Nordic Journal of Computing

MODELING AND VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS USING COLOURED PETRI NETS AND *Design/CPN*

Issam Al-Azzoni Douglas G. Down
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.

ACM CCS Categories and Subject Descriptors: D.4.6 [Security and Protection]: Authentication, Verification; D.2.2 [Design Tools and Techniques]: Petri nets; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages; C.2.2 [Network Protocols]: Protocol verification;

Key words: 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 communication systems. They are used in the Internet and in wired and wireless networks 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 [5, 11].

Generally, a cryptographic protocol involves two communicating agents who exchange a few messages, with the help of a trusted server. The exchanged messages are composed from components such as keys, random numbers, timestamps, and signatures [42, 44]. At the end of the protocol, the agents involved may deduce certain properties such as the secrecy and authenticity of an exchanged message [41].

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 attacks [41]: prevent a message from being delivered, copy messages, intercept a message by preventing it from reaching its destination and making a copy, fake a message, modify

Received 26th October 2005.

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.

1.1 Literature review and motivation

The verification of cryptographic protocols has gained a lot of interest in the research community, due to several factors. First, these protocols play a major role in the security of communication systems. Second, although these protocols are simple looking (only a few lines), they are extremely difficult to verify. Finally, such protocols are excellent candidates for formal analysis methods. In fact, most of the ongoing research about cryptographic protocols is on *formal methods* of verification. Formal methods used for the analysis of cryptographic protocols can be classified as follows:

1) *Methods based on logic*: these methods build a logic model for the protocol, and reason in terms of logical propositions. Such methods include *BAN logic* [5], *GNY logic* [17], and *MAO logic* [30].

2) *Methods based on algebra*: these methods involve modeling the protocol as an algebraic system, and reason in terms of the algebraic properties of the model. Such methods include the *CSP algebra* [41].

3) *Methods based on state machines*: these methods involve modeling the protocol in terms of a general modeling tool that enumerates the state space, and then analyzing the model in terms of state invariants. Such methods include *Inajo* [23], and *NRL Analyzer* [32].

A good survey of the different formal methods for verification of cryptographic protocols is provided in [31], and [40].

There is no one method that can be used to model all aspects of cryptographic protocols, and thus detect all types of flaws [31]. The best a formal method can do is to guarantee that a security property is satisfied by a certain cryptographic protocol, given that a set of assumptions hold. For instance, one of the assumptions all formal methods make is that secure cryptographic algorithms are used. Usually, more than one formal method is used to prove different security properties of a protocol.

The following lists a few aspects in which formal methods may differ:

1) Automated tools: several formal methods have a computerized tool to help in the analysis. Most of the state machine based methods use an automated tool to construct and analyze the state space. On the other hand, logic based methods are hard to automate since they involve non-trivial proofs.

2) Proof abilities: some methods, especially those based on logic and algebra, can be used to formally prove that a security property is satisfied by a given cryptographic protocol [15]. Such methods state the properties of intruder actions and reason in terms of deduction rules. Other methods, most of those based on state machines, are geared toward determining the existence of certain flaws rather than guaranteeing that flaws do not exist in a given cryptographic protocol [18]. Such methods require explicitly stating the possible intruder attacks. Thus, they will not be of any help in detecting attacks not included in the model.

3) Systematic approach: protocol analyzers may find certain methods more systematic than others in constructing the required model of the cryptographic protocol. For instance, a protocol needs to be converted to an idealized form before being analyzed under the BAN logic. Rubin and Honeyman claim that "there is no clear transformation method presented" [40] to convert a protocol into an idealized form. This can be attributed to the fact that the idealization step depends on the verifier's understanding of the protocol as well as its assumptions.

In this paper, we explore the use of coloured Petri nets [19, 20] in the verification of cryptographic protocols. In [19, page 69], Jensen states that these nets are analogous to directed graphs and non-deterministic finite automata. Therefore, from the perspective of the classification of formal methods given on page 2, the techniques that are based on coloured Petri nets belong to the class of methods based on state machines.

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.

In the literature, we find many papers where the authors report on projects that investigated the practicability of using CP-nets and the CPN tools for the specification, verification, validation, or performance analysis of the considered system. For examples of industrial use of CP-nets and *Design/CPN*, we refer the reader to [8, 7, 6], and [21, Chapter 7]. In [47], Wheeler describes a technique to use CP-nets and the CPN tools to model and analyse a sub-network reconfiguration protocol for a Metropolitan Area Network. In [36], Mnaouer et al. report on the usage of CP-nets and the CPN tools in modeling the centralised architecture of the timed-token Fieldbus protocol. In [9] and [21, Chapter 3], the authors outline how they used CP-nets and the CPN tools to verify a protocol used in audio/video

systems. de Figueiredo et al. [14] report on a part of a large modeling project conducted in co-operation with Hewlett-Packard. They give the models and the analysis performed on a number of variants of the TCP protocol. In [25, 26], we find applications of CP-nets and their supporting computer tools in the development of communication protocols for interconnecting backbone networks and mobile ad-hoc networks. An exhaustive summary on the usage of CP-nets as well as CPN tools in a variety of areas can be found at the website [13].

During the 1990s, several researchers applied coloured Petri nets in the verification of cryptographic protocols [4, 38, 43, 3]. The techniques developed by these researchers use a form of coloured Petri nets that is lower level than Jensen's CP-nets [19]. For instance, the following features have not been used: arc inscription, guard expression, CPN/ML statements, fusion places, and functions on the values of the coloured tokens. Having such features would result in having smaller, easier to understand, and extendable models.

Furthermore, the computer tool *Design/CPN* [12, 33] has not been explored as a potential automated verification tool to deal with cryptographic protocols. 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 presented in [4, 38, 43]. We focus on benefiting from the high level constructs of Jensen's coloured Petri nets, as well as using *Design/CPN*.

1.2 Background

A *Petri net* is a directed, weighted, bipartite graph consisting of two kinds of nodes: places and transitions. It is *formally defined* [37] as a 5-tuple, (P, T, F, W, M_0), where: P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, $W : F \to \{1, 2, 3, \ldots\}$ is the weight function, and $M_0 : P \to \{0, 1, 2, \ldots\}$ is the initial marking.

There are many varieties of coloured Petri nets. We, however, are concerned with one particular variety: *Jensen's coloured Petri nets* [19]. From now on, we refer to Jensen's coloured Petri nets as *CP-nets*; *CPN* refers to a single Jensen's coloured Petri net. CP-nets use *CPN ML* as an inscription language. CPN ML is an extended version of *ML* (Meta Language) which is a popular functional language standard [39, 46]. This is what distinguishes CP-nets from other varieties of coloured Petri nets [19].

1.2.1 Jensen's Coloured Petri Nets

CP-nets are one of many different modeling languages. Here, we provide a formal definition for non-hierarchical CP-nets [19].

A *multi-set ms* over a domain X is a function that maps each element $x \in X$ into a number $ms(x) \in \mathcal{N}$. We represent a multi-set as a formal sum $\sum_{x \in X} ms(x)'x$, where $ms(x)$ is the number of occurrences of x in ms. $ms(x)$ is called the coefficient of x in ms. For example, $ms1 = 2'a + 1'c$ and $ms2 = 3'b+1'c$ are two multi-sets defined on $\{a, b, c\}$. Note that $|ms|$ denotes the size of the multi-set ms, $|ms| = \sum_{x \in X} ms(x)$, *e.g.* $|ms1| = 3$. Use use X_{MS} to denote the set of all multi-sets over X.

