THE VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS USING COLOURED PETRI NETS

THE VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS USING COLOURED PETRI NETS

By
ISSAM AL-AZZONI, B.Eng.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree
Master of Applied Sciences (M.A.Sc.)

McMaster University

MASTER OF APPLIED SCIENCES (2004) McMaster University
(Software Engineering) Hamilton, Ontario

TITLE: The Verification of Cryptographic Protocols Using Coloured Petri Nets

AUTHOR: Issam Al-Azzoni, B.Eng. (McMaster University)

SUPERVISOR: Dr. Doug Down and Dr. Ridha Khedri

NUMBER OF PAGES: cxlvii, 147

Contents

Contents iii

List of Tables vii

List of Figures viii

Chapter 1. Introduction 1

1.1 Background on Cryptography and Cryptographic Protocols 1

1.1.1 Cryptography and Cryptanalysis 2

1.1.2 Cryptographic Algorithms 3

1.1.3 Cryptographic Protocols 4

1.1.4 Properties of Cryptographic Protocols 7

1.1.5 Remarks 8

1.2 Background on the Verification of Cryptographic Protocols 9

1.2.1 Intruder Attacks on Cryptographic Protocols 9

1.2.2 Example Intruder Attacks 11

1.2.3 The Difficulty of Cryptographic Protocol Verification 14

1.2.4 Formal Methods for Cryptographic Protocol Verification . . . 15

1.2.5 Concluding Remarks 17

1.3 The Verification Using Coloured Petri Nets 18

1.4 Thesis Motivation 19

1.5 Research Objective 20

1.6 Contributions 20
1.7 Thesis Outline 21

Chapter 2. Petri Nets and Coloured Petri Nets **22**
2.1 Petri Nets 22
2.2 Analysis of Petri Nets 26
2.3 High Level Petri Nets 26
2.4 Jensen's Coloured Petri Nets 29
2.5 A CPN Example Model 30
2.6 Formal Definition of CP-nets 33
2.7 Hierarchical CP-nets 36
2.8 Design/CPN 41
2.9 Remarks 44

Chapter 3. Cryptographic Protocol Verification using Coloured Petri Nets: A Review **46**
3.1 Varadharajan Contributions 46
3.2 Behki Contributions 47
3.3 Nieh Contributions 47
3.3.1 Features 48
3.3.2 Assumptions 49
3.3.3 Modeling the Intruder 49
3.3.4 Verification 49
3.3.5 Limitations 50
3.4 Doyle Contributions 51
3.5 Basyouni Contributions 52
3.5.1 Basyouni *vs.* Nieh Models 54
3.6 Remarks 56

Chapter 4. Cryptographic Protocol Verification Using Design/CPN: A New Approach **59**

4.1 CPN Declarations 61

4.2 The Model with No Intruder 64

4.2.1 The Top-Level Model 64

4.2.2 Defining the Top-Level Substitution Transitions 65

4.2.3 Summary 71

4.2.4 Simulation and Debugging 72

4.2.5 Generating the Occurrence Graph 75

4.3 The Model with an Intruder 78

4.3.1 The Top Level Model with an Intruder 78

4.3.2 Extending the CPN/ML Definitions 80

4.3.3 Defining the Intruder 80

4.3.4 Identifying Security Requirements 86

4.3.5 Simulation and Debugging 86

4.3.6 Analyzing the Occurrence Graph 89

4.4 Approaches for Reducing the Occurrence Graph 92

4.4.1 Applying a Token-Passing Scheme 92

4.4.2 A Further Improvement: No Storage of Intruder Constructed Ciphers 99

4.4.3 Fixing a Non-Intended Constraint on the Intruder Behaviour . 101

4.4.4 The Final TMN Model 101

4.5 Discussion 110

Chapter 5. Conclusion **112**

5.1 Discussion 112

5.2 Future Work 114

Appendices **116**

Appendix A. Functional Definitions **116**

Appendix B. Verification of the Needham-Schroeder Public Key Protocol **118**

B.1 The Model with no Intruder 119

B.2 The Model with an Intruder- A Initiates a Session with B 119

B.3 The Model with an Intruder- A Initiates a Session with I 129

Bibliography **133**

List of Tables

4.1 Port assignments for page *TMN* 71
4.2 Instance fusion sets 72
4.3 The intruder subprocesses 81

List of Figures

1.1 The message sequence diagram of the NSSK protocol 7
1.2 The message sequence diagram of the TMN protocol 12
1.3 The message sequence diagram of the NSPK protocol 15

2.1 A sample Petri net 23
2.2 The sample Petri net after firing transition t 25
2.3 A Petri net model of a simple communication protocol 25
2.4 A sample coloured Petri net 28
2.5 The sample coloured Petri net from Figure 2.4 after firing transition $T1$ 28
2.6 A CPN model of an intruder intercepting two ciphertexts: $C(1)$ and $C(2)$ 31
2.7 The CPN model from Figure 2.6 after firing trasnition *Decrypt* . . . 32
2.8 The CPN model from Figure 2.6 with a different initial marking . . . 35
2.9 The CPN model from Figure 2.8 after firing the step Y_1 37
2.10 The CPN model from Figure 2.8 after firing the step Y_2 37
2.11 Page *Net* describes a primitive network of two entities 38
2.12 Page *Entity* describes the behaviour of an entity 40
2.13 The hierarchy page of the net system 41
2.14 Using the page fusion set MFS in page *Entity* 42
2.15 The occurrence graph for the CPN of Figure 2.8 44

3.1 Two PNOs with a communication channel 48
3.2 An inhibitor arc 51

3.3 Nieh's and Basyouni's usage of place colours 54
3.4 An alternative usage from figure 3.3 54
3.5 An intruder process in Basyouni's model 56
3.6 The same process of figure 3.5 in Nieh's model 57

4.1 The declarations for the TMN model 61
4.2 Transition T will fire infinitely. 63
4.3 By using the E set, transition T can fire at most one time. 63
4.4 The TMN top-level model with no intruder 66
4.5 Page *EntityA* 68
4.6 Page *EntityB* 68
4.7 Page *EntityJ* 69
4.8 The hierarchy page 70
4.9 The marking of *EntityA* after firing $T3$ 73
4.10 The marking of *EntityJ* after firing T4 74
4.11 The final marking of *EntityA* 75
4.12 The final marking of *EntityB* 76
4.13 The final marking of *EntityJ* 76
4.14 The occurrence graph of the TMN model with no intruder 77
4.15 The TMN top-level model with an intruder 79
4.16 Declarations used in the TMN model with an intruder 81
4.17 The *intruder* page 82
4.18 Page *intruder_mi* 83
4.19 Page *intruder_m* 84
4.20 Page *intruder_i* 84
4.21 The hierarchy page of the TMN model with an intruder 85
4.22 The *TMN* page after adding the S-places 93
4.23 The *EntityA* page after adding the S-places 94
4.24 The *EntityB* page after adding the S-places 95

4.25 The *EntityJ* page after adding the S-places 96
4.26 The *intruder* page after adding the S-places 97
4.27 The *intruder_mi* page after adding the S-places 97
4.28 The *intruder_m* page after adding the S-places 98
4.29 The *intruder_i* page after adding the S-places 98
4.30 The *intruder_m* page after preventing the intruder from storing ciphers it constructs 100
4.31 The page *intruder_m* after using a global fusion set SG 102
4.32 The page *intruder_i* after using a global fusion set SG 102
4.33 The result of executing SecrecyViolation1 and SecrecyViolation2 . . . 104
4.34 A path from the initial marking to the insecure marking 19170 105
4.35 A path from the initial marking to the insecure marking 9571 109

B.1 The NSPK model with no intruder (with declarations) 120
B.2 Page *EntityA* 121
B.3 Page *EntityB* 122
B.4 The hierarchy page 122
B.5 The NSPK top-level model with an intruder 123
B.6 Declarations used in the NSPK model with an intruder 123
B.7 Page *EntityA* in the NSPK model with an intruder 124
B.8 Page *EntityB* in the NSPK model with an intruder 125
B.9 The *intruder* page 126
B.10 Page *intruder_c* 127
B.11 The hierarchy page of the NSPK model with an intruder 128
B.12 The result of executing AuthViolation2 130
B.13 A path from the initial marking to the insecure marking 6410 131

Chapter 1

Introduction

This chapter provides the background necessary to state the research objective of this thesis. In Section 1.1, cryptographic protocols are introduced. Then, Section 1.2 discusses how these protocols are attacked and what formal methods exist to aid the verification process. In Section 1.2, we give the benefits of using coloured Petri nets in the verification, and briefly mention the existing techniques that use these nets. Section 1.4 states the motivation of this thesis. This is followed by a section that states the thesis objective (Section 1.5). The contributions of this thesis are listed in Section 1.6. The outline of this thesis is given in Section 1.7.

1.1 Background on Cryptography and Cryptographic Protocols

This section gives an introduction to cryptography. First, key concepts such as encryption and decryption are introduced. Then, we introduce cryptographic algorithms and how they are related to encryption and decryption. In the third section, we explain cryptographic protocols. Finally, properties that these protocols aim to satisfy are explained.

1.1.1 Cryptography and Cryptanalysis

Cryptography is the "art and science of keeping messages secure" [Sch96]. Cryptographic techniques allow a sender to send a message to a receiver securely: the sender makes sure that only the receiver can read the message and an eavesdropper cannot read it. *Encryption* is the process of transforming a message in order to hide its meaning. *Plaintext* refers to the original message before encryption, and *ciphertext* refers to the encrypted message. *Decryption* is the process of recovering the plaintext from a ciphertext. Hence, to send a message securely, a sender first encrypts the message and sends the ciphertext to the receiver. Subsequently, using decryption techniques, the receiver decrypts the ciphertext to obtain the original plaintext.

Cryptography attempts to hide the meaning of a message, but not its existence. This is achieved by encrypting the message. This is opposed to *steganography* where the aim is to hide the existence of the message itself; leaving the plaintext untransformed. Sometimes, it is infeasible to hide the existence of the communicated messages. In such cases, cryptographic techniques are used. Note that it is even possible to combine cryptography and steganography to maximize security.

Cryptanalysis is the process of breaking ciphertext. *Cryptanalysts* attempt to find the plaintext of an encrypted message. Modern cryptography and cryptanalysis depend heavily on mathematics. Cryptology is the branch of mathematics that consists of cryptography and cryptanalysis.

Cryptography plays a vital role in achieving security in today's communication systems. For instance, when a computer sends data to another computer on the Internet, data passes through many computers along the path from the sender to the receiver. If data is not encrypted, the computers along the path of transmission will have the opportunity to access and observe the data. Another application area of cryptography is in wireless networks where data is broadcast from one station to another. Such data needs to be encrypted to achieve secure communication between communicating stations [Pat97, Ros04].

1.1.2 Cryptographic Algorithms

A *cryptographic algorithm*, or a *cipher*, is the mathematical function used for encryption and decryption. Modern cryptographic algorithms use keys for the encryption and decryption processes. A sender uses an encryption key to encrypt a message using a cryptographic algorithm. The receiver uses the decryption key to decrypt the ciphertext. Thus, if M denotes the plaintext, then $C = E(M, K)$ denotes the ciphertext which is obtained by encrypting M with the encryption key K. The function E has two parameters: the plaintext and the key. Note that $M = E^{-1}(C, K)$ denotes the plaintext M which is obtained by decrypting the ciphertext C using the key K.

There are two types of key based cryptographic algorithms: *symmetric* and *public key* algorithms. In symmetric key algorithms, the same key is used for both encryption and decryption. The key is kept secret. The security of the algorithm depends on securing the key; forging the key means anyone could encrypt and decrypt messages. The *Data Encryption Standard (DES)* is an example of a symmetric key cryptographic algorithm [US 99].

In public key algorithms, the key used for encryption is different from the key used for decryption. The encryption key is kept public: anyone can use it to encrypt a message. However, only someone with the decryption key can decrypt the message. The decryption key cannot be calculated from the encryption key. Encrypting the plaintext M using the public key K^+ results in the ciphertext $C = E(M, K^+)$. Subsequently, only someone with the private key, $K^- = AssociatedKey(K^+)$ can decrypt C to obtain M. *RSA* (invented by Ron Rivest, Adi Shamir, and Leonard Aldeman in 1977) is an example of a public key cryptographic algorithm [Hel78, Sch96].

Given a ciphertext C, cryptanalysts attempt to find the plaintext M such that $C = E(M, K)$, where K, unknown to the cryptanalyst, is the key used to encrypt M. Cryptanalysts break a cryptographic algorithm by observing ciphertext messages generated by the algorithm, or having access to a collection of plaintext-ciphertext pairs. Many of the currently used cryptographic algorithms are very difficult to break. An

extensive discussion of various topics on cryptography and cryptographic algorithms can be found in [Lab95, Sch96, Sti02]. An excellent introduction with plenty of historical background, but less technical presentation, can be found in [Kah97, Sin99].

There are different *standards* that govern the development and use of cryptographic algorithms and protocols. For instance, the Computer Security Division (*CSD*), a division of the National Institute of Standards and Technology (*NIST*), has the mission to develop standards and validation requirements for cryptographic systems [The04b]. CSD is currently in the process of developing a comprehensive cryptographic toolkit that will guide US government agencies and others to select cryptographic algorithms and protocols for protecting their communication systems [The04a].

1.1.3 Cryptographic Protocols

A *cryptographic protocol* is a sequence of message exchanges between a set of communicating agents that attempts, using cryptographic algorithms, to distribute secrets to some of those agents in order to meet specified security properties [BAN90, CJ97]. Generally, a cryptographic protocol involves two communicating agents who exchange three to six messages, with the help of a trusted server. The exchanged messages are composed from components such as keys, random numbers, timestamps, and signatures. At the end of the protocol, the agents involved may deduce certain properties such as the secrecy and authenticity of an exchanged message.

Consider the following problem. Assume a symmetric key algorithm is used to secure communication in a network of agents, or computers. In this setting, each pair of agents uses a distinct, shared secret key to encrypt and decrypt messages. Thus, if there are N agents in the network, then roughly N^2 distinct keys should be distributed to the network agents [RS01]. This is impractical especially if N is large. Clearly, there is a need for a mechanism to distribute secret keys when required. This is done using a key exchange cryptographic protocol. Thus, if two agents want to

communicate, then they follow the protocol to establish a secret, session key. Key exchange cryptographic protocols must ensure the secrecy of the exchanged key.

Even if the cryptographic algorithm is secure, *flaws* in the cryptographic protocol may result in compromising the security of the communication system. Flaws in cryptographic protocols may allow an intruder to authenticate as someone else, or gain information that should not be otherwise revealed. Thus, the security of a communication system depends not only on the security of the cryptographic algorithm, but also on the security of the cryptographic protocol. In this research, 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 analyzed protocol structure.

An example cryptographic protocol is the *Needham-Schroeder Secret Key (NSSK)* protocol [NS78]. It uses a symmetric key cryptographic algorithm, and is designed to enable two agents, A and B, to establish secure session keys with the help of a trusted server, say J. Initially, both of the registered agents, A and B, share private, long-term keys with J (K_{AJ} and K_{BJ}, respectively). Thus, each of A and B can communicate securely with J, but they are not able to communicate directly with each other. We will use the following notation: i. *Send* (X, Y, *data*) to indicate that in the ith step of the protocol agent X sends message *data* to agent Y. The protocol proceeds as follows:

1. $Send(A, J, ID(A) \oplus ID(B) \oplus RN_A)$
2. $Send(J, A, E(RN_A \oplus ID(B) \oplus K_{AB} \oplus E(K_{AB} \oplus ID(A), K_{BJ}), K_{AJ}))$
3. $Send(A, B, E(K_{AB} \oplus ID(A), K_{BJ}))$
4. $Send(B, A, E(RN_B, K_{AB}))$
5. $Send(A, B, E(RN'_B, K_{AB}))$

Here, $E(data, key)$ stands for the ciphertext of the message *data* encrypted using the key *key*. Refer to Appendix A for information on the notation we use to describe

cryptographic protocols. Inside a message, $ID(A)$ and $ID(B)$ refer to the identifications of agents A and B, respectively. RN_A and RN_B are random numbers; generated and originally known only by A and B, respectively. These are also called *nonces*. RN'_B is a nonce obtained by applying a standard function on RN_B as specified in the protocol implementation. For instance, if RN_B represents an integer, then it may be agreed that RN'_B is $RN_B - 1$. K_{AB} denotes the newly established session key. A message $W \oplus Z$ denotes a message composed of the text W concatenated by Z.

Thus, in the first step of the NSSK protocol, A tells J that it wants to communicate with B. A supplies J with the message containing the ids of A and B, and the nonce RN_A. Then, in the second step, J creates a new session key K_{AB} and returns to A the message $RN_A \oplus ID(B) \oplus K_{AB} \oplus E(K_{AB} \oplus ID(A), K_{BJ})$ encrypted using K_{AJ}. At this point, A uses its secret key, K_{AJ}, to decrypt the message it has received and thus obtains the session key K_{AB}. Note that A has also received $E(K_{AB} \oplus ID(A), K_{BJ})$, but this can only be decrypted with K_{BJ}; which is not known by A. A forwards $E(K_{AB} \oplus ID(A), K_{BJ})$ to B in the third step of the protocol. When B receives this message, it uses its secret key K_{BJ} to decrypt it. Thus, B obtains K_{AB} and believes that this session key is intended for communication with A. Now, B creates a new nonce, RN'_B, encrypts it under K_{AB} and sends the encrypted message to A as indicated by the fourth step. A decrypts this message, calculates RN'_B and sends B the message $E(RN'_B, K_{AB})$. Finally, A and B can use K_{AB} to encrypt and decrypt exchanged messages, thus if the protocol is correct, A and B should communicate securely.

Figure 1.1 shows the *message sequence diagram* for the NSSK protocol. The message sequence diagram is a graphical representation that aids in describing cryptographic protocols. A box represents an agent, it is labeled by the agent's identity. An arc represents a message flow, it is labeled by the message number. We show the keys that are assumed to be initially owned by an agent, beside its box. We do not include public keys, nor keys originated by the agent itself.

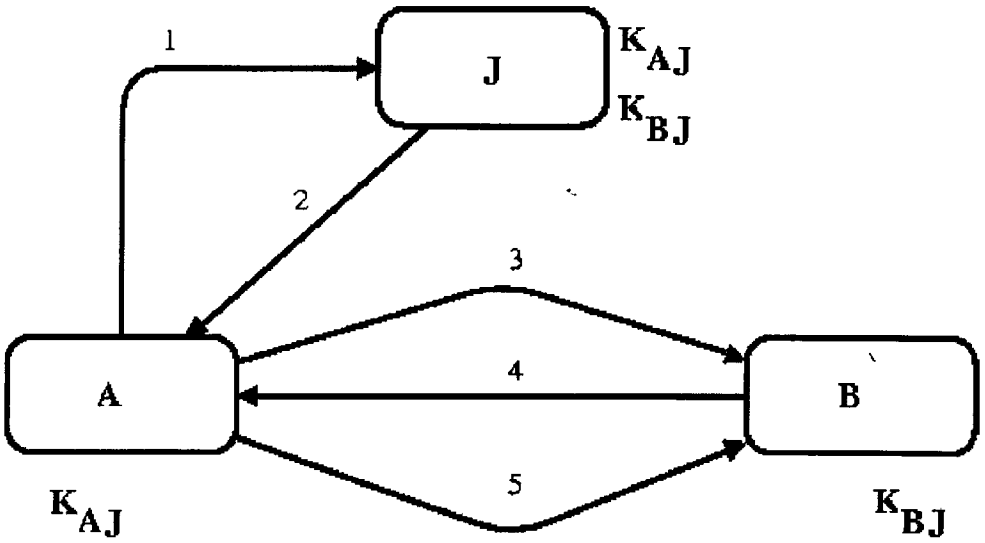


Figure 1.1: The message sequence diagram of the NSSK protocol

1.1.4 Properties of Cryptographic Protocols

A cryptographic protocol is designed to provide a number of *security properties*. The following explains informally some of these properties [RS01]:

1) *Secrecy* (or *confidentiality*): the cryptographic protocol ensures that secret exchange of information is restricted to the appropriate communicating parties. This means that an intruder must be prevented from deriving the plaintext of the encrypted messages passing between the communicating agents. This is the most important property for a cryptographic protocol.

2) *Authentication*: the cryptographic protocol ensures that a message that is claimed to be from a communicating agent was indeed originated by that agent. That is to say, an intruder cannot impersonate an agent.

3) *Integrity*: the cryptographic protocol ensures that data sent during communication cannot be modified, or at least the protocol is able to detect any such modification.

4) *Non-repudiation*: After a protocol run, non-repudiation means that the parties involved should not have the ability to deny their involvement in a communication session. This means that after a protocol run, the recipient should have proof that the received message was indeed sent by the claimed sender, even if the sender denies it.

There are many cryptographic protocols. They can be categorized based on many factors such as the cryptographic approach taken *i.e.* symmetric or public key, the number of parties involved, the number of exchanged messages, and whether or not one or both of the communicating parties are to be authenticated. Also, some protocols use a combination of symmetric and public key cryptographic algorithms. Furthermore, cryptographic protocols differ also in the features they use to satisfy security properties *e.g.* timestamps, nonces, and signatures. Also, they may differ in the security properties they satisfy. The choice of a protocol also depends on other factors, including the underlying communication architecture *e.g.* number and size of messages [Sch96].

1.1.5 Remarks

The cryptographic protocols studied in this research are sometimes referred to as *authentication and key exchange protocols*. Here, agents A and B are on opposite ends of a network and want to communicate securely. The aim of these protocols is to allow A and B to exchange a secret, and at the same time each of A and B is confident that he or she is talking to the other and not to an intruder. Such protocols aim to satisfy at least the following basic security properties: secrecy, authentication, and integrity. These protocols include the following, noting that some of these protocols have different versions: *Needham-Schroeder* [NS78], *Wide-Mouth Frog* [BAN90], *Yahalom* [BAN90], *Otway-Rees* [OR87], *Kerberos* [NT94], *Newman-Stubblebine* [NS93], *TMN* [TMN90], *IKE* (Internet Key Exchange) [HC98], and handshaking protocols in the *SSL (Secure Socket Layer Protocol)* [FKK96] which runs on

top of *TCP/IP(Transmission Control Protocol/Internet Protocol).*

In the next section, we discuss how an intruder can attack a protocol. Also, we discuss reasons that make the verification of cryptographic protocols difficult. Formal methods for the verification of cryptographic protocols are also introduced.

1.2 Background on the Verification of Cryptographic Protocols

In this section, we introduce the verification of cryptographic protocols. First, the notion of an intruder is introduced. Then, we list attack strategies that an intruder might employ to attack a cryptographic protocol. Example attacks are explained. We discuss issues and formal methods related to the verification of such protocols.

1.2.1 Intruder Attacks on Cryptographic Protocols

A cryptographic protocol is implemented by a network of agents, for example computers, who want to communicate securely. Typically, the protocol involves two agents with another agent acting as a server that facilitates the distribution of a secret. It is assumed that this network uses an *open addressing scheme* [RS01]. This means that messages carry fields that indicate the source and destination. Thus, a dishonest agent may modify the source field of a message to make it appear to come from a different source.

In analyzing a cryptographic protocol, all possible actions by an intruder must be considered. This is problematic since there is an infinite number of possible intruder actions. An *intruder* is an attacker who wants to undermine the security of a protocol. An intruder can perform the following *actions* to mount attacks [RS01]: 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.

These actions are not independent. Furthermore, some of these actions may not be possible depending on the underlying communication system. For instance, in a wireless system, where messages are broadcast over the air, the intruder may not be able to prevent messages from reaching their destination.

The intruder manipulates messages as outlined above to mount an attack on the protocol. In this research, we are concerned with attacks that result from flaws inherent in the protocol. Here is a partial list of *attack strategies* that an intruder might use:

1) *Man-in-the-middle attack*: the intruder behaves as an eavesdropper who intercepts, and possibly alters messages sent from one agent to another. Thus, the intruder gains information that may be useful in undermining the security of the protocol.

2) *Oracle attack*: the intruder tricks an honest agent into giving away some information that is not obtainable otherwise. This style of attack may require that the intruder uses messages from a different run of the protocol, or messages from an entirely different protocol. An example oracle attack will be discussed later when analyzing the TMN protocol.

3) *Replay attack*: the intruder replays one or more messages obtained from a previous run of the protocol. This attack is possible if the protocol has no mechanism to distinguish between separate runs. Nonces and timestamps are used to prevent such attacks. An example replay attack will be discussed later when analyzing the TMN protocol.

4) *Interleave attack*: the intruder simultaneously engages in two or more runs of the protocol to undermine its security.

5) *Forging of keys attack*: the intruder compromises an old session key and uses it to undermine security of future protocol runs. A session key may be com-

promised due to successful cryptanalysis or by other means, such as burglary, blackmail, or bribery [RS01].

6) *Algebraic attacks*: the intruder exploits algebraic properties of the cryptographic algorithms.

7) *Type attacks*: the intruder exploits type properties of the message fields [HLS00].

8) *Guessing attacks*: the intruder guesses secrets used by the communicating agents, and uses them in attacking the protocol [CMAFE03, Low02].

These attacks are not mutually exclusive and an intruder may mount a combination of two or more of these attacks on a given protocol. A good discussion of such attacks is provided in [CJ96].

1.2.2 Example Intruder Attacks

We illustrate different strategies an intruder might use to mount an attack on the *TMN protocol* [LR97, TMN90]. The protocol involves two agents, A and B, and a server, J, to facilitate the distribution of the session keys. Its message sequence diagram is shown in Figure 1.2. Initially, the protocol assumes that both A and B know the public key of J, K_J^+. Here are the protocol steps:

1. $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+))$
2. $Send(J, B, ID(A))$
3. $Send(B, J, ID(A) \oplus E(K_{AB}, K_J^+))$
4. $Send(J, A, ID(B) \oplus E(K_{AB}, K_{AJ}))$

K_{AJ} and K_{AB} are symmetric keys freshly created by A and B, respectively. K_{AJ} must be known only to both A and J; and is used to send K_{AB} in an encrypted form as indicated in step 4 of the protocol. K_{AB} must be known only to A, B and J; and is used as a session key. Thus, A uses K_{AB} to encrypt plaintext messages it sends to

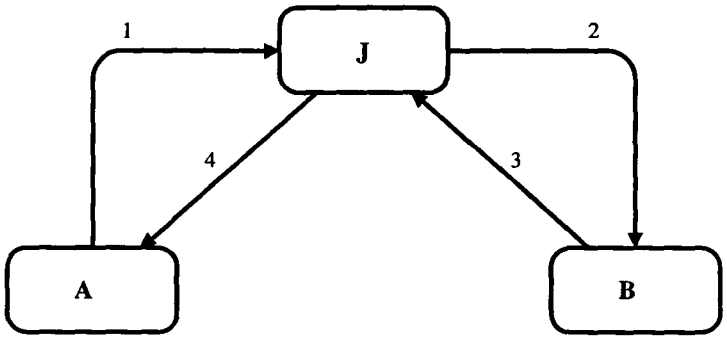


Figure 1.2: The message sequence diagram of the TMN protocol

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 instance.

There are more than four known attacks on the TMN protocol. A simple attack is illustrated as follows, noting that $PosesAs(I, A)$ denotes I impersonating A. Thus, a step of the form "$Send(I, B, X), where\ PosesAs(I, A)$" means that I poses as A and sends X to B, whereas a step of the form "$Send(B, A, X), where\ PosesAs(I, A)$" means that I intercepts the message X; originally sent from B to A.

1. $Send(I, J, ID(B) \oplus E(K_I, K_J^+)), where\ PosesAs(I, A)$
2. $Send(J, B, ID(A))$
3. $Send(B, J, ID(A) \oplus E(K_{AB}, K_J^+))$
4. $Send(J, A, ID(B) \oplus E(K_{AB}, K_I)), where\ PosesAs(I, A)$

This attack is an oracle attack. First, the intruder posing as A, sends the message $ID(B) \oplus E(K_I, K_J^+)$ to J. The server, J, thinks that A wants to start a communication session with B. Thus, J sends B the message $ID(A)$ as indicated in the second step of the protocol. Now, B thinks that A wants to initiate a connection. Therefore, as indicated by the third step of the protocol, B sends J the message $ID(A) \oplus E(K_{AB}, K_J^+)$. Thus, J uses its private key to obtain K_{AB} and sends the message $ID(B) \oplus E(K_{AB}, K_I)$ to A. However, the intruder I intercepts this message

and uses its session key, K_I, to obtain K_{AB}. Note that this is considered an oracle attack since the intruder I has tricked the honest agent J to supply K_{AB}.

