otherwise binding elements are disabled and can not be occurred.
Tokens can move between places and transitions using arcs. However, when tokens need to move between different pages of model, arcs are not useful. Port and socket places are designed to move tokens in complete models. 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.

Transitions initially are disabled and when all of their input tokens are provided they are enabled. Then output arcs that are connected to the transition move the input token to the output places places. It is possible to consider special inscriptions named guards for transitions. This 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.

To execute a CPN model different occurring steps and the reached intermediate markings must be described by means of an occurrence sequence. If a marking via an occurrence sequence is reachable and it starts from the initial marking then it is called a reachable marking[16].

2.3 State Space Analysis

Simulation of a CPN model just analysis a finite number of executions. This helps validating, detecting and finding errors of CPN model as well as increasing confidence about the model. It can demonstrate model is working correctly. However, using that it is impossible to guarantee correctness of model with 100% of certainty because all the possible executions are not covered[16].

Full state space 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 set of reachable markings and the arcs are corresponding to the occurring binding elements.

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. It means that, there are a large number of reachable states in the state spaces of the system that they can not easily be handled by the available computing resources like CPU and memory. Designing new methods to optimize and reduce the number of states in an state space is an active area of research. These methods are led to design a large collection of state space reduction methods. The invention of these methods has made analysis of large scale industrial systems possible.

In most cases after producing full state spaces the Strongly Connected Component Graph (SCC-graph)is generated. This graph is derived from the state space graph. 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 CPN/Tools uses the SCC-graph to determine standard behavioral properties of the model. The structure of the SCC-graph can provide information about the behavior of the model [16]. To create this graph a special toolbox is designed in CPN simulation and analysis tools such as CPN Tools.

2.4 What is ASK/CTL

State spaces of coloured Petri nets provide a number of properties that can be evaluated. Temporal logics like CTL are also convenient for specifying CPN properties or for defining new properties for them [7]. CTL provides a model of time 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. They can determine whether an artifact possess safety or liveness properties or not. ASK-CTL is an extension of CTL [9] 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. For more information about the ASK-CTL, [8] can be studied. This research uses ASK-CTL to define and verify the authentication property of the OSAP protocol.

3 Modeling OSAP using CPN

To create the CPN model of the OSAP protocol the following steps are followed:
1. A CPN model for the protocol and attacker 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 and ML functions.
2. Simulating the CPN model to find its bugs.
3. Analyzing the model using state space.
4. Evaluating the analysis results to investigate whether designated places (based on the definition of the evaluation property) contain predicted tokens or not.
5. Validating the results using ASK-CTL to be sure that the evaluation outcomes are valid in all the time slots of the analysis.

The mentioned steps are illustrated in more detail in the next paragraphs and sub-sections.

To design and implement the CPN model of the OSAP protocol all the entities participating in one protocol are identified. Three main entities are considered in this protocol: user, TPM and intruder. To model messages sent and received by these entities the following colour sets are defined:
colset csTERMS = with null | ah | ahi | no_osap | ne | ne_osap | ne_osap1 | no | nel | nil | ni2 | pkh_user | pkh_tpm | pkh_j | keyblob | keyblobi | pkh_authdata | newauth_user | authdatai;

This colour set defines all the terms used in the protocol. These terms are components of each sent and received message and it does not allow other terms to be used. The terms ending with ‘tpm’ are created or stored by the TPM. The ones ending with ‘user’ are created or stored by the user. The existence of letter ‘i’ at the end of the term, means that the term has been created by the intruder.

colset csATTACK = with posattack | negattack;

The csATTACK colour set determines whether an attack has occurred or not. When an attack has occurred a token with the value of posattack is put in a designated place. When no attack has occurred a token with the value of negattack is put in the designed place.

colset csSEQ = with user1 | user2 | user3 | user4 | user41 | user42 | user43 | user5 | int1 | int2 | int3 | int4 | intx3 | tpm1 | tpm2 | tpm3 | tpm4 | tpm5 | err | endex;

The state space explosion is one of the most important problems of modeling protocols by CPN. To prevent this problem csSEQ colour set is designed to define in any state which transitions can be run. When in a typical state the user1 transition should be enabled, all the other transitions will be deactivated to decrease the state space nodes.

colset csAUTH_HANDLE = subset csTERMS with [ah, ahi];

This colour set defines which authorization handles can be created or used. The ‘ah’ is used by TPM and the user. The ‘ahi’ is another authorization handle that is faked by the intruder.

colset csNONCE = subset csTERMS with [no_osap, ne, ne_osap, no, nel, nil, ni2];

This colour set defines all the nonces that are used or created by the user, TPM and intruder.

colset csPUBKH = subset csTERMS with [pkh_user, pkh_tpm, pkh_j];

This colour set defines the public key handle used by the user, TPM and intruder.

colset csKEYBLOB = subset csTERMS with [keyblob, keyblobi];

This colour set defines two terms: the first one is used by the TPM and the user and the second one is faked by the intruder.

colset csAUTH_DATA = subset csTERMS with [pkh_authdata, newauth_user, authdatai];

This colour set defines the authorization data that is created by the TPM or is faked by the intruder.

colset csOSAP_MSG = product csPUBKH * csNONCE;

In the first exchange of the protocol shown in figure 7 format of the sent message is defined by this colour set.

colset csOSAP_RESPONSE = product csAUTH_HANDLE * csNONCE * csNONCE;