A *CPN* [19] is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where Σ is a set of non-empty types (*colour sets*), P is a set of places, T is a set of transitions, and A is a set of arcs. $N : A \longrightarrow P \times T \cup T \times P$ is a node function that maps each arc $a \in A$ into a pair ($source(a)$, $destination(a)$), where $source(a)$ and $destination(a)$ are the source and destination nodes of a, respectively. Given an arc a, the source and destination nodes of a must be of different types, *i.e.* one must be a place while the other is a transition. C is a colour function that maps each place p into a colour set $C(p)$ specifying the type (colour set) of tokens that can reside in p. $G : T \longrightarrow \mathcal{B}$ is a guard function that maps each transition t into a boolean expression (a predicate) $G(t)$. E is an arc expression function that maps each arc a into an expression $E(a)$, which must be of type $C(p(a))_{MS}$ where $p(a)$ denotes the place of $N(a)$. Finally, I is the initialization function which maps each place p to a multi-set $I(p)$ of type $C(p)_{MS}$ specifying the initial marking of the place p.

An example of a Design/CPN model is provided in [22]. A comprehensive guide to the practical use of CP-nets and Design/CPN can be found in [24]. Analysis methods and theoretical background are provided in [16, 19, 20, 37].

The set of variables of a transition t is denoted $Var(t)$. $Type(v) \in \Sigma$ denotes the type of v. A *binding element* (t, b) is a pair consisting of a transition t and a binding b of values to all of the variables of t such that the evaluation of the guard $G(t)$ returns true. We write binding elements in the form $(t, \langle v_1 = c_1, \cdots, v_n = c_n \rangle)$ where $Var(t) = \{v_1, \cdots, v_n\}$ and $c_1, \cdots, c_n$ are colours (data values) such that $c_i \in Type(v_i)$ for $1 \leq i \leq n$. The initial marking of a CPN is denoted by M_0. $M(p)$ denotes the marking of a place p in a given marking M. A *step* Y is a non-empty and finite multi-set of binding elements. It is represented by listing the pairs $(t, Y(t))$ where $Y(t) \neq \emptyset$. $Y(t)$ is the multi-set of bindings for t in Y. A step Y is *enabled* in a given marking M if all of its binding elements are enabled in M. A binding element (t, b) is *enabled* in a marking M if each input place p of t is marked with at least the multi-set $E(a)\langle b \rangle$ of tokens, where a is the arc whose source node is p and destination node is t. $E(a)\langle b \rangle$ returns the multi-set of tokens that results from evaluating the arc expression $E(a)$ using the binding b. A transition t is *enabled* in a marking M if there is an enabled binding element (t, b), with a binding b, in M. In this case, we say that t is enabled in M for the binding b. When a step Y is enabled in a marking M_1, it may *occur*. If it occurs, it changes the marking M_1 to another marking M_2 by removing tokens from the input places and adding tokens to the output places, as determined by the arc expressions evaluated for the step bindings. In this case, we say that M_2 is *directly reachable* from M_1 by the occurrence of the step Y, we denote this by $M_1[Y\rangle M_2$. A *finite occurrence sequence* is a sequence of markings and steps $M_1[Y_1\rangle\ M_2[Y_2\rangle M_3 \cdots M_n[Y_n\rangle M_{n+1}$ where $n \in \mathcal{N}$ and $M_i[Y_i\rangle M_{i+1}$ for $i \in 1, \cdots, n$. An *infinite occurrence sequence* is an infinite sequence of markings and states $M_1[Y_1\rangle M_2[Y_2\rangle M_3 \cdots$, such that $M_i[Y_i\rangle M_{i+1}$ for

$i \in \mathcal{N}^+$. A marking M'' is *reachable* from a marking M' if there exists a finite occurrence sequence having M' as an initial marking and M'' as a final marking.

1.2.2 Hierarchical CP-nets

Hierarchical nets allow us to construct large CP-nets by combining smaller nets. Two CPN constructs exist that allow us to create hierarchical nets: substitution transitions and fusion sets. A non-hierarchical CPN is called a *page*. In *Design/-CPN*, each page has a *page name* and a *page number*. In this paper, we use the page name to refer to a page. A *substitution transition* is a transition which is described in more detail in a separate CPN page. The page with the substitution transition is called the *superpage*. The page that defines a substitution transition is called a *subpage*. Each substitution transition corresponds to an *instance* of the subpage it is related to. Information on how a superpage is glued to a subpage is provided in the *port assignment*. The port assignment for a substitution transition assigns a place from the subpage (called *port*) to a place in the superpage (called *socket*). We note that, in Design/CPN, if the socket and its related port have identical names, then their assignment is not shown in the hierarchical inscriptions. In our work, we consider two types of ports: *input* and *output* ports. An input port is a place in the subpage that must be related to a socket that is an input place to the corresponding substitution transition. On the other hand, an output port is a place in the subpage that must be related to a socket that is an output place from the corresponding substitution transition. To enable a specifier to state that a set of places are considered to be identical, the notion of *fusion of places* is used. The set of such places is called a *fusion set*. The places that participate in a fusion set may therefore belong to one or more pages.

1.2.3 Petri nets analysis approaches

There are three different approaches to analyzing Petri nets. The first approach involves enumerating all possible states. A *reachability tree*, sometimes referred to as an *occurrence graph*, is used to record all reachable states and dependencies among the states. A reachability tree is a directed graph where a node identifies the current marking, while an edge identifies a transition. An edge going from node *A* to node *B* labeled *t* in a reachability tree indicates that this transition *t* moves the net from the state identified by node *A* to the state identified by node *B*.

The second approach involves analyzing the Petri net by means of *state invariants*. A Petri net state invariant is a statement that is satisfied by all reachable markings of the net. This approach requires representing the net as well as its dynamic behaviour as matrix equations or other mathematical constructs. State invariants are then formally proved using established rules.

The third approach involves the reduction of the Petri net to a standard form that preserves the properties of the original net. Assume a Petri net is to be analyzed in terms of a specified property, then certain *transformation rules* may exist that preserve such a property in the reduction process. Once arriving at a reduced version

of the Petri net, the verification of the property under investigation is simplified. The results then apply to the original Petri net.

In this work we use the first analysis approach which employs occurrence graphs.

1.3 Structure of the paper

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 Tatebayashi, Matsuzaki, and Newman (TMN) [45] protocol. Finally, Section 4 summarises the benefits of the technique and suggests possible extensions. This paper is an extended version of [2]. A more extended version of this work can be found in [1].

2. Outline of the Technique

Our technique is a finite-state analysis method [18]. 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.

We provide 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 [4, 38, 43] 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, 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) we add the intruder transition to the top-level model;
   (c) we define the intruder substitution transition.
   (d) we Implement a token-passing scheme.
   (e) we specify security requirements stated in terms of CPN markings.
   (f) we 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, the TMN protocol, and then, using our technique, we build its model. The selection of the TMN protocol to illustrate our technique is motivated by its familiarity. Any other protocol listed in [28] could have been used.

