Modeling and Verification of Cryptographic
Protocols Using Coloured Petri Nets and
Design/CPN

Issam Al-Azzoni, Douglas G. Down, and Ridha Khedri
McMaster University
1280 Main Street West, Hamilton, Ontario, Canada L8S 4K1
{alazzoi, downd, khedri}@mcmaster.ca

Abstract
In this paper, we present a technique to model and analyse cryptographic
protocols using coloured Petri nets. A model of the protocol is constructed
in a top-down manner: first the protocol is modeled without an intruder,
then a generic intruder model is added. The technique is illustrated on the
TMN protocol, with several mechanisms introduced to reduce the size of the
occurrence graph. A smaller occurrence graph facilitates deducing whether
particular security goals are met.
Keywords: Cryptographic protocols, Protocol analysis, Coloured Petri nets,
Design CPN, Security goals.

1 Introduction
Cryptographic protocols play a crucial role in achieving security in today’s com-
munication systems. They are used in the Internet and in wired and wireless net-
works to ensure privacy, integrity and authentication. A cryptographic protocol
is a communication protocol that uses cryptographic algorithms (encryption and
decryption) to achieve certain security goals.

Generally, a cryptographic protocol involves two communicating agents who
exchange a few messages, with the help of a trusted server. The exchanged mes-
sages are composed from components such as keys, random numbers, timestamps,
and signatures [13]. At the end of the protocol, the agents involved may de-
duce certain properties such as the secrecy and authenticity of an exchanged mes-
se [12].

In analysing a cryptographic protocol, all possible actions by an intruder must
be considered. An intruder is an attacker who wants to undermine the security
of a protocol. An intruder can perform the following actions to mount at-
tacks [12]: prevent a message from being delivered, make a copy of messages,
intercept a message by preventing it from reaching its destination and making a
copy, fake a message, modify a message, replay a message, delay the delivery of a message, and reorder messages. A fake message is fully generated using material gleaned from past exchanged messages while a modified message is a genuine message that the intruder partially altered.

The intruder manipulates messages as outlined above to mount an attack on the protocol. In this paper, we are concerned with attacks that result from flaws inherent in the protocol. Flaws in cryptographic protocols may allow an intruder to authenticate as someone else, or gain information that should not be otherwise revealed. We assume cryptographic algorithms are secure; i.e. it is not possible to decrypt a ciphertext without knowledge of the decryption key. This assumption allows us to focus on finding flaws inherent in the analysed protocol structure.

In this paper, we explore the use of coloured Petri nets [5] in the verification of cryptographic protocols. The ability to model concurrent behaviour has made coloured Petri nets an appropriate analysis tool for cryptographic protocols. There are two distinctive advantages of using coloured Petri nets: they provide a graphical presentation of the protocol, and they have a small number of primitives making them easy to learn and use. Furthermore, there exists a large variety of algorithms for the analysis of coloured Petri nets. Several computer tools aid in this process.

The computer tool Design/CPN [4, 10] had not been explored as a potential automated verification tool. We claim that given the power of Design/CPN, one can construct a coloured Petri net model of a cryptographic protocol and use advanced features to allow stronger and more efficient verification. Examples of such features include: inscriptions, occurrence graph tools, hierarchical features, and ML queries.

In this paper, we are motivated to explore the use of Jensen’s form of coloured Petri nets and Design/CPN in the verification of cryptographic protocols. In the process, we develop a new technique that addresses limitations of the techniques developed in [2, 14]. We focus on benefiting from the high level constructs of Jensen’s coloured Petri nets, as well as using Design/CPN.

In the next section, we give an outline of the new technique. In Section 3, we demonstrate the technique by using it in the modeling and analysis of the TMN protocol. Finally, Section 4 summarises the technique’s benefits and suggests possible extensions. An extended version of this work can be found in [1].

2 Outline of the Technique

Our technique is a finite-state analysis method. Thus, it involves modeling the protocol as a coloured Petri net, then an automated tool (Design/CPN) is used to generate all possible states. Insecurities are discovered if an insecure state is reachable in the CPN occurrence graph.