The format of the TPM response to the OSAP message (the 2nd interchange of figure 7) is defined by the csOSAP_RESPONSE.
01 colset csTERMS = with null | ah | ahi | no_osap | ne | ne_osap | ne_osap1 | no | nei | ni1 | ni2 | pkh_user | pkh_tpm | pkh_i | keyblob | keyblobi | pkh_authdata | neauth_user | 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 | err | endses;
04 colset csAUTH_HANDLE = subset csTERMS with [ah, ahi];
05 colset csNONCE = subset csTERMS with [no_osap, ne, ne_osap, no, nei, ni1, ni2];
06 colset csPUBKH = subset csTERMS with [pkh_user, pkh_tpm, pkh_i];
07 colset csKEYBLOB = subset csTERMS with [keyblob, keyblobi];
08 colset csAUTH_DATA = subset csTERMS with [pkh_authdata, neauth_user, 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 csXOR_OUTPUT = product csSHARED_SECRET * csNONCE * csAUTH_DATA;
13 colset csHMAC_OUTPUT = product csSHARED_SECRET * csNONCE * csNONCE;
14 colset csWRAPKEY_INPUT = product csAUTH_HANDLE * csPUBKH * csNONCE * csXOR_OUTPUT * csHMAC_OUTPUT;
15 colset csWRAPKEY_MSG = product csWRAPKEY_INPUT * csKEYBLOB;
16 colset csWRAPKEY_RESPONSE = product csKEYBLOB * csNONCE * csHMAC_OUTPUT;

Fig. 2. List of CPN model colour sets
11 colset csSHARED_SECRET = product csAUTH_DATA * csNONCE * csNONCE;
The TPM and the user are both able to create a shared secret. The csSHARED_SECRET colour set is used to define the format of the secret.

12 colset csXOR_OUTPUT = product csSHARED_SECRET * csNONCE * csAUTH_DATA;
This is a colour set designed to temporarily store the result of the XOR function.

13 colset csHMAC_OUTPUT = product csSHARED_SECRET * csNONCE * csNONCE;
This is a colour set designed to temporarily store the result of the HMAC function.

14 colset csWRAPKEY_INPUT = product csAUTH_HANDLE * csPUBKH * csNONCE * csXOR_OUTPUT;
This colour set stores all the inputs of the TPM_CreateWrapKey function.

15 colset csWRAPKEY_MSG = product csWRAPKEY_INPUT * csHMAC_OUTPUT;
The message sent in exchange number 3 of the protocol is sent by the user to the TPM and is stored in this colour set.

16 colset csWRAPKEY_RESPONSE = product csKEYBLOB * csNONCE * csHMAC_OUTPUT;
The response of the TPM to TPM_CreatWrapKey function is stored in a message of type csWRAPKEY_RESPONSE.

The intruder has a database in which stores all his knowledge. This database is a location to accumulate all the sent and received messages through intruder. It also stores the initial knowledge of the intruder. The colour set csINTDB is designed for this purpose. The initial value of intruder’s database is shown in figure 3. In this figure the part 1'fiauth_handle(ahi) means that one token with value of ahi should be put in the fiauth_handle field of intruder’s database. The other fields have the same meaning. For example, 1'finonce1(ni1) means that a nonce with value of ni1 is put in the finoce1 field of the database.

Variables that are used in CPN model are shown in figure 4.

\[
\begin{align*}
1'fkeyblob(keyblobi) &+ \\
1'fiauth_handle(ahi) &+ \\
1'finonce1(ni1) &+ \\
1'finonce2(ni2) &+ \\
1'fisnewauth(authdatai) &+ \\
1'fipubkh(pkh_i) &+ \\
1'fiss(authdatai,ni1,ni2) &+
\end{align*}
\]

Fig. 3. Initial value of the intruder’s database
OSAP CPN model

The next step, after the colour set definition is to design the CPN models of the user, intruder and TPM. Figure 5 demonstrates the OSAP protocol composed of four different exchanges. In any of them TPM and the 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. The first substitution transition of this model is used to create the $TPM_{OSAP}(pkh, n_{osap}^{o})$ message. This message is sent to the TPM. However, intruder can intercept this message. The Intruder_1 substitution transition is modeling the intruder functionality. Intruder is able to send original, faked or changed message toward TPM or user. If it send the message to the TPM, because of the specific format of the message it can be received by ‘Process TPM_{OSAP}’ substitution transition. When the message is sent to the user again, this new message should be created by intruder. Intruder_2 substitution transition is the only transition that can do this, thus the control of message movement is changed from exchange #1 (figure 6 part (a) ) to the exchange #2 (figure 6 part (b)). When the message movement is like part (a) of figure 6 it is processed by ‘Process TPM_{OSAP}’ substitution transition. Then the message of 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 in its way. It can send 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 3rd exchange 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. If the message is processed by ‘Process TPM_CreateWrapKey message’ in the next step ‘Send TPM_CreateWrapKey Response’ will be executed otherwise after Intruder 4, ‘Process TPM_CreateWrapKey(...) Response’ is executed and the protocol will be ended. The main page of CPN model shows the sequences of the OSAP protocol demonstrated in figure 7.

![Fig. 5. Different exchanges of OSAP protocol](image)

![Fig. 6. message sequence in exchange #1 and exchange #2](image)
Fig. 7. OSAP protocol CPN model
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 should be run. This token during the simulation and analysis moves from one transition to the other and specifies the sequence of protocol run. The application of this approach prevents concurrent run of transitions and dramatically decreases the state space during the analysis.