3.1 The TMN Protocol

The 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 [28, 29]. Moreover, three other known attacks on this protocol are given in [28].

Initially, the TMN protocol assumes that both A and B know the public key of J, K_J^{Pb}. We use the following notation: (i) $A \rightarrow B : X$ to indicate that in the ith 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". Furthermore, *key*(*data*) denotes the message *data* encrypted using the key *key*. The protocol proceeds as follows:

$$\begin{array}{ll} (1)\ A \rightarrow J : B, K_J^{\text{Pb}}(K_{AJ}) & (3)\ B \rightarrow J : A, K_J^{\text{Pb}}(K_{AB}) \\ (2)\ J \rightarrow B : A & (4)\ J \rightarrow A : B, K_{AJ}(K_{AB}) \end{array}$$

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_J^{Pb}), and sends it along with the identity of B to the server J (step 1). Upon receiving the first message, the server decrypts $K_J^{\text{Pb}}(K_{AJ})$ using its private key (K_J^{Pr}) 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_J^{Pb}, and sends it along with the identity of A to J (step 3). Upon receiving the third message, the server decrypts $K_J^{\text{Pb}}(K_{AB})$ using its private key (K_J^{Pr}) 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

3.2.1 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 ciphertext $K(A)$ can be viewed as an ordered pair (A, K), where A is the identity and K is the encryption key.

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, $K = \{K_{AB}, K_{AJ}, K_J^{\text{Pb}}, K_J^{\text{Pr}}\}$.
   b) All ciphers have the same format: $k1(k2)$, where $k1$ and $k2$ are of type K. For instance, $K_J^{\text{Pb}}(K_{AJ})$ is the ciphertext of the first message, *etc*. Thus, the ciphertext 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 $(B, K_J^{\text{Pb}}(K_{AJ}))$ and the third message is $(A, K_J^{\text{Pb}}(K_{AB}))$. 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, $(B, K_J^{\text{Pb}}(K_{AJ}))$, 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 3.1, 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 are used to indicate tokens inherent to an entity, or tokens that are to be used in subsequent subtasks performed by the entity.

2) Variables: We use variables of the defined colour sets as inscriptions for arcs of the CPN. They are: $k1$, $k2$, and k of type K, c of type C, i of type I, and m of type M.

3) Functions:
   a) The function *DecryptionKey*$(k : K)$ returns the decryption key of a given key k.

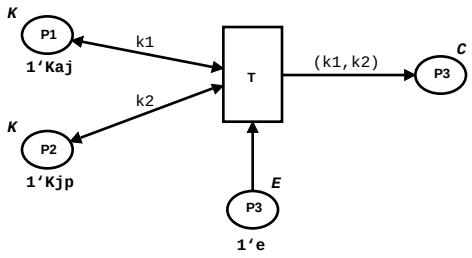

Figure 3.1: By using the E set, transition T can fire at most one time.

b) The function $SharedKey(i : I)$ returns the shared key between entity B and the entity i. For instance, $SharedKey(A)$ 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 3.2 using *CPN ML* notation.

```
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;
fun SharedKey(i:I):K= case i of A => Kab;
```

Figure 3.2: The declarations for the TMN model

3.2.2 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

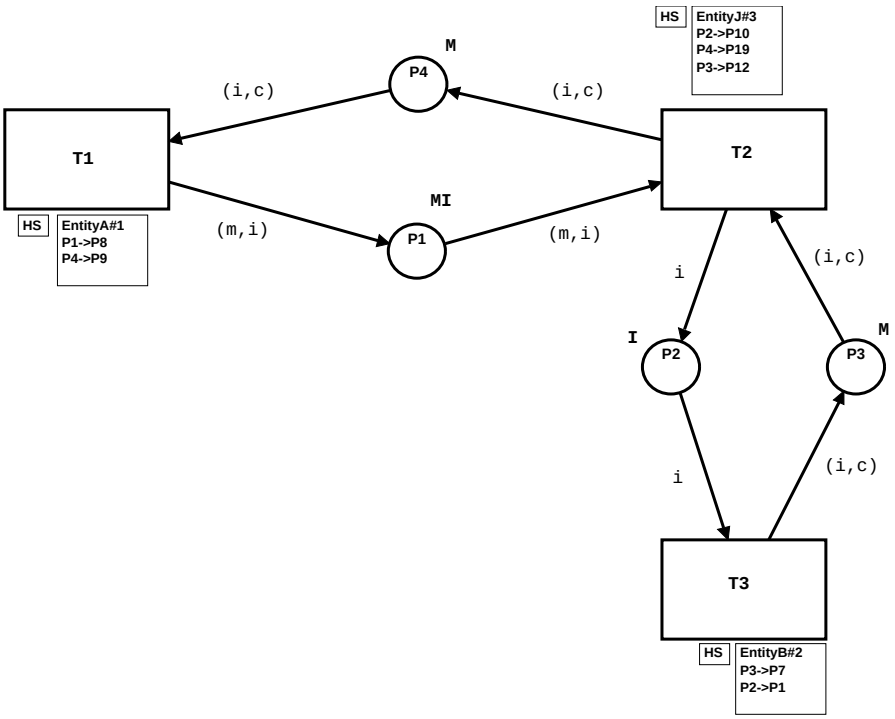

Figure 3.3: The TMN top-level model with no intruder

level, protocol entities are modeled as transitions. Figure 3.3 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.

3.2.3 Defining the Top-Level Substitution Transitions

We consider in detail the model of the initiator A and we give the models of entities B and J without further elaboration (see [1] for more details about the models

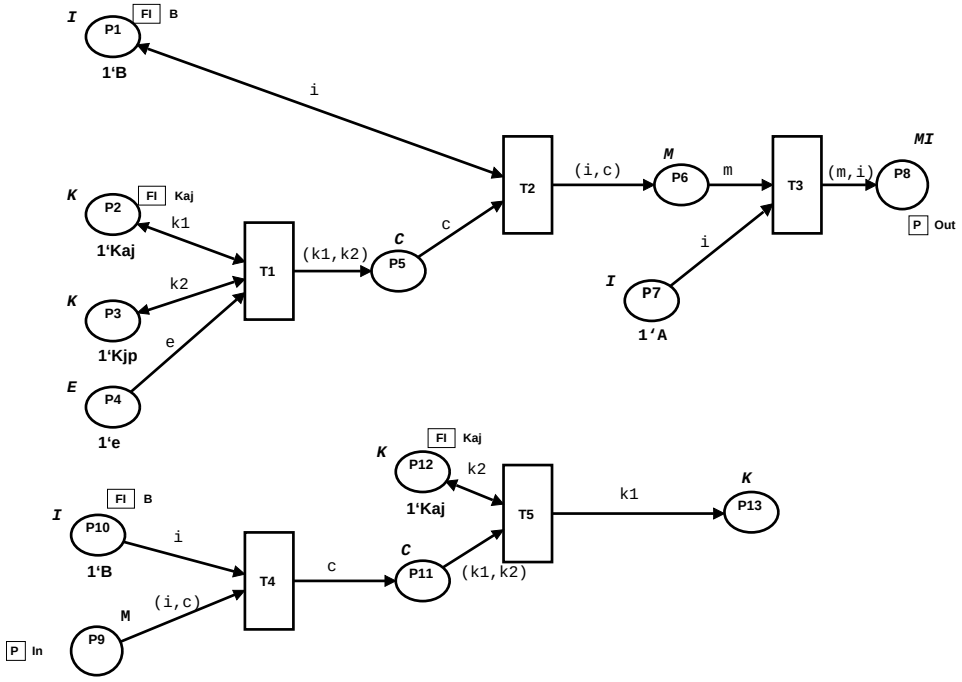

Figure 3.4: Page `EntityA`

of entities B and J). The following is an informal description for the behaviour the initiator of the communication session, A. Thus, A always sends the message $B, K_J^{\text{Pb}}(K_{AJ})$. 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 3.4 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 3.3), the socket $P1$ is related to the output port $P8$ of `EntityA`, while the socket $P4$ is related to the input port $P9$ of `EntityA`.