The technique has several technical features not existing in other cryptographic protocol verification techniques using Petri nets. One of these features is the use
of a central place to hold the tokens intercepted by the intruder; we call this place a DB-place. Its marking models the accumulated intruder knowledge. It is implemented by using a global fusion set of places. Although the pages of the illustrative example presented here are not big enough to fully illustrate the advantages of the use of fusion places, their use is extremely advantageous when one deals with more complex protocols. The colour set of this fusion set is defined to be the union of the colour sets of tokens that can be possessed by the intruder. The use of the DB-place makes the intruder model simple and clear.

We implement a token-passing scheme to prevent unnecessary interleaving of the firings of protocol entity transitions. This results in a smaller occurrence graph. Other techniques [2, 14] handle the issue of state explosion differently. They restrict the behaviour of the intruder by introducing new assumptions. On the other hand, the intruder model in our technique is less restricted. This implies that our technique may capture a larger variety of attacks.

We use a top-down modeling approach. At the highest level of abstraction, an entity is modeled as a substitution transition. Each substitution transition is defined in a separate subpage that provides a lower level description of the behaviour of the entity.

In modeling a cryptographic protocol using our technique, we follow these steps:

1. Build a model with no intruder: In this step a) using CPN ML notation, we declare the colour sets, functions, variables, and constants that will be used in the net inscriptions of the CPN model; b) we build a top-level model in which the protocol entities are modeled as substitution transitions; c) we define the substitution transitions from the top-level model.

2. Add the intruder to the model: In this step, a) we extend the CPN declarations to include the intruder; b) add the intruder transition to the top-level model; c) define the intruder substitution transition.

3. Implement a token-passing scheme.

4. Specify security requirements stated in terms of CPN markings.

5. Analyse the resulting occurrence graph by using OG queries to locate markings that violate a security requirement.

3 The Technique

In this section, we first present our sample protocol, TMN protocol, and then, using our technique, we propose a model of this protocol. The selection of the TMN protocol to illustrate our technique is motivated by its familiarity. Any other protocol listed in [7] can be used for the illustration of our technique.
3.1 The TMN Protocol

The Tatebayashi, Matsuzaki, and Newman (TMN) protocol is a key exchange cryptographic protocol for mobile communication systems. The protocol involves two entities, \(A\) and \(B\), and a server, \(J\), to facilitate the distribution of a session key, \(K_{AB}\). The attack illustrated in this paper is a known one. The reader can find very similar attacks in [7, 8]. Moreover, three other known attacks on this protocol are given in [7].

Initially, the TMN protocol assumes that both \(A\) and \(B\) know the public key of \(J\), \(K_P^B J\). We use the following notation: \((i)\) \(A \rightarrow B : X\) to indicate that in the \(i\)th step of the protocol agent \(A\) sends message \(X\) to agent \(B\). We write \(A \rightarrow B : X, Y\) to denote “\(A\) sends \(B\) the message \(X\) along with the message \(Y\)”.

The protocol proceeds as follows:

1. \(A \rightarrow J : B, K_P^B J(K_{AJ})\)
2. \(J \rightarrow B : A\)
3. \(B \rightarrow J : A, K_P^B J(K_{AB})\)
4. \(J \rightarrow A : B, K_{AJ}(K_{AB})\)

When \(A\) (the initiator) wants to start a session with \(B\) (the responder), \(A\) chooses a key \(K_{AJ}\), encrypts it using the public key of \(J\) \(K_P^B J\), and sends it along with the identity of \(B\) to the server \(J\) (step 1). Upon receiving the first message, the server decrypts \(K_P^B J(K_{AJ})\) using its private key \(K_{Pr}^B J\) and obtains \(K_{AJ}\). Then, in the second step, \(J\) sends a message to \(B\) containing the identity of \(A\). When \(B\) receives this message, it chooses a session key \(K_{AB}\), encrypts it using \(K_P^B J\), and sends it along with the identity of \(A\) to \(J\) (step 3). Upon receiving the third message, the server decrypts \(K_P^B J(K_{AB})\) using its private key \(K_{Pr}^B J\) and obtains \(K_{AB}\). Then, \(J\) sends to \(A\) the key \(K_{AB}\) encrypted under \(K_{AJ}\) along with the identity of \(B\) (step 4). When \(A\) receives this message, it decrypts it using the key \(K_{AJ}\) to obtain the session key \(K_{AB}\).