Another attack is described as follows:

1. $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+))$
2. $Send(J, B, ID(A)), where\ PosesAs(I, B)$
3. $Send(I, J, ID(A) \oplus E(K_I, K_J^+)), where\ PosesAs(I, B)$
4. $Send(J, A, ID(B) \oplus E(K_I, K_{AJ}))$

This attack, as opposed to the previous one, allows the intruder I to authenticate as B. Similar to the previous attack, this attack is an oracle attack.

The following attack is a replay attack:

1.($I1$) $Send(I, J, ID(B) \oplus E(K_I, K_J^+)), where\ PosesAs(I, A)$
2.($I2$) $Send(J, B, ID(A))$
3.($I3$) $Send(B, J, ID(A) \oplus E(K_{AB}, K_J^+))$
4.($I4$) $Send(J, A, ID(B) \oplus E(K_{AB}, K_I)), where\ PosesAs(I, A)$
5.($II1$) $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+))$
6.($II2$) $Send(J, B, ID(A)), where\ PosesAs(I, B)$
7.($II3$) $Send(I, J, ID(A) \oplus E(K_{AB}, K_J^+)), where\ PosesAs(I, B)$
8.($II4$) $Send(J, A, ID(B) \oplus E(K_{AB}, K_{AJ}))$

This attack involves two separate runs of the protocol; labeled I and II. It is also considered a replay attack since the message $ID(A) \oplus E(K_{AB}, K_J^+)$ is replayed in the second run, as indicated by steps $I3$ and $II3$. After the completion of the two runs, A and B communicate with the key K_{AB}, which has been compromised by I. Thus, I is able to fake a message and claim that it has been originated by A or B; a compromise of the authentication property. Furthermore, I is able to know secret communication between A and B; a compromise of the secrecy property. Also, I is able to modify

or corrupt communication messages between A and B; a compromise to the integrity property. Note that the ability of the intruder I to eavesdrop on the communication between A and B did not exist in the previous attacks.

There is a fourth attack compromising secrecy of the TMN protocol. This attack depends on the algebraic properties of the cryptographic algorithms that are used in the protocol. For more information on this attack, refer to [Sec04b].

1.2.3 The Difficulty of Cryptographic Protocol Verification

The literature is rich with flawed cryptographic protocols. Some of these protocols appear to be secure; some have even been implemented in actual networks, but subsequently have been found to contain flaws. Refer to [CJ97] and [Sec04a] for a list of possible attacks on many cryptographic protocols.

The difficulty of verifying cryptographic protocols is a result of many factors [RS01]:

1) The concurrent nature of cryptographic protocols increases analysis complexity.

2) It is extremely difficult to capture the different ways by which an intruder may attack a protocol.

3) Informal reasoning on cryptographic protocols is prone to errors. Consider the following example: the *Needham-Schroeder Public Key (NSPK) protocol*. Note that this protocol is different from the Needham-Schroeder Secret Key (NSSK) protocol described earlier. The NSPK protocol is as follows:

   1. $Send(A, B, E(ID(A) \oplus RN_A, K_B^+))$
   2. $Send(B, A, E(RN_A \oplus RN_B, K_A^+))$
   3. $Send(A, B, E(RN_B, K_B^+))$

   Its message diagram is shown in Figure 1.3. Initially, the protocol assumes that A and B know each other's public key. After completing the protocol steps, agent B may reason as follows: since RN_B has only been generated by B

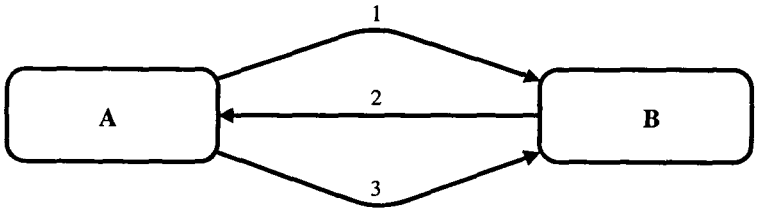


Figure 1.3: The message sequence diagram of the NSPK protocol

and sent to A encrypted using the public key of A, A should be the only one to know RN_B. Thus, by receiving the third message, B should be confident that it is interacting with A, that A knows RN_B and no one else knows RN_B. Similarly, A should be confident that it is interacting with B, that B knows RN_A and no one else knows RN_A. Thus, informal reasoning results in the conclusion that this protocol satisfies the authentication property; it authenticates A to B and vice versa. For many years the NSPK protocol was thought to be a correct authentication protocol. However, this protocol has been shown to be vulnerable to a combination of interleaving and oracle attacks [CJ97].

4) It is extremely difficult to capture the precise meaning of a security property in a verification tool. For instance, *BAN logic* [BAN90], one of the tools for verifying cryptographic protocols, was used to analyze and prove the authentication property of the NSPK protocol. It turns out that the NSPK protocol satisfies the *entity authentication property*; a weaker form of the authentication property. The entity authentication property means that A is confident of B's identity only recently, and vice versa [RS01].

1.2.4 Formal Methods for Cryptographic Protocol Verification

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 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* [BAN90], *GNY logic* [GNY90], and *MAO logic* [MB93].

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* [RS01].

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* [Kem89], and *NRL Analyzer* [Mea96].

A good survey of the different formal methods for verification of cryptographic protocols is provided in [Mea94], and [RH93].

There is no one method that can be used to model all aspects of cryptographic protocols, and thus detect all types of flaws [Mea94]. 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 [Fid01]. 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 [Hel98]. 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" [RH93] 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.

1.2.5 Concluding Remarks

An analyzer should be aware of the limitations and strengths of the formal method being used in the verification of a cryptographic protocol. For instance, an analyzer may prove a flawed cryptographic protocol correct if the analyzer is not aware of certain assumptions that should be met. Instances where an analysis has failed to detect certain flaws due to failed assumptions have occurred. Refer to [BM94] for an example of such failures in the BAN logic.

We conclude this section by emphasizing that there is no one formal method for proving all aspects of a cryptographic protocol. Furthermore, there is no one method that would guarantee that a protocol is flawless. As Schneier states, "the application of formal methods to cryptographic protocols is still a fairly new idea and it is really hard to figure out where it is headed. At this point, the weakest link seems to be in the formalization process." [Sch96]

1.3 The Verification Using Coloured Petri Nets

In this thesis, we explore the use of coloured Petri nets in the verification of cryptographic protocols. Coloured Petri nets are introduced in Chapter 2.

This approach is a finite-state analysis method. Thus, it involves modeling the protocol as a coloured Petri net, then an automated tool is used to generate all possible states. Insecurities are discovered if an insecure state is reachable.

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.

The graphical nature of coloured Petri nets adds understandability to the models. It aids communication between designers in the same way as flowcharts. Furthermore, coloured Petri net graphs resemble the informal drawings that designers use in the construction and analysis of systems. For instance, the notions of states, actions, and flow are represented in a straightforward way in coloured Petri nets.

Furthermore, there exists a large variety of algorithms for the analysis of coloured Petri nets. Several computer tools aid in this process. These factors have created an interest in applying coloured Petri nets in the verification of cryptographic protocols.

In the 1990s, several researchers at Queen's University applied coloured Petri nets in the verification of cryptographic protocols. They developed a technique to

model and analyze cryptographic protocols. A review of their technique is provided in Chapter 3.

1.4 Thesis Motivation

Several varieties of coloured Petri nets exist. The most established of these is the form developed by Jensen [Jen96a, Jen96b, Jen96c] . In the literature, many methods have been developed to analyze properties of such nets [Jen81, Jen96b]. Furthermore, there exist powerful automated tools that aid in the construction and analysis, such as Design/CPN [Col04e, Met93a]. Design/CPN is one of the most popular Petri net tools. Jensen's form of Petri nets and Design/CPN are introduced in detail in Chapter 2.

The technique developed by researchers at Queen's University uses a form of coloured Petri nets that is lower level than Jensen's CP-nets. 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. These features are explained in Chapter 2. Having such features would result in having smaller, easier to understand, and extendable models.

Furthermore, Design/CPN has not been explored as a potential automated verification tool. Instead, they developed a generic Petri net software 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 a stronger and more efficient verification. Examples of such features include: inscriptions, recurrence graph tools, hierarchical features and ML queries.

In this thesis, 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 technique developed at Queen's University. We focus on benefiting from the high level constructs of Jensen's

coloured Petri nets, as well as using Design/CPN.

1.5 Research Objective

We will pursue the following *research objective: Provide a technique for verifying cryptographic protocols using Jensen's form of coloured Petri nets.* This would be advantageous since Jensen's coloured Petri nets provide more suitability to cope with large net structures, and powerful and efficient computer tools; such as Design/CPN, exist to aid in the analysis and construction of CP-nets.

The new technique should address limitations of the other techniques that use coloured Petri nets. We aim to develop a better technique in terms of clarity of the model, the size of the generated occurrence graph, and the automation tools.

1.6 Contributions

The main contribution of this thesis 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.

- introduce the concept of a DB-place and used it to hold the intruder's knowledge. Other Petri net models do not use such a concept. This adds simplicity and clarity to the intruder models.

- 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 and the Needham-Schroeder public key authentication protocols.

1.7 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 introduces Petri nets and coloured Petri nets. First, Petri nets are introduced. This is followed by a detailed introduction to Jensen's form of coloured Petri nets and Design/CPN.

Chapter 3 serves as a literature review on the verification of cryptographic protocols using Petri nets.

Chapter 4 describes our new technique. We demonstrate the technique by using it in the modeling and analysis of the TMN protocol.

Chapter 5 is the conclusion chapter. It includes a discussion on the technique, as well as suggestions for possible future work.

Appendix A summarizes the functional notation we use to describe the protocols.

Appendix B presents the application of our technique in the verification of the Needham-Schroeder authentication protocol.

Chapter 2

Petri Nets and Coloured Petri Nets

In this chapter, Petri nets and coloured Petri nets are introduced. First, Petri nets are defined. Then, a brief overview of methods for their analysis is provided. Jensen's coloured Petri nets are then introduced. We discuss their definition, the concept of hierarchical nets, and a powerful tool for their analysis (Design/CPN).

2.1 Petri Nets

A *Petri net* is a graphical and mathematical tool. It has been used in many applications including the modeling and analysis of discrete-event systems [BBD01], concurrent systems [DJ02], fault tolerant systems [LFK91], communication protocols [MN90], distributed-software systems [PC92], control systems [RPD95] and parallel systems [WH94].

An inherent advantage of Petri nets is the generality of their models. As a mathematical tool, a Petri net model can be described using mathematical equations. These equations may then be analyzed to study their behaviour. As a graphical tool, Petri nets aid in visualizing the dynamic behaviour of systems. Furthermore, Petri nets are used as a specification tool to aid communication between designers in the same way as flow charts and networks. There exist many computer tools to assist in the

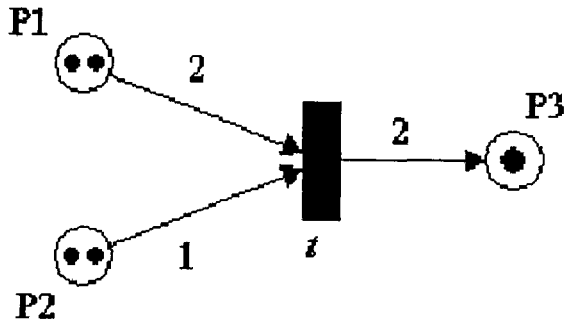


Figure 2.1: A sample Petri net

drawing and analysis of Petri nets.

The following definitions are burrowed from [Mur89]. A *Petri net* is a directed, weighted, bipartite graph consisting of two kinds of nodes: places and transitions. It is *formally defined* 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, $W : F \rightarrow \{1, 2, 3, \ldots\}$ is the weight function, and $M_0 : P \rightarrow \{0, 1, 2, \ldots\}$ is the initial marking. Figure 2.1 shows a sample Petri net. This graph represents the Petri net (P', T', F', W', M_0') where $P' = \{P1, P2, P3\}$, $T' = \{t\}$, $F' = \{(P1, t), (P2, t), (t, P3)\}$, $W' = \{((P1, t), 2), ((P2, t), 1), ((t, P3), 2)\}$, and $M_0' = \{(P1, 2), (P2, 2), (P3, 1)\}$.

Places are drawn as circles and *transitions* as rectangles. *Arcs* are either from a place to a transition or from a transition to a place. Arcs are labeled with their *weights* (positive integers). A *token* is represented as a black dot. Tokens occupy places. A *marking* (*state*) of a Petri net assigns a number of tokens to each place. A Petri net has an initial state called the *initial marking*.

The *dynamic behaviour* of a Petri net is described by its marking changes. There is

only one rule that governs the dynamic behaviour of Petri nets: the rule of *transition enabling and firing*. This rule is quoted directly from [Mur89] as follows:

1) A transition t is said to be *enabled* if each input place p of t is marked with at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc from p to t.

2) An enabled transition may or may not fire (depending on whether or not the event actually takes place).

3) A *firing* of an enabled transition t removes $w(p, t)$ tokens from each input place p of t, and adds $w(t,p)$ tokens to each output place p of t, where $w(t,p)$ is the weight of the arc from t to p.

For example, transition t, in Figure 2.1, is enabled since each of its input places $P1$ and $P2$ is marked with at least the required number of tokens. Place $P1$ should be marked with at least two tokens and place $P2$ with at least one token for transition t to become enabled. Since place $P1$ is marked with two tokens and place $P2$ is marked with at least one token, transition t is enabled. When transition t fires, two tokens are removed from place $P1$, one token is removed from place $P2$ and two tokens are added to place $P3$. Figure 2.2 shows the Petri net after firing transition t. Note that transition t cannot fire again since place $P1$ is not marked with the minimum required number of tokens, two in this case. Since no more transitions can fire, the Petri net is said to be *deadlocked*.

Figure 2.3 shows a Petri net modeling a simple communication protocol [Mur89]. The protocol involves two communicating processes: a sender and a receiver. The protocol is informally described as follows: "When the sender sends a message, it waits until it receives an acknowledgment. When the receiver receives a message, it sends back an acknowledgment". Note that the sender performs process 1 when the send operation is complete. Similarly, the receiver performs process 2 when the receive operation is complete.

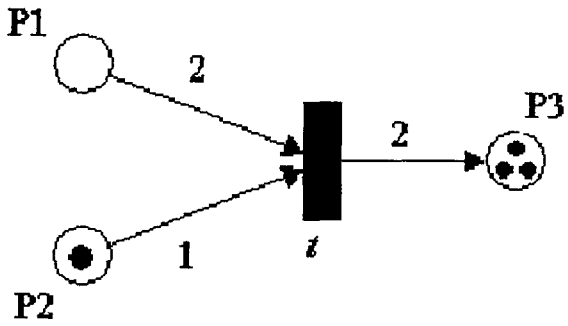


Figure 2.2: The sample Petri net after firing transition t

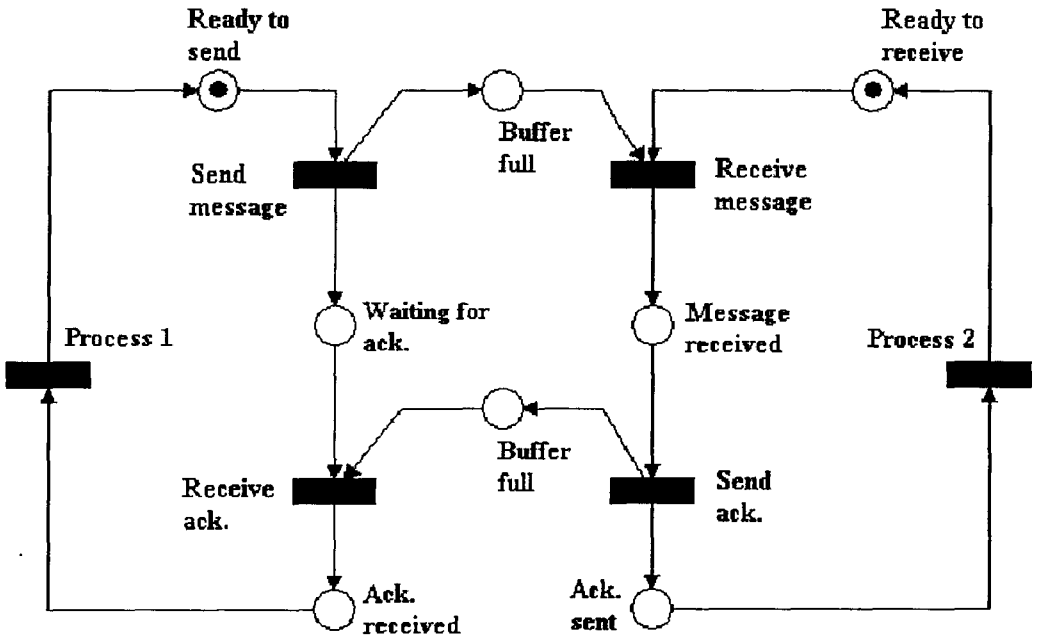


Figure 2.3: A Petri net model of a simple communication protocol

2.2 Analysis of Petri Nets

A Petri net can be analyzed for properties that reflect the concurrent behaviour of the system it is modeling. These include reachability, boundedness, liveness, reversibility, coverability, persistence, and fairness. A detailed explanation of these properties along with analysis methods can be found in [Mur89].

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.

2.3 High Level Petri Nets

Several extensions have been added to Petri nets to address the following factors. First, the majority of real life systems are complex in nature. Modeling a complex

system requires the use of a large Petri net (with a large number of places and transitions). As the net becomes larger, it becomes impractical to manipulate and analyze. The second factor is to provide more suitability for specific applications. For instance, the notion of time is introduced in timed Petri nets to help evaluate the performance of distributed networks [Zub98]. Examples of extended Petri net models are continuous nets [AD98], place/transactor nets [Fus87], FIFO nets [MF85], and timed Petri nets [Zub98].

As extensions were being added, the new Petri net models became more suitable for a single, and often narrow, application. This has created a need for general purpose Petri nets. This was met by the introduction of *high level Petri nets*.

One of the main features of high level Petri nets is the distinguishability of tokens. Tokens may belong to different *types*, or *colours*. Each colour consists of a set of values. Tokens of the same colour may take different values. A place contains only tokens of the same colour. *Inscriptions* are added to the transitions and arcs to indicate the colour of tokens to be removed by the firing events.

There are two types of high level Petri nets: *predicate-transition Petri nets* and *coloured Petri nets*. Predicate-transitions Petri nets are defined using the notion of sigma algebra [GL81]. Coloured Petri nets, on the other hand, are defined using lambda calculus; including the notion of types, variables, and expressions as known from functional programming languages [Jen96a, Pet04]. We should note that the two forms of high level Petri nets are very similar; in fact, they can be viewed as dialects of each other.

Figure 2.4 shows a sample coloured Petri net. After firing transition $T1$, the coloured Petri net becomes marked as shown in Figure 2.5; assuming the binding $x = a \wedge y = A$ holds.

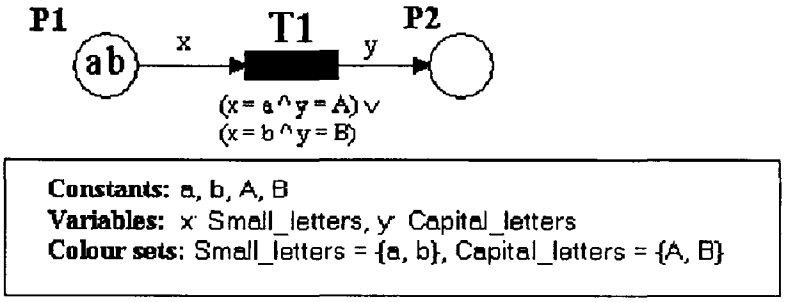


Figure 2.4: A sample coloured Petri net

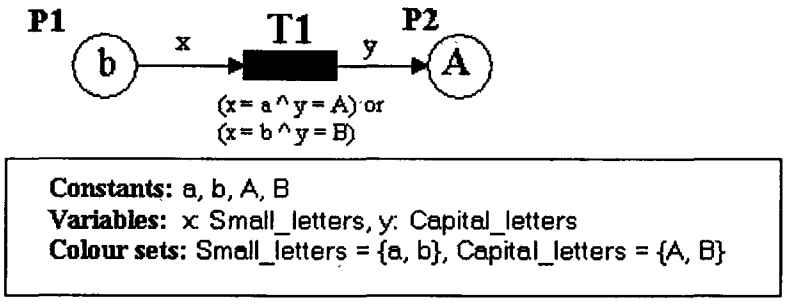


Figure 2.5: The sample coloured Petri net from Figure 2.4 after firing transition $T1$

2.4 Jensen's Coloured Petri Nets

There are many varieties of coloured Petri nets. We, however, are concerned with one particular variety: *Jensen's coloured Petri nets* [Jen96a]. 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 [Pau96, Var91]. This is what distinguishes CP-nets from other varieties of coloured Petri nets [Jen96a].

CP-nets are one of many different modeling languages. The following lists several features that make CP-nets a valuable modeling tool [Jen96a]:

1) CP-nets have a very appealing graphical form.

2) CP-nets provide a general modeling tool; they have been used to model and analyze a large variety of systems [Col04b, GV03, Jen96c].

3) Powerful computer tools exist to aid in the construction, analysis and simulation of CP-nets. One of these computer tools is *Design/CPN*. Design/CPN is a computer and graphical tool supporting the practical use of CP-nets. It has four integrated parts: the *CPN Editor* supports the construction and drawing of CP-nets, the *CPN Simulator* supports the simulation of CP-nets, the *Occurrence Graph Tool* supports the construction and analysis of occurrence graphs (reachability trees); and the *Performance Tool* supports the performance analysis of CP-nets. Design/CPN is a powerful tool, available free of charge. It supports all aspects of CP-nets, and has been used in several applications [Col04b, Col04a, Jen96c]. More information about Design/CPN is provided in section 2.8.

4) A large number of formal methods exist to analyze and prove properties of CP-nets [Jen96b].

5) CP-nets offer a semantics suitable to describe true concurrency.

6) CP-nets have a small number of powerful primitives.

7) CP-nets have a well-defined semantics which precisely describe their behaviour.

8) CP-nets provide mechanisms for abstraction and hierarchical design that help in coping with complex systems.

A formal definition of CP-nets is provided in Section 2.6. An example of a Design/CPN model is provided in [Jen97]. A comprehensive guide to the practical use of CP-nets and Design/CPN can be found in [KCJ98]. Analysis methods and theoretical background are provided in [GV03, Jen96a, Jen96b, Mur89].

2.5 A CPN Example Model

A simple CPN modeling an intruder decryption process is shown in Figure 2.6. The CPN model is created and simulated using Design/CPN. The CPN models an intruder who has intercepted two ciphertexts: $C(1)$ and $C(2)$. The intruder has the decryption key $K(1)$, which can be used to decrypt the ciphertext $C(1)$. As a result, the intruder obtains the plaintext $P(1)$. Since the intruder does not have the decryption key $K(2)$, the intruder will not be able to decrypt $C(2)$.

The CPN model has three places. The places are named: *Intercepted*, *Possessed Keys* and *Decrypted*. The place *Intercepted* contains tokens of type *Ciphertext*, and initially has two tokens: $C(1)$ and $C(2)$. The place *PossessedKeys* contains tokens of type *Key*, and initially has one token: $K(1)$. The place *Decrypted* contains tokens of type *Plaintext*, and initially has no tokens.

There is only one transition, *Decrypt*. To become enabled, the following three conditions must be met:

a) A token is present in place *Intercepted*; as indicated by the variable c on the arc from place *Intercepted* to transition *Decrypt*.

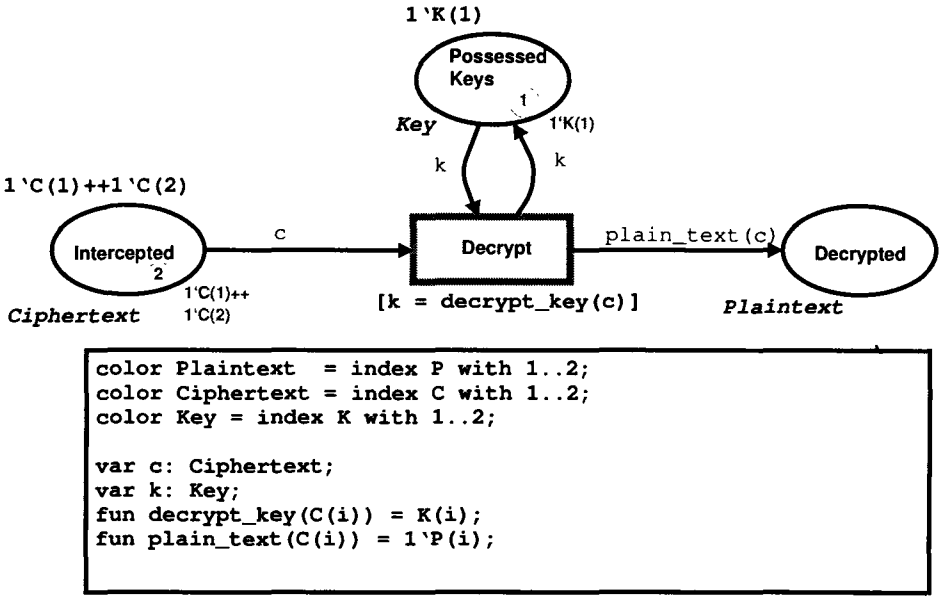


Figure 2.6: A CPN model of an intruder intercepting two ciphertexts: $C(1)$ and $C(2)$

b) A token is present in place *PossessedKeys*; as indicated by the variable k on the arc from place *PossessedKeys* to transition *Decrypt*.

c) The key token k is the correct decryption key of the ciphertext token c; as indicated by the guard of the transition *Decrypt* ($[k = decrypt\_key(c)]$). Thus, assuming the binding $c = C(1)$ then k must be bound to $K(1)$ to enable transition *Decrypt*; similarly, if $c = C(2)$ then k must be bound to $K(2)$.

Upon firing the transition Decrypt, the following actions happen:

a) The key token k is returned to place *PossessedKeys*; as indicated by the arc from transition *Decrypt* to place *PossessedKeys*.

b) A token of type *Plaintext* is created and added to the place *Decrypted*; as indicated by the function $plain\_text(c)$ on the arc from transition *Decrypt* to place *Decrypted*. Note that the added token represents the plaintext corresponding to the decrypted ciphertext. Thus, if $c = C(1)$ and $k = K(1)$ then $P(1)$ is

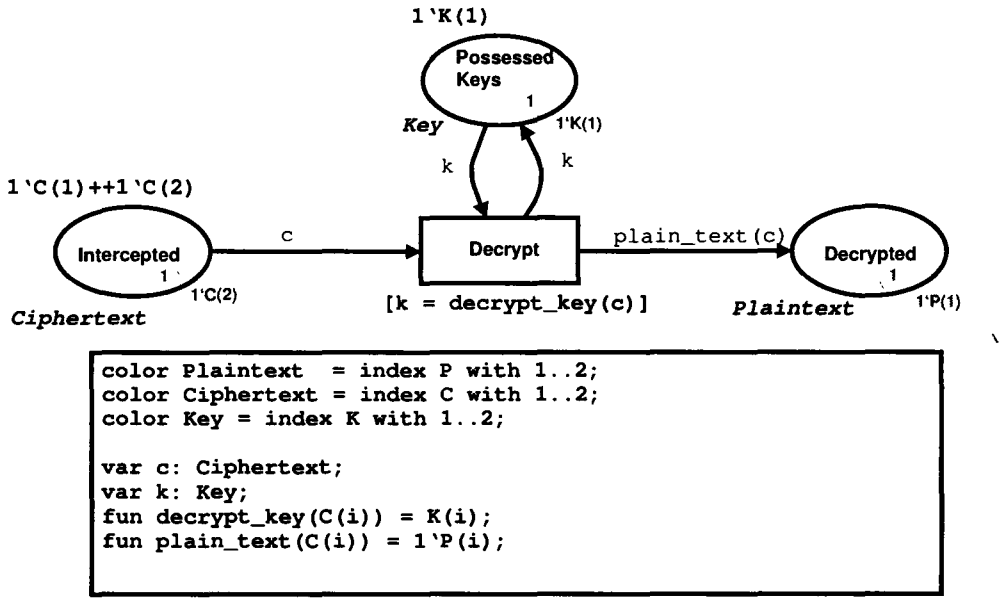


Figure 2.7: The CPN model from Figure 2.6 after firing trasnition *Decrypt*

added to place *Decrypted*. Similarly, if $c = C(2)$ and $k = K(2)$ then $P(2)$ is added to place *Decrypted*.