![Fig. 8. TPM_OSAP(pkh, no_osap) hierarchical transition CPN model](image)

**Fig. 8.** TPM_OSAP(pkh, no_osap) hierarchical transition CPN model

**modeling TPM_OSAP(pkh, no_osap) substitution transition**

The CPN model of TPM_OSAP(pkh, no_osap) substitution transition is shown in figure 8. The TPM_OSAP transition needs three tokens from the no_osap_user, pkh_user and call TPM_OSAP places to be run. The token of the ‘call TPM_OSAP’ place is provided from the ‘Start Session 1’ in the upper page of the model. This token is the sequence token that will be passed through different transitions to determine the correct sequence of the protocol. The value of this token is compared by a specific constant (such as user1) provided by the guard [vseq=user1] for transition TPM_OSAP. When the value of this guard is true, the transition can be run. After running the transition TPM_OSAP, this token is moved to the place CS (Current Sequence) with its colour changes to int1. This means that int1 (the first transition of the intruder) is the next transition that should be enabled. The content of the CS place has become a global value in the model using the GF_seq fusion set. The other two required tokens, no_osap and pkh_user,
are stored in their places as the initial value. This assumption is acceptable because the \texttt{no\_osap} is a nonce that is created by the user and the \texttt{pkh\_user} is a public value that is known publicly. These two tokens are returned by double arcs to their initial places after the \texttt{TPM\_OSAP} transition is enabled to provide these tokens for other stages of the protocol when the user needs them again. As \texttt{no\_osap} is used by the user to create the shared secret and to make the model as simple and readable as possible, this place is tagged as a global fusion set named \texttt{no\_osap\_user}. The result of \texttt{TPM\_OSAP} transition will be a message containing input parameters of the \texttt{TPM\_OSAP} function. This message is sent to the place \texttt{’TPM\_OSAP message’}, then using the output port of this place it will be sent back to the \texttt{’Sent TPM\_OSAP message’} place in the main page of the OSAP model.

**modeling Intruder\_1 substitution transition**

The stored message in the \texttt{’Sent TPM\_OSAP message’} place is transmitted over the network. This provides the ability for the intruder to intercept the message. The intruder model is based on the Dolev-Yao [27] model. The Dolev-Yao model assumes the intruder is the medium that transmit all the messages and has the most possible functions. It is able to intercept, store, change, forward and create a message. The CPN model of the intruder in the OSAP model is demonstrated in figure 9.

The OSAP message token enters the intruder sub-transition using the input port \texttt{’Sent TPM\_OSAP Packet’}. This token is stored in the \texttt{’tmp storage’} place and using the \texttt{Tmp\_echg1} fusion set, it becomes public in the page of the intruder. This makes creating the model simpler and prevents long arcs that cross each other to be created. The \texttt{’Store message parts in the DB’} transition stores token parts, the parent key handle and the created nonce, in the \texttt{inToken pubkh} and \texttt{’inToken nonce’} places. The content of these places will be stored in the intruder database using the \texttt{GF\_intDB} fusion set. They are stored in the \texttt{fipubkh} and \texttt{finonce\_en} fields of the intruder’s database respectively. The arc that connects \texttt{’Store message parts in DB’} transition to the \texttt{’tmp storage’} place is a double arc that after a token is consumed by the transition returns it again to the place. In some cases (that all the other required tokens to fir the transition are provided) this makes a transition permanently enabled. To prevent this situation the \texttt{’JO 1’} enabler place, with just one token, is connected to the transitions. This mechanism, just once, allows a transition to only run once. It is impossible to run it again. The transitions located in the intruder’s page can only be run if the sequencer token value is \texttt{int1} and it has been sent from another page to this page. To enforce this condition all the transitions have the guard \texttt{[vseq=int1]}. This sequencer token using places with colour set \texttt{csSEQ} and with connected arcs to them is moved from the first transition of the page to the last one.

The transition \texttt{’Store Whole message in DB’} after \texttt{’Store message parts in DB’} transition stores the complete message in the intruder database. The \texttt{’JO 2’} and \texttt{’Sent TPM\_OSAP’} places connected to this transition act the same as the illustrated \texttt{’JO 1’} and \texttt{’tmp storage’} places. The intruder, after storing the
Fig. 9. The CPN model of the intruder located in the first message exchange of OSAP protocol
message parts or the complete message, can take several actions. These actions are forwarding one of the stored messages to the TPM, creating new messages and sending them to the TPM, or bypassing the TPM completely. One of the ‘Forward stored message’, ‘Create new message’ or ‘Bypass TPM’ transitions will be run randomly to do any of the mentioned actions. The sequence of the protocol run is based on whether the first two transitions are run or the last one will be different.

The result of forwarding a message or creating a message will be stored in the ‘tmp output TPM OSAP message’ place. The ‘check attack’ transition will check whether the new message is completely faked by the intruder or not. If the attack has occurred, a posattack token will be put in the ‘int1 change’ place, otherwise a token with negattack value will be added to the place. Function fakedxchg1 in figure 10 checks whether the content of the sent token is fully based on the intruder’s knowledge or not. If ‘yes’ then it means the token is faked. If ‘no’ it means the token is genuine.

```haskell
fun fakedxchg1 (vossap_msg: csOSAP_MSG):csATTACK =
  case vossap_msg of
  | (pkh_i, ni1) => posattack
  | (pkh_i, ni2) => posattack
  | _ => negattack;
```

Fig. 10. function fakedxchg1 to check whether the exchange one is faked or not.

After verification the resulting output token will be sent to the ‘output TPM_OSAP message’ place. Then using the output port the token will be forwarded to the TPM. The current sequence of the model will change to the tpm2 after the message is sent to the TPM. This change happens by sending tpm2 token to the CSO1 place. This change causes just the transitions of page T2 (first TPM substitution transition of TPM) to run.

The ‘Bypass TPM’ transition is selected to be run when the intruder decides not to send any message to the TPM. In this case the current sequence of the model is put in the ‘Run intruder 2’ place. In CPN/Tools version 3.0.2 it is not possible to connect a port directly to a fusion set. Thus a few transitions and places are inserted between Intruder_1 and Intruder_2 transitions to connect them to each other. As mentioned before in this model it is necessary to determine when each transition can be run. Thus, before entering to the Intruder_2 it is assumed that the current state of the system to is still int1. In page ‘Session_1 Run Intruder 2’ transition changes the current sequence of the model from int1 to int2 and triggers the Intruder_2 sub-transition. The model of the Intruder_2 sub-transition will be illustrated later.
modeling ‘Process TPM\_OSAP’ substitution transition

The transition ‘Process TPM\_OSAP’, after receiving the TPM\_OSAP command message in ‘Received TPM\_OSAP Message’ stores its parts in pkh\_tpm and no\_osap\_tpm places. The CSI and CSO places are used to determine input and output sequence tokens. When the transition is ended an empty token is put in the ‘End processing TPM\_OSAP msg’ to transfer the protocol run to the next sub-transition of the protocol, ‘Send TPM\_OSAP Response’.