The keys \(K_{AJ}\) and \(K_{AB}\) are symmetric keys freshly created by \(A\) and \(B\), respectively. The key \(K_{AJ}\) must be known only to \(A\) and \(J\); and is used to send \(K_{AB}\) in an encrypted form as indicated in step 4 of the protocol. The key \(K_{AB}\) must be known only to \(A\), \(B\) and \(J\), and it is used as a session key. Thus, \(A\) uses \(K_{AB}\) to encrypt messages it sends to \(B\), and vice versa. When the communication session between \(A\) and \(B\) is over, \(K_{AB}\) is discarded. A new session key is used in every protocol run.

3.2 Modeling the Protocol with no Intruder

CPN ML Declarations

In our modeling of cryptographic protocols, messages are composed of fields. Some of these fields are atomic, they include entity identities, keys, and nonces. The other fields are constructed from the atomic fields. For instance, a cipher \(K(A)\) can be viewed as an ordered pair \((A, K)\), where \(A\) is the identity and \(K\) is the encryption key.

4
For the TMN protocol, we define the following:

1. Colour sets:

   a) The atomic fields are the set of identities, \( I = \{A, B\} \), and the set of keys, \( K = \{K_{AB}, K_{AJ}, K_{Pb}^p, K_{Pr}^p\} \).

   b) All ciphers have the same format: \( k_1(k_2) \), where \( k_1 \) and \( k_2 \) are of type \( K \). For instance, \( K_{Pb}^p(K_{AJ}) \) is the cipher of the first message, etc. Thus, the cipher colour set \( C \) is defined as \( C = K \times K \).

   c) Messages are generally composed of an identity and a cipher. For instance, in the TMN protocol, the first message is \( (B, K_{Pb}^p(K_{AJ})) \) and the third message is \( (A, K_{Jp}^p(K_{AB})) \). Thus, the message colour set \( M \) is defined as \( M = I \times C \). Note that the second message of the protocol only includes an identity. Hence, \( I \) is used as the colour set for such messages.

   d) The TMN protocol implicitly assumes that \( J \) knows the originator of the first message it receives. To model this, we define the colour set \( MI = M \times I \). Thus, the first message that \( J \) receives is actually composed of two fields: the message contents, \( (B, K_{Pb}^p(K_{AJ})) \), and the sender’s identity \( A \).

   e) We use a special colour set \( E = \{e\} \) to prevent an infinite number of transition firings. For instance, by using a construct such as in Figure 1, we force the transition \( T \) to fire at most once. Using such constructs is needed whenever a transition has double input arcs. As we show later, the labels of double arcs indicate tokens inherent to an entity, or tokens that are to be used in subsequent subtasks performed by the entity.

![Figure 1](image-url)  
Figure 1: By using the \( E \) set, transition \( T \) can fire at most one time.

2. Variables: We use variables of the defined colour sets as inscriptions for arcs of the CPN. They are: \( k_1, k_2, \) and \( k \) of type \( K \), \( c \) of type \( C \), \( i \) of type \( I \), and \( m \) of type \( M \).

3. Functions:

   a) The function \( \text{DecryptionKey}(k : K) \) returns the decryption key of a given key \( k \). 

---

5
b) The function $\text{SharedKey}(i : I)$ returns the shared key between entity $B$ and the entity $i$. For instance, $\text{SharedKey}(A)$ is $K_{AB}$. We use this function to model the behaviour of $B$ in which it generates a session key based on the initiator’s identity, as shown in step 2 of the protocol.

The TMN protocol declarations are given in Figure 2 using $CPN ML$ notation.

```plaintext
color I = with A | B;
color K = with Kaj | Kjp | Kjpr | Kab;
color C = product K*K;
color M = product I*C;
color MI = product M*I;
color E = with e;
var k1,k2,k:K;
var c:C;
var i:I;
var m:M;
fun DecryptionKey(k:K):K = case k of Kaj=> Kaj
| Kjp => Kjpr | Kjpr => Kjp;
fun SharedKey(i:I):K= case i of A => Kab;
```

Figure 2: The declarations for the TMN model

The Top-Level Model

The computer tool $Design/CPN$ supports hierarchical net construction. This makes it possible to model cryptographic protocols in a modular way. Thus, the model of a protocol is constructed by using sub-models of its agents. In CP-nets, this is implemented by using substitution transitions.