Since the intruder has only $K(1)$, as indicated by the initial marking, only $C(1)$ can be decrypted. Thus, transition *Decrypted* becomes enabled with the bindings $c = C(1)$ and $k = K(1)$. When it fires, token $C(1)$ is removed from place *Intercepted*, $K(1)$ is returned to place *PossessedKeys*, and token $P(1)$ is added to place *Decrypted*. Thus, the CPN becomes marked as shown in Figure 2.7.

Finally, note the CPN ML code used to define the colours, variables and functions of the CPN. Three colours are defined: $Plaintext = \{P(1),\ P(2)\}$, $Ciphertext = \{C(1),\ C(2)\}$, and $Key = \{K(1),\ K(2)\}$. Two variables are defined: c of type *Ciphertext* (can only be bound to $C(1)$ or $C(2)$), and k of type *Key* (can only be bound to $K(1)$ or $K(2)$). Two functions are defined: the function $decrypt\_key$: $Ciphertext \rightarrow Key$, returns the decryption key of a given ciphertext, and the function $plain\_text$: $Ciphertext \rightarrow Plaintext$, returns the plaintext corresponding to a given ciphertext.

2.6 Formal Definition of CP-nets

In this section, we provide a formal definition for non-hierarchical CP-nets [Jen96a].

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.* $|ms1| = 3$. X_{MS} denotes the set of all multi-sets over X.

A *CPN* [Jen96a] 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.

In this section, the examples provided use Figure 2.8. This CPN is represented as follows:

(i) $\Sigma = \{Plaintext, Ciphertext, Key\}$.

(ii) $P = \{Intercepted, PossessedKeys, Decrypted\}$.

(iii) $T = \{Decrypt\}$.

(iv) $A = \{InterceptedtoDecrypt, PocessedKeystoDecrypt, DecrypttoPocessedKeys, DecrypttoDecrypted\}$.

(v) $N(a) = (SOURCE, DESTINATION)$ where $a \in A$ is in the form $SOURCE$-$toDESTINATION$.

(vi)

$$C(p) = \begin{cases} Ciphertext & \text{if } p = Intercepted, \\ Plaintext & \textit{if } p = Decrypted, \\ Key & \text{if } p = PocessedKeys. \end{cases}$$

(vii) $G(Decrypt) = [k = decrypt\_key(k)]$.

(viii)

$$E(a) = \begin{cases} c & \text{if } a = InterceptedtoDecrypt, \\ plain\_text(c) & \text{if } a = DecrypttoDecrypted, \\ k & \text{if } a \in \{PocessedKeystoDecrypt, DecryptoDecrypted\}. \end{cases}$$

(ix)

$$I(p) = \begin{cases} 1'C(1) + 1'C(2) & \text{if } p = Intercepted, \\ 1'K(1) + 1'K(2) & \text{if } p = PocessedKeys, \\ \emptyset & \text{if } p = Decrypted. \end{cases}$$

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

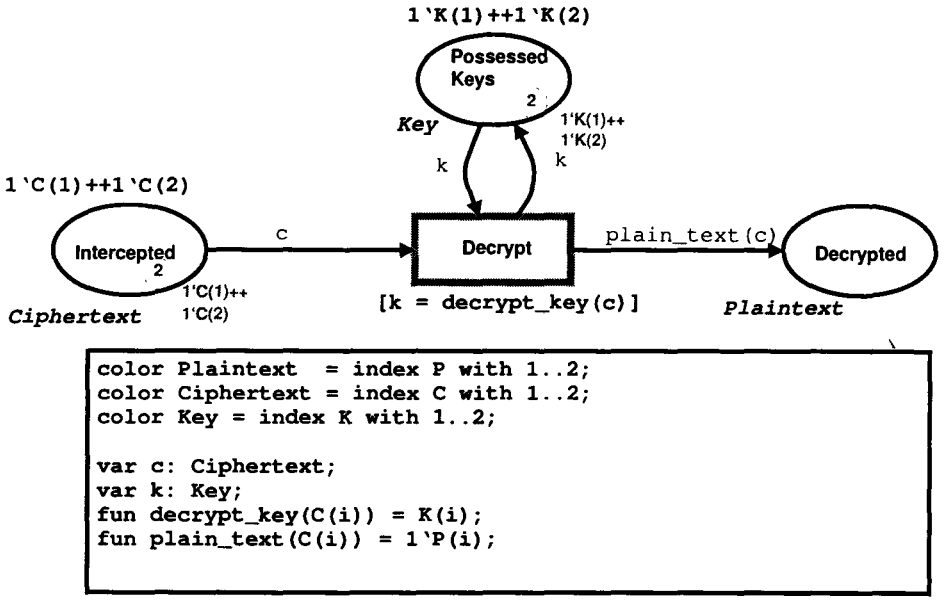


Figure 2.8: The CPN model from Figure 2.6 with a different initial marking

$\leq i \leq n$. For instance, $(Decrypt, \langle c = C(1), k = K(1)\rangle)$ and $(Decrypt, \langle c = C(2), k = K(2)\rangle)$ are two different binding elements.

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. For example, $M_0(Intercepted) = 1'C(1) + 1'C(2)$ and $M_0(\text{Decrypted}) = \emptyset$.

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. For instance, $(Decrypt, 1'\langle c = C(1), k = K(1)\rangle, 1'\langle c = C(2), k = K(2)\rangle)$ is a step.

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. For instance, the step $(Decrypt, 1'\langle c = C(1), k = K(1)\rangle)$ is enabled in M_0.

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. For instance, transition *Decrypt* is enabled in the marking M_0 for the binding $\langle c = C(1), k = K(1)\rangle$.

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. For instance, let $Y_1 = (Decrypt, 1'\langle c = C(1), k = K(1)\rangle)$ and $Y_2 = (Decrypt, 1'\langle c = C(2), k = K(2)\rangle)$ be two steps. Thus, $M_0[Y_1\rangle M_1[Y_2\rangle M_2$ is an occurrence sequence, where the markings M_1 and M_2 are shown in Figures 2.9 and 2.10, respectively. Both M_1 and M_2 are reachable from M_0.

2.7 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. In this section, we demonstrate concepts related to hierarchal CP-nets using the net system in Figure 2.11. It models two entities that exchange one token m.

A non-hierarchical CPN is called a *page*. In Design/CPN, each page has a *page name* and a *page number*. In this thesis, we use the page name to refer to a page.

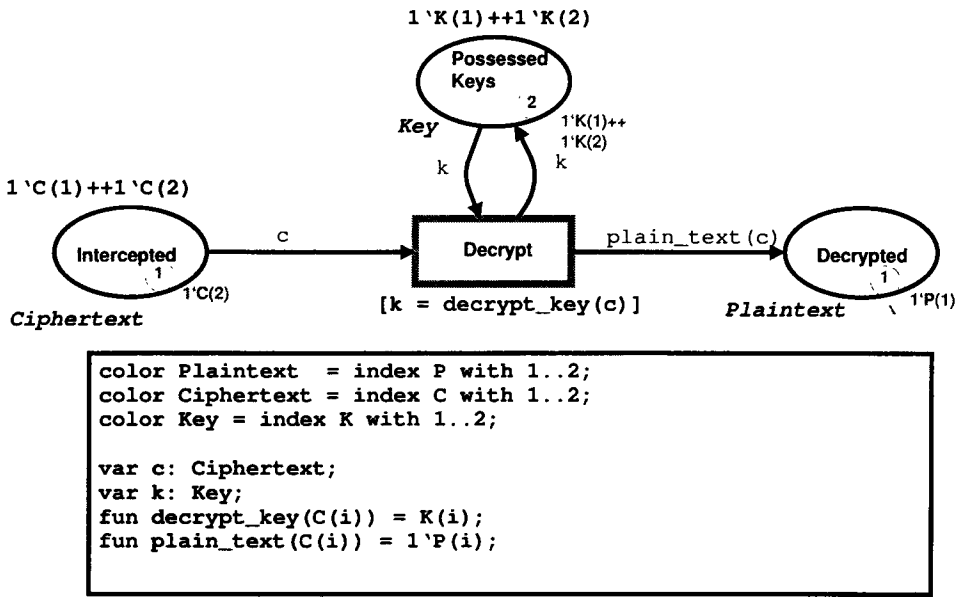


Figure 2.9: The CPN model from Figure 2.8 after firing the step Y_1

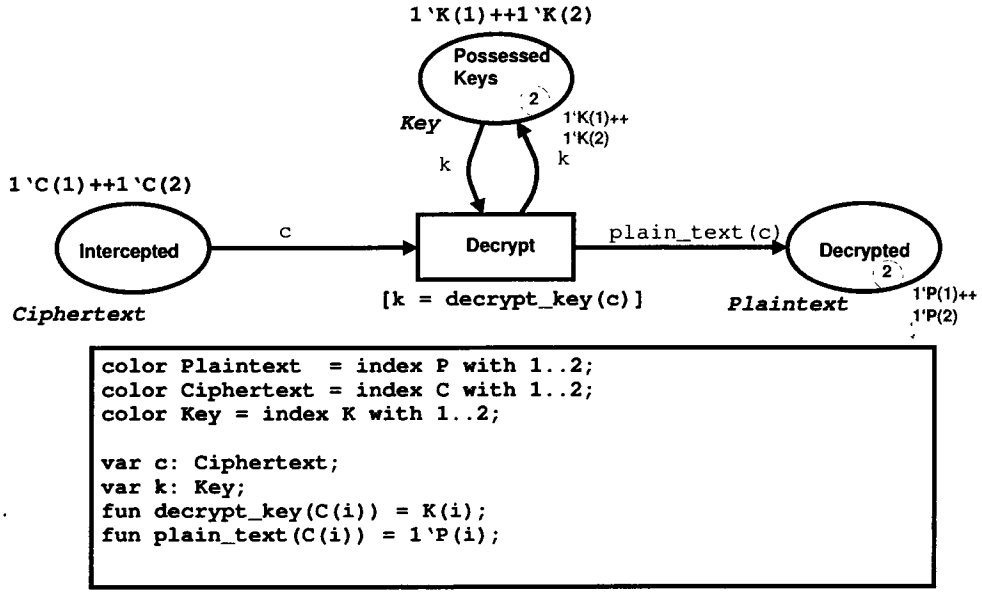


Figure 2.10: The CPN model from Figure 2.8 after firing the step Y_2

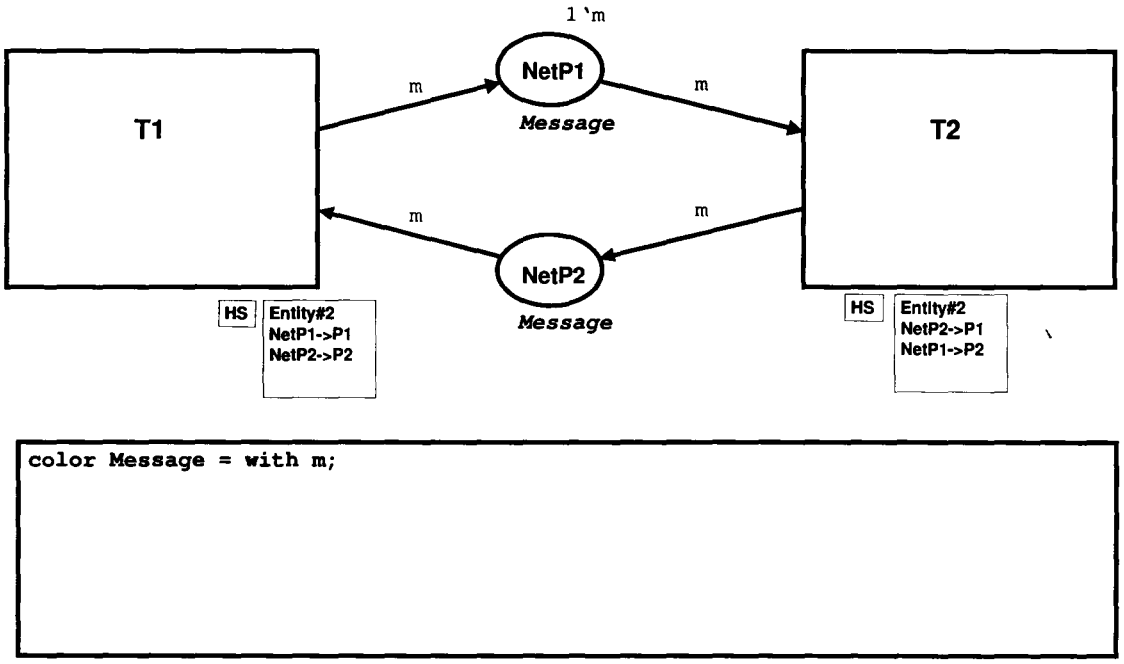


Figure 2.11: Page *Net* describes a primitive network of two entities

The page *Net* has two transitions $T1$ and $T2$, and two places $NetP1$ and $NetP2$. The two transitions are *substitution transitions*. Each of them has a small HS-tag adjacent to it, HS stands for Hierarchy + Substitution.

A substitution transition is a transition which is described in more detail in a separate CPN page. For instance, in Figure 2.11, $T1$ is defined in the page *Entity* shown in Figure 2.12. The page with the substitution transition is called the *super-page*. The page that defines a substitution transition is called a *subpage*. Thus, in the net system example, *Net* is a super page, while *Entity* is a subpage.

Each substitution transition corresponds to an *instance* of the subpage it is related to. For example, the page *Net* contains two instances of *Entity*: one corresponds to $T1$, and the other corresponds to $T2$. Each instance is independent from the other page instances. Hence, at simulation, the marking of an instance is independent from the other instances.

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*). For instance, consider the substitution transition $T1$ in page Net. Place $NetP1$ is a socket. It is related to the port $P1$ in the subpage instance $Entity$ that corresponds to $T1$. This is shown in the *hierarchical net inscriptions box* beside the HS-tag of $T1$. The first line indicates the name of the subpage that corresponds to the substitution transition. The following lines are socket/port assignments. Each line is of the form $SocketPlace- > PortPlace$ where $SocketPlace$ is the socket place and $PortPlace$ is the related port place. Thus, both $SocketPlace$ and $PortPlace$ will represent the same conceptual place, *i.e.* they always have an identical marking. Thus, for instance, the socket $NetP1$ and its related port $P1$ are conceptually the same place. 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 this thesis, 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. For example, place $P1$ is an output port for the $Entity$ page, while place $P2$ is an input port. This is indicated by the Out-tag and In-tag beside $P1$ and $P2$, respectively. Considering $T1$ in Net, the socket $NetP1$ is assigned to the output port $P1$, while the socket $NetP2$ is assigned to the input port $P2$. This is expressed by the hierarchal net inscriptions of the substitution transition $T1$ in Figure 2.11.

There can be multiple levels of substitutions. Thus, a superpage can have a substitution transition that corresponds to a subpage, which in turn has a substitution transition corresponding to another subpage, *etc.* In Design/CPN, the *page hierarchy* is very useful in visualizing how pages relate to each other. A page hierarchy is a directed graph which contains a node for each page and an arc for each direct

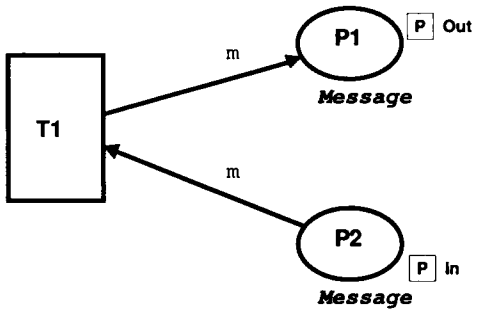


Figure 2.12: Page *Entity* describes the behaviour of an entity

superpage/subpage relationship. A page p_0 is a *direct subpage* of a page p_1 if there is a substitution transition in p_1 that corresponds to p_0. The page p_1 is said to be a *direct superpage*.

Figure 2.13 shows the hierarchy page of the net system. Note that an arc in the hierarchy is labeled with the name of the corresponding substitution transition. Thus, if an arc a from page p_0 to page p_1 is labeled with l, then this indicates that the superpage p_0 has a substitution transition named l which corresponds to an instance of the subpage p_1. Note that the hierarchy page can also help in finding the number of instances a page has in the system. For instance, using the hierarchy page, one can determine that there are two instances of *Entity*.

Looking at the page hierarchy of Figure 2.13, one notes that the page *Net* is labeled with the inscription "prime". This indicates that this page is a *prime page*. A prime page acts like a main function in programming languages; it is the first page that starts executing in the system. Each CPN must have a prime page.

The second construct to support hierarchical CPN construction is the fusion set construct. A *fusion set* is a set of places that are conceptually the same *i.e.* they always have the same marking. There are three types of fusion sets:

1) *Global fusion sets*: if two places p and q belong to the same global fusion set, then they are the same resulting place, regardless of being in different pages or page instances.

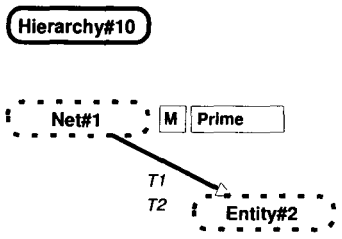


Figure 2.13: The hierarchy page of the net system

2) *Page fusion sets*: if two places p and q belong to the same page fusion set and to the same page, then they are the same resulting place, regardless of being in different page instances.

3) *Instance fusion sets*: if two places p and q belong to the same instance fusion set, to the same page, and to the same page instance, then they are the same resulting place.

We demonstrate the use of fusion sets in Figure 2.14. The place $P3$ belongs to the page fusion set MFS, as indicated by the tag FP and its inscription beside it. Note that the tag FI indicates an instance fusion set, while the tag FG indicates a global fusion set. The name to the side of the tag indicates the name of the fusion set.

Thus, as Figures 2.11 and 2.14 show, $P3$ in the first and second instances of *Entity* are conceptually the same resulting place. This implies that the two entities in our modified net system store the received message tokens into a common place.

2.8 Design/CPN

Design/CPN is a computerized package that supports the construction and analysis of CP-nets. It enables the automated editing, simulation, and verification of CP-nets. It is one of the most elaborate Petri net tools, with over 50 person-years of design and implementation [Col04e]. It has been used in many industrial applications [Col04b].

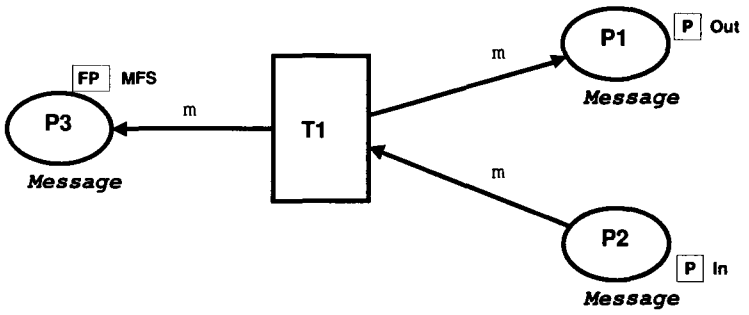


Figure 2.14: Using the page fusion set MFS in page *Entity*

Design/CPN was developed by the *Meta Software Corporation* [Met04] in cooperation with the *CPN group* at *Aarhus University*, Denmark [CPN04]. It was sold commercially during the period 1989 to 1995, with a price tag of over 24,000 US Dollars [Col04c]. Since 1996, Design/CPN has been available free of charge. It is used by over 750 organizations, including 200 commercial organizations [CPN04]. It is currently supported by the CPN group at Aarhus University.

Design/CPN has four integrated parts. One of these is the CPN Editor. It supports all issues related to the editing of CP-nets, including construction, modification, and syntax check. All CPN figures in this thesis have been produced using this editor. The editor is menu-driven. Thus, the user uses the menu to insert an object to the CPN, then the user moves and resizes it using the mouse. More information on using the CPN Editor is provided in [Met93a].

The second part of Design/CPN is the CPN Simulator. The simulator supports *interactive* and *automatic* simulation of CP-nets. The interactive simulation allows the user to select the firing steps. In automatic simulation, these steps are chosen automatically by the simulator. For instance, Figures 2.9 and 2.10 represent the state of the CPN after firing the steps $Y_1 = (Decrypt, 1'\langle c = C(1), k = K(1)\rangle)$ and $Y_2 = (Decrypt, 1'\langle c = C(2), k = K(2)\rangle)$, respectively. These steps have been selected by using the interactive simulation commands of the CPN Simulator. The user can also control the simulation by setting features such as breakpoints, code segments,

size of steps, *etc.* More information about the CPN Simulator is provided in [Jen96a] and [Met93a].

The third part of Design/CPN is the CPN Occurrence Graph (OG) tool. It supports all aspects related to the construction and analysis of occurrence graphs. As explained previously, a node in the occurrence graph represents a marking (state) of the CPN, while an arc represents a firing step that has one binding element. For example, Figure 2.15 shows the occurrence graph for the CPN of Figure 2.8. The dashed boxes beside the nodes (arcs) display the corresponding markings (firing steps). These are not initially displayed in the occurrence graph, but the user can double-click a node, or an arc, to display these boxes.

The user of the OG tool controls the construction of the occurrence graph by setting the *stop options* and *branching criteria*. Using the stop options, the user can set the OG tool to stop generating new nodes after a certain period of time, or when a condition becomes satisfied. The branching criteria, on the other hand, allows the user to control the shape of the occurrence graph. For instance, the user can request that the OG tool discovers at most one successor marking for all markings.

The user queries the generated occurrence graph using *OG queries*. An OG query enables the user to locate all the nodes in the occurrence graph that satisfy a certain condition. For instance, after generating the occurrence graph for a CPN modeling a cryptographic protocol, a query is run that locates all insecure nodes *e.g.* nodes where a key is not supposed to be in a certain place. Also, standard OG queries exist which can be used to check the dynamic properties of the CPN *e.g.* fairness, boundedness, *etc.* More information about queries can be found in [Met96].

Note that the OG tool is integrated with the CPN Simulator. For instance, one can select a node from the occurrence graph, move its state into the simulator, and continue simulation from the selected marking. Similarly, one can start generating an occurrence graph starting from a simulated state. This may be helpful in cases where only a partial occurrence graph, starting from a specific state, is needed.

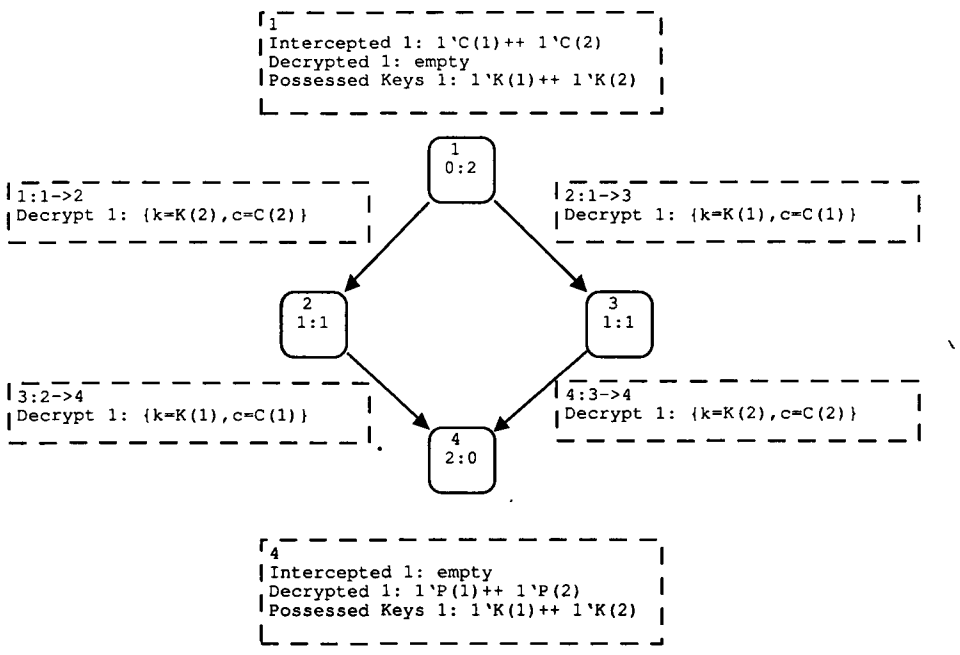


Figure 2.15: The occurrence graph for the CPN of Figure 2.8

The fourth part of Design/CPN is the Performance tool. It supports performance analysis of CP-nets with time extensions. In this thesis, such analysis is not used, thus we choose not to discuss this tool further at this point.

A tutorial on using Design/CPN can be found in [Met93b]. It is very informative, and discusses most issues related to using the editor and simulator. A full reference manual on Design/CPN is in [Met93a].

2.9 Remarks

There are several features of CP-nets that help the modeling of cryptographic protocols. First, CPN support for hierarchical construction allows us to build the models in a hierarchical manner. Thus, for instance, a CPN page can be constructed to model a single entity. Then, the full model is constructed by integrating the CPN models of the entities.

Another feature of CP-nets is their use of colour sets. This allows us to reduce the size of the model by using a single place to hold tokens that belong to the same colour set.

The use of fusion sets also helps reduce the size of the model. As will be seen in Chapter 4, fusion sets allow us to define a central place in the intruder CPN page to hold all tokens intercepted by the intruder. The result of this is a great simplification to the intruder model.

Finally, the existence of rich transition-related constructs, such as guards and arc expressions, allows us to specify complex actions of the protocol entities. For instance, a transition can be used to model the decryption of ciphertexts encrypted under a given key, where properties of the decryption are stated in terms of arc expressions.

In the following chapters, we apply the concepts introduced in this chapter to develop a new technique for verifying cryptographic protocols using coloured Petri nets. We also demonstrate how Design/CPN is used in this context.

Chapter 3

Cryptographic Protocol Verification using Coloured Petri Nets: A Review

This chapter provides a literature review on the verification of cryptographic protocols using coloured Petri nets.

3.1 Varadharajan Contributions

To the best of our knowledge, *Varadharajan* [Var90] was the first to suggest that Petri nets are a useful tool to model the flow of secret information in a security system. He suggested using coloured Petri nets as a security analysis tool; and proposed a Petri net formalism that can be used to model security policies. A *security policy* is a "set of rules and practices that regulate how a system is to manage, protect, and distribute sensitive information" [Var90].

In Varadharajan's extended Petri net formalism [Var90], tokens are associated with *security classes*. Insecurities are discovered if the security class of a token delivered to a place does not belong to the set of security classes of that place. Since security classes can be considered as token colours, coloured Petri nets are very suitable for Varadharajan's formalism.

The proposed framework can be used in the modeling and verification of security policies in a wide range of applications in a network environment. The verification of security policies is not directly related to cryptographic protocols. Nevertheless, Varadharajan was the first to suggest using Petri nets in the verification of security related requirements [LL97].

3.2 Behki Contributions

In the 1990s, several researchers at Queen's University applied coloured Petri nets in the verification of cryptographic protocols.

Behki was the first among these researchers to suggest a method for the design of general communication protocols using Petri nets [BT89]. He suggested using Petri nets to model the control flow of protocols. In addition, he proposed combining Petri nets and other modeling methods, especially methods based on programming languages and formal grammars, such as *LOTOS* [ISO89], to provide formal models at different levels of abstraction.

To aid in the hierarchical construction of Petri net models, Behki proposed the notion of *modules*, which later evolved into the notion of *Petri Net Objects* (*PNOs*). A PNO is a "black box" subsystem from which protocol entities are constructed. It is a Petri net enclosed in a box with transitions at the perimeter called ports [DTM95]. External access to the PNOs is only via their ports. Thus, communication between protocol entities is modeled by connecting the port on the sending PNO to the port of the receiving PNO through intermediate places, as shown in Figure 3.1.

3.3 Nieh Contributions

Although a Petri net formalism was presented by Behki to model communication protocols, it was Nieh who presented a coloured Petri net technique for the modeling

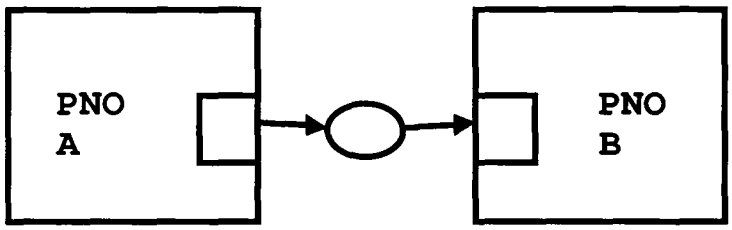


Figure 3.1: Two PNOs with a communication channel

and analysis of cryptographic protocols [Nie92, NT92].