In `EntityA`, we use the instance fusion sets $B = \{P1, P10\}$ and $Kaj = \{P2, P12\}$. 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 ciphertext it receives in the last step.

In a similar way, we build the models of entities B and J given respectively in Figure 3.5 and Figure 3.6. Figure 3.7 shows the hierarchy page.

TABLE I: Instance fusion sets in `EntityA`

Instance Fusion Set	Members of the Fusion Set
B	P1, P10
Kaj	P2, P12

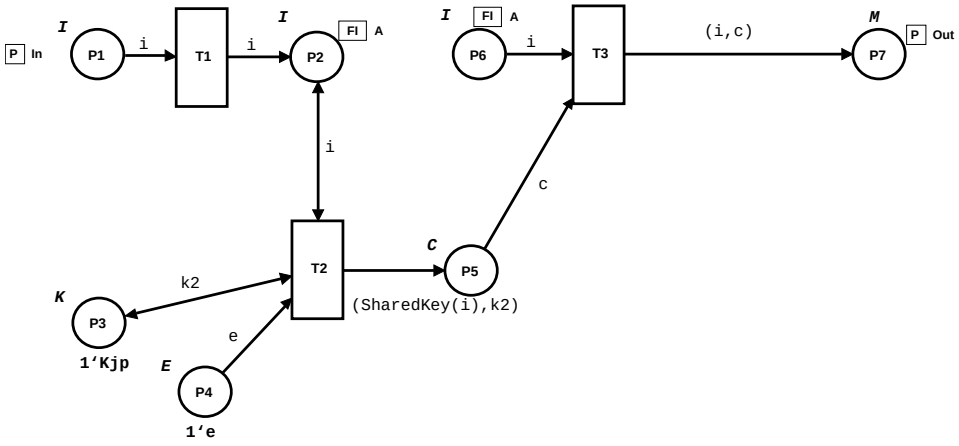

Figure 3.5: Page *EntityB*

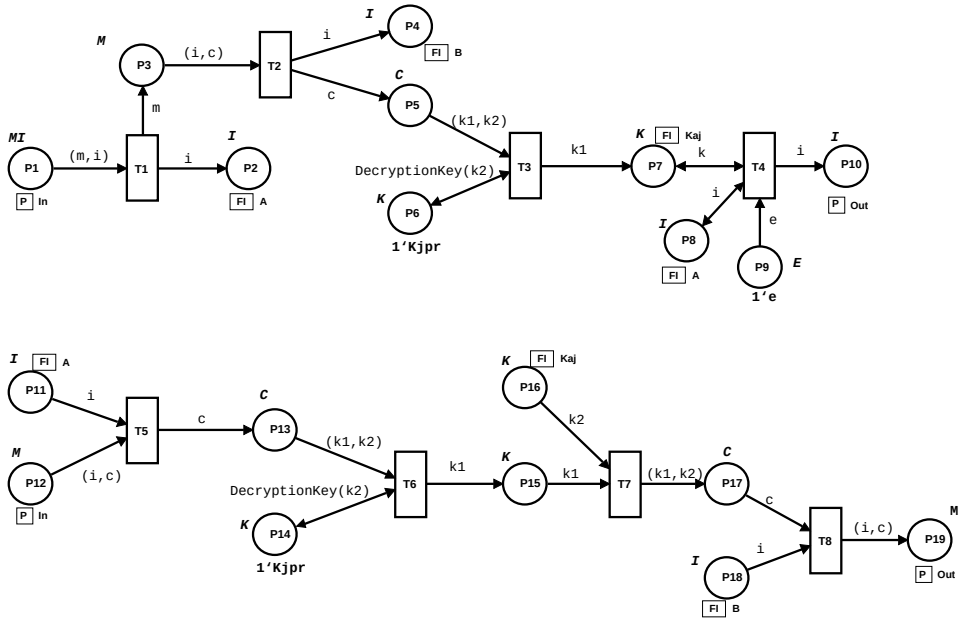

Figure 3.6: Page *EntityJ*

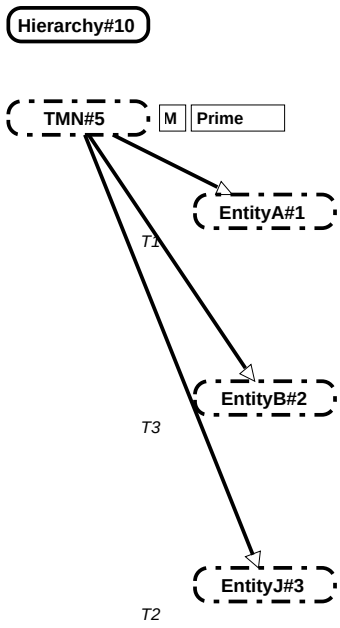

Figure 3.7: The hierarchy page

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 to generate new messages to replace the intercepted ones.

3.3.1 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:

color DB=union cI:I + cK:K + cC:C;

Here, *cI*, *cK*, and *cC* are selectors [33]. Thus, the intruder's possession of K_{AB} is modeled as reaching a marking where a token *cK*(*Kab*) is in a DB-place. Figure 3.8 contains the final CPN ML declarations.

```
color I = with A | B | In;
color K = with Kaj | Kjp | Kjpr | Kab | Ki;
color C = product K*K;
color M = product I*C;
color MI = product M*I;
color DB = union cI:I + cK:K + cC:C;
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 | Kab=>Kab | Ki=>Ki;
fun SharedKey(i:I):K=case i of A=> Kab | In =>
Ki | B=>Kab;
color S=with s;
```

Figure 3.8: Declarations used in the TMN model with an intruder

3.3.2 The Top-Level Model with an Intruder

Figure 3.9 shows the top level model of the TMN protocol with an intruder. The substitution transition *T*4 represents the intruder, which was not included in the earlier top level model given in Figure 3.3.

Each place in Figure 3.3 is replaced with two corresponding places as shown in Figure 3.9: one is an input place to the intruder while the second is an output place. For instance, place *P*1 in Figure 3.3 is replaced with *P*1 and *P*2 in Figure 3.9. This is needed to model the intruder's ability to receive a message (the input place), to deal with it (transition *T*4), and to substitute it with a new message (the output place).

3.3.3 Defining the Intruder Substitution Transition

The intruder substitution transition (*T*4 in Figure 3.9) is defined by the subpage `intruder` shown in Figure 3.10.