First, we focus on the messages exchanged between the protocol entities. At this level, protocol entities are modeled as transitions. Figure 3 shows a top-level model of the TMN protocol. This net is described as follows:

1. Transition $T1$ represents entity $A$. In the first step of the protocol, $A$ generates a token of type $MI$. This corresponds to the first message that $A$ sends to $J$, along with the identity of $A$ to inform $J$ about the initiator. In the last step of the protocol, $A$ consumes a token of type $M$.

2. Transition $T2$ represents entity $J$. In the first step of the protocol, $J$ consumes a token of type $MI$. Then, $J$ sends a token of type $I$, modeling the second protocol step. In the third step of the protocol, $J$ consumes a token of type $M$ generated by $B$. Finally, $J$ generates a token of type $M$ modeling the last message of the protocol.

3. Transition $T3$ represents entity $B$. Entity $B$ consumes a token of type $I$ in the second step, and generates a token of type $M$ in the third step of the protocol.
Figure 3: The TMN top-level model with no intruder

Defining the Top-Level Substitution Transitions

Due to space constraints, we consider in detail the model of the initiator $A$ but we refer the reader to [1] for the models of entities $B$ and $J$. The following is an informal description for the behaviour of $A$ which is the initiator of the communication session. Thus, $A$ always sends the message $B, K_B^p(K_{AJ})$. The reply $A$ receives is in the format $(i, c)$, where $i$ is an identity and $c$ is a cipher. Entity $A$ checks that $i = B$. If this is true, $A$ decrypts $c$ with $K_{AJ}$. If $c$ was decrypted with $K_{AJ}$, $A$ accepts the received session key, and uses it for communication with $B$ in the current session.

Figure 4 shows the CPN model of entity $A$. It contains two subnets: one models the subtask of $A$ initiating a protocol run in step 1, while the second models the subtask of $A$ receiving the last message from $J$.

Port assignments are used to relate the top-level page, named TMN, with the entity models. As the port assignments for the substitution transition of $A$ show (Figure 3), the socket $P_1$ is related to the output port $P_8$ of $EntityA$, while the socket $P_4$ is related to the input port $P_9$ of $EntityA$.

In $EntityA$, we use the instance fusion sets $B = \{P_1, P_{10}\}$ and $Kaj = \{P_2, P_{12}\}$. Fusion sets are used to allow an entity to control the order of subtasks and check the validity of messages. For instance, $A$ has to remember the key it chooses ($K_{AJ}$) in the first step, in order to decrypt the cipher it receives in the last step.
3.3 Modeling the Protocol with an Intruder

The intruder is modeled as a separate entity that controls the communication channels between the protocol entities. Thus, it intercepts the exchanged messages and stores them for future use. Then, it attempts to decrypt the encrypted portions of the intercepted messages. Finally, it attempts to modify the message contents, or even generate new messages to replace the intercepted ones.

Extending the CPN ML Declarations

In order to add the intruder to the model, one must extend the CPN ML declarations. The identity of the intruder $In$ is added to the colour set $I$. Also, an intruder key $Ki$ is added to the colour set $K$. The $DecryptionKey$ and $SharedKey$ functions are extended to handle the new colours: $DecryptionKey(Ki) = Ki$ and $SharedKey(In) = Ki$.

During the execution of the protocol, the intruder stores the intercepted messages for future use. We model the intruder memory as a global fusion set that we call the DB fusion set (DB stands for database). We refer to a place that is a member of the DB fusion set as a $DB$-place.

A $DB$-place is expected to hold tokens of atomic and non-atomic types. In the TMN protocol, a $DB$-place should hold keys, identities, and ciphers. Thus, we define the DB colour set as $DB = I \cup K \cup C$, and we use DB as the colour set of
a DB-place.

In CPN ML, DB is declared as follows: \( \text{color DB} = \text{union cI:I + cK:K + cC:C} \); Here, \( cI \), \( cK \), and \( cC \) are selectors [10]. Thus, the intruder’s possession of \( K_{AB} \) is modeled as reaching a marking where a token \( cK(Kab) \) is in a DB-place. The reader can find the final CPN ML declarations in [1].