Fig. 11. Substitution transition ‘Process TPM\_OSAP’

modeling ‘Send TPM\_OSAP Response’ substitution transition

TPM after processing TPM\_OSAP message prepares the response. The ‘Send TPM\_OSAP Response’, figure 12, is a substitution transition that creates the response. The required token to create the response is provided by ne\_osap\_tpm, ne\_tpm, ah\_tpm places. The result will be stored in the ‘TPM\_OSAP Response’ place. The CSI and CSO places are used like before to change the current sequence of the model in the GF\_seq fusion set. ‘Start Seq 2’ place is used to the transfer protocol run from the previous transition to the next one. The ‘Create Shared Secret’ place is considered to transfer a sequence to a substitution transition ‘Create Shared Secret TPM’ after triggering ‘SEND TPM\_OSAP RESPONSE’.
modeling ‘Create Shared Secret TPM’ substitution transition

The ‘Create Shared Secret TPM’ substitution transition, figure 13, produces the shared secret of the TPM. To generate the shared secret required token are fetched from ne_osap_tpm, no_osap_tpm, ‘authdata_pkh_tpm’ places. The double arcs are used to return tokens after enabling ‘GENERATE SHARED SECRET TPM’ transition to their places. The usage of CSI, CSO, ‘Create Shared Secret’ and ‘End Creating SC’ places are like previous places. The result will be stored in the global fusion set ‘shared_secret_TPM’.

When both of ‘Send TPM.OSAP Response’ and ‘Create Shared Secret TPM’ substitution transitions of page Session 1 are run the ‘Hash is Done’ transition in the page can be enabled. Then the response of the TPM to the TPM.OSAP message will be stored in the ‘Sent TPM.OSAP Response’ place and Intruder_2 substitution transition, figure 14, will be run.

modeling Intruder_2 substitution transition

The page Intr_2 contains the CPN model of the Intruder_2 substitution transition in page Session 1. This intruder can be enabled from two different paths. The former path is when the TPM is bypassed by the Intruder_1 substitution transition. As the result of bypassing TPM there is no message send from the
TPM to user and intruder will create the message based on its database information itself. The ‘TPM is Bypassed’ transition is the first enabled transition. Its input is just a sequence token coming from the previous transition and transfers the flow of tokens to it. When the sequence token reaches the ST3 place, it randomly chooses either the ‘Forward stored message’ (to fetch one message from the intruder’s database and send it to the user) or ‘Create new message’ (to fetch different fields from the intruder database, create a new message and send it to the user) transitions to be run. After running either transition the sequence token will be transmitted to the CSO place and an output token will be stored in ‘Tmp Received TPM OSAP Response’ place. This token is checked by the ‘check attack’ transition using fakedxchg2(vosap res) function, figure 15, to investigate whether it is faked by the intruder or whether it is a genuine message created by TPM. When the token is faked by the intruder, fakedxchg2 function puts a token with the colour set of csATTACK and posattack value in the ‘int2 change’ place. The final token will be sent to the user from ‘Received TPM_OSAP Response’ place as an output port.

The latter path is when the TPM sends a response of the TPM_OSAP instruction to the user. This response will be stored in the ‘Sent TPM_OSAP Response’ place. The ‘Store Whole message in DB’ transition puts the whole input token into the intruder’s database. After that, the ‘Store message parts in DB’ transition will be enabled to store message parts in the intruder’s database using ‘inToken ah’, ‘inToken ne’ and ‘inToken ne_osap’ places. Then the se-
Fun fakedxchg2 (vosap_res: csOSAP_RESPONSE): csATTACK =
case vosap_res of
  (ahi, n1, n1) => posattack
| (ahi, n1, n1) => posattack
| (ahi, n1, n2) => posattack
| (ahi, n2, n1) => posattack
| _ => negattack;

Fig. 14. Intruder 2 substitution transition

Fig. 15. Function fakedxchg2
sequence token will be moved to the ‘ST3’ place. The following parts of the model are similar to the models illustrated before.

**modeling ‘Process TPM_OSAP Response’ substitution transition**

The TPM response to the OSAP command will be processed by ‘Process TPM_OSAP Response’ user substitution transition. Its CPN model is shown in page ‘U2’, figure 16. It demonstrates how different parts of input token, are stored in suitable places and fusion sets.

![Fig. 16. ‘Process TPM_OSAP Response’ CPN model.](image.png)

The next step is creating the shared secret by user. The ‘Create Shared Secret User’, figure 17, substitution transition does this.

**modeling ‘Create Shared Secret User’**

This substitution transition generates the shared secret and puts it in the ‘SHARED SECRET USER’ place. The shared secret is generated based on content of ‘AUTHDATA_PKH USER’, ne_osap_user and no_osap_user places. The CSI place is used to move the current state token of model to the ‘GENERATE SHARED SECRET USER’. The CSO place stores the next state token of the model, user41, in the GF_seq global fusion set. The current sequence of the protocol is moved to the ‘GENERATE SHARED SECRET USER’ by ‘Start creating Shared Secret’ place and then will be moved to the next transitions by ‘Start Seq 3’.