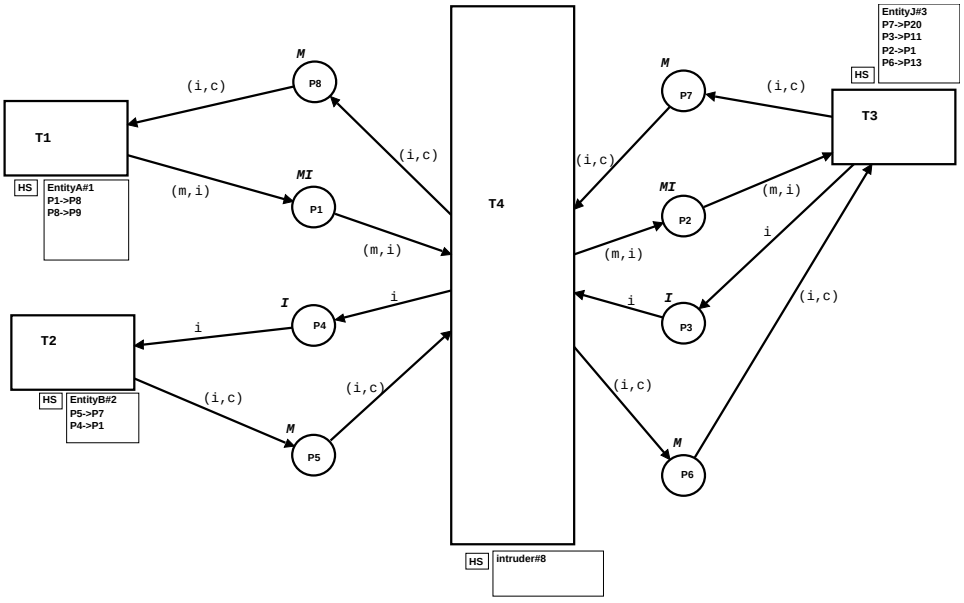

Figure 3.9: The TMN top-level model with an intruder

TABLE II: The intruder subprocesses

Pair Places		Colour	The Corresponding
Input	Output	Set	Intruder Subprocess
P1	P2	MI	intruder_mi
P3	P4	I	intruder_i
P5	P6	M	intruder_m
P7	P8	M	intruder_m

The intruder model is constructed by using several intruder subprocesses. Each intruder subprocess models the intruder's possible actions to intercept tokens that belong to a given colour set (type). Table II lists the intruder subprocesses, along with their input/output places.

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 $T1$), one instance of `intruder_i` (which defines $T2$), and two instances of `intruder_m` (which define $T3$ and $T4$).

The `intruder_m` subprocess is given in Figure 3.11. 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 ciphertext using one of the keys stored in its database. Finally, the intruder forms a new message to be sent in place of the intercepted

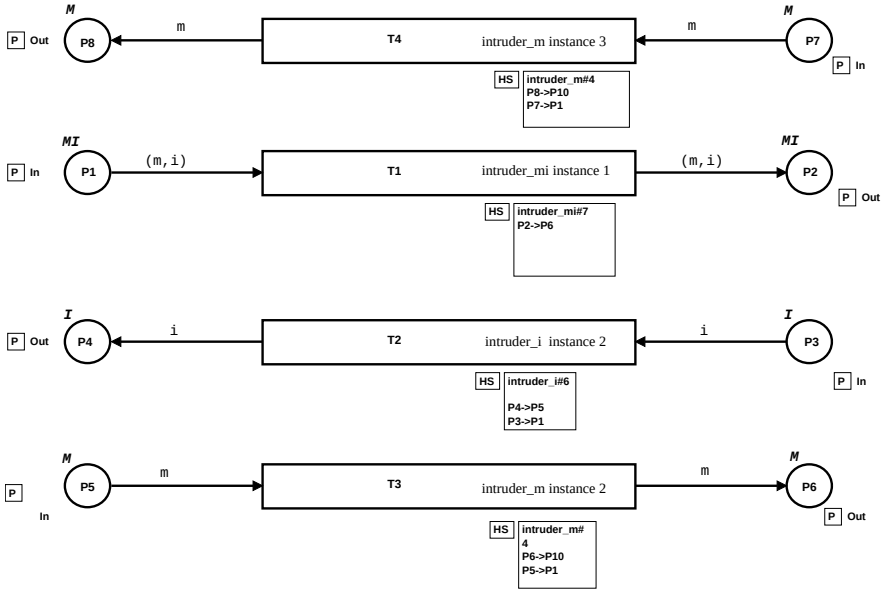

Figure 3.10: The `intruder` page

one. The intruder uses one of the ciphers stored in the database, or constructs a new ciphertext 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 3.12. 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 *P*4 in `intruder_m` indicates (Figure 3.11), the `DB` is set initially to $\{K_i, K_J^{\text{pb}}, A, B, In\}$.

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_J^+, and it has three identities: *A*, *B*, and *I*.

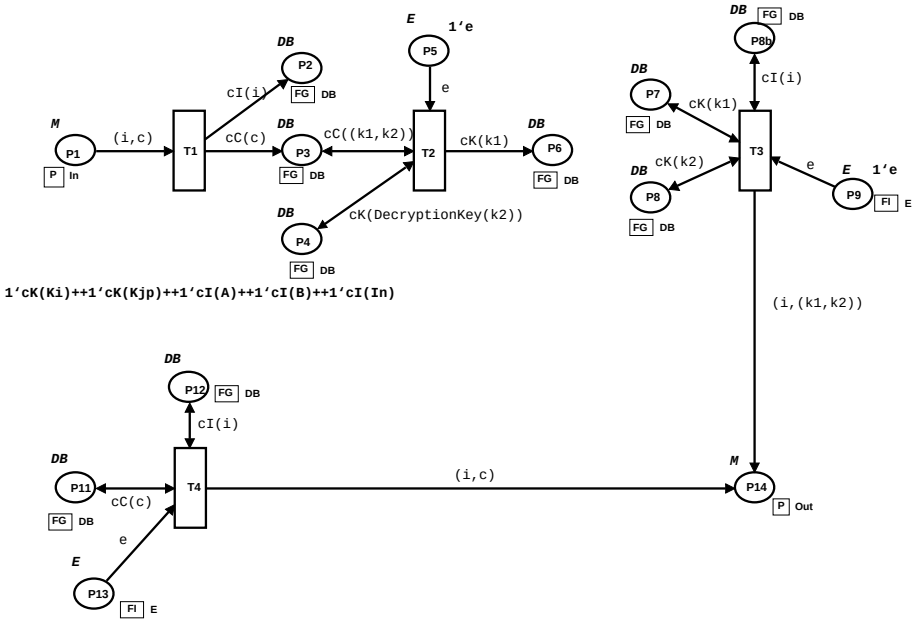

Figure 3.11: Page `intruder_m`

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 $T1$ 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 $T2$ of *EntityA*, etc.

Let $\mathcal{E}$ be the set of finite occurrence sequences of the possible execution of the agents *A*, *B*, and *J*. For every sequence α, the intruder observes the set S_α of relevant information (keys, messages, and agent identities) that are carried by α. Let $R = \{(\alpha_1, \alpha_2) \mid \alpha_1 \in \mathcal{E} \wedge \alpha_2 \in \mathcal{E} \wedge S_{\alpha_1} = S_{\alpha_2}\}$. The relation R is an equivalence relation. It is clear that every sequence in the set of sequences generated by an interleaving of subprocesses belongs to the same R-equivalence class as any sequence in their sequential execution. Hence, there is no need to include all of 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.