**The Top-Level Model with an Intruder**

Figure 5 shows the top level model of the TMN protocol with an intruder. The substitution transition \( T4 \) represents the intruder, which was not included in the earlier top level model given in Figure 3.

![Figure 5: The TMN top-level model with an intruder](image)

Each place in Figure 3 is replaced with two corresponding places as shown in Figure 5: one is an input place to the intruder while the second is an output place. For instance, place \( P1 \) in Figure 3 is replaced with \( P1 \) and \( P2 \) in Figure 5. This is needed to model the intruder’s ability to receive a message (the input place), to deal with it (transition \( T4 \)), and to substitute it with a new message (the output place).

**Defining the Intruder Substitution Transition**

The intruder substitution transition (\( T4 \) in Figure 5) is defined by the subpage *intruder* shown in Figure 6.

The intruder model is constructed by using several intruder subprocesses. Each intruder subprocess models the intruder’s possible actions to intercept tokens
Table 1: The intruder subprocesses

<table>
<thead>
<tr>
<th>Pair Places</th>
<th>Colour Set</th>
<th>The Corresponding Intruder Subprocess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Output</td>
<td>Set</td>
</tr>
<tr>
<td>P1</td>
<td>P2</td>
<td>MI</td>
</tr>
<tr>
<td>P3</td>
<td>P4</td>
<td>I</td>
</tr>
<tr>
<td>P5</td>
<td>P6</td>
<td>M</td>
</tr>
<tr>
<td>P7</td>
<td>P8</td>
<td>M</td>
</tr>
</tbody>
</table>

that belong to a given colour set (type). Table 1 lists the intruder subprocesses, along with their input/output places.

![Figure 6: The intruder page](image)

The intruder subprocesses `intruder_mi`, `intruder_m`, and `intruder_i` are defined in separate pages. The `intruder` page has one instance of `intruder_mi` (which defines $T_1$), one instance of `intruder_i` (which defines $T_2$), and two instances of `intruder_m` (which define $T_3$ and $T_4$).

The `intruder_m` subprocess is given in Figure 7. It models what an intruder can do to intercepted tokens of type $M$. A token of type $M$ has two fields: an identity and a cipher. The intruder first stores these fields of the intercepted token. Then, it tries to decrypt the cipher using one of the keys stored in its database. Finally, the intruder forms a new message to be sent in place of the intercepted one. The intruder uses one of the ciphers stored in the database, or constructs a new cipher by using keys stored in the database.

The `intruder_i` subprocess models what an intruder can do to intercepted
tokens of type $I$. It is constructed in a similar manner as $intruder_m$ (for details, see [1]). The $intruder_mi$ subprocess models what an intruder can do to intercepted tokens of type $MI$. A token of type $MI$ has two fields: an identity and a message. Thus, $intruder_mi$ can be constructed using instances of $intruder_i$ and $intruder_m$, as shown in Figure 8. An instance of $intruder_i$ is used to handle the identity field, and an instance of $intruder_m$ is used to handle the message field.

The last step in defining the intruder is to specify its initial knowledge. One specifies the initial intruder knowledge by setting the initial marking of a DB-place. As the initial marking of $P4$ in $intruder_m$ indicates (Figure 7), the $DB$ is set initially to \{$K_I, K_Pb, A, B, In$\}.

### 3.4 Applying a Token-Passing Scheme

Using our technique as outlined up to this point, most models of cryptographic protocols result in a large occurrence graph. The large size of the occurrence graph can be explained by two aspects of the model: the nondeterministic behaviour of the intruder, and the interleaved subprocesses.

The intruder model is nondeterministic in the sense that there are many possible actions the intruder can take at a given time. For instance, assume that in the TMN model the intruder has the keys $K_I$ and $K_Pb$, and it has three identities: $A$, $B$, and $I$. Then, there are 12 possible messages $(i, c)$ the intruder can use. Each
choice will have different implications in terms of the resulting markings.

The second factor attributing to the size of the occurrence graph is the interleaving of subprocesses. Transitions of an entity and the intruder instances can be interleaved, causing an unnecessary increase in the size of the occurrence graph. For instance, consider a state where transition $T_1$ of $EntityA$ has not yet fired. At this state, many transitions of the intruder instances are enabled. The different order of firing such transitions will result in different markings and paths in the occurrence graph. The same thing happens after firing $T_2$ of $EntityA$, etc.