3.3.1 Features

Nieh's model is described using three different levels of abstraction: entity, conceptual, and functional.

In the entity level description, the model includes high-level information on the communicating entities and the message sequences. In the conceptual level description, a lower level of abstraction is provided in which processes associated with each entity are listed *e.g.* send a message, receive a message, *etc.* In the lowest level of abstraction, the functional level, message contents are modeled as token colours. Moreover, cryptographic processes, such as encrypting messages, are modeled as transition firings.

Nieh's technique uses a top-down modeling approach, in which decomposition is based on the required level of abstraction. In a top-down approach, the system to be modeled is decomposed into subsystems, and these subsystems are further decomposed until the point is reached where the subsystems can simply be modeled as nets. In the model of Nieh, each entity is modeled as an independent subsystem (a PNO). Communication between protocol entities is modeled as token exchange between the PNOs.

In a cryptographic protocol system, entities perform actions *e.g.* send messages, receive messages, decrypt and encrypt ciphers, *etc.* In Nieh's technique, an action is

abstracted as a process that performs operations on a set of input data to generate a set of output data [NT92]. A cryptographic process is modeled as a transition. For example, a "send message" transition models the process of an agent sending a message. Also, an input/output medium is modeled as a place, while data items are modeled as tokens. For example, a place can be designated to hold an agent's keys, while a token of colour K_A represents the key created by an agent A.

3.3.2 Assumptions

The intruder is assumed to have full knowledge of the protocol. Furthermore, the intruder is assumed to have a record of all previous transactions between the protocol entities. It can also perform any operation defined in the protocol, such as signing messages, *etc.* Finally, the intruder is considered as a legitimate user in the system, and thus, it has full control of the communication medium.

3.3.3 Modeling the Intruder

The intruder is modeled as a PNO in the same way as the other protocol entities. Within this PNO, there is an intruder process corresponding to each message of the protocol. There are two subprocesses in each intruder process: *"Message Interception and Manipulation"* and *"Spurious Message Synthesis"*. The *"Message Interception and Manipulation"* subprocess models the intruder's ability to intercept and delete a message, or send a manipulated version of the message. The other subprocess, namely *"Spurious Message Synthesis"*, models the intruder action of synthesizing a spurious message.

3.3.4 Verification

After creating the model for a cryptographic protocol, its security criteria must be defined. The security criteria of a protocol are defined as the requirements that must

be met by the protocol to achieve its security goals. These criteria are then translated into requirements on the markings (states) of the Petri net model. For instance, if a protocol must ensure the secrecy of a key, then this can be stated, in terms of tokens, as the requirement that the token corresponding to the key never reaches a certain place in the intruder model.

An exhaustive search strategy is applied to find if an insecure state is reachable. An insecure state is a state where security criteria are violated. This search is implemented by constructing the state reachability graph corresponding to the Petri net model.

3.3.5 Limitations

The model of Nieh has two main drawbacks. First, the Petri net model is very large, with a large number of places and transitions. For instance, a large net is required to model the Yahalom protocol (a 5-line protocol) [CJ97], with over 160 transitions and 240 places. This significantly increases analysis complexity. Note that analysis complexity depends on the size of the occurrence graph. Furthermore, the larger the model, the harder it becomes for users to comprehend.

Second, the model only verifies man-in-the-middle attacks. Clearly, there is a need to verify against other attack strategies so the model can be reliably used to verify any generic cryptographic protocol.

To overcome these limitations, a number of contributions have been made by other researchers at Queen's University. All of these contributions build on the model of Nieh. Some contributions aim to decrease analysis complexity by using new analysis techniques [Sta94, STM94, TZ97, Zha97]. Other contributions aim to develop tools to automate the construction and analysis of the models [Edw98, ETM98]. The remainder of the contributions focus on generalizing the model by including more intruder attacks [Doy96, DTM95, Sha99].

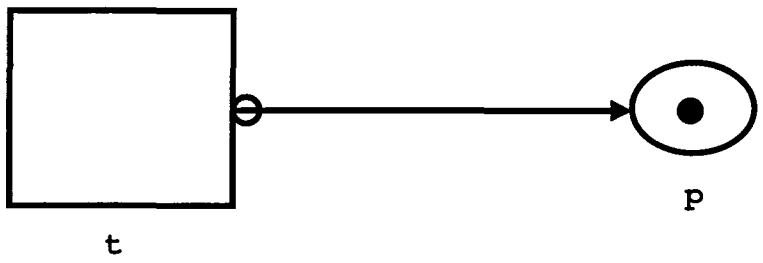


Figure 3.2: An inhibitor arc

3.4 Doyle Contributions

Doyle in [DTM95] aimed to extend Nieh's technique by modeling and analyzing new forms of attacks. In addition, Doyle used Prolog to parse and analyze the state reachability tree [Doy96, DTM95].

One of the Petri net extensions used by Doyle is the *inhibitor arc*. An inhibitor arc, a special arc drawn with a small circle on its tail, prevents a transition t from firing until the corresponding place p is empty. Thus, transition t in Figure 3.2 cannot fire.

Doyle applied the extended technique in the modeling and verification of authentication protocols. Authentication protocols are simpler than key exchange protocols, however, they are especially susceptible to interleaving attacks, including multiple iterations and parallel session attacks. While Nieh's intruder model detects man-in-the-middle attacks, Doyle's model was designed specifically to model interleaving attacks on selected authentication protocols.

Doyle used *Prolog* to automate analysis of the reachability tree [Doy96, DTM95]. Breadth first search is used to detect duplicate nodes and cut the search effort. Terminal states (leaf nodes) are categorized into classes. Terminal states are considered to belong to the same class if they have the same marking. Analyzing in terms of classes of terminal states reduces complexity resulting in shorter execution time of the tree analysis.

Finally, Doyle suggested the use of an object oriented language to construct an automated tool. He attributed this to the suitability of such languages to implementing objects. This is beneficial since PNOs can be implemented as objects.

3.5 Basyouni Contributions

One of the last researchers from Queen's University to extend the technique, Basyouni aimed to reduce analysis complexity and the size of Nieh's model [Bas97, BT97]. Basyouni achieved this by using backward state reachability analysis [STM94], and the reachability matrix description for coloured Petri nets [Jen81].

Basyouni's method involves solving the *incidence matrix equation*:

$$(M_n - M_0) = A \times \sigma^T$$

where M_0 and M_n represent the initial and the n^{th} marking after firing a sequence of n transitions, respectively. A is the incidence matrix which describes token changes after firing a transition. Marking vectors, M_0 and M_n, and the incidence matrix are as defined by Jensen [Jen81]. The transpose of the vector σ is denoted as σ^T. To solve this equation, all possibilities for the vector σ are tried. If σ exists, then M_n is reachable from M_0.

Basyouni claims that the elements of σ are either zero or one [Bas97, BT97]. This is because any insecure state can be reached by firing each transition, from the model's set of transitions, at most once [BT97].

Basyouni's technique uses two steps. The first one is the *Acceptance Check Step* (*ACS*), and the second is the *Matrix Analysis Step* (*MAS*).

The Acceptance Check Step (ACS) involves finding the *set of vulnerable data* (*VDS*) *i.e.* message contents, or fields, that can be modified by an intruder and still be accepted by a legitimate entity. Instead of using a general intruder model, the idea is to include only the VDS. This results in a smaller intruder model.

An example of a vulnerable data set is as follows. Assume the first step in an arbitrary protocol is of the form: *Send* $(A, B, K_A^+ \oplus A)$. Clearly, the intruder can substitute $\widetilde{m} = K_I^+ \oplus A$ for the message $m = K_A^+ \oplus A$, pose as A, and B still accepts $\widetilde{m}$ as being sent by A. Thus, m is vulnerable, *i.e.* $m \in VDS$. On the other hand, if the following first step is used: *Send* $(A, B, K_A^+ \oplus A \oplus C_A)$, the intruder will not be able to substitute any message for $m = K_A^+ \oplus A \oplus C_A$ without being rejected by B. This is since B uses the certificate C_A to check the validity of the key K_A^+ and the identity A. Thus, this message is not vulnerable, *i.e.* $m \notin VDS$. In the first case, $\widetilde{m}$ will be included in the intruder model, while in the second case, no modified message is included.

The Matrix Analysis Step (MAS) involves solving the incidence matrix equation. This requires stating an insecure marking M_n. The result indicates if this insecure state is reachable from the initial state.

To summarize, Basyouni's technique consists of the following steps, as quoted from [Bas97, BT97]:

Step 1: Model the protocol using coloured Petri nets.

Step 2: Apply the ACS to determine the VDS.

Step 3: Stop if the VDS is empty. Otherwise, check if the VDS represents a potential weakness in the protocol. If we find a potential weakness in the protocol, add the model of the intruder to the net.

Step 4: Specify the initial state, the insecure final state, and matrix description for the net.

Step 5: Apply the MAS. The results will tell if this insecure state is reachable from the initial state or not. It will also give the sequence of transitions which will fire to reach this state.

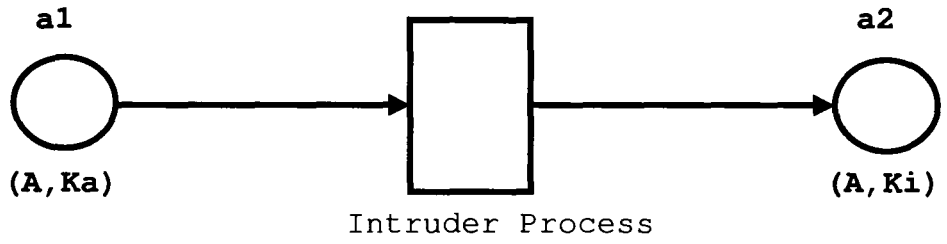
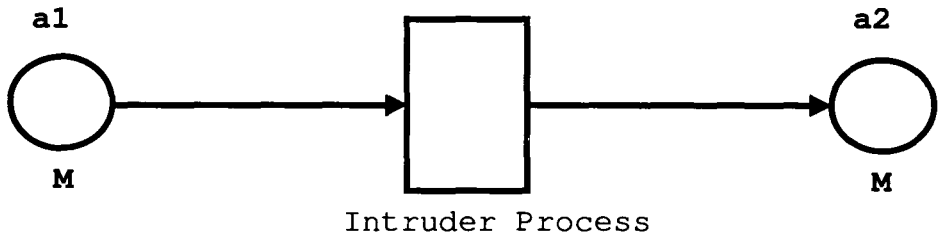


Figure 3.3: Nieh's and Basyouni's usage of place colours



M = set of all records (x,k), where x is an id and k is a key

Figure 3.4: An alternative usage from figure 3.3

3.5.1 Basyouni *vs.* Nieh Models

The models of Nieh and Basyouni are similar in many ways. Both use similar modeling concepts *e.g.* PNO, colouring of places, intruder process modeling, *etc.* Both use the colour of places to represent a message. For instance, in Figure 3.3, if a token is in place $a1$, then the message received by the intruder is (A, K_A). Similarly, if a token is in place $a2$, then the message produced by the intruder is (A, K_I). This strategy is different from using the colour of tokens to represent the message, as sketched in Figure 3.4.

Both models require an explicit model of a specific intruder attack to verify against, rather than discovering unforeseen attacks. For instance, to test the ex-

istence of a man-in-the-middle attack, one should build a model that captures this type of attack.

The following are key differences between Nieh's and Basyouni's models:

1) Nieh's model includes all possible behaviours of the intruder. This explains the large size of Nieh's model and its analysis complexity. On the other hand, Basyouni's model only includes intruder behaviours that may result in a flow. This is done by using the vulnerable data set in the ACS step.

2) Nieh's model is more detailed than Basyouni's. For instance, the model in Figure 3.5 represents the intruder process of modifying the contents of the message $M = A \oplus K_A \oplus C_A$, as outlined in Basyouni's technique. The same process is modeled in Nieh's technique as shown in Figure 3.6.

3) Nieh's technique involves forward execution of the reachability tree; starting from an initial state to check the reachability of an insecure final state. On the other hand, Basyouni improves on this by using backward state execution and solving the incidence matrix equation.

4) Nieh's technique is automated, *i.e.* a computer program can be used to parse a protocol and build a corresponding model to check for specific vulnerabilities. On the other hand, Basyouni's model is semi-automated and requires human input to adaptively build the intruder model, especially in the ACS and MAS steps.

5) A result of (1) and (2) is that Basyouni's method is more efficient in terms of model size and analysis complexity; this is of course at the expense of generality and automation of the model.

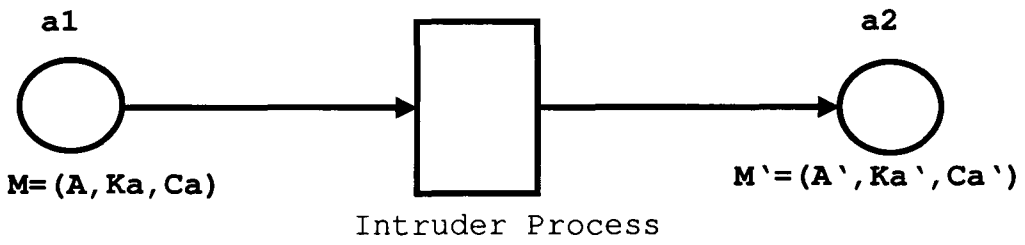


Figure 3.5: An intruder process in Basyouni's model

3.6 Remarks

The techniques presented in this chapter build on Nieh's model [Nie92, NT92]. The technique of Nieh uses a form of coloured Petri nets that is lower level than CP-nets. The use of lower level nets results in an increase in the size of the intruder model, making it harder for users to cope with. Furthermore, as the model becomes larger, the reachability tree also becomes larger causing an increase in analysis complexity. In the next chapter, we focus on using high level constructs of CP-nets that allow us to build smaller and clearer intruder models.

Some of the analysis methods used in these techniques are not applicable to CP-nets. For instance, the state equation method, developed by Basyouni [Bas97], requires determining a matrix representation of the model and solving a matrix equation to check for reachability, where the vector σ (Section 3.5) is a vector of integers $\{0, 1\}$ [Bas97, BT97]. This is different from Jensen's matrix equation where σ is a vector of binding elements [Jen81].

Other high-level forms of Petri nets have been used to verify cryptographic protocols. Aura [Aur95] used Predicate-Transition nets to construct and analyze models for cryptographic protocols. Aura's approach is based on *PROD* (a Predicate-Transition net reachability analysis tool). The stubborn set method is used to reduce the complexity of state reachability analysis.

Another high-level Petri net model was introduced by Lee and Lee [LL97]. This

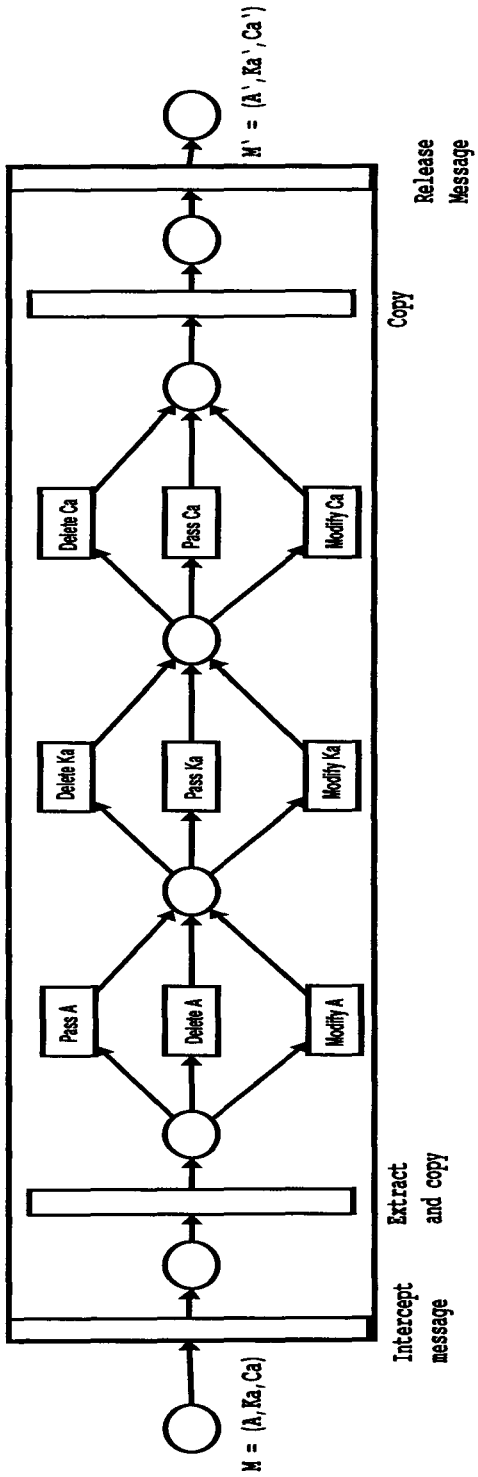


Figure 3.6: The same process of figure 3.5 in Nieh's model

approach is based on a new type of Petri net called *Cryptographic Timed Petri Net (CPTN)*, which is an extended timed Petri net that includes new types of objects such as ciphertext-places and crypto-transitions. This approach includes a technique for specifying cryptographic protocols (*CPTN-language*), and a computer tool to aid in the analysis of CPTN models (*CPTN-analyzer*).

Finally, Crazzolara and Winskel [CW01] used the event-based semantics of Petri nets to prove the properties of cryptographic protocols. Their approach is based on the inductive method [Pau98]. Petri nets were used because of their ability to keep track of concurrent events [CW01]. Since a cryptographic protocol can be considered as a set of events, and as each event specifies a set of pre/post conditions, the trace of events can be viewed as a Petri net. We note that only secrecy properties can be verified using this approach. This work is related in many ways to the strand-space approach [HG99].

Chapter 4

Cryptographic Protocol Verification Using Design/CPN: A New Approach

In this chapter, we demonstrate a new technique for modeling and verifying cryptographic protocols using CP-nets. This 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. 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 simpler and clearer than in the other techniques.

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 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, similar to [BT89, BT97, DTM95, NT92].

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. Declare the colour sets, functions, variables, and constants that will be used in the net inscriptions of the CPN model.

2. Build a model with no intruder.

   a) Build a top-level model.

   b) Define the substitution transitions from the top-level model.

3. Add the intruder to the model.

   a) Add the intruder transition to the top-level model.

   b) Define the intruder substitution transition.

   c) Specify security requirements stated in terms of CPN markings.