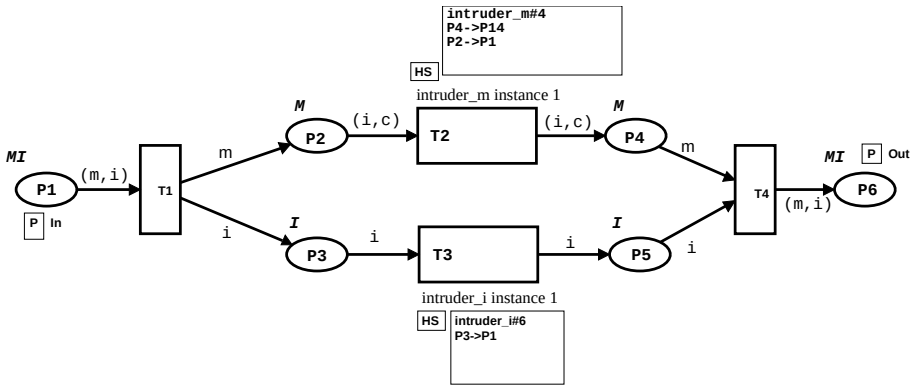

Figure 3.12: Page `intruder_mi`

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 3.13.

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 3.14 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, the creation of S is carried out by performing the following actions:

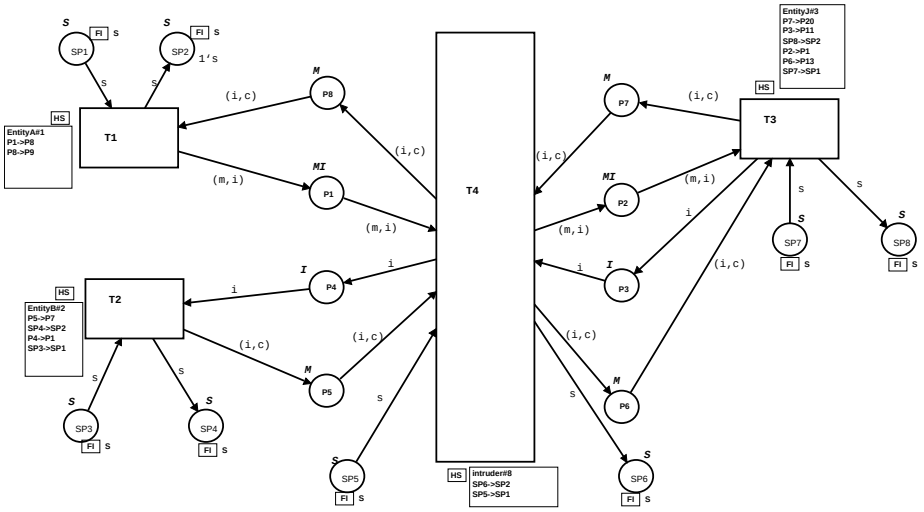

Figure 3.13: The *TMN* page after adding the S-places

(a) Add an output arc from the first transition of the intruder subpage to a place that belongs to the fusion set S.
(b) Add an input arc from a place that belongs to the fusion set S to the last transition of the intruder subpage.
(c) Add double arcs from a place that belongs to the fusion set S to the intermediate intruder transitions.

These changes are demonstrated in Figure 3.16. For example, transitions $T2$ and $T3$ of `intruder_m` will not fire until the s-token arrives to the subprocess, which means transition $T1$ fires, consuming the s-token from the input port $SP1$. When the s-token is returned back by the intruder subprocess (*i.e.* transition $T4$ fires and the s-token is deposited back to the output port $SP2$), transitions $T2$ and $T3$ 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 3.15.

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,

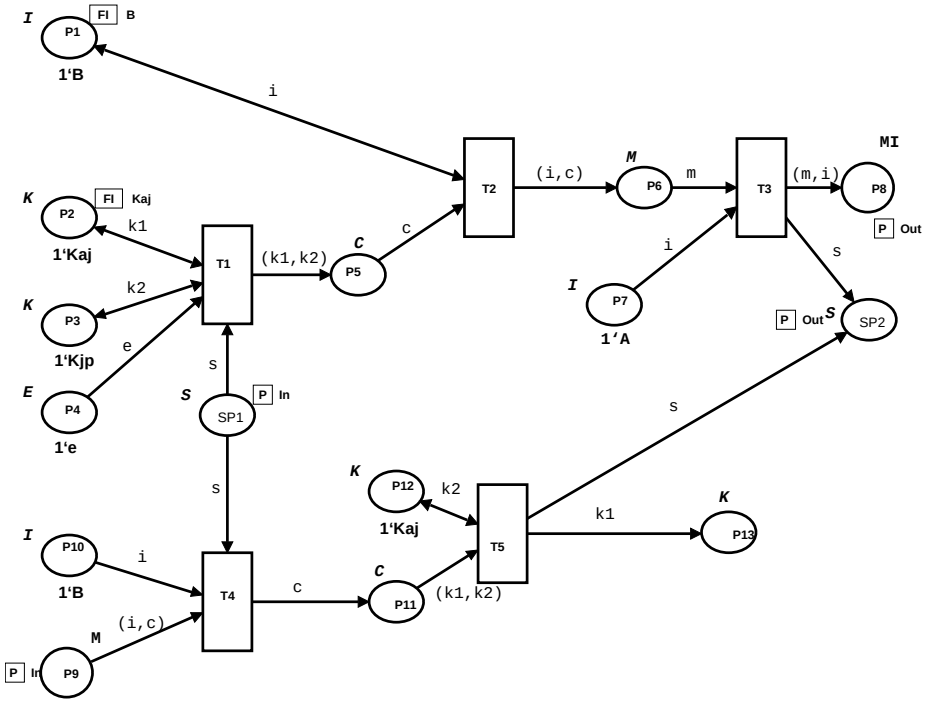

Figure 3.14: The `EntityA` page after adding the *S*-places

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 [34]. 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 *P*4 of `intruder_m` (first instance) contains at least one token *cK*(*Kab*), and evaluates to false