**modeling **TPM_CreateWrapKey(...) substitution transition**

**TPM_CreateWrapKey(...)** is the next substitution transition from the user side to be run. It produces the following message:
Fig. 17. ‘Create Shared Secret User’ CPN model

Fig. 18. ‘Create Shared Secret User’ CPN model
\[\text{TPM\_CreateWrapKey}(ah, pkh, n_0, \ldots, \text{SHA}1(S, n_e) \oplus \text{newauth}), \text{hmac}_S(n_e, n_0, \ldots)\]

Figure 19 demonstrates page U4 containing the whole model.

To simplify modeling, \textit{Create\_XOR} substitution transition, produces the \textit{SHA}1(S, n_e) \oplus newauth part of message. Its model is designed in page U4, figure 20. In this page ‘\textit{Create\_XOR\_WrapKey}’ transition fetches the required tokens from ‘\textit{new\_auth}’, ‘\textit{SHARED\_SECRET\_USER}’ and ‘\textit{ne\_user}’ places. The created result will be sent to the ‘\textit{xor\_output}’ place then using the output port the result is sent back to the ‘\textit{xor\_result}’ place in page U4 and will be used to create other parts of the message exchange.
The Prepare TPM\_CreateWrapKey substitution transition, figure 21, produces

\[TPM\_CreateWrapKey(ah, pkh, n_o, ..., SHA1(S, n_e) \oplus newauth)\]

part of message. The result of Create\_XOR is sent to ‘Prepare TPM\_CreateWrapKey’ substitution transition using the ‘xor\_result’ input port. The \(ah\), \(pkh\) and \(n_o\) are provided by \(ah\_user\), \(pkh\_user\) and \(no\_user\) places. The result is stored in ‘WrapKey\_SHA’. This token will remain in WrapKey\_SHA till the other part, \(hmac_S(n_e, n_o, ...)\), has been created.

The ‘Compute HMAC’ transition produces \(hmac_S(n_e, n_o, ...)\). The detailed model is shown in page U4\_3, figure 22. Like the previous models, required inputs are provided from ‘ne\_osap\_user’, ‘SHARED SECRET USER’ and ‘no\_user’ places. The result is sent to page U4, figure 19 and will be stored in \(hmac_S.1\) place. The ‘Send TPM\_CreateWrapKey Message’ transition can now be enabled to store the result in ‘TPM\_CreateWrapKey Packet’. The provided output port will move the token to the ‘TPM\_CreateWrapKey Sent Packet’ place in Session\_1 page of the model. The token can be intercepted by the Intruder\_3.

**modeling Intruder\_3 substitution transition**

The functionalities of the Intruder\_3, figure 23, is exactly the same as previous intruders functionalities. The intruder at first, stores the complete message, \(TPM\_CreateWrapKey(ah, pkh, n_o, ..., SHA1(S, n_e) \oplus newauth), hmac_S(n_e, n_o, ...)\), in its database. Then the intruder starts storing smaller parts of a message recursively. The \(TPM\_CreateWrapKey(ah, pkh, n_o, ..., SHA1(S, n_e) \oplus newauth)\) and \(hmac_S(n_e, n_o, ...)\) parts are processed separately.
Analysis of OSAP using CPN

Fig. 21. Prepare TPM\_Create\_WrapKey

Fig. 22. The CPN model of the Compute HMAC Substitution transition
Fig. 23. Intruder_3 CPN model
The $hmac_S(n_e, n_o, ...)$ is stored completely in $fihmac$ output field of the intruder’s database. Its arguments $S$, $n$ and $n_o$ are stored in $fiss$ and $finonce$ in the intruder’s database fields respectively.

The complete $TPM_{CreateWrapKey}(ah, pkh, n_o, ..., SHA_1(S, n_e) \oplus newauth)$ message is stored in $fiwrapkey$ input field of intruder’s database in parallel with $hmac$. However, its parts are stored after the $hmac$ components. The $ah$, $pkh$ and $n_o$ will be stored in $fiauth$ handle, $fipubkh$ and $finonce$ fields of the database respectively. Then $SHA_1(S, n_e) \oplus newauth$ will be stored in $fixor$ output field. The transition ‘extract XOR fields’ will put $S$, $n_e$ and $newauth$ in the defined $fiss$, $finonce$ and $finewauth$ database fields.

The end of storing messages is the start of producing new messages to be sent to the TPM or bypassing it. When the sequence token is in state ST6 the run of ‘Bypass TPM’ transition moves the token flow to the Intruder_4 substitution transition in page Session_1. However, if ‘Forward stored message’ is to be enabled a message is fetched from $fiwrapkey$ field of the intruder’s database and will be sent to the TPM. The selection of ‘Int create new exchange 3’ substitution transition produces a new message based on the intruder’s knowledge and sends it to the TPM. The CPN model of this transition is designed in $Int_3$ page and is shown in figure 24.

The operation of $Int_3$ page is similar to page U4 designed for $TPM_{CreateWrapKey}(...)$. At first $SHA_1(S, n_e) \oplus newauth$ part of faked message is reproduced by $Create\_XOR\_WrapKey$ transition. The required inputs are fetched from the intruder $fiss$, $finonce1$ and $finonce2$ database fields. The result will be used to produce $TPM_{CreateWrapKey}(ah, pkh, n_o, ..., SHA_1(S, n_e) \oplus newauth)$ part. The $ah$, $pkh$ and $n_o$ are fetched from the $fiauth$ handle, $fipubkh$ and $finonce1$ fields. The result is temporarily stored in $WrapKey_{SHA}$ place till ‘Generate HMAC S_1’ computes $hmac_S(n_e, n_o, ...)$. This message is stored in ‘$TPM_{CreateWrapKey} Packet$’ place and finally will be sent to the $Int_3$ page using the output port of $TPM_{CreateWrapKey}$ place. Storing $tpm4$ token in $CSO$ place will change the global state of model. The ‘End create Exchg’ place by receiving the $tpm4$ token from $end$ place finishes the $Int_3$ page and returns to the $Int3$ page.