4. Implement a token-passing scheme.

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

We demonstrate our technique by using it in modeling and analyzing the TMN protocol, which was described in Section 1.2.2. The TMN protocol is a key exchange protocol. It must guarantee the secrecy of the session key K_{AB}. Furthermore, the protocol must satisfy the authentication requirement that an intruder cannot impersonate A or B. Thus, A and B should never use a session key other than K_{AB}.

```
color T = with A | B | Kjp | Kjpr | Kaj | Kab;
color I = subset T with [A,B];
color K = subset T with [Kaj,Kjp,Kjpr,Kab];
color C = product T*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 4.1: The declarations for the TMN model

4.1 CPN Declarations

The first step in modeling the TMN protocol is to declare the colour sets, functions, variables, and constants that will be used in the net inscriptions of the CPN. The declarations are written in CPN ML. They are provided in Figure 4.1.

In 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 $E(A, K)$ 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, $K = \{K_{AB}, K_{AJ}, K_J^+, K_J^-\}$. Thus, $T = I \cup K$ is the set of atomic fields for the TMN protocol.

   b) All ciphers have the same format: $E(k1, k2)$, where $k1$ and $k2$ are of type K. For instance, $E(K_{AJ}, K_J^+)$ is the cipher of the first message, *etc.* Thus, the cipher colour set C is defined as $C = T \times K$.

   c) Messages are generally composed of an identity and a cipher. For instance, the first message is $ID(B) \oplus E(K_{AJ}, K_J^+)$, the third message is $ID(A) \oplus$

$E(K_{AB}, K_J^+)$, *etc.* 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. Therefore, I is used as the colour set for such messages.

d) The TMN protocol implicitly assumes that J is able to know 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 $ID(B) \oplus E(K_{AJ}, K_J^+)$, 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, transition T in Figure 4.2 will fire endlessly, generating an infinite number of ciphers (Kaj, Kjp). By using a construct such as in Figure 4.3, we force the transition T to fire only once.

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 *i.e.* it returns $AssociatedKey(k)$.

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.

In summary, the following needs to be declared in modeling cryptographic protocols:

1. Atomic fields which include identities, keys, and nonces.

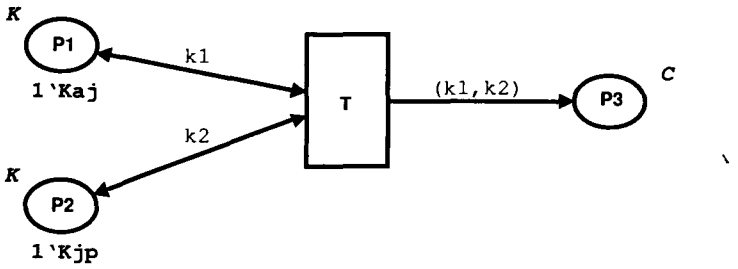


Figure 4.2: Transition T will fire infinitely.

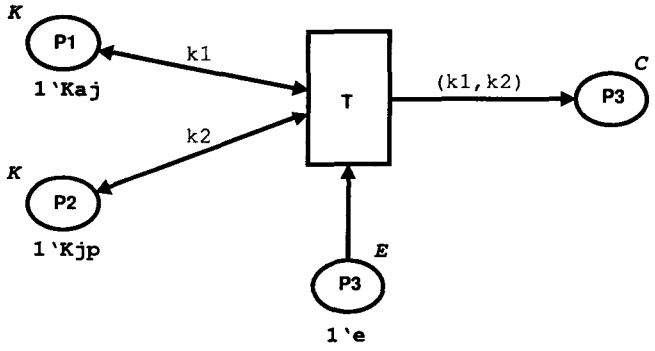


Figure 4.3: By using the E set, transition T can fire at most one time.

2. Ciphers: these are defined using the atomic fields.

3. Messages: these are defined using the atomic fields and ciphers.

4. Variables to be used as arc inscriptions.

5. A special colour set E to control the firing of transitions.

6. Functions such as decryption functions, shared keys, *etc.*

The techniques presented in [BT97, NT92] do not directly include declarations in modeling a cryptographic protocol. We believe that such declarations add clarity to the models. Furthermore, these declarations allow us to specify complex properties of encryption and decryption functions, *e.g.* algebraic properties. Also, as will be seen later, the intruder model can be systematically constructed based on the colour of tokens it intercepts. Thus, having declarations will benefit us in constructing the intruder's model. Finally, we note that such declarations can be automatically generated by parsing a high-level description of the cryptographic protocol.

4.2 The Model with No Intruder

We first present a model of the TMN protocol with no intruder.

4.2.1 The Top-Level Model

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. We note that the idea of using a substitution transition to model an entity is similar to the concept of a PNO introduced in other techniques (See Section 3.2).

First, we focus on the messages exchanged between the protocol entities. At this level, protocol entities are modeled as transitions. Figure 4.4 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.

4.2.2 Defining the Top-Level Substitution Transitions

In order to define the substitution transitions, one must clearly understand the behaviour of the protocol entities. The following is an informal description for their behaviour:

1. Entity A is the initiator of the communication session. Thus, A always sends the message $ID(B) \oplus E(K_{AJ}, K_J^+)$. The reply A receives is in the format (i, c), where i is an identity and c is a cipher. 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.

2. Entity B is the responder. The first message it receives is of type I. It indicates the entity that wants to initiate a communication session with B. Based on

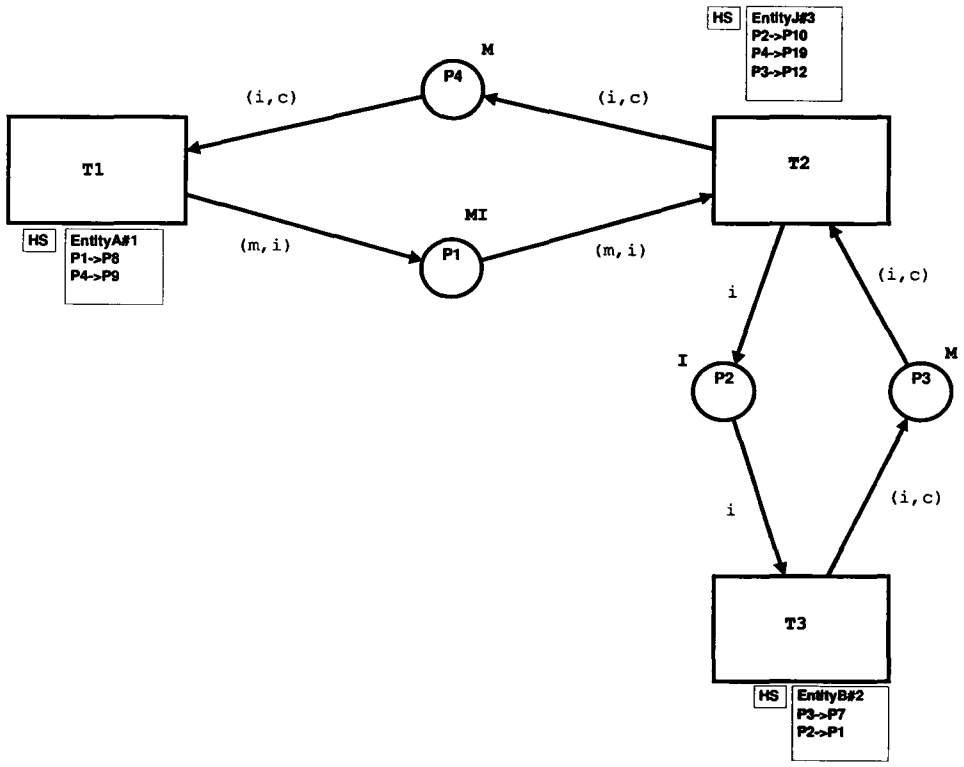


Figure 4.4: The TMN top-level model with no intruder

this, B generates a message of type M, containing the initiator's identity and a cipher containing a session key. Thus, if B receives the message $ID(X)$, it sends J the message $ID(X) \oplus E(K_{XB}, K_J^+)$ where K_{XB} is a session key to be used to encrypt messages between entity X and B.

3. Entity J is the server. The first message it receives is of type MI. Assume J has received the token (m, i); i indicates the sender (the initiàtor) and m is the message. The colour m contains two fields: the first is the identity of the responder, while the second is a cipher containing an auxiliary key encrypted using K_J^+. At this point, J uses its private key to decrypt the cipher and obtain the auxiliary key. Then, J sends a message of type I to the responder indicating the identity of the initiator.

   Afterwords, the responder sends J a message of type M, containing two fields: the first is the initiator's identity and the second is the session key encrypted under K_J^+. J checks that the first field is the initiator's identity, then it uses its private key to decrypt the second field in order to obtain the session key. Finally, J encrypts this key with the auxiliary key it has received from the first message, and sends it along with the last message containing the identity of the responder.

Figure 4.5 shows the CPN model of entity A. Figure 4.6 shows the CPN model of entity B. Figure 4.7 shows the CPN model of entity J. Figure 4.8 shows the hierarchy page.

Note that one of the benefits of building a CPN model is establishing formalism. The CPN model establishes a precise specification of the behaviour of the protocol entities. This removes ambiguities faced when using informal descriptions of the protocol.

As the models of Figures 4.5, 4.6, and 4.7 show, the CPN model of an entity is composed from multiple subnets. Each subnet has its own input and output ports.

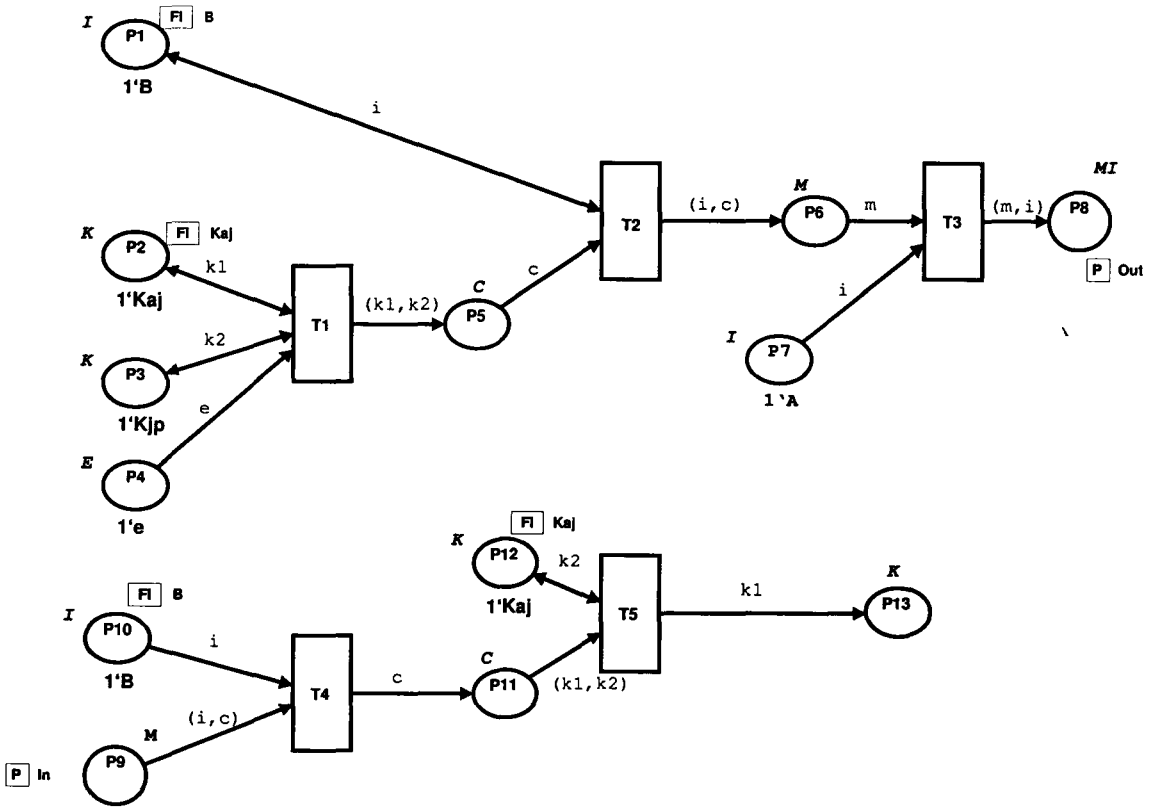


Figure 4.5: Page *EntityA*

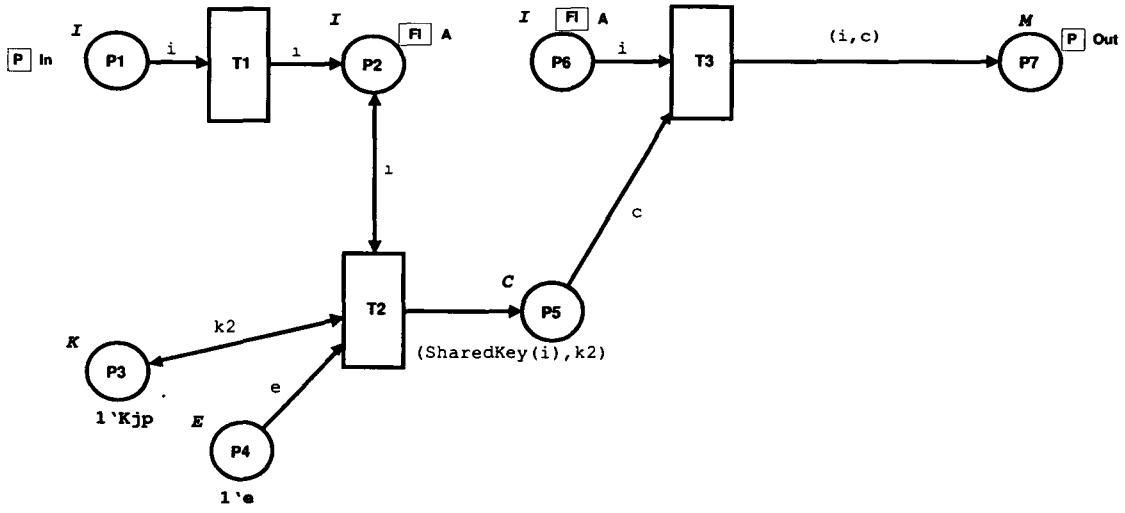


Figure 4.6: Page *EntityB*

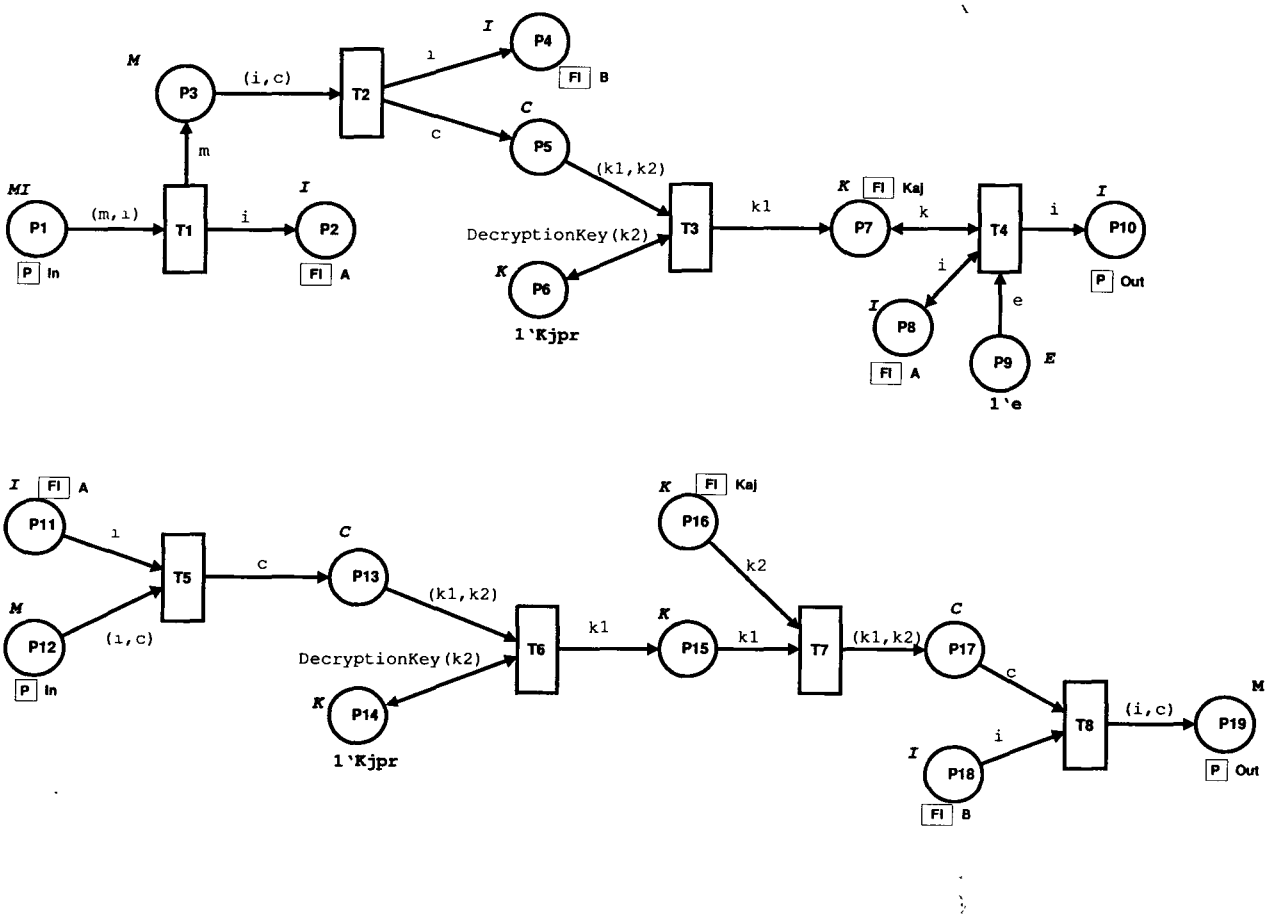


Figure 4.7: Page *EntityJ*

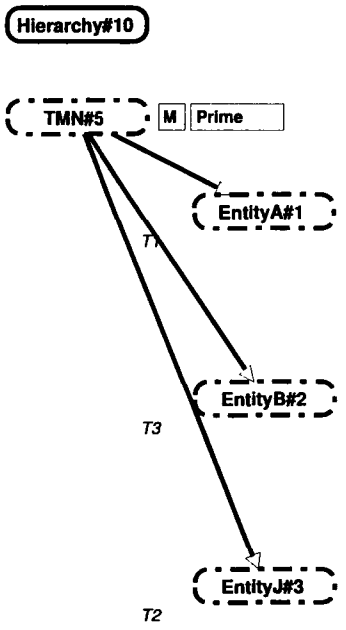


Figure 4.8: The hierarchy page

Also, each subnet models a subtask that is performed by the corresponding entity. For instance, the model of entity A, Figure 4.5, contains two subnets: one subnet models the subtask of A initiating a protocol run in step 1, while the second subnet models the subtask of A receiving the last message from J.

The model for entity B (Figure 4.6) contains one subnet. The entity J model, Figure 4.7, is composed of two subnets: one models the subtask of J receiving an initiation request and sending a message to the responder. This corresponds to what J does in steps 1 and 2 of the protocol. The second models the subtask of J receiving the session key from the responder and sending it in an encrypted form to the initiator. This corresponds to what J does in steps 3 and 4 of the protocol.

Port assignments are used to relate the top-level page, named *TMN*, with the entities models. This information is summarized in Table 4.1.

Table 4.2 lists the instance fusion sets that are used in the model. Places that belong to the same fusion set are conceptually the same place. Fusion sets are used

Table 4.1: Port assignments for page *TMN*

Socket	Subpage	Port Name	Port Type
P1	EntityA	P8	Output
	EntityJ	P1	Input
P2	EntityJ	P10	Output
	EntityB	P1	Input
P3	EntityB	P7	Output
	EntityJ	P12	Input
P4	EntityJ	P19	Output
	EntityA	P9	Input

to allow an entity to control the order of subtasks and check the validity of messages. For instance, J has to remember the identity of the responder that it receives in the first step, in order to send it back to the initiator in the last step.

Note that we have used the E colour set to prevent, when possible, transitions from firing infinitely often. This is done whenever a transition has double input arcs. For instance, transition $T1$ from *EntityA* requires the use of the E set construct. Note that double arcs are used to store tokens inherent to an entity, or tokens that are to be used in subsequent subtasks performed by the entity.

4.2.3 Summary

In summary, to build a model with no intruder, we follow these steps:

1. Construct a top level model of the protocol. Here, transitions are used to model entities. Message exchange is modeled as token movements via the places that connect the transitions.

2. Informally describe the individual behaviour of protocol entities.

Table 4.2: Instance fusion sets

Page	Instance Fusion Set	Members of the Fusion Set
EntityA	B	P1, P10
	Kaj	P2, P12
EntityB	A	P2, P6
EntityJ	A	P2, P8, P11
	Kaj	P7, P16
	B	P4, P18

3. Build a CPN model for the entities.

4. Define the substitution transitions of the top level page. Use port and socket assignments to relate this page and the subpages modeling protocol entities.

5. Define the instance fusion sets.

6. Use the E colour set to prevent the infinite firing of transitions with double input arcs.

4.2.4 Simulation and Debugging

We use the CPN Simulator to simulate the current model. This allows us to informally check its behaviour.

Since no intruder is present, we can simulate a complete run initiated by A. During simulation, one can view the contents of passing tokens, as if actual messages are being exchanged.

We use the *Interactive Simulation Options* menu to set the breakpoints to stop between steps. We then use the *Interactive Run* command from the *Sim* menu to have an interactive simulation.

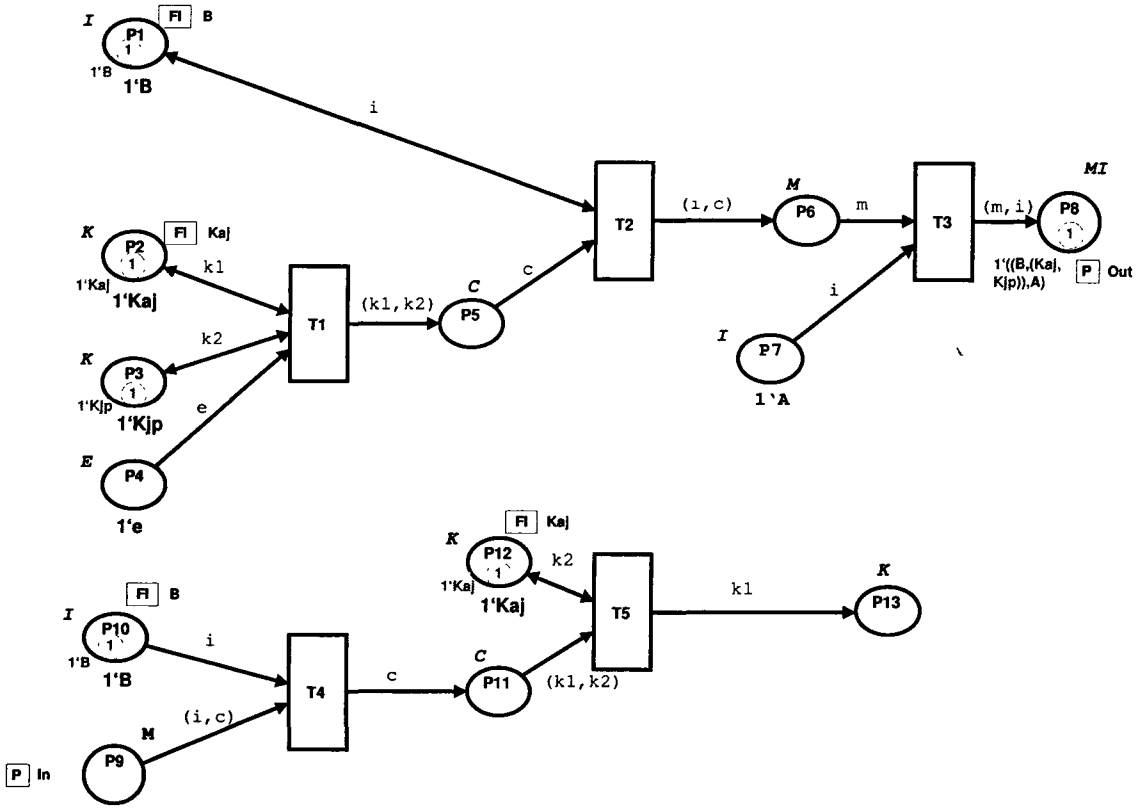


Figure 4.9: The marking of *EntityA* after firing $T3$

Figure 4.9 shows the marking of *EntityA* after firing $T3$. Place $P8$ contains one token with colour $((B,(Kaj,Kjp)),A)$. This corresponds to the message $ID(B) \oplus E(K_{AJ}, K_J^+)$ that A sends in the first step. Since $P8$ is a port, its marking is the same as its corresponding sockets, which can be checked by viewing the markings of place $P1$ in *TMN* and place $P1$ in *EntityJ*.

Continuing the simulation, Figure 4.10 shows the marking of Entity J after firing $T4$. As this shows, place $P10$ contains one token of colour A corresponding to the message that J sends to B in the second step of the protocol. The marking of $P10$ is the same as that of its socket places, $P2$ of *TMN* and $P1$ of *EntityB*. Also, places in a fusion set have the same marking; for instance, places in fusion set A, which include $P2$, $P8$, and $P11$, all have the same marking $1'A$. A similar observation can be made

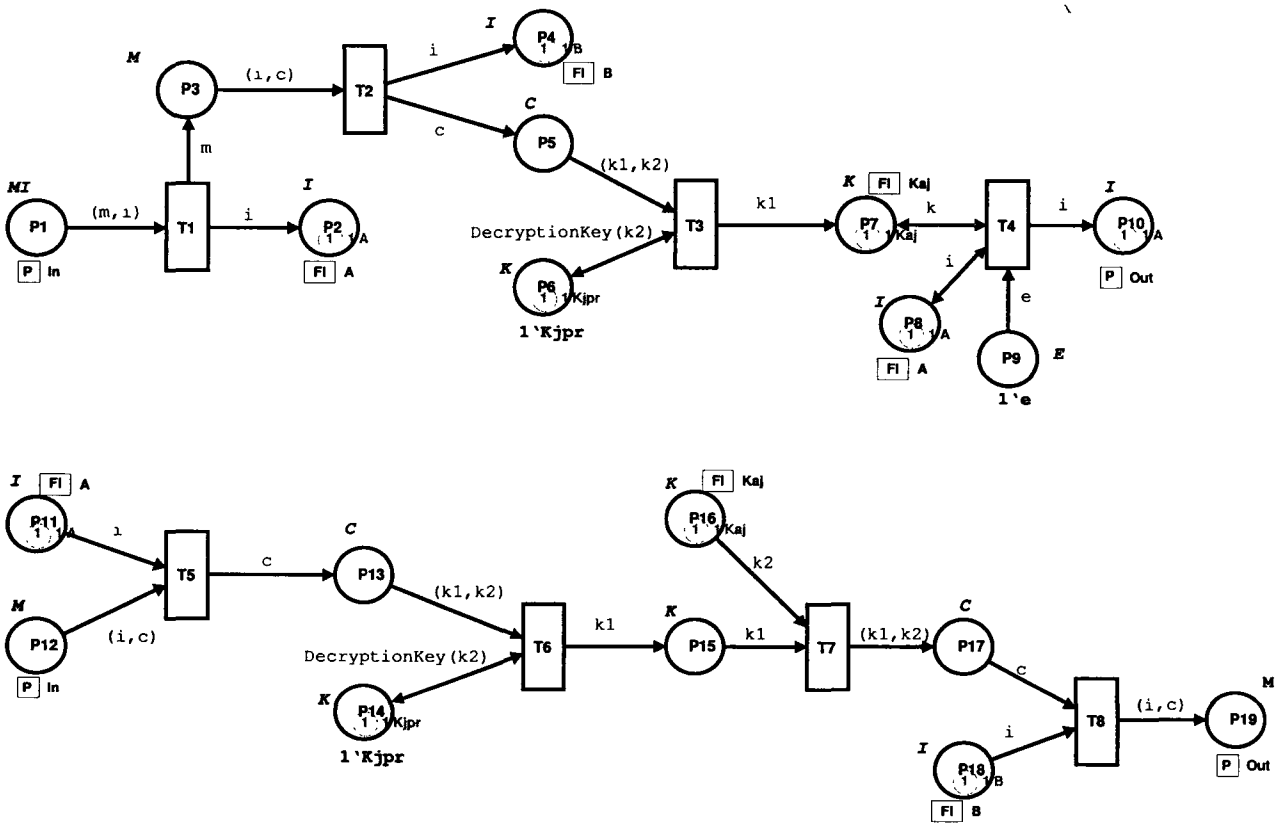


Figure 4.10: The marking of *EntityJ* after firing T4

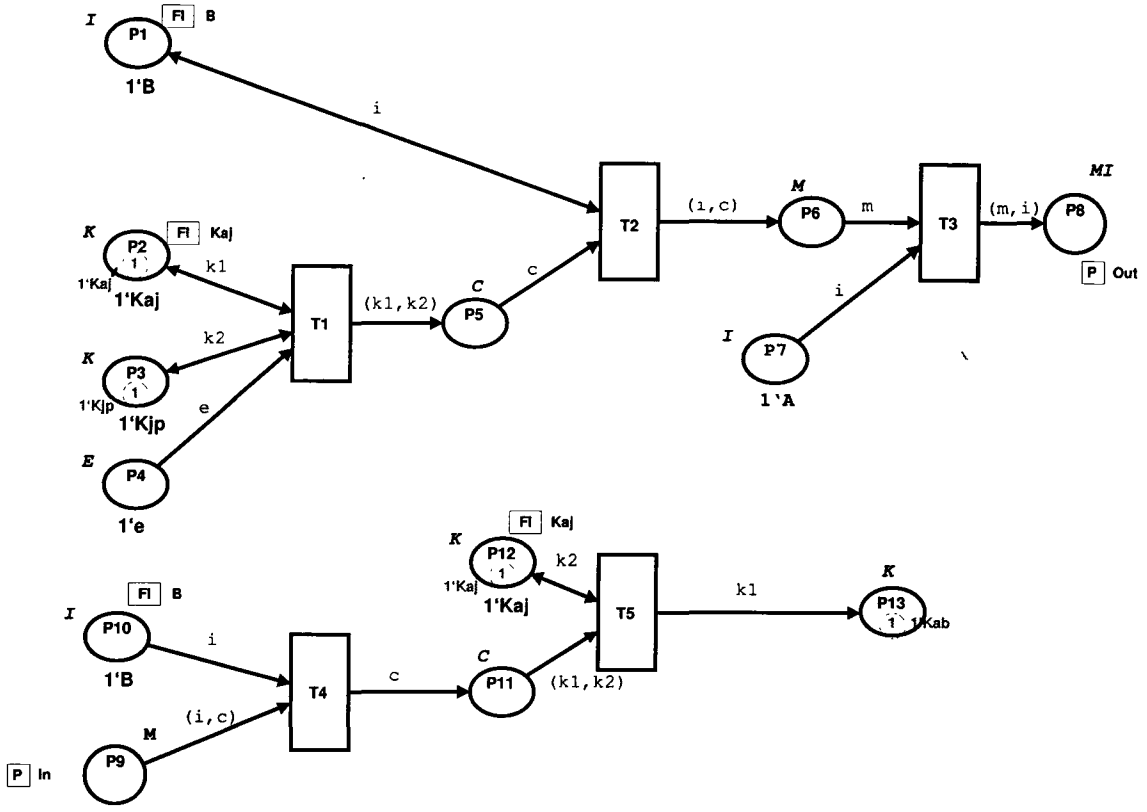


Figure 4.11: The final marking of *EntityA*

about the places of fusion set B.

Figures 4.4, 4.11, 4.12, and 4.13 show the final markings of *TMN*, *EntityA*, *EntityB*, and *EntityJ*, respectively. At this state, place $P13$ in *EntityA* contains one token of colour Kab. This represents the acceptance of the session key K_{AB} by A.

4.2.5 Generating the Occurrence Graph

Finally, we use the Design/CPN Occurrence Graph tool to construct the full occurrence graph of the CPN model, shown in Figure 4.14. Since the model represents the behaviour of protocol entities without the presence of the intruder, there should be

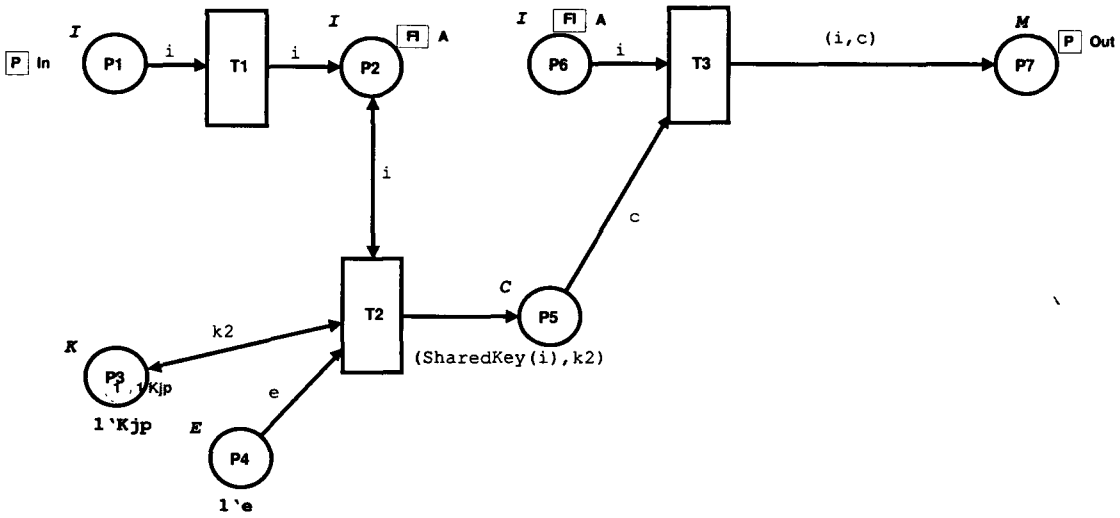


Figure 4.12: The final marking of *EntityB*

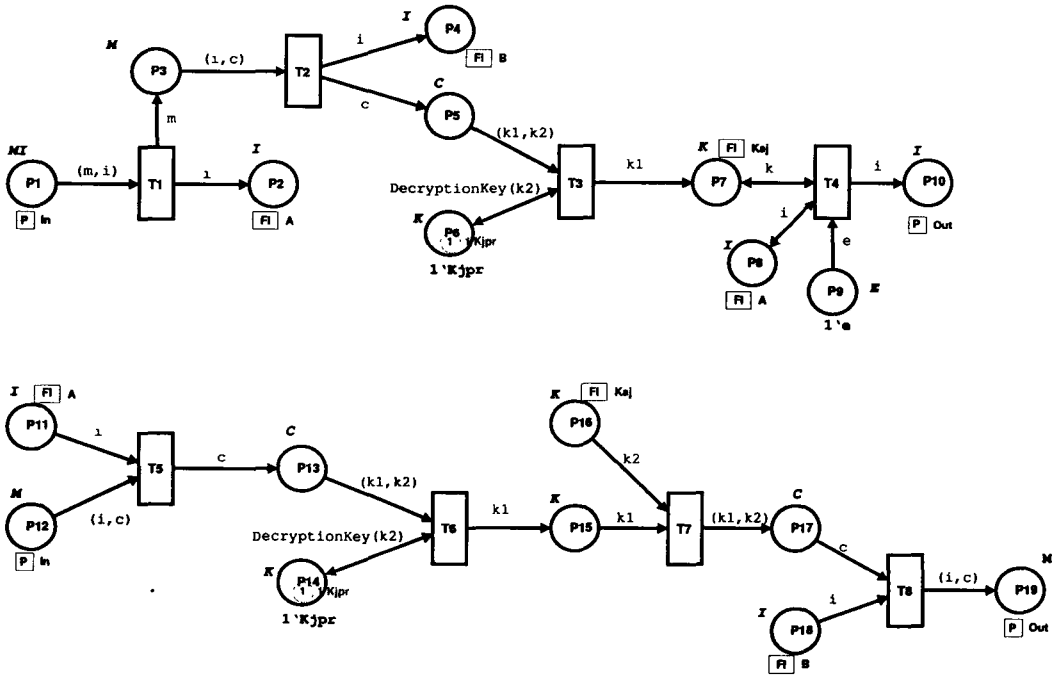


Figure 4.13: The final marking of *EntityJ*

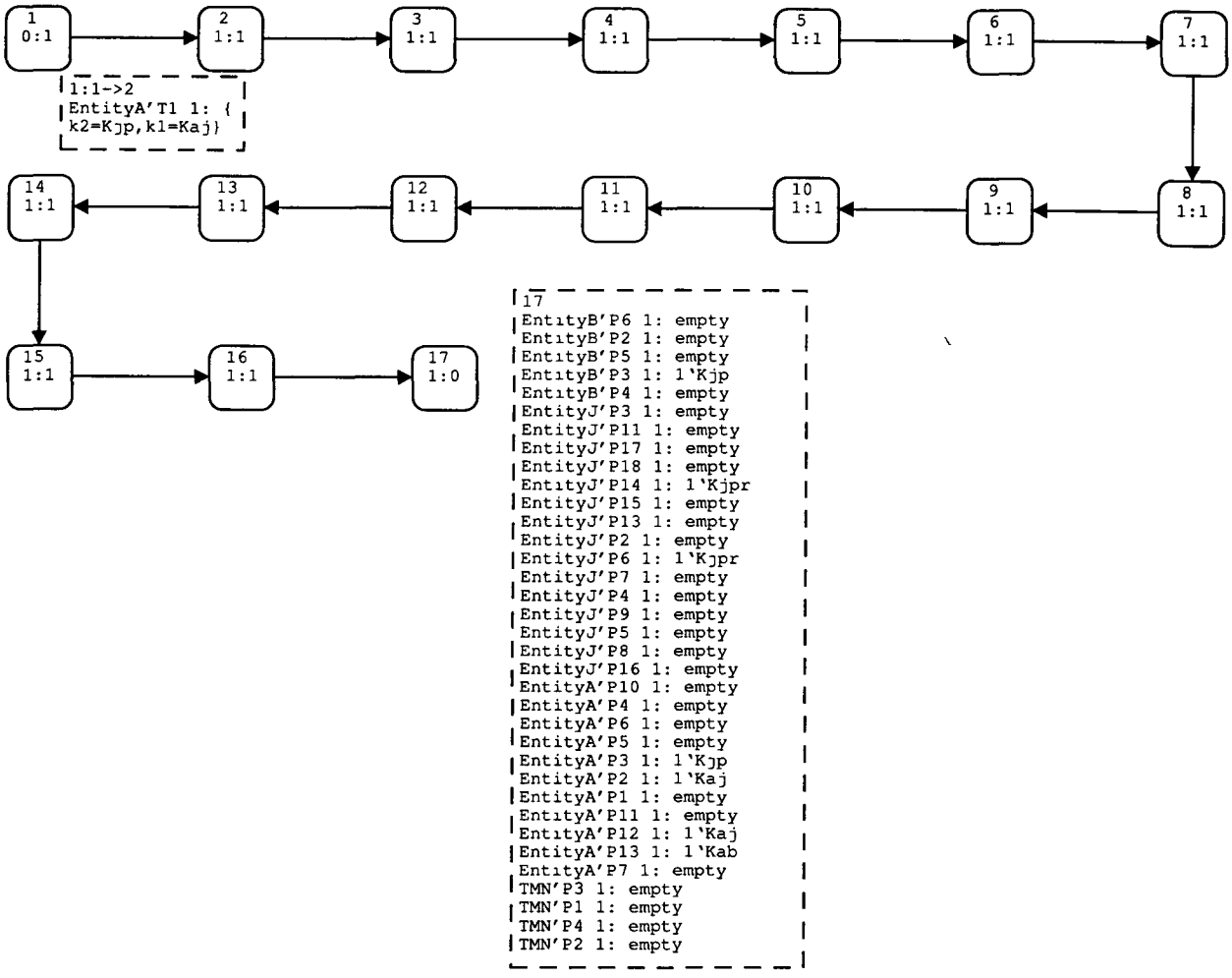


Figure 4.14: The occurrence graph of the TMN model with no intruder

only one path from the initial state to the final state.

The occurrence graph contains 17 nodes (markings). By double-clicking an arc, one can view the corresponding step. For instance, the arc from node 1 to node 2 has the inscription *"EntityA'T1 1:k2=Kjp, k1=Kaj"*. Thus, the CPN changes from the first marking to the second by firing $T1$ of $EntityA$ with the binding $k1 = Kaj$ and $k2 = Kjp$.

By double-clicking a node from the occurrence graph, one can view the corresponding marking. For instance, Figure 4.14 shows the marking of node 17, which is

the final reachable state. Alternatively, the *Occ State to Sim* command from the *Sim* menu can be used to select a node, and move its state to the simulator.

4.3 The Model with an Intruder

The next step in modeling a cryptographic protocol is to add the 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.

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 also 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.

4.3.1 The Top Level Model with an Intruder

Figure 4.15 shows the top level model of the TMN protocol with an intruder. $T4$ represents the intruder, which was not included in the earlier top level model in Figure 4.4.

Each place in Figure 4.4 is replaced with two corresponding places as shown in Figure 4.15: one is an input place to the intruder while the second is an output place.

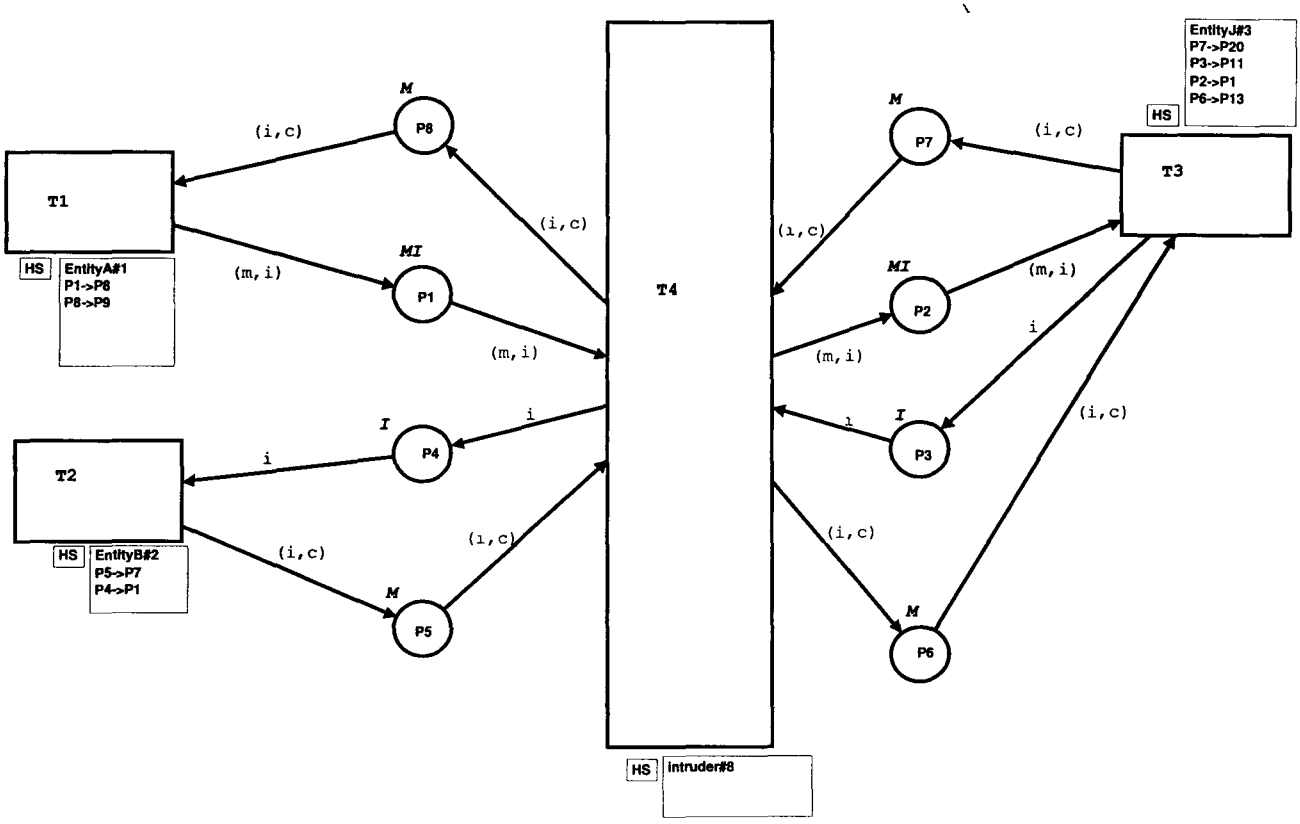


Figure 4.15: The TMN top-level model with an intruder

For instance, place $P1$ in Figure 4.4 is replaced with $P1$ and $P2$ in Figure 4.15. 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).

4.3.2 Extending the CPN/ML Definitions

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 use DB as the colour set of a DB-place.

In CPN ML, DB is declared as follows:

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

Here, *cI*, *cK*, and *cC* are selectors [Met93a]. 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 4.16 contains the final CPN ML declarations.

4.3.3 Defining the Intruder

The substitution transition $T4$ in the top-level model, Figure 4.15, is defined by the subpage *intruder* shown in Figure 4.17. Note that the places in Figure 4.17 are ports