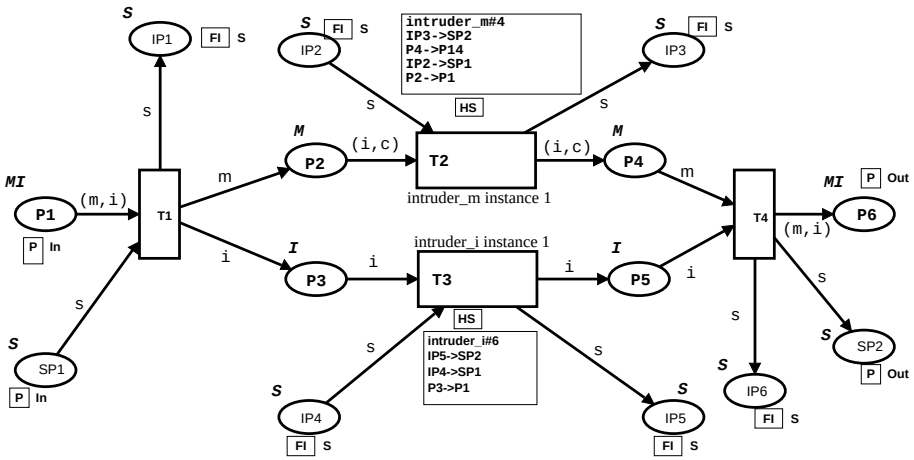

Figure 3.15: The `intruder_mi` page after adding the S-places

otherwise. Note that `cf` is the coefficient function [33]. 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,5'A)` 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.

```
fun SecrecyViolation1(k:K):
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`, $\langle k1 = Kab, k2 = Ki \rangle$).

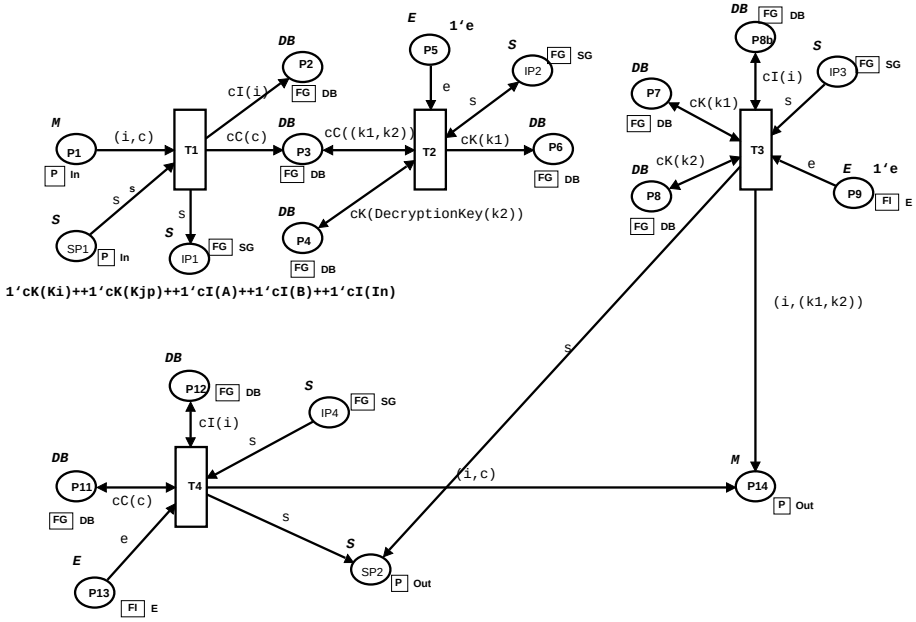

Figure 3.16: The `intruder_m` page after adding S-places

`EntityA`	`T1`	$k1 = Kaj, k2 = Kjp$
`EntityA`	`T2`	$i = B, c = (Kaj, Kjp)$
`EntityA`	`T3`	$i = A, m = (B, (Kaj, Kjp))$
`intruder_mi`	`T1`	$m = (B, (Kaj, Kjp)), i = A$
`intruder_i 1`	`T1`	$i = A$
`intruder_i 1`	`T2`	$i = A$
`intruder_m 1`	`T1`	$i = B, c = (Kaj, Kjp)$
`intruder_i 2`	`T2`	$i = A$
`EntityB`	`T1`	$i = A$
`EntityB`	`T2`	$i = A, k2 = Kjp$
`EntityB`	`T3`	$i = A, c = (Kab, Kjp)$
`intruder_m 2`	`T1`	$i = A, c = (Kab, Kjp)$
`intruder_m 1`	`T3`	$k1 = Ki, k2 = Kjp, i = A$
`intruder_mi`	`T4`	$i = A, m = (A, (Ki, Kjp))$
`EntityJ`	`T1`	$i = A, m = (A, (Ki, Kjp))$
`EntityJ`	`T2`	$i = A, c = (Ki, Kjp)$
`EntityJ`	`T3`	$k1 = Ki, k2 = Kjp$
`EntityJ`	`T4`	$i = A, k = Ki$
`intruder_i 2`	`T1`	$i = A$
`intruder_m 2`	`T4`	$i = A, c = (Kab, Kjp)$
`EntityJ`	`T5`	$i = A, c = (Kab, Kjp)$
`EntityJ`	`T6`	$k1 = Kab, k2 = Kjp$

`EntityJ`	`T7`	$k1 = Kab, k2 = Ki$
`EntityJ`	`T8`	$i = A, c = (Kab, Ki)$
`intruder_m 3`	`T1`	$i = A, c = (Kab, Ki)$
`intruder_m 1`	`T2`	$k1 = Kab, k2 = Ki$ `(*)`
`intruder_m 2`	`T2`	$k1 = Kab, k2 = Ki$
`intruder_m 3`	`T3`	$i = B, k1 = Ki, k2 = Kjp$
`EntityA`	`T4`	$i = B, c = (Ki, Kjp)$

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.

$(I1)\ A \rightarrow I(J) : B, K_J^{\text{Pb}}(K_{AJ})$ $\quad$ $(II2)\ J \rightarrow I(A) : A$

$(I2)\ I(J) \rightarrow B : A$ $\quad$ $(II3)\ I(A) \rightarrow J : A, K_J^{\text{Pb}}(K_{AB})$

$(I3)\ B \rightarrow I(J) : A, K_J^{\text{Pb}}(K_{AB})$ $\quad$ $(II4)\ J \rightarrow I(A) : A, K_I(K_{AB})$

$(II1)\ I(A) \rightarrow J : A, K_J^{\text{Pb}}(K_I)$

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 $K_J^{\text{Pb}}(K_{AB})$ 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.

The main contribution of this work is the development of a new technique to verify cryptographic protocols using Jensen's form of coloured Petri nets. Furthermore, we:

- show how to use *Design/CPN* in the construction and verification of the net models.
- simplify the representation of the intruder's knowledge by using the concept of DB-place.
- apply a token passing scheme to reduce the size of the occurrence graph. This reduction has no effect on the security assumptions. Without using this scheme, the resulting occurrence graph would be extremely large and it would be impractical to use *Design/CPN*.
- use the technique to model and verify the TMN key exchange protocol. This technique was also used to model and verify the Needham-Schroeder public key authentication protocols [1].

Our technique compares well with other finite-space methods [29, 32, 35]. It includes the same verification assumptions and reveals the same patterns of attacks. 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 [10], 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 ciphertext 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 [20] seem to be useful. Other techniques to yield a reduced representation of the occurrence graph are applicable. These include the stubborn set method [27], and occurrence graphs with equivalence classes [20].

Acknowledgements

The first author was supported by an Ontario Graduate Scholarship. This research is also supported by grants from the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank Ryszard Janicki and the anonymous reviewers of `MOMPES 2005` for their comments that helped increase the quality of the paper.

References

[1] I. Al-Azzoni. The verification of cryptographic protocols using coloured Petri nets. Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2004.

[2] I. Al-Azzoni, D. G. Down, and R. Khedri. Modeling and verification of cryptographic protocols using coloured Petri nets and Design/CPN. In J. Lilius, R. J. Machado, D. Truscan, and J. ao M. Fernandes, editors, *2nd International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2005)*, volume 39 of *TUCS General Publication*, pages 1–19, Rennes, France, June 2005. Turku Centre for Computer Science. ISBN 952-12-1556-9, ISSN 1239-1905.

[3] S. Aly and K. Mustafa. Protocol verification and analysis using colored Petri nets. Technical Report TR-04-003, The School of Computer Science, Telecommunication and Informations, August 2004. http://www.cs.depaul.edu/research/technical.asp. (Accessed August 23, 2005).

[4] A. Basyouni and S. Tavares. New approach to cryptographic protocol analysis using coloured Petri nets. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE'97)*, pages 334–337, St. John's, Newfoundland, May 1997.

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[6] C. Capellmann, S. Christensen, and U. Herzog. Visualising the behaviour of intelligent networks. In T. Margaria, B. Steffen, R. Rückert, and J. Posegga, editors, *Services and Visualization, Towards User-Friendly Design*, volume 1385 of *Lecture Notes in Computer Science*, pages 174–189. Springer-Verlag, 1998.

[7] C. Capellmann and H. Dibold. Petri net based specifications of services in an intelligent network: Experiences gained from a test case application. In M. Ajmone-Marsan, editor, *Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 542–551. Springer-Verlag, 1993.

[8] C. Capellmann, H. Dibold, and U. Herzog. Using high-level Petri nets in the field of intelligent networks. In J. Billington, M. Diaz, and G. Rozenberg, editors, *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*, pages 1–36. Springer-Verlag, 1999.

[9] S. Christensen and J. Jørgensen. Analysing Bang & Olufsen's BeoLink Audio/Video system using coloured Petri nets. In G. Balbo and P. Azema, editors, *Application and Theory of Petri Nets. Proceedings of the 18th International Petri Net Conference, Toulouse 1997*, Lecture Notes in Computer Science, pages 387–406. Springer-Verlag, 1997.