The produced tokens by both ‘Forward stored message’ and ‘Int create new exchange 3’ transitions are stored in ‘tmp $TPM_{CreateWrapKey} Received Packet’ place. The ‘check attack’ transition by calling fakedxchg3(vwrapkey_msg), 25, ML-function investigates whether the third message sequence of protocol is faked or not. If the packet is faked a $posattack$ token will be put in the ‘int3 change’ place. Then the produced packet is stored in ‘$TPM_{CreateWrapKey} Received Packet$’ place and then will be sent to the ‘$TPM_{CreateWrapKey} Received Packet$’ place in the Session_1 page using the output port. This packet is processed by ‘Process $TPM_{CreateWrapKey} message’ that its model is shown in T4 page.
Fig. 24. Int_3.1 CPN model
fun fakedxchg3 (vwrapkey_msg:csWRAPKEY_MSG) : csATTACK =
case vwrapkey_msg of
  ((ahi, pkh_i, ni1, ((authdatai, ni1, ni1), ni1, authdatai)),
   ((authdatai, ni1, ni1), ni1, ni2)) => posattack
| _ => negattack;

Fig. 25. fakedxchg3(...) ML-Function

modeling ‘Process TPM CreateWrapKey message’ substitution transition

This page, figure 26, is the forth processing page of TPM. It processes the TPM CreateWrapKey(...) message and stores its input parameters in specified places. The TPM does not need the exact amount of input parameters like ah and can find them in its memory. Thus the CPN model of T4 just stores TPM_CreateWrapKey(ah, pkh, no, ..., SHA1(S, ne)⊕ newauth) and hmac_S(no, ne, ...). The ‘Retrieve no’ transition stores hmac_S(no, ne, ...) in hmac_S_user place and TPM_CreateWrapKey(ah, pkh, no, ..., SHA1(S, ne)⊕ newauth) in ‘tmp Wrapkey’ place. The no nonce, used by TPM to produce last message of protocol, is extracted by ‘extract no’ transition and will be stored in no_TPM place. The TPM creates final message and sends it to the user in ‘Send TPM CreateWrapKey Response’ transition.

Fig. 26. Process TPM_CreateWrapKey message CPN model

modeling ‘Send TPM_CreateWrapKey Response’ substitution transition

In this page, figure 27, at first hmac_S(no', ne, ...) is generated by ‘Generate HMAC_S_2’ place. To generate it S from shared_secret_TPM, no' from net_TPM
and \( n_o \) from \( no\_TPM \) fusion sets are fetched. The ‘Generate HMAC_S_2’ transition is enabled when \([\text{vseq} = \text{tpm5}]\) is evaluated to \( \text{TRUE} \). A double arc provides access of this transition to the sequence token to make \([\text{vseq} = \text{tpm5}]\) evaluation possible. The sequence token is used by other transitions of this page thus double arc is used to return the token back to its original place, \( GF\_seq \) global fusion set.

The other required tokens, \( n'_e \) and \( \text{keyblob} \), are provided by \( ne1\_TPM\_b \) and \( \text{keyblob}\_TPM \) places. They will be combined with \( \text{hmac}\_S\_2\_TPM \) place token to produce the final message, \( \text{keyblob, n'}_e, \text{hmac}_S(n'_e, n_o, ...) \). The final token will be stored in ‘\( \text{TPM}\_\text{CreateWrapKey Response} \)’ place and using the output port will be sent to the ‘\( \text{TPM}\_\text{CreateWrapKey Sent Response} \)’ place in Session_1 page. The next transition that will be enabled is Intruder_4.

![Fig. 27. Send TPM_CreateWrapKey Response CPN model](image)

**modeling Intruder_4 substitution transition**

This page like the other intruder pages illustrates the functionality of intruder. The \( n'_e \) and \( \text{keyblob} \) parts of the input message at first are stored in \( \text{fkeyblob} \) and \( \text{finonce} \) in fields of the intruder’s database by ‘\( \text{Store message parts in DB} \)’ transition. In the next step the transition ‘\( \text{Store whole hmac} \)’ stores the \( \text{hmac}_S(n'_e, n_o, ...) \) part of the input in \( \text{fihmac_output} \) field of database. Then
‘Store hmac parts in DB’ is enabled to store hmac parts in the database. Thus, $S$, $n'_e$ and $n_o$ will be stored in $fiss$, $finonce_in$ and $finonce_in$ database fields respectively. Then ‘Store Whole message in DB’ transition is enabled and the complete $keyblob, n'_e, hmac_S(n'_e, n_o, ...)$ is stored in the intruder database $fiwrap-key_rsp$ field.

The sequence control after storing the input message and its parts reaches to the ST5 place. The existence of one token in ST5 can enable one of the ‘Forward stored token’ or ‘Create new exchange’ transitions. If ‘Forward stored token’ be enabled one message is fetched by the intruder from its database and will be forwarded to the user. The movement of sequence token from ST5 toward ‘Create new exchange’ causes a new token based on the intruder knowledge be faked and sent to the user. To produce this faked token ‘Generate output HMAC’ transition produces the $hmac_S(n'_e, n_o, ...)$ and ‘Create new exchange’ transition creates the final message, $keyblob, n'_e, hmac_S(n'_e, n_o, ...)$.

The sequence control token can move to the ST5 place from ‘TPM is By-passed’ transition. In this case Intruder_3 has bypassed the TPM in the previous exchange and has directly sent the flow of messages to the Intruder_4. The method of moving tokens to the ST5 place does not change the flow of running other transitions. All the time after getting token by ST5 either a new message will be faked and sent to the user or a complete stored message in the database will be fetched and forwarded to the user. The final created message by the intruder is stored in the ‘TPM CreateWrapKey Response’ place. Then ‘check attack’ is run to check whether new message is completely faked by intruder (all the fields are created by the intruder) or not. As the result of a faked message, function $\text{fakedxchg}(\text{vwrapkey_rsp})$ puts a token with the value of posattack in ‘int4 change’ place. Otherwise a token with negattack value is inserted in the ‘int4 change’ place. The final message is stored in ‘TPM_CreateWrapKey final Response’ place and using the output port it will be sent to the Session_1 page. In this page the response will be stored in ‘TPM_CreateWrapKey Rec Response’ place and then is finally processed by ‘Process TPM_CreateWrapKey(...) Response’ substitution transition.