```
color T = with A | B | In | Kjp | Kjpr | Kaj |
Kab | Ki;
color I = subset T with [A,B,In];
color K = subset T with [Kaj,Kjp,Kjpr,Kab,Ki];
color C = product T*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 | Kjpr => Kjp | Kab=>Kab | Ki=>Ki;
fun SharedKey(i:I):K=case i of A=> Kab | In =>
Ki | B=>Kab;
```

Figure 4.16: Declarations used in the TMN model with an intruder

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

Table 4.3: The intruder subprocesses

related to the sockets of the top level page. Also, note that for each input/output pair of intruder places, there is a corresponding substitution transition modeling what an intruder can do to the intercepted tokens. Table 4.3 lists these pairs along with their corresponding subprocesses.

The intruder subprocesses *intruder_mi*, *intruder_m*, and *intruder_i* are defined in separate pages, see Figures 4.18, 4.19, and 4.20. Instances of such pages are used to define the substitution transitions in *intruder*. Thus, the intruder page has one instance of *intruder_mi* (which defines $T1$), one instance of *intruder_i* (which

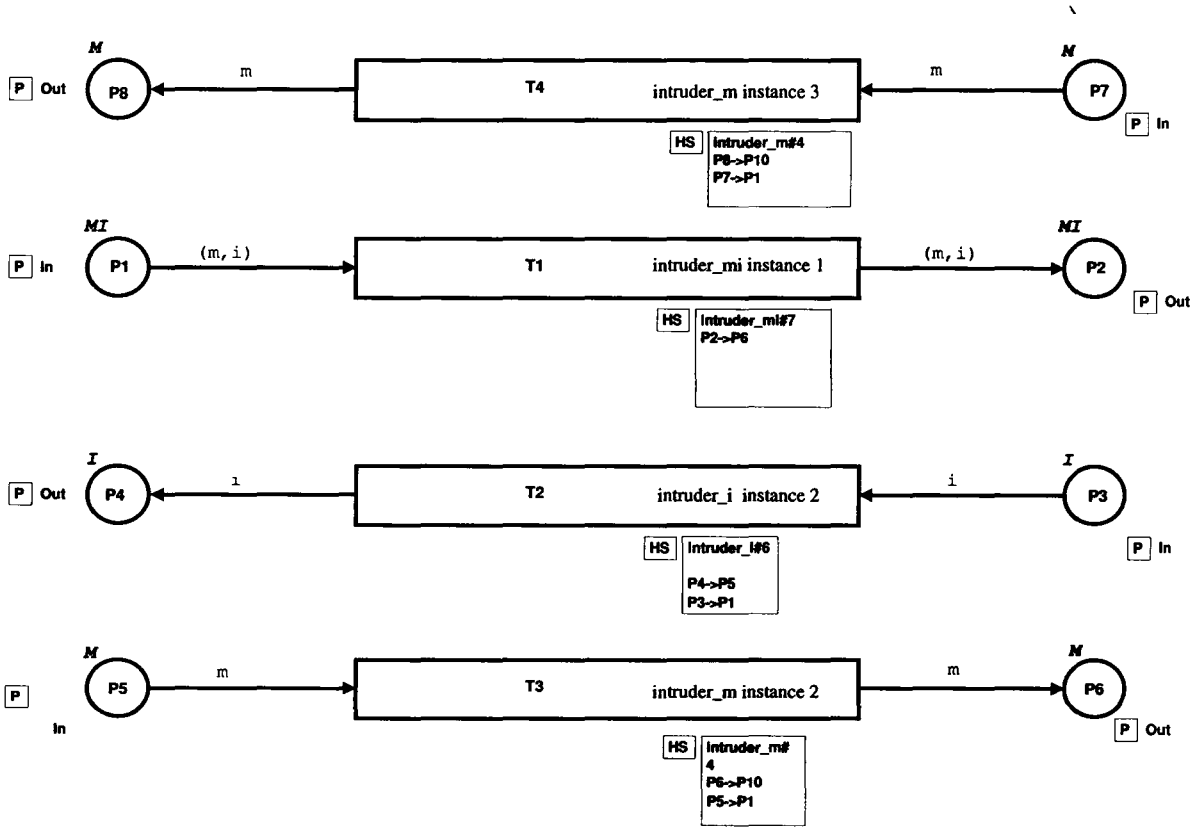


Figure 4.17: The *intruder* page

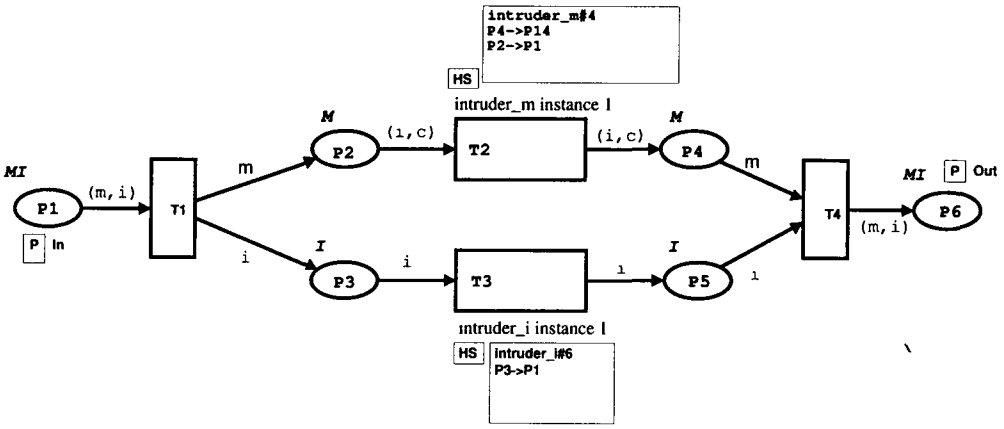


Figure 4.18: Page *intruder_mi*

defines $T2$), and two instances of *intruder_m* (which define $T3$ and $T4$).

The *intruder_m* subprocess is defined in Figure 4.19. 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. The intruder can also construct new ciphers by using keys stored in the database. Finally, the intruder uses one of the stored ciphers, and forms a new message to be sent in place of the intercepted one.

The *intruder_i* subprocess is defined by the *intruder_i* subpage. It models what an intruder can do to intercepted tokens of type I. As Figure 4.20 shows, the intruder first stores the intercepted identity. Then, it uses one of the identities stored in its database to replace the intercepted one.

Figure 4.18 shows the *intruder_mi* page. It 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. The intruder handles both separately; 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.

Several instances of *intruder_i* and *intruder_m* are used in constructing *intruder*

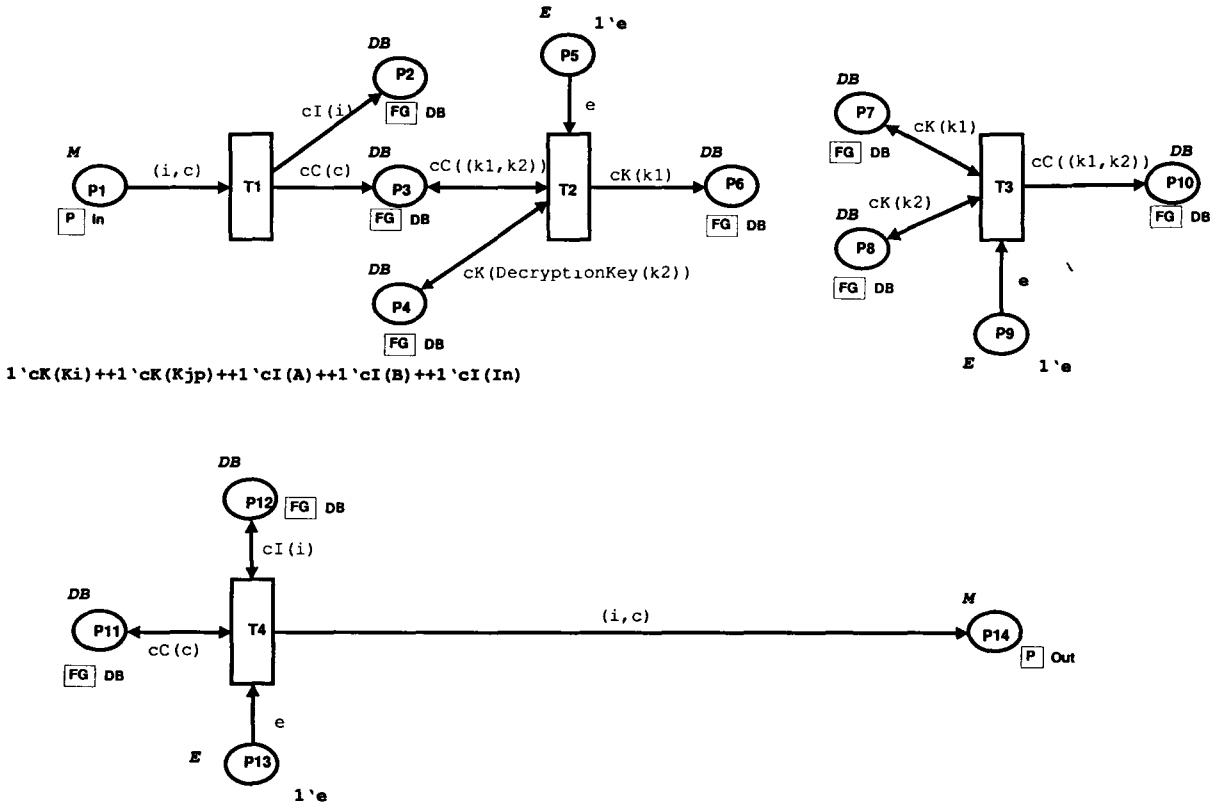


Figure 4.19: Page *intruder_m*

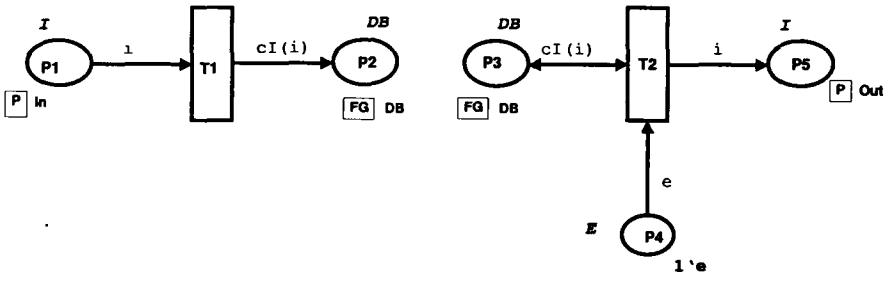


Figure 4.20: Page *intruder_i*

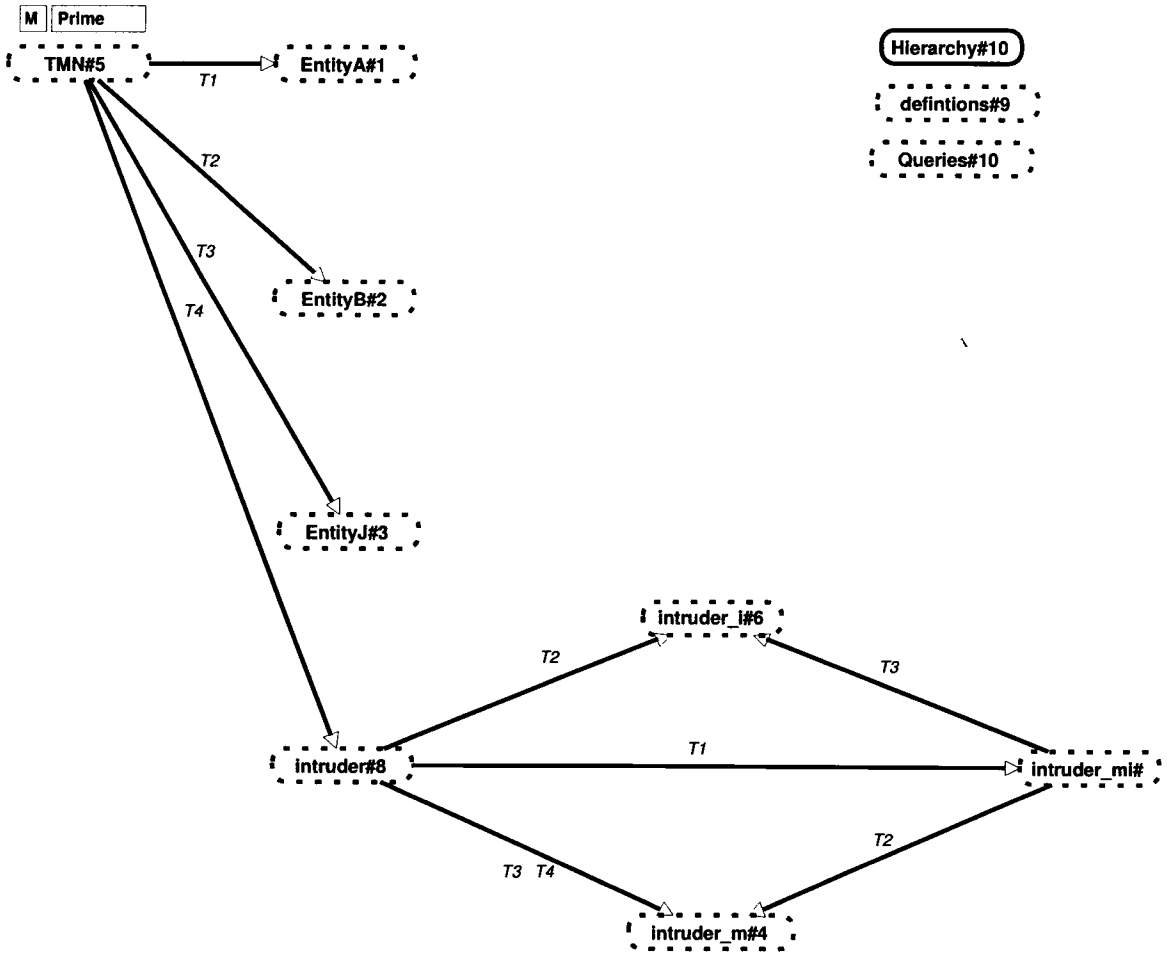


Figure 4.21: The hierarchy page of the TMN model with an intruder

and *intruder_mi*. To identify the instances, the corresponding substitution transition is labeled with the instance number. For instance, $T2$ in Figure 4.17 is related to the second instance of *intruder_i*, while $T3$ in Figure 4.18 is related to the first instance of *intruder_i*. Figure 4.21 shows the hierarchy page.

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 4.19), the DB is set initially to $\{K_I, K_J^+, A, B, In\}$.

4.3.4 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 note that the violation of a security requirement can be stated as the reachability of a given token to a particular place. For instance, the secrecy property of a protocol is violated if a token holding secret information reaches a place that it should not reach. Similarly, the authentication property is violated if a token holding the identity of an unauthorized entity is accepted by a given entity.

The following are the requirements that must be verified for the TMN protocol:

1. The protocol must guarantee the secrecy of the session key K_{AB}. Thus, K_{AB} must be known only by A,B, and J, in a given session. 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.

2. The protocol must guarantee the mutual authentication of A and B. Thus, entity A never accepts a session key other than K_{AB}. In terms of markings, place $P13$ in $EntityA$ must never have a token with colours other than K_{AB}.

3. The protocol must guarantee the secrecy of the auxiliary key K_{AJ}. Thus, K_{AJ} must be known only by A and J. Equivalently, the intruder must never know K_{AJ}. In terms of CPN markings, this translates into the requirement that a token with colour Kaj never reaches a DB-place.

4.3.5 Simulation and Debugging

At this stage, the model is ready for simulation and occurrence graph analysis. We present, using the CPN simulator, occurrence sequences that result in a violation of the TMN protocol requirements. This is not typically done when verifying new

protocols with unknown attacks, however, we choose to do so to emphasize that: (1) our model actually detects known attacks on the TMN protocol, and (2) simulation using Design/CPN allows for human input in pruning the state space.

The following shows an occurrence sequence that results in the violation of one of the requirements.

`intruder_m`	1	T3	$k1 = Ki, k2 = Kjp$
`intruder_m`	1	T4	$c = (Ki, Kjp), i = B$
`intruder_i`	1	T1	$i = A$
`intruder_i`	1	T2	$i = A$
`intruder_mi`		T4	$i = A, m = (B, (Ki, Kjp))$
`EntityJ`		T1	$m = (B, (Ki, Kjp)), i = A$
`EntityJ`		T2	$i = B, c = (Ki, Kjp)$
`EntityJ`		T3	$k1 = Ki, k2 = Kjp$
`EntityJ`		T4	$i = A, k = Ki$
`intruder_i`	2	T1	$i = A$
`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	$c = (Kab, Kjp), i = A$
`intruder_m`	2	T4	$i = A, c = (Kab, Kjp)$
`EntityJ`		T5	$c = (Kab, Kjp), i = A$
`EntityJ`		T6	$k1 = Kab, k2 = Kjp$
`EntityJ`		T7	$k1 = Kab, k2 = Ki$
`EntityJ`		T8	$i = B, c = (Kab, Ki)$
`intruder_m`	3	T1	$c = (Kab, Ki), i = B$
`intruder_m`	3	T2	$k1 = Kab, k2 = Ki$

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 following line represents the step ($T4$ in the second instance of $intruder\_m$, $\langle i = A,\ c = (Kab, Kjp)\rangle$):

`intruder_m`	2	`T4`	$i = A, c = (Kab, Kjp)$

This occurrence sequence models the following known attack on the TMN protocol.

1. $Send(I, J, ID(B) \oplus E(K_I, K_J^+)), where\ PosesAs(I, A)$
2. $Send(J, B, ID(A))$
3. $Send(B, J, ID(A) \oplus E(K_{AB}, K_J^+))$
4. $Send(J, A, ID(B) \oplus E(K_{AB}, K_I)), where\ PosesAs(I, A)$

This attack allows the intruder to impersonate A. This represents a violation of the first requirement. Note the reachability of a token Kab to a DB-place in the last step of the occurrence sequence.

The following shows an occurrence sequence that results in the violation of a different requirement.

`EntityA`		`T1`	$k1 = Kaj, k2 = Kjp$
`EntityA`		`T2`	$c = (Kaj, Kjp), i = B$
`EntityA`		`T3`	$i = A, m = (B, (Kaj, Kjp))$
`intruder_mi`		`T1`	$i = A, m = (B, (Kaj, Kjp))$
`intruder_m`	1	`T1`	$c = (Kaj, Kjp), i = B$
`intruder_m`	1	`T4`	$i = B, c = (Kaj, Kjp)$
`intruder_i`	1	`T1`	$i = A$
`intruder_i`	1	`T2`	$i = A$
`intruder_mi`		`T4`	$i = A, m = (B, (Kaj, Kjp))$
`EntityJ`		`T1`	$i = A, m = (B, (Kaj, Kjp))$
`EntityJ`		`T2`	$i = B, c = (Kaj, Kjp)$

EntityJ		T3	$k1 = Kaj, k2 = Kjp$
EntityJ		T4	$i = A, k = Kaj$
intruder_m	2	T4	$i = A, c = (Ki, Kjp)$
EntityJ		T5	$i = A, c = (Ki, Kjp)$
EntityJ		T6	$k1 = Ki, k2 = Kjp$
EntityJ		T7	$k1 = Ki, k2 = Kaj$
EntityJ		T8	$i = B, c = (Ki, Kaj)$
intruder_m	3	T1	$i = B, c = (Ki, Kaj)$
intruder_m	3	T4	$i = B, c = (Ki, Kaj)$
EntityA		T4	$i = B, c = (Ki, Kaj)$
EntityA		T5	$k1 = Ki, k2 = Kaj$

This occurrence sequence models the following known attack on the TMN protocol.

1. $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+))$
2. $Send(J, B, ID(A)), where\ PosesAs(I, B)$
3. $Send(I, J, ID(A) \oplus E(K_I, K_J^+)), where\ PosesAs(I, B)$
4. $Send(J, A, ID(B) \oplus E(K_I, K_{AJ}))$

This attack allows the intruder to impersonate B. It represents a violation of the second requirement. Note the reachability of a token Ki to $P13$ in $EntityA$ in the last step of the occurrence sequence. This represents the acceptance of K_I, by A, as a session key.

4.3.6 Analyzing the Occurrence Graph

The final step in the analysis of the model is to construct and analyze the occurrence graph. The goal is to find nodes (markings) that violate a security requirement.

We use the *Occ Menu* to invoke commands related to the occurrence graphs. 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 first security requirement of the TMN protocol 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. 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 first security requirement.

The other security requirements can be stated in a similar way. We discuss the second requirement; it states that no other key other than Kab arrives to place $P13$ in $EntityA$. In CPN ML, we can use the following predicate:

```
fn n => cf(Ki, Mark.EntityA'P13 1 n) >0
```

Given a marking n, this predicate evaluates to true if place $P13$ of $EntityA$ contains at least one token Ki, and false otherwise.

The following function returns all nodes of the occurrence graph where place $P13$ of $EntityA$ has at least one token k. It uses the predicate defined above.