Let $E$ be the set of finite occurrence sequences of the possible execution of the agents $A$, $B$, and $J$. For every sequence $e$, the intruder observes the set $S_e$ of relevant information (keys, messages, and agent identities) that are carried by $e$. Let $R = \{(e_1, e_2) \mid e_1 \in E \land e_2 \in E \land S_{e_1} = S_{e_2}\}$. The relation $R$ is an equivalence relation. It is clear that interleaving of subprocesses belong to the same $R$-equivalence class as their sequential execution. Hence, there is no need to include all the the unnecessary interleaving of subprocesses in the occurrence graph.

The model can be extended to prevent the unnecessary interleaving of subprocesses. The goal is to allow a single subprocess to be enabled at a given time. This is achieved using a token-passing scheme. For instance, if $EntityA$ has the token, no transitions from other subprocesses should fire. This results in a reduction in the size of the occurrence graph.

We note that applying the token-passing scheme does not restrict the model assumptions. This is because it is assumed that an intruder would not obtain more knowledge by the simultaneous execution of protocol entities than it would by the interleaving of such executions. In other words, true concurrency is assumed not to affect properties of cryptographic protocols.

To apply the token passing strategy, a new colour set is defined, $S = \{s\}$. We will refer to a place of colour $S$ as an $S$-place. The token $s$ is the token exchanged among entities.
The following rules are the changes required to apply this scheme.

1. Add an input $S$-place to every substitution transition in the top level page. Similarly, add an output $S$-place from every substitution transition in the top level page. All of these $S$-places should be added to a single instance fusion set. Thus, there is one resulting $S$-place. It must be initialised with one $s$-token. This rule is demonstrated in Figure 9.

2. Add an $S$-place input port and an $S$-place output port to every subpage. The input port should have an outgoing arc to the first transition in every subprocess of the subpage. Similarly, the output port should have an incoming arc from the last transition in every subprocess of the subpage. Figure 10 shows the application of this rule to the EntityA page.

3. Applying the first two rules does not prevent the intermediate intruder transitions from firing. These are the transitions that have double input arcs coming from DB-places, e.g. transitions $T2$ and $T3$ in intruder$_m$. We must allow these transitions to fire only when the corresponding subprocess has the $s$-token. To apply this, we create an instance fusion set $S$, in every intruder subpage, to hold the $s$-token that is passed to the active intruder subprocess. To be more precise, a) we add an output arc from the first transition of the intruder subpage to a place that belongs to the fusion set $S$; b) we add an input arc from a place that belongs to the fusion set $S$ to the last transition of the intruder subpage; and c) we add double arcs from a place that belongs to the fusion set $S$ to the intermediate intruder transitions.
These changes are demonstrated in Figure 12. For example, transitions $T_2$ and $T_3$ of $intruder.m$ will not fire until the $s$-token arrives to the subprocess, which means transition $T_1$ fires, consuming the $s$-token from the input port $SP_1$. When the $s$-token is returned back by the intruder subprocess (i.e. transition $T_4$ fires and the $s$-token is deposited back to the output port $SP_2$), transitions $T_2$ and $T_3$ become disabled.

Note that the intruder intermediate subpages, e.g. $intruder.mi$, must be extended to pass the received token to the lower level subpages. This is demonstrated in Figure 11.

The application of these rules to the pages $EntityB$, $EntityJ$, $intruder$ and $intruder.i$ is provided in [1].

3.5 Identifying Security Requirements

Before simulating the model, one needs to identify the security requirements that must be met by the protocol. These requirements should be stated in terms of conditions on the CPN markings.

We consider the following requirement. The protocol must guarantee the secrecy of the session key $K_{AB}$. Thus, in a given session, $K_{AB}$ must be known only by $A$, $B$, and $J$. In other words, the intruder should never know $K_{AB}$. In
terms of CPN markings, this translates into the requirement that a token with colour $Kab$ never reaches a DB-place.

Other security requirements that the TMN protocol aims to satisfy are discussed and verified in [1].