### Modeling ‘Process TPM_CreateWrapKey(...) Response’ substitution transition

This page, figure 28, like other pages, processes the input token and its fields are put in different places. The input token comes from the ‘TPM_CreateWrapKey Response’ input port and is stored in ‘Whole rsp msg’ place. Its $keyblob, n'_e$ and $hmac_S(n'_e, n_o, ...)$ parts are stored in ‘keyblob_user’, $n'_e_user$ and $hmac_S_s2_user$ places respectively. The ‘Retreive WrapKey Response fields’ transition after decomposing the input token puts an $endses$ token in $CSO$ place to indicate that the protocol is finalized and there is no more transitions. An empty token is put
in ‘End WrapKey Proc’ place. This token is sent to the Session_1 page from output port and is stored in the ‘End Session 1’ place. The existence of a token in Session_1 page demonstrates that the protocol has finished.

![Diagram](image)

Fig. 28. Process TPM_CreateWrapKey(...) Response CPN model

4 Verification of the model using state-space

Simulation of a CPN model just analyses a finite number of executions. This helps validating, detecting and finding errors of CPN model as well as increasing confidence about the model. It can demonstrate model is working correctly. However, using that it is impossible to guarantee correctness of model with 100% of certainty because all the possible executions are not covered[16].

Full state space 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 set of reachable markings and the arcs are corresponding to the occurring binding elements.

In this research the CPN/Tools state space is used to evaluate the authentication property of OSAP CPN model.
4.1 Authentication property

The designed CPN model in this research checks the authentication property of OSAP protocol. To check this property any sent and received message is checked by the model to find whether it is faked by the intruder or not. If the TPM is bypassed in transitions 2 and 4, or what is sent to the TPM in transitions 1 and 3 is faked by intruder it means that authentication property is violated. If any message is faked by the intruder it means that the authentication property is violated. This illustration can be formulated as follows:

\[ CS = \{ \text{colourset}_i \mid 1 \leq i \leq k \} \]

where \( k \) is the number of defined colour sets in CPN model.

\[ CS(f_i) : \text{colourset of } f_i \]

\[ CS(f_m^i) : \text{colour set of the } i\text{-th } \]

\[ M(n) = \{ \forall_{i=1}^n f_i \mid (f_i \in F) \} \]

any message \( M \) with \( n \) fields is the union of \( n \) different fields and each field is a member of \( F \).

\[ M'(h) = \{ \forall_{j=1}^h f'_j \mid (f'_j \in F) \} \]

any message \( M' \) with \( h \) fields is the union of \( h \) different fields and each field is a member of \( F \).

\[ \text{equivalency} : m \equiv m' \text{ iff } m \in M(n) \AND m' \in M'(h) \AND n = h \AND \forall 1 \leq i \leq n, CS(f_m^i) = CS(f_m'^i) \]

\[ M_{A,B} : \text{Message } M \text{ sent from } A \text{ to } B \]

\[ M_{B,A} : \text{Message } M \text{ sent from } B \text{ to } A \]

\[ M_{I,A} : \text{Message } M \text{ sent from intruder } I \text{ to principal } A \]

\[ M_{I,B} : \text{Message } M \text{ sent from intruder } I \text{ to principal } B \]

Authentication property definition:

\[ O : \text{the number of messages sent from principal } A \text{ to principal } B \]

\[ P : \text{the number of messages sent from principal } B \text{ to principal } A \]

\[ M_{i,A,B} : \text{the } i\text{-th message sent from } A \text{ to } B \]

\[ M_{i,B,A} : \text{the } i\text{-th message sent from } B \text{ to } A \]

Authentication property is hold if \( O \leq P \)

\[ M_{A,B} \equiv M_{I,B} \AND (\forall_{i=1}^O M_{A,B} \AND \exists M_{I,A} \mid M_{i,A,B} \equiv M_{i,A}) \]

To check the authentication property, four different functions are implemented. They are all shown in figure 29.

This definition of authentication can be used for any protocol. It only checks whether a message is sent from an authentic source or not. However, for OSAP when authentication property is violated both of (1 and 3) or (1 and 4) exchanges are faked by the intruder. Thus in the ASK-CTL formula both of the situations are checked.

4.2 using ASK-CTL for validating OSAP authentication property

The ASK-CTL formula is used to check authentication property of the OSAP protocol. There are two different situations that if any of them happens the authentication property will be violated:

1. When the intruder has bypassed the TPM and based on its knowledge in database has faked 2 and 4 message exchanges. The marking of the model in this situation is shown in figure 30.
fun fakedxchg1 (vosap_msg: csOSAP_MSG): csATTACK =
    case vosap_msg of
    | (pkh_i, ni1) => posattack
    | (pkh_i, ni2) => posattack
    | _ => negattack;

fun fakedxchg2 (vosap_res: csOSAP_RESPONSE): csATTACK =
    case vosap_res of
    | (ahi, ni1, ni1) => posattack
    | (ahi, ni1, ni2) => posattack
    | (ahi, ni2, ni1) => posattack
    | (ahi, ni2, ni2) => posattack
    | _ => negattack;

fun fakedxchg3 (vwrapkey_msg: csWRAPKEY_MSG) : csATTACK =
    case vwrapkey_msg of
    | ((ahi, pkh_i, ni1, ((authdatai, ni1, ni1), ni1, authdatai)),
        ((authdatai, ni1, ni1), ni1, ni2)) => posattack
    | _ => negattack;

fun fakedxchg4 (vwrapkey_rsp: csWRAPKEY_RESPONSE) : csATTACK =
    case vwrapkey_rsp of
    | (keyblobi, ni1, ((authdatai, ni1, ni2), ni1, ni2)) => posattack
    | _ => negattack;