```
fun SecrecyViolation2(k:K):Node list
PredAllNodes (fn n => cf(k, Mark.EntityA 1 n) >0);
```

Thus, $SecrecyViolation2(Ki)$ returns all nodes of the occurrence graph that violate the second security requirement.

Using the model developed so far, without modification, the size of the resulting occurrence graph is huge. Using a 1-GHz, 16-GB machine, we ran the OG tool for 162 hours. This generated a partial occurrence graph with 434,351 nodes and 1,537,319 arcs. Also, no node violating a security requirement was found in this partial occurrence graph.

Using our technique as outlined up to this point, most models of cryptographic protocols result in a large occurrence graph. This problem exists in all state-based methods. It also exists in other techniques of cryptographic protocol verification using CP-nets. The large size of the occurrence graph can be explained by two aspects of the model: the nondeterministic behaviour of the intruder, and the interleaving of the 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, in the TMN model, that the intruder has the keys K_I and K_J^+, 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 $T1$ of $EntityA$ has not fired yet. 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.*

In the next section, we present a solution to prevent the unnecessary interleaving.

4.4 Approaches for Reducing the Occurrence Graph

4.4.1 Applying a Token-Passing Scheme

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 obtain by the interleaving of such executions. In other words, true concurrency is assumed not to affect properties of cryptographic protocols. This assumption is made by other state-based methods too.

The following sentence provides a formal explanation for the claim that using a token passing scheme does not restrict the model assumptions. Let E be the set of all possible executions of entities A, B, and J, where an execution is defined to be a sequence of actions (transition firings). The intruder observes the set of actions in an execution $e \in E$, denoted by S_e. If, for instance, there is an execution represented by the following actions, $a.b.c$, the intruder sees $\{a, b, c\}$. Let $R = \{(e_1, e_2) \mid e_1 \in E \wedge e_2 \in E \wedge S_{e_1} = S_{e_2}\}$. It is clear that R is an equivalence class. Hence, there is no need to take into considerations all the executions. We need only to consider a representative from each equivalence class associated to R. An execution e that represents an interleaving execution of entities belongs to the same equivalence class as the sequence that represents the sequential execution of these entities.

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.

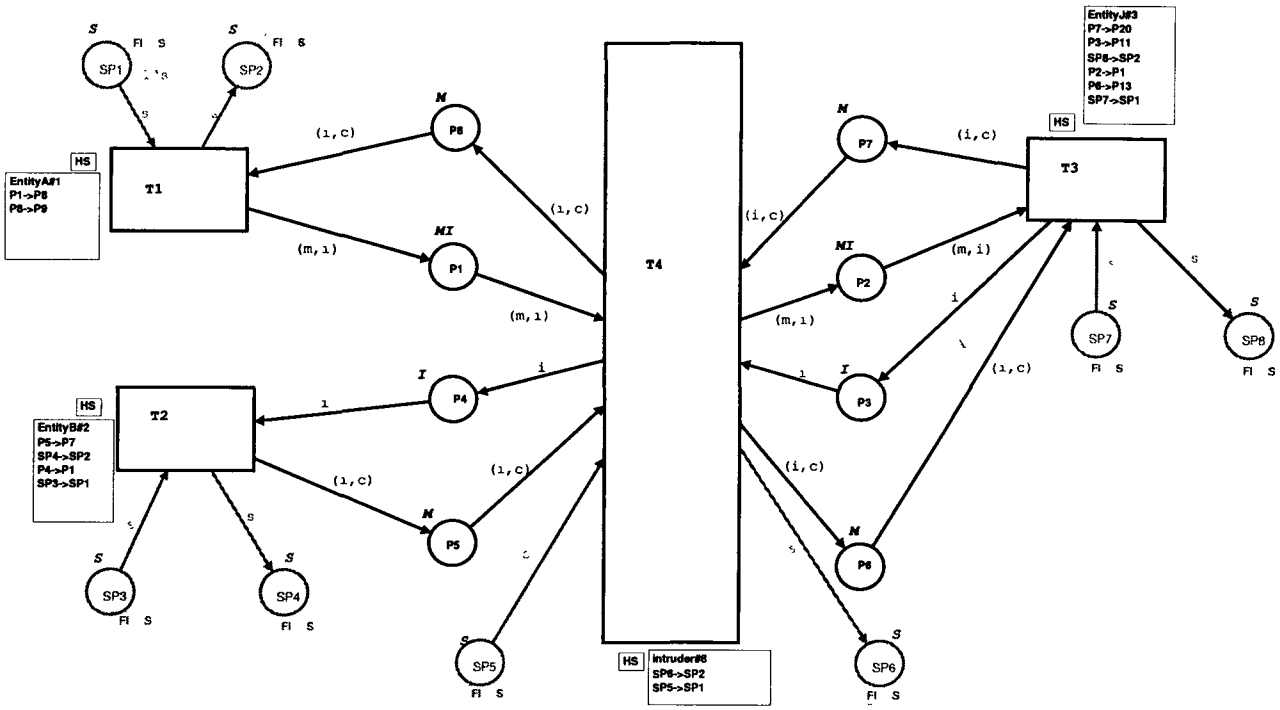


Figure 4.22: The *TMN* page after adding the S-places

The following rules are the changes required to apply this scheme. Although we demonstrate them on the TMN model, these are applicable in modeling any cryptographic protocol.

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 initialized with one s-token. This rule is demonstrated in Figure 4.22.

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. This rule is applied to the

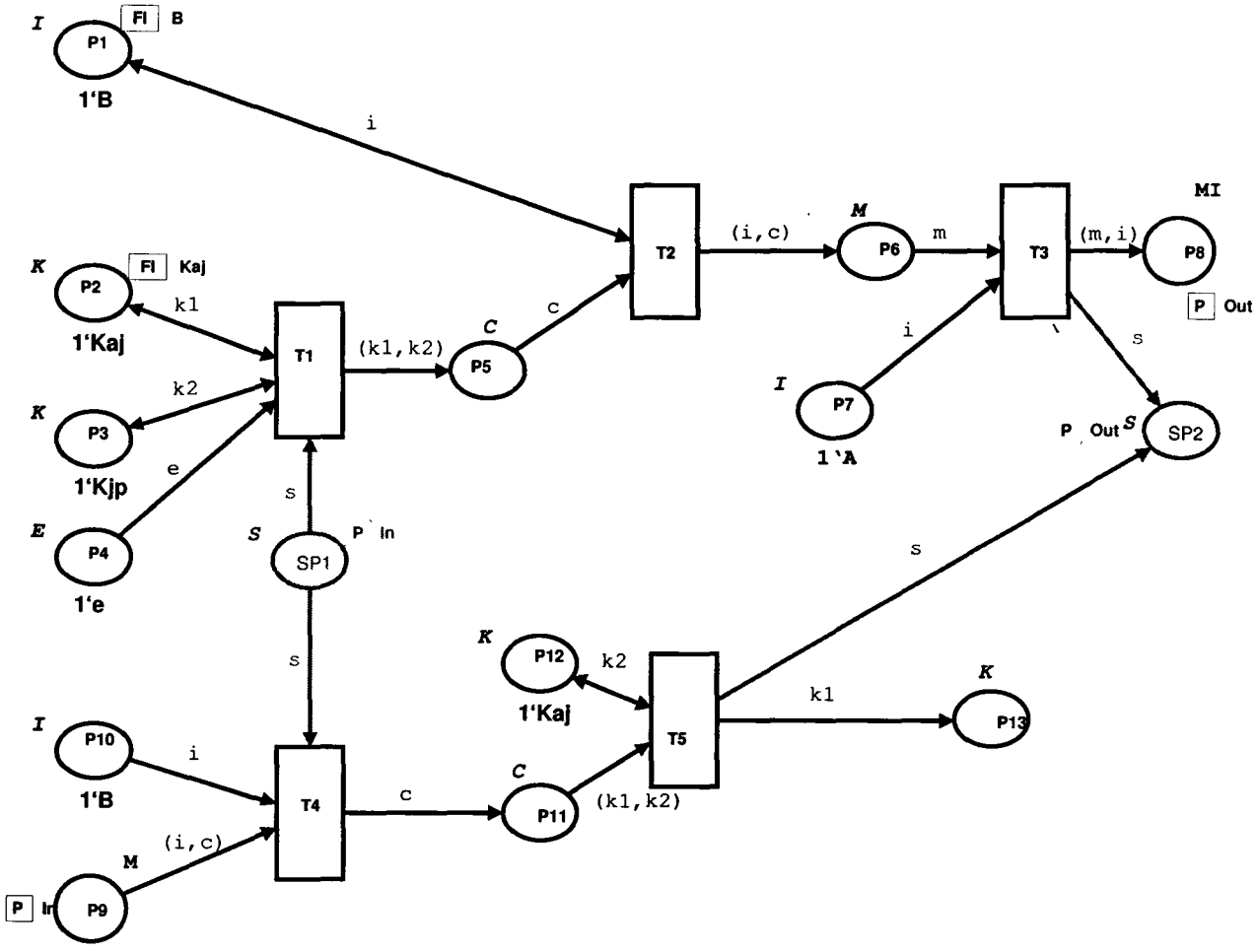


Figure 4.23: The *EntityA* page after adding the S-places

pages: *EntityA* (Figure 4.23), *EntityB* (Figure 4.24), *EntityJ* (Figure 4.25), *intruder* (Figure 4.26), *intruder_mi* (Figure 4.27), *intruder_m* (Figure 4.28), and *intruder_i* (Figure 4.29).

3. Applying the above 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

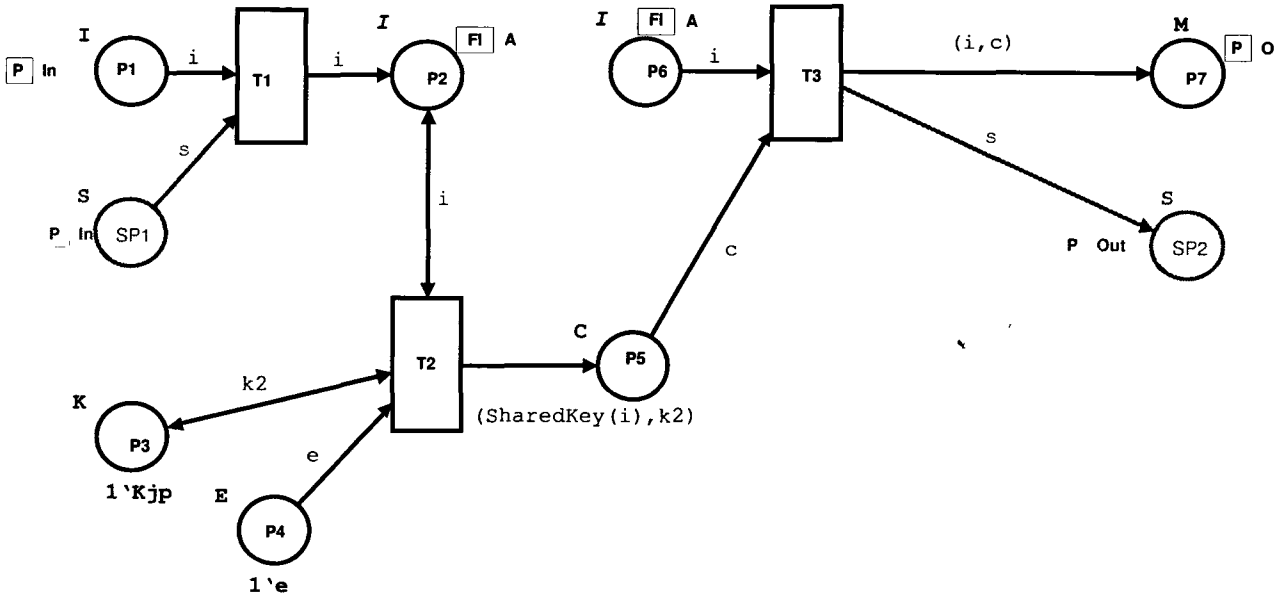


Figure 4.24: The *EntityB* page after adding the S-places

hold the s-token that is passed to the active intruder subprocess. The following outlines the required changes:

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 S fusion set to the last transition of the intruder subpage.

c) Add double arcs from a place that belongs to the S fusion set to the intermediate intruder transitions.

These changes are demonstrated in Figures 4.28 and 4.29. 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.

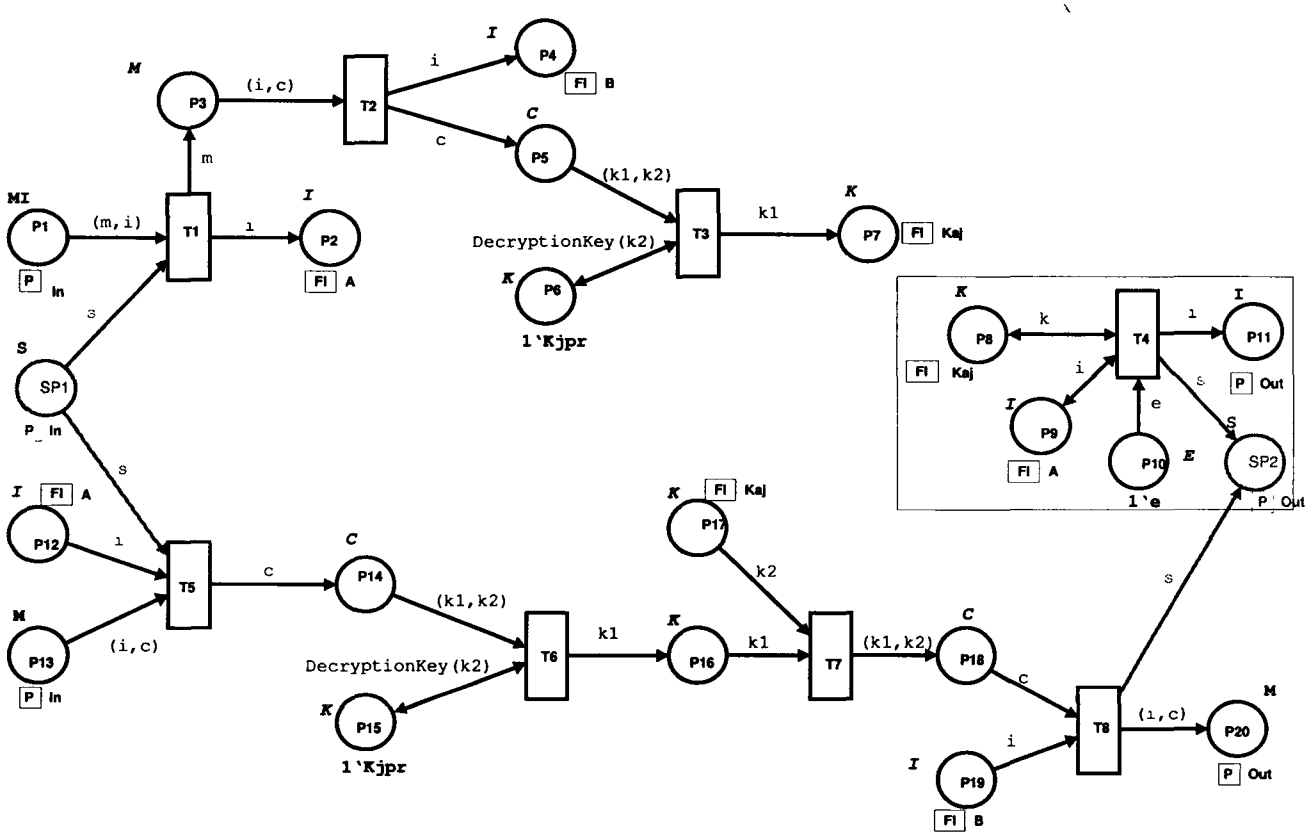


Figure 4.25: The *EntityJ* page after adding the *S*-places

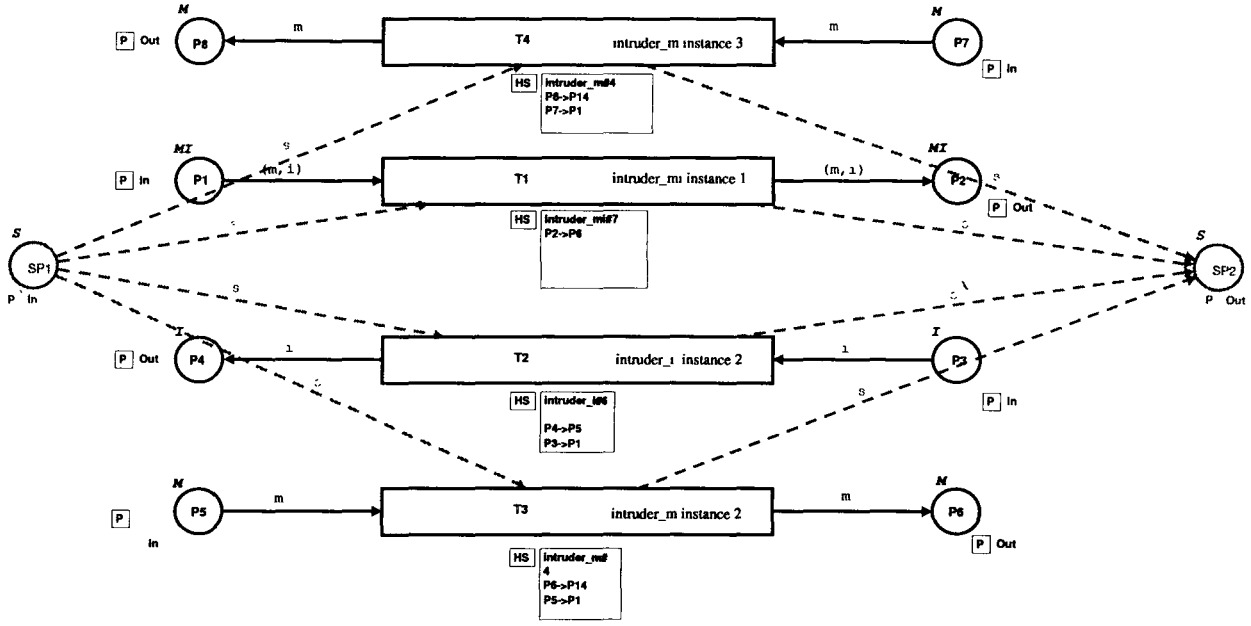


Figure 4.26: The *intruder* page after adding the S-places

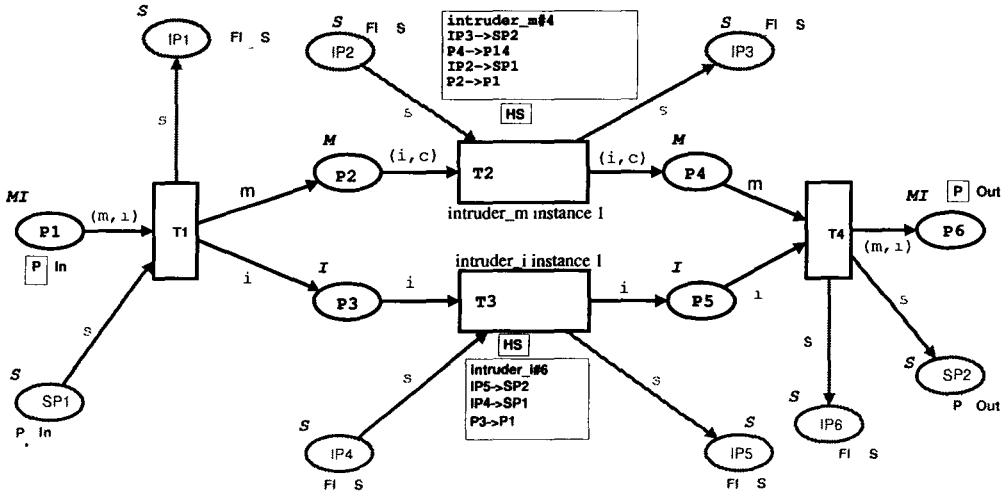


Figure 4.27: The *intruder_mi* page after adding the S-places

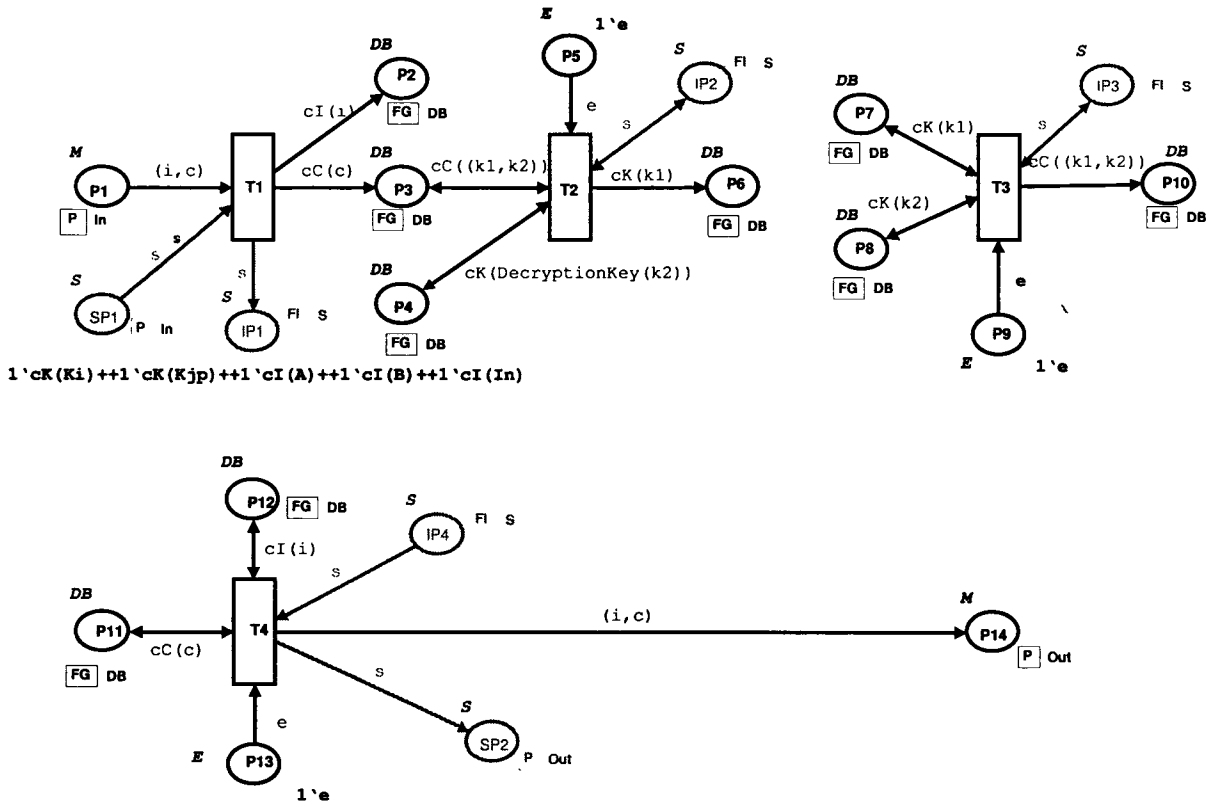


Figure 4.28: The *intruder_m* page after adding the S-places

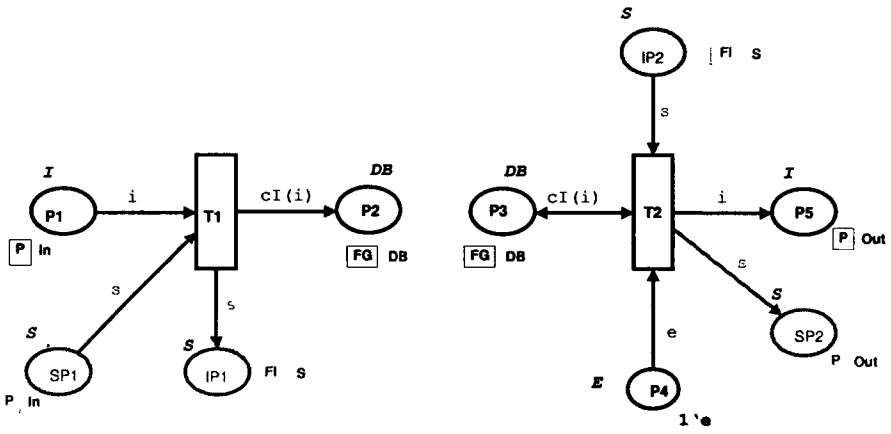


Figure 4.29: The *intruder_i* page after adding the S-places

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 4.27.

After applying the above rules to the TMN model, we were able to produce a full occurrence graph that has 137,092 nodes and 140,736 arcs. It took 535 seconds (almost 9 minutes) to generate the occurrence graph, using the 1-GHz, 16GB machine. Executing *SecrecyViolation1* and *SecrecyViolation2*, defined in Section 4.3.6, return a non-empty node list, containing the nodes that violate the corresponding security requirement.

4.4.2 A Further Improvement: No Storage of Intruder Constructed Ciphers

Before showing attack paths leading to insecure states, we discuss a change to the TMN model that will result in a further decrease of the size of the occurrence graph.

The change is simple. We prevent the intruder from storing the ciphers that it has constructed itself. This does not restrict the intruder since storing an intruder's constructed cipher in a DB-place does not increase its knowledge . Thus, the effect of storing ciphers that an intruder constructs itself is the same as the effect of using these ciphers without storing them.

Thus, transition $T3$ of *intruder_m* in Figure 4.28 is modified, as shown in Figure 4.30, to deposit the output token directly to the output port $P14$. Also, the E-places $P9$ and $P13$ are added to a single instance fusion set E. This is to allow only one of $T3$ or $T4$ to fire in a single subprocess run. This forces the intruder to either send a cipher it has constructed, or to replay an intercepted cipher, but not both. Finally, transition $T3$ is modified to deposit the s-token to the output port $SP2$, since $T3$ is now the last transition in the *intruder_m* subprocess.

After applying this modification, the size of the occurrence graph is reduced to

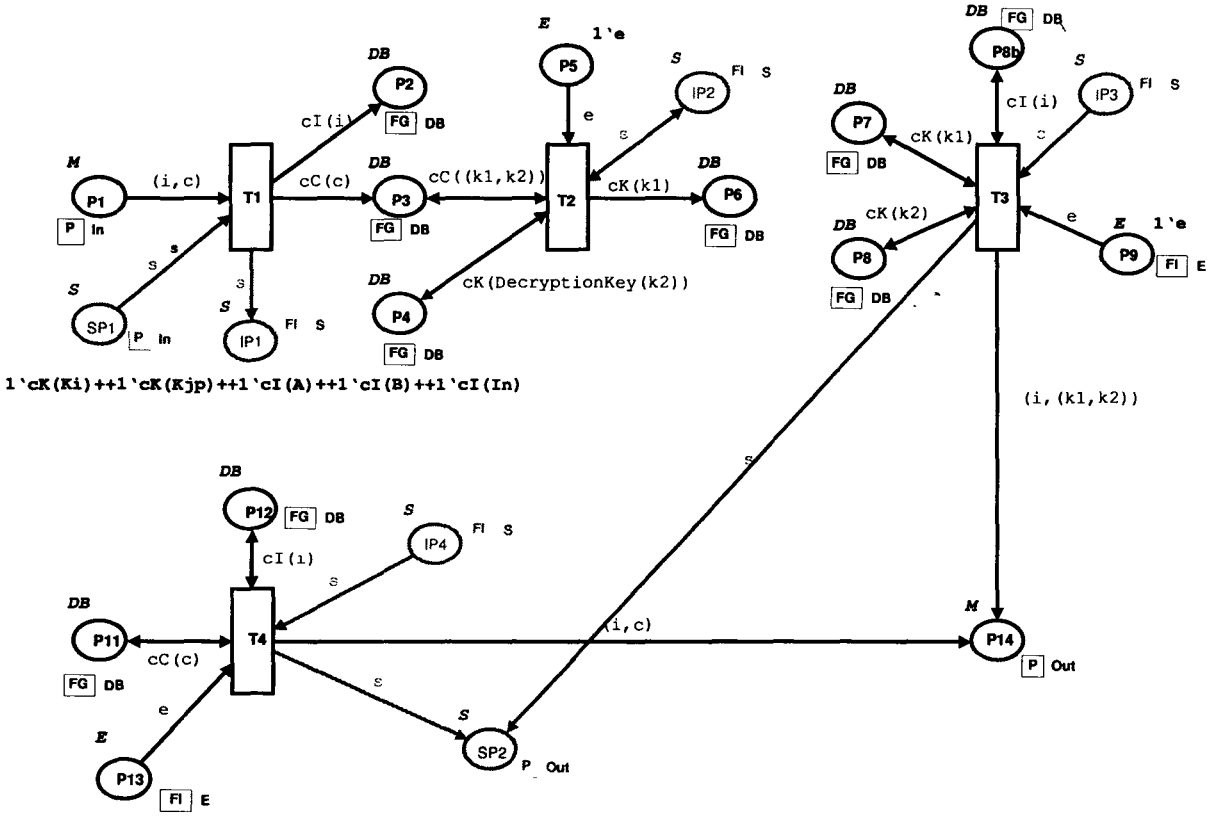


Figure 4.30: The *intruder_m* page after preventing the intruder from storing ciphers it constructs

4,183 nodes and 4,632 arcs. It took less than 3 seconds to generate the occurrence graph, using the 1-GHz, 16GB machine.

4.4.3 Fixing a Non-Intended Constraint on the Intruder Behaviour

The model, as constructed so far, has an unintentional restriction on the intruder's behaviour. The problem is that the intruder can only modify a message without being able to redirect traffic. Thus, an attack which involves I intercepting a message sent from A to J, modifying it then re-sending it to B, will not be detected using our model.

The reason for this problem is the use of instance fusion sets in $intruder\_i$ and $intruder\_m$, see Figures 4.29 and 4.30. By using a global fusion set instead, this problem can be solved. This is shown in Figures 4.31 and 4.32. Applying this change results in the final TMN model.

The existence of this accidental constraint and its fix show that small changes in the CPN model result in having different assumptions and restrictions on the behaviour of entities. This can be beneficial since the modeler has the ability to introduce and test several assumptions and restrictions, which can be helpful to better understand a protocol and test it under a variety of settings. However, this also means that the modeler should make sure that the model captures the specified behaviour accurately.

4.4.4 The Final TMN Model

The final TMN model is shown in Figures 4.15, 4.16, 4.21, 4.23, 4.24, 4.25, 4.26, 4.27, 4.31, and 4.32.

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

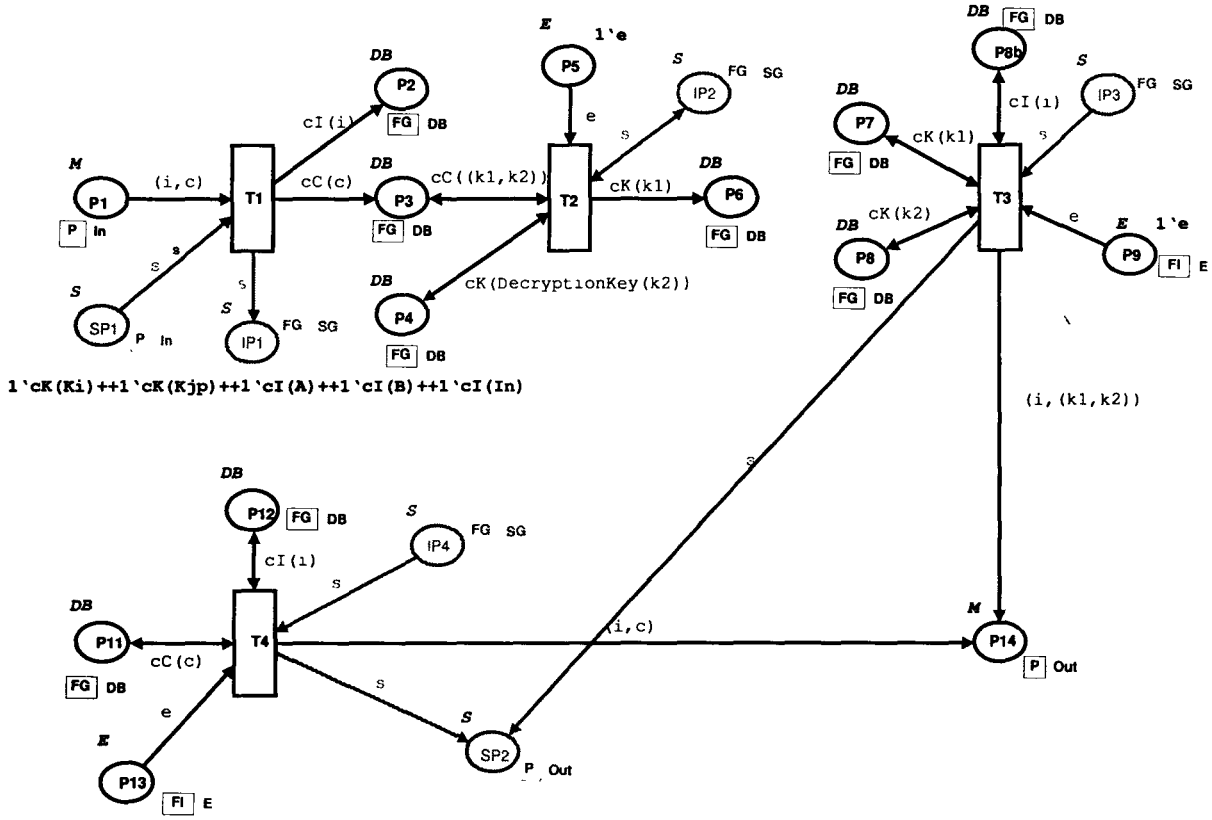


Figure 4.31: The page *intruder_m* after using a global fusion set *SG*

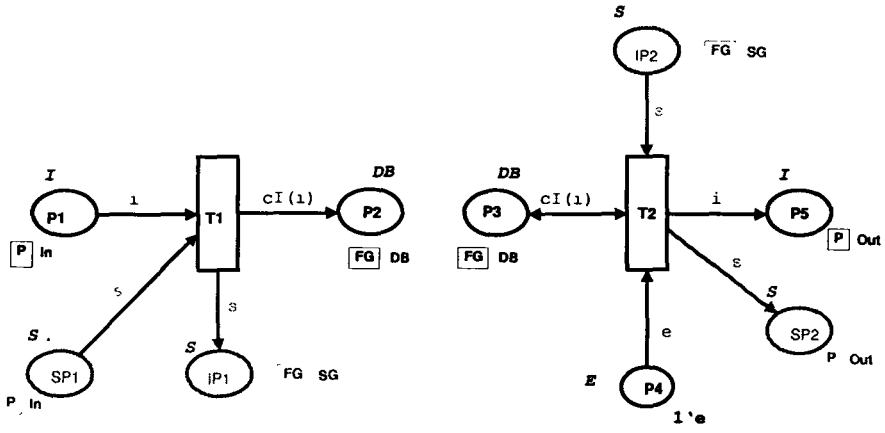


Figure 4.32: The page *intruder_i* after using a global fusion set *SG*

machine. Using a different machine (100MHz, 64MB), it took a minute and a half. Executing SecrecyViolation1 and SecrecyViolation2 return a non empty node list.

As a summary of the model changes, the following lists the different versions constructed before arriving at the final model. Note that the execution time to generate the occurrence graphs is based on runs on a 1-GHz, 16GB machine.

1. The original TMN model. We ran the *OG tool* for 162 hours and constructed a partial occurrence graph, with 500,000 nodes and 1,500,000 arcs.

2. Version 1 of the TMN model. In this model, the S-constructs are added to prevent unnecessary interleaving of subprocesses. It took 9 minutes to generate a full occurrence graph, with 137,092 nodes and 140,736 arcs.

3. Version 2 of the TMN model. In this model, the intruder is prevented from storing ciphers it has generated itself. It took 3 seconds to generate a full occurrence graph, with 4,183 nodes and 4,632 arcs.

4. The final version of the TMN model. In this model, the fusion sets in $intruder\_m$ and $intruder\_i$ are made global. This allows the intruder to redirect traffic. The resulting final occurrence graph has 19,237 nodes and 22,419 arcs. It took 19 seconds to generate the occurrence graph.

Figure 4.33 shows the result of executing SecrecyViolation1 and SecrecyViolation2. SecrecyViolation1 returns all nodes violating the first security requirement (the intruder knows K_{AB}). For instance, node 19170 represents an insecure state. Figure 4.34 shows a path on the occurrence graph from the initial marking to this insecure marking. The corresponding occurrence sequence is as follows:

`EntityA`	`T1`	$k1 = Kaj, k2 = Kjp$
`EntityA`	`T2`	$i = B, c = (Kaj, Kjp)$
`EntityA`	`T3`	$i = A, m = (B, (Kaj, Kjp))$
`intruder_mi`	`T1`	$m = (B, (Kaj, Kjp)), i = A$

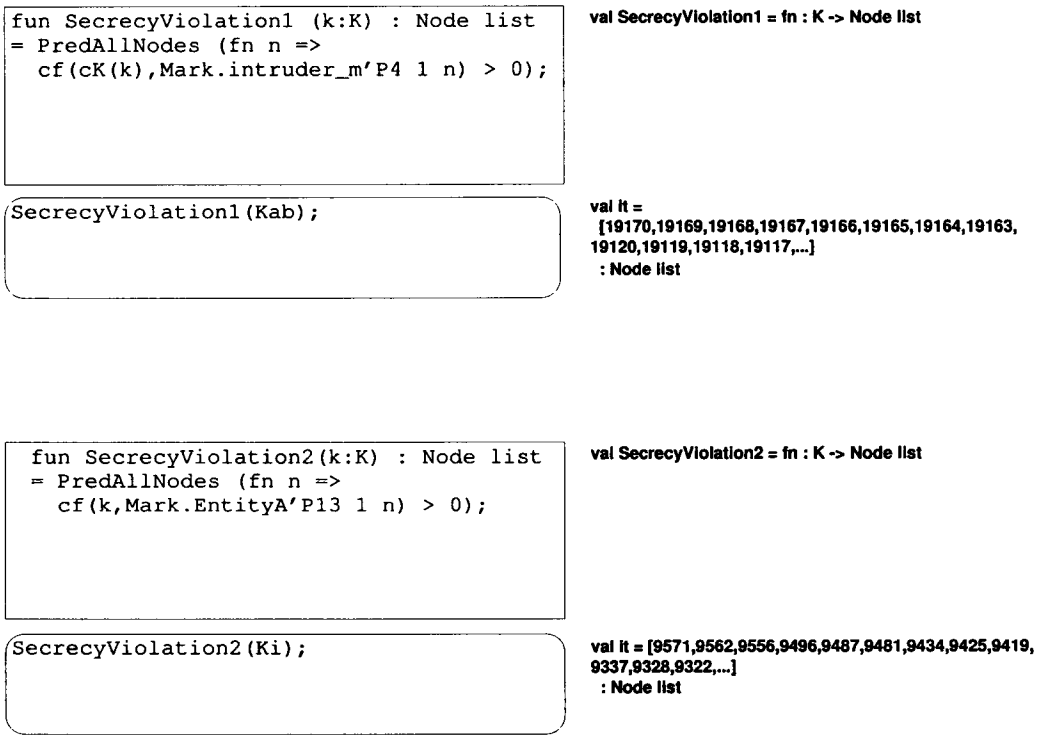


Figure 4.33: The result of executing SecrecyViolation1 and SecrecyViolation2

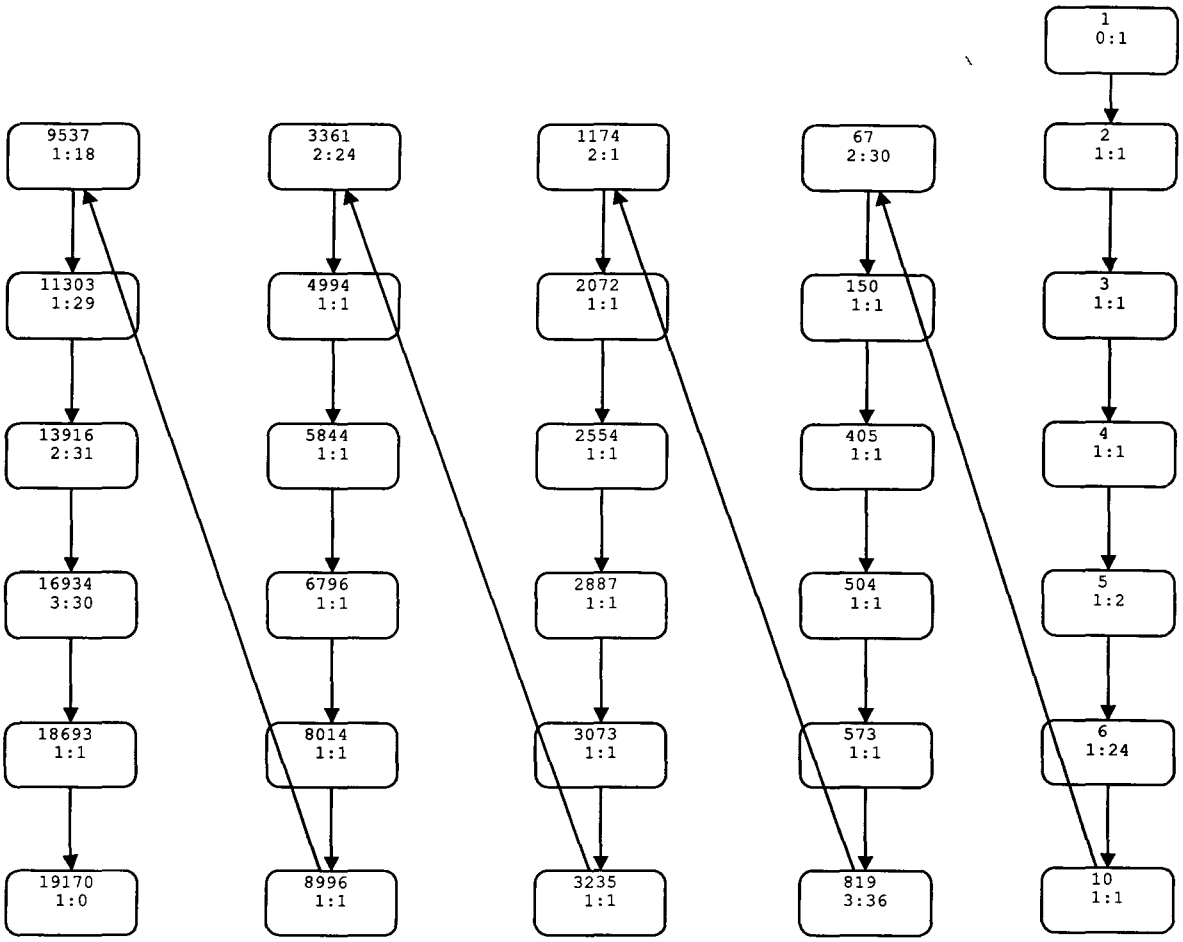


Figure 4.34: A path from the initial marking to the insecure marking 19170

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:

1.($I1$) $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+)), where\ PosesAs(I, J)$

2.($I2$) $Send(I, B, ID(A)), where\ PosesAs(I, J)$

3.($I3$) $Send(B, J, ID(A) \oplus E(K_{AB}, K_J^+)), where\ PosesAs(I, J)$

4.($II1$) $Send(I, J, ID(A) \oplus E(K_I, K_J^+)), where\ PosesAs(I, A)$

5.($II2$) $Send(J, A, ID(A)), where\ PosesAs(I, A)$

6.($II3$) $Send(I, J, ID(A) \oplus E(K_{AB}, K_J^+)), where\ PosesAs(I, A)$

7.($II4$) $Send(J, A, ID(A) \oplus E(K_{AB}, K_I)), where\ PosesAs(I, A)$