### 3.6 Analysing the Occurrence Graph

The final step in the analysis of the model is to construct and analyse the occurrence graph. We use the OG tool in Design/CPN to automate this process. The goal is to find nodes (markings) that violate a security requirement.

We use the Occ Menu to invoke commands related to the occurrence graph [11]. Given the CPN model for a cryptographic protocol, we construct the full occurrence graph, and then run CPN queries to find the insecure markings.

The security requirement that we consider states that a token with colour $Kab$ never reaches a DB-place. In CPN ML, we use the following predicate:

$$ fn\ n\ =>\ cf(cK(Kab),\ Mark.intruder.m .P4\ 1\ n)\ >0. $$

Given a marking $n$, this predicate evaluates to true if the DB-place $P4$ of $intruder.m$ (first instance) contains at least one token $cK(Kab)$, and evaluates to false otherwise. Note that $cf$ is the coefficient function [10]. It takes two arguments: a colour and a multiset of tokens, and returns the coefficient of the specified colour in the specified multiset. For instance, $cf(A, 5A)$ returns 5. Thus, $cf(cK(Kab),\ Mark.intruder.m .P4\ 1\ n)$ returns the coefficient of $cK(Kab)$ in the multiset of tokens in $P4$ of the first instance of $intruder.m$ in marking $n$.

The following function returns all nodes of the occurrence graph where the DB-place has at least one token $cK(k)$. It uses the predicate defined above.

$$ \text{fun SecrecyViolation1}(k:K): $$
Figure 12: The `intruder.m` page after adding S-places

Node list = PredAllNodes (fn n => cf(cK(k), Mark.intruder.m’P4 1 n) >0);

Thus, SecrecyViolation1(Kab) returns all nodes of the occurrence graph that violate the considered security requirement.

The full occurrence graph generated for the model has 19,237 nodes and 22,419 arcs. It took 19 seconds to construct the occurrence graph using a 1-GHz, 16GB machine.

Executing SecrecyViolation1 returns a non empty node list. One of the nodes returned by SecrecyViolation1 is node 19170. We use the Design/CPN Occurrence Graph (OG) tool to find a path from the initial marking (node 1 in the OG) to the insecure marking (node 19170). This path is represented by the following occurrence sequence. Each line in the occurrence sequence represents a step that has a single binding element. Each line contains the following information: the page name, the instance number (if missing, then there is a single instance), the transition, and the binding. For instance, the line identified by (*), on its right side, represents the step (T2 in the first instance of `intruder.m`, (k1 = Kab, k2 = Ki)).

EntityA T1 k1 = Kaj, k2 = Kjp
EntityA T2 i = B, c = (Kaj, Kjp)
EntityA T3 i = A, m = (B, (Kaj, Kjp))
intruder mi T1 \( m = (B, (K_{aj}, K_{jp})), i = A \)
intruder i 1 T1 \( i = A \)
intruder i 1 T2 \( i = A \)
intruder m 1 T1 \( i = B, c = (K_{aj}, K_{jp}) \)
intruder i 2 T2 \( i = A \)
EntityB T1 \( i = A \)
EntityB T2 \( i = A, k2 = K_{jp} \)
intruder m 2 T1 \( i = A, c = (K_{ab}, K_{jp}) \)
intruder m 1 T3 \( k1 = K_{ij}, k2 = K_{jp}, i = A \)
intruder m 1 T4 \( i = A, m = (A, (K_{i}, K_{jp})) \)
EntityJ T1 \( i = A, m = (A, (K_{i}, K_{jp})) \)
intruder m 2 T2 \( i = A, c = (K_{i}, K_{jp}) \)
intruder m 2 T3 \( k1 = K_{ij}, k2 = K_{jp} \)
intruder m 2 T4 \( i = A, k = K_{i} \)
intruder i 2 T1 \( i = A \)
intruder m 2 T4 \( i = A, c = (K_{ab}, K_{jp}) \)
EntityJ T5 \( i = A, c = (K_{ab}, K_{jp}) \)
intruder m 2 T6 \( k1 = K_{ab}, k2 = K_{jp} \)
intruder m 2 T7 \( k1 = K_{ab}, k2 = K_{ij} \)
intruder m 2 T8 \( i = A, c = (K_{ab}, K_{i}) \)
intruder_m 1 T1 \( i = A, c = (K_{ab}, K_{i}) \)
intruder m 1 T2 \( k1 = K_{ab}, k2 = K_{ij} \) (*)
intruder m 1 T2 \( k1 = K_{ab}, k2 = K_{ij} \)
intruder m 3 T1 \( i = A, c = (K_{ab}, K_{i}) \)
intruder m 1 T3 \( k1 = K_{ab}, k2 = K_{ij} \)
intruder m 3 T3 \( i = B, k1 = K_{i}, k2 = K_{jp} \)
EntityA T4 \( i = B, c = (K_{i}, K_{jp}) \)