Fig. 29. Function to check first exchange of the protocol

2. When the intruder does not bypass the TPM but it is able to fake all the messages sent from user to the TPM. In this case not only message exchanges number 2 and 4 but also number 1 and 3 are faked by the intruder. The marking of the model in these situations is shown in figure 31.

\[
\text{Marking}(M, P^{int1\_change}) = 1'\text{posattack AND Marking}(M, P^{int3\_change}) = 1'\text{posattack AND EndSession}\{\text{Marking}(M, P^{CSO})\} = 1'\text{endses = FakedExchange2\_4}(M)\]

Fig. 30. Marking of the CPN model when authentication property is violated by bypassing TPM in 2 and 4 exchanges

To check these situations the ASL-CTL formula of figure 32 is written:

The first line of the SML formula opens the ASK/CTL library. Line 2 defines function \text{FakedExchange2\_4} to find is there any state in the state space that at least one token is available in \text{int1\_change} and \text{int3\_change} places or not. When this function is \text{TRUE} the marking of figure 30 can be found in the occurrence graph. Line 3 defines function \text{FakedExchange1\_3} to find whether at least in one marking both of the 1 and 3 exchanges have been faked by the intruder or not. When this function is evaluated to \text{TRUE}, one marking in the occurrence graph can be found that both of the exchanges are faked. The marking of the OG in this situation is shown in figure 31. The \text{EndSession} function (line 4) by checking the number of tokens with the value of \text{endses} in \text{CSO} place of
\[
\text{Marking}(M_1, P_{\text{int}1}^{\text{initial}}) = 1' \text{posattack AND}
\text{Marking}(M_3, P_{\text{int}3}^{\text{initial}}) = 1' \text{posattack AND}
\text{EndSession} \{ \text{Marking}(M_i, P_{\text{CSO}}^{\text{U5}}) \} = 1' \text{endses} = \text{FakedExchange1}_3(M_i)
\]

**Fig. 31.** Marking of the CPN model when authentication property is violated by bypassing TPM in 1 and 3 exchanges

```sml
01 use (ogpath^"ASKCTL/ASKCTLloader.sml");
02 fun FakedExchange2_4 n = cf (posattack, Mark.Int_2'int2_change 1 n) > 0 andalso cf (posattack, Mark.Int_4'int4_change 1 n) > 0;
03 fun FakedExchange1_3 n = cf (posattack, Mark.Int_1'int1_change 1 n) > 0 andalso cf (posattack, Mark.Int_3'int3_change 1 n) > 0;
04 fun EndSession n = cf (endses, Mark.U5'CSO 1 n) > 0;
05 val TransitionsFaked2_4 = NF ("", FakedExchange2_4);
06 val TransitionsFaked1_3 = NF ("", FakedExchange1_3);
07 val SessionEnds = NF ("", EndSession);
08 val myASKCTLformula = POS (OR(AND(TransitionsFaked2_4, SessionEnds), AND(AND(TransitionsFaked1_3, TransitionsFaked2_4), SessionEnds)));
09 eval_node myASKCTLformula InitNode;
```

**Fig. 32.** ASK-CTL-formula

the page U9 determines whether the protocol has finished or not. The node function (NF) in lines 5, 6 and 7 evaluates FakedExchange2_4, FakedExchange1_3 and EndSession functions. The result which is TRUE or FALSE will be stored in TransitionsFaked2_4, TransitionsFaked1_3 and SessionEnds. The POS function in line 8 checks either the first situation of authentication property or the second situation is violated or not. This condition is shown in figure 33. The last line of SML code evaluates the myASKCTLformula starting from InitNode.

\[
\text{POS(FakedExchange2}_4(M_i) \text{ AND EndSession) OR (FakedExchange1}_3(M_i) \text{ AND FakedExchange2}_4(M_i) \text{ AND EndSession}}
\]

**Fig. 33.** ML function to check the authentication property of OSAP

### 5 Conclusion and future works

This research analyses the OSAP protocol using CPN. The results of the analysis shows that authentication property of this protocol can be violated. This model is designed based on [6] assumptions. The analysis can be done by different assumptions to study the protocol in more detail.

The used approach can be applied for other security properties such as secrecy. Analyzing other properties needs a few refinements in the model to add required place, transitions and colour sets. It is necessary to write new ASK/CTL formulas to validate results.
The designed attacker model based on Dolev-Yao approach can be replaced by other models. However, this replacement needs lots of changes in the model that requires more efforts and times. Because the Dolev-Yao attacker model is the most powerful and popular attacker model used in analyzing protocols, it is not recommended to change it.

The OSAP protocol is just 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 composing OSAP with other trusted computing protocols and analyze the created model.

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 messages structure can cause lots of changes in the CPN model. It leads to inevitable cascaded changes in the CPN model. However, this firm connection helps the designers of the model 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.

6 Previous works

Coloured Petri Nets have been used by [12] 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 authentication protocols and analyzes them.

The Station-to-Station (STS) security protocol is analyzed in [2] using CPN. The authors 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 [1] 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. Application of the token passing scheme is used to resolve the problem of state space explosion that during the simulation in Design/CPN occurs. The Al-Azzoni approach is used for this research. Moreover, a current state token mechanism is used to determine current page of the model that should be run. In this mechanism a guard is added to transitions of a nominated page. This guard enables transitions just when container page of transition is the active page of model. This mechanism is implemented using GF_seq global fusion set and csSEQ colour set.
References