[10] S. Christensen, L. M. Kristensen, and T. Mailund. A sweep-line method for state space exploration. In *Proceedings of TACAS 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer-Verlag, 2001.

[11] J. Clark and J. Jacob. A survey of authentication protocol literature. Technical Report 1.0, SVRC, The University of Queensland, School of Information Technology and Electrical Engineering, Queensland, November 1997. http://www.cs.york.ac.uk/~jac/papers/ drareview.ps.gz. (Accessed August 23, 2005).

[12] CPN Group at the University of Aarhus. Design/CPN Online, 2004. http://www.daimi.au.dk/designCPN/. (Accessed February 17, 2004).

[13] CPN Group at the University of Aarhus. Main page, 2005. http://www.daimi.au.dk/CPnets/cpngroup.html. (Accessed August 25, 2005).

[14] J. de Figueiredo and L. Kristensen. Using coloured Petri nets to investigate behavioural and performance issues of TCP protocols. In K.Jensen, editor, *Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN, Aarhus 1999*, pages 21–40. Department of Computer Science, University of Aarhus, 1999.

[15] C. Fidge. A survey of verification techniques for security protocols. Technical Report 01-22, Software Verification Research Centre, The University of Queensland, July 2001.

[16] C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications.* Springer-Verlag, 2003.

[17] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and*

Privacy, pages 234–248, 1990.

[18] K. Heljanko. Can finite-state system verification methods help cryptographic protocol analysis? Technical report, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Finland, 1998.

[19] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts. Springer-Verlag, 2nd edition, 1996.

[20] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 2: Analysis Methods. Springer-Verlag, 2nd edition, 1996.

[21] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 3: Practical Use. Springer-Verlag, 2nd edition, 1996.

[22] K. Jensen. A brief introduction to coloured Petri nets. In Brinksma, E., editor, *Lecture Notes in Computer Science: Tools and Algorithms for the Construction and Analysis of Systems. Proceedings of the TACAS'97 Workshop, Enschede, The Netherlands 1997*, volume 1217, pages 201–208. Springer-Verlag, 1997.

[23] R. Kemmerer. Analzying encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[24] L. Kristensen, S. Christensen, and K. Jensen. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer: Special section on coloured Petri nets*, 2(2):98–132, 1998.

[25] L. Kristensen and K. Jensen. Specification and validation of an edge router discovery protocol for mobile ad hoc networks. In *Integration of Software Specification Techniques for Applications in Engineering: Priority Program SoftSpez of the German Research Foundation (DFG)*, volume 3147, pages 248–269. Springer-Verlag, 2004.

[26] L. Kristensen, J. Jørgensen, and K. Jensen. Application of coloured Petri nets in system development. In *Lectures on Concurrency and Petri Nets - Advanced in Petri Nets. Proceedings of 4th Advanced Course on Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 626–685. Springer-Verlag, 2004.

[27] L. M. Kristensen and A. Valmari. Finding stubborn sets of coloured Petri nets without unfolding. *Lecture Notes In Computer Science*, 1420:104–123, 1998.

[28] Laboratoire Spécification et Vérification. SPORE security protocols open repository. http://www.lsv.ens-cachan.fr/spore/table.html, 2005. (Accessed August 23, 2005).

[29] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, October 1997.

[30] W. Mao and C. Boyd. Towards the formal analysis of security protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, pages 147–158. IEEE Computer Society Press, June 1993.

[31] C. Meadows. Formal verification of cryptographic protocols: A survey. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1994.

[32] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, Feb 1996.

[33] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*, Version 2.0, 1993.

[34] Meta Software Corporation. *Design/CPN Occurrence Graph Manual*, Version 3.0, 1996.

[35] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, pages 141–153. IEEE Computer Society Press, May 1997.

[36] A. Mnaouer, T. Sekiguchi, Y. Fujii, T. Ito, and H. Tanaka. Coloured Petri nets based modelling and simulation of the static and dynamic allocation policies of the asynchronous bandwidth in the Fieldbus protocol. In J. Billington, M. Diaz, and G. Rozenberg, editors, *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*, pages 93–130. Springer-Verlag, 1999.

[37] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[38] B. Nieh and S. Tavares. Modelling and analyzing cryptographic protocols using Petri nets. In *Advances in Cryptology-ASIACRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 275–295. Springer, 1992.

[39] L. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.
[40] A. Rubin and P. Honeyman. Formal methods for the analysis of authentication protocols. Technical Report 93–7, Center for Information Technology Integration (CITI), October 1993. http://www.citi. umich.edu/techreports/reports/citi-tr-93-7.ps.gz. (Accessed August 23, 2005).
[41] P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.
[42] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley, 2nd edition, 1996.
[43] D. M. Stal, S. E. Tavares, and H. Meijer. Backward state analysis of cryptographic protocols using coloured Petri nets. In *Workshop on Selected Areas in Cryptography, SAC '94 Workshop Record*, pages 107–118, May 1994.
[44] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 2nd edition, 2002.
[45] M. Tatebayashi, N. Matsuzaki, and D. Newman. Key distribution protocol for digital mobile communication systems. In *Advances in Cryptology–CRYPTO'89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 324–334, 1990.
[46] P. Varhol. ML and colored Petri nets for modeling and simulation: a little language for a big job. (ML: Meta Language functional programming language). *Dr. Dobbs Journal*, 16(9):76–81, September 1991.
[47] G. Wheeler. The modelling and analysis of IEEE 802.6's configuration control protocol with coloured Petri nets. In J. Billington, M. Diaz, and G. Rozenberg, editors, *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*, pages 69–92. Springer-Verlag, 1999.