Note the reachability of \( K_{ab} \) to a DB-place in the step identified by (*). This attack is stated in a high level description as follows, noting that \( I(A) \) denotes I impersonating \( A \). Thus, a step of the form “\( I(A) \rightarrow B : X \)” means that I poses as \( A \) and sends \( X \) to \( B \), whereas a step of the form “\( B \rightarrow I(A) : X \)” means that I intercepts the message \( X \); originally sent from \( B \) to \( A \).

\[
\begin{align*}
(I1) \; & A \rightarrow I(J) : B, K_{pp}^{i}(K_{AJ}) & (II2) \; & J \rightarrow I(A) : A \\
(I2) \; & I(J) \rightarrow B : A & (II3) \; & I(A) \rightarrow J : A, K_{pp}^{j}(K_{AB}) \\
(I3) \; & B \rightarrow I(J) : A, K_{pp}^{j}(K_{AB}) & (II4) \; & J \rightarrow I(A) : A, K_{i}(K_{AB}) \\
(II1) \; & I(A) \rightarrow J : A, K_{pp}^{j}(K_{i})
\end{align*}
\]

This attack involves two separate runs of the protocol; labelled \( I \) and \( II \). At the end of \( II4 \), the intruder decrypts \( K_{i}(K_{AB}) \) to obtain \( K_{AB} \). Thus, the intruder is able to impersonate \( A \). Note the replay of \( K_{pp}^{j}(K_{AB}) \) in step \( II3 \) from \( I3 \).
4 Discussion

In this paper we have presented a promising technique that uses coloured Petri nets for the verification of cryptographic protocols.

Our technique compares well with other finite-space methods [8, 9]. It includes the same verification assumptions. The same approach of reachability analysis is used. The generated number of states is acceptable compared with other methods. Furthermore, Design/CPN fits well to our technique, with several advantageous features such as the ability to control the construction of the occurrence graph and the ability to stop searching when certain criteria are met. In other terms, the capabilities of Design/CPN enable us to grasp the theoretical power of CP nets in practice for dealing with complex systems. The state explosion problem can be slightly managed using for instance a token-passing scheme, but not significantly reduced. In [3], the sweep-line method is introduced to reduce both the space and the time used during state space exploration. One avenue for future investigation would be to apply this method in the exploration of the state space of more complex protocols modeled using the technique proposed in this paper.

There are two features in our technique that facilitate the construction of the intruder model for cryptographic protocols. The first feature is the use of a DB-place to hold all intercepted tokens. The second feature is that the intruder model is constructed by using several intruder subprocesses, where each intruder subprocess is defined based on the colour of the intercepted token. For instance, if the intercepted token is an identity, then the intruder first stores it and then it replays any other identity it possesses. If the intercepted token is a cipher, the intruder has the ability to try to decrypt the cipher and to form new ciphers. The net result of this is clarity and simplicity of the intruder model, and the ability to construct the intruder model in a systematic way.

Finally, the model presented for the TMN protocol involves a single instance of each entity. Thus, an attack that involves multiple instances of a given entity in multiple runs will not be captured under this restriction. Our model can easily be extended to include more than one instance of a given entity by adding tokens to the entity’s $E$-places. However, this would result in a dramatic increase in the size of the occurrence graph. This problem also arises in other finite-state methods. In such cases, analytic methods are applied to avoid generating the full reachability tree. For the case of CP-nets, methods such as the matrix equation [5] seem to be useful. Other techniques to yield a reduced representation of the occurrence graph are applicable. These include the stubborn set method [6], and occurrence graphs with equivalence classes [5].

References