Now the intruder decrypts $E(K_{AB}, K_I)$ to obtain K_{AB}. Thus, the intruder is able to impersonate A. Note the replay of $K_J^+(K_{AB})$ in step $II3$.

SecrecyViolation2 returns all nodes violating the second security requirement (entity A never accepts K_I as a session key). For instance, node 9571 represents a marking violating this requirement. Figure 4.35 shows a path on the occurrence graph from the initial marking to this insecure marking. The corresponding occurrence sequence is as follows:

`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_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	$k = Ki, i = A$
intruder_i	2	T1	$i = A$
intruder_m	2	T4	$i = A, c = (Kaj, Kjp)$
EntityJ		T5	$i = A, c = (Kaj, Kjp)$
EntityJ		T6	$k1 = Kaj, k2 = Kjp$
EntityJ		T7	$k1 = Kaj, k2 = Ki$
EntityJ		T8	$i = A, c = (Kaj, Ki)$
intruder_m	3	T1	$i = A, c = (Kaj, Ki)$
intruder_m	2	T2	$k1 = Kaj, k2 = Ki$ (*)
intruder_m	3	T2	$k1 = Kaj, k2 = Ki$
intruder_m	3	T3	$i = B, k1 = Ki, k2 = Kaj$
EntityA		T4	$i = B, c = (Ki, Kaj)$
EntityA		T5	$k1 = Ki, k2 = Kaj$

Note the reachability of Kaj to a DB-place in the step identified by (*). Also, note the acceptance of Ki by entity A as a session key in the last step of the occurrence sequence.

This attack is stated in a high level description as follows:

1.($I1$) $Send(A, J, ID(B) \oplus E(K_{AJ}, K_J^+)), where\ PosesAs(I, J)$

2.($I1'$) $Send(I, J, ID(A) \oplus E(K_I, K_J^+)), where\ PosesAs(I, A)$

3.($I2$) $Send(J, A, ID(A)), where\ PosesAs(I, A)$

4.($I3$) $Send(I, J, ID(A) \oplus E(K_{AJ}, K_J^+)), where\ PosesAs(I, A)$

5.($I4$) $Send(J, A, ID(A) \oplus E(K_{AJ}, K_I)), where\ PosesAs(I, A)$

6.($I4'$) $Send(I, A, ID(B) \oplus E(K_I, K_{AJ})), where\ PosesAs(I, J)$

Note the replay of $E(K_{AJ}, K_J^+)$ in step $I3$. Note also that the intruder gets to know K_{AJ} in step $I4$. This is a violation of the third security requirement; that K_{AJ} should be secret to A and J only.

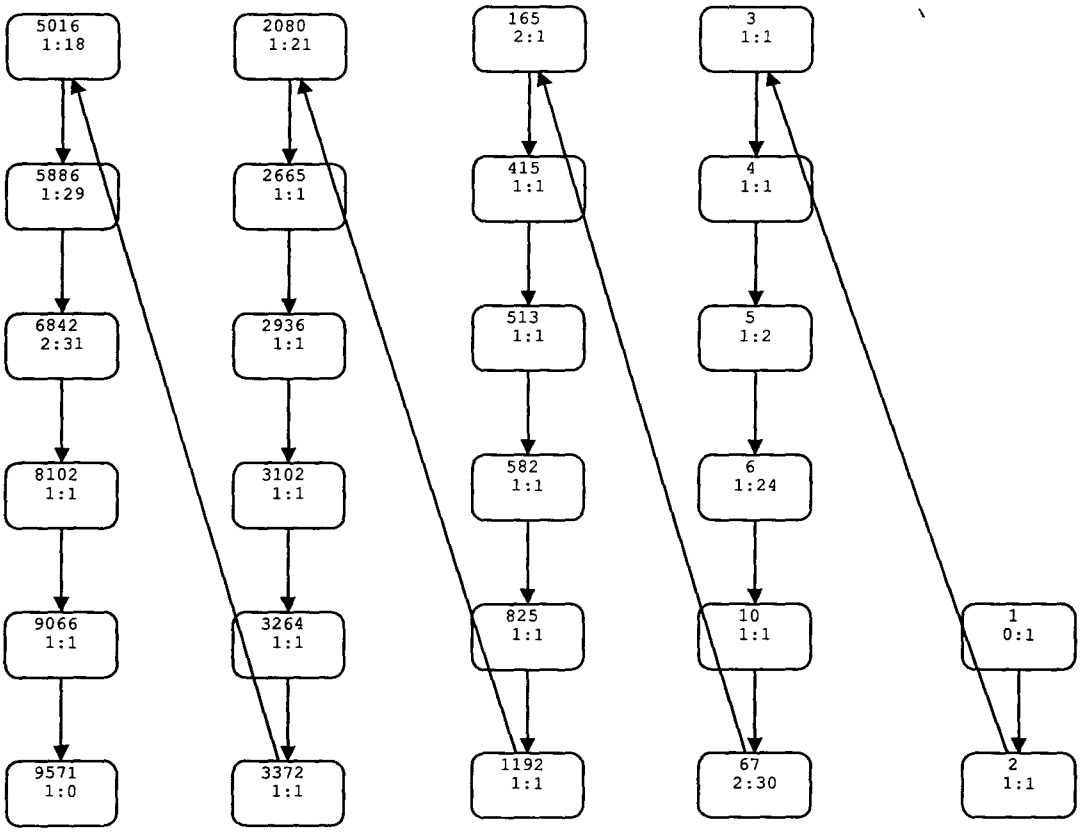


Figure 4.35: A path from the initial marking to the insecure marking 9571

4.5 Discussion

Our technique which is outlined in this chapter can be applied in the verification of the known cryptographic protocols [CJ97, Sec04a]. Appendix B applies this technique in the verification of the Needham-Schroeder authentication protocol.

We make several verification assumptions. First, cryptographic algorithms are assumed to be secure. We do not include algebraic properties of the algorithms used. However, our model can be extended to capture such properties since such operators can be defined in CPN/ML.

Another assumption is that nothing can be said about the contents of a ciphertext without decrypting it with the correct decryption key. Thus, if an agent does not have the decryption key, it does not apply further encryption or decryption to the ciphertext. This assumption is typically made by finite-state verification methods to limit the state space.

In our technique, we assume that an entity can distinguish fields. Thus, it can know whether a given token is a key, a nonce, a ciphertext, *etc.* This assumption is typical in other finite state methods. We mention that our model can be extended so this is not possible, by using appropriate colour definitions and variables. It allows for the inclusion of attacks such as type attacks [HLS00].

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 [Jen96b] seem to be useful. Other techniques to yield a reduced representation of the occurrence graph are applicable. These include the stubborn set method [KV98], and occurrence graphs

with equivalence classes [Jen96b].

Chapter 5

Conclusion

5.1 Discussion

The use of coloured Petri nets in the verification of cryptographic protocols has many advantages. The most important benefit is the graphical nature of their models. The use of graphical models adds simplicity and clarity to the modeled protocols. Furthermore, we believe that coloured Petri nets, with the small number of primitives they have, are easier to use and learn.

Furthermore, CP-nets support several aspects that are of great value to the design and construction of the models. For instance, CP-nets support hierarchical net construction. This enables the construction of models at different levels of abstraction. Furthermore, this makes it possible to create a model for an entity and use several instances of it. Thus, this benefits the design in the same way information hiding and separation of concerns help the design of software systems.

Design/CPN has been extremely beneficial in constructing and analyzing models of cryptographic protocols. The CPN Editor provides a graphical user interface that makes it easy to construct and check the model. The CPN Simulator and the Occurrence Graph Tool are easy to use.

Our technique uses Jensen's form of coloured Petri nets, whereas the models de-

veloped by Nieh [NT92] and Basyouni [BT97] use a different form. The following features of Jensen's form have not been used in their models: fusion of places, arc expressions, guards, and functional expressions. These features make our models clearer and simpler. Furthermore, our models are concise, since we effectively use colour sets and variables. These benefits are attributed to our use of a higher level form of coloured Petri nets.

The use of Jensen's form allows our models to be constructed and analyzed using Design/CPN which is a powerful automated tool. Furthermore, the use of established formal methods for CP-nets becomes applicable to our models.

Our intruder model is more generic than Nieh's [NT92] and Basyouni's [BT97] models. This is a result of using the DB place and arc variables. The intruder model specifies what an intruder can do to intercepted tokens *e.g.* decrypting ciphertexts, storing plaintexts, constructing new messages, *etc.* Intercepted tokens are stored in the DB-place. This adds clarity and simplicity to the intruder model.

Our technique is a finite-state verification method [Hel98]. In a finite-state method, a model is built to capture the abstracted possible behaviours of the modeled system. The main analysis method used to verify the model is reachability analysis. When verifying a cryptographic model, several verification assumptions are used to limit the state space of the model. For instance, the model usually involves a limited number of protocol instances and runs. Then, the question that these methods try to answer is: does this model under the verification assumptions contain flaws? If it does, then there is a flaw in the modeled protocol. If it does not, then we can say nothing about the protocol correctness for an arbitrary number of concurrent runs.

Our technique compares well with other finite-space methods. 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

search when a certain criteria is met.

5.2 Future Work

We believe that the use of coloured Petri nets in the verification of cryptographic protocols is promising. The following areas could be of interest for other researchers:

- Building a library of intruder attacks. For instance, Bird *et al.* [BGH+91] characterizes families of attacks on authentication protocols. Syverson [Syv94] presents a taxonomy of replay attacks on cryptographic protocols. The verifier can use these libraries of attacks to test different configurations of the protocol entities and runs. The hierarchical and graphical nature of our models are of great help in constructing such configurations.

- Using other search techniques and analytical methods. There are many formal CP-net analysis methods. Some of these deal with reducing the size of the occurrence graph by exploiting symmetries, equivalence classes, and stubborn sets [Jen96b, KV98]. Currently, Design/CPN supports the construction of occurrence graphs with equivalence classes [Col04d]. Other analytical techniques can also be applied. For instance, the matrix equation method [Jen96b] can be used to check the reachability of a given marking.

  Most of the security properties we analyze involve testing the reachability of a certain token to a given place. New analytical techniques can be developed to apply this kind of analysis. Thus, by using these techniques, we can analyze protocols involving multiple runs without generating the full occurrence graph.

- Modeling new protocols and discovering new attacks.

- Exploring the use of other automated tools in the verification process.

- Analyzing properties of the CP-net models other than security related ones. These include dynamic properties, such as fairness and liveness. Performance

analysis, such as protocol delay times, can also be studied using CP-nets with time extensions [Jen96b]. We note that Design/CPN provides automatic support to both kinds of analysis.

Appendix A

Functional Definitions

α : Set of Alphabets

α is the set of alphabets from which message strings are built. For instance, in computer networks, α would be $\{0, 1\}$.

Ω : Set of Agents (Entities)

Δ : Set of Keys

Note that $\Delta \subseteq \mathcal{P}(\alpha^*)$.

Λ : Set of nonces

Note that $\Lambda \subseteq \mathcal{P}(\alpha^*)$.

KeyOriginator : $\Delta \longrightarrow \Omega$

$KeyOriginator(k)$, where $k \in \Delta$, returns $a \in \Omega$ such that a is the agent who has originated the key k.

KeyOwner : $\Delta \longrightarrow \mathcal{P}(\Omega)$

$KeyOwner(k)$, where $k \in \Delta$, returns all agents $a \in \Omega$ such that a has the key k.

$\mathbf{E} : \alpha^* \times \mathbf{\Delta} \longrightarrow \alpha^*$

$E(X, k) = Y$ if and only if $Y \in \alpha^*$ is the ciphertext that results from encrypting $X \in \alpha^*$ using the key $k \in \Delta$. Equivalently, $E^{-1}(Y, k) = X$ if and only if $X \in \alpha^*$ is the plaintext that results from decrypting $Y \in \alpha^*$ using the key $k \in \Delta$.

$\mathbf{AssociatedKey} : \mathbf{\Delta} \longrightarrow \mathbf{\Delta}$

$AssociatedKey(k) = k'$, where $k, k' \in \Delta$, if and only if k' is the key related to k. The following result defines this relation:

$$E(X, k) = y \Longleftrightarrow E^{-1}(Y, AssociatedKey(k)) = X.$$

Note that a key symbol without a superscript denotes a key used in symmetric key algorithms, *e.g.* k_A. In this case, $k_A = AssociatedKey(k_A)$. On the other hand, a key symbol with a superscript denotes a key used in public key algorithms e.g. k_A^+ and k_A^-. The superscript '+' indicates a public key, while '$-$' indicated a private key. Thus, $k_A^+ = AssociatedKey(k_A^-)$, and $k_A^- = AssociatedKey(k_A^+)$.

$\mathbf{Send} \subseteq \mathbf{\Omega} \times \mathbf{\Omega} \times \alpha^*$

$Send(A, B, M) \Longleftrightarrow$ Agent $A \in \Omega$ sends the message $M \in \alpha^*$ to agent $B \in \Omega$.

$\mathbf{PosesAs} : \mathbf{\Omega} \times \mathbf{\Omega} \longrightarrow \mathbf{Boolean}$

$PosesAs(I, A) \Longleftrightarrow I \in \Omega$ poses as $A \in \Omega$.

$\oplus : \alpha^* \times \alpha^* \longrightarrow \alpha^*$

$\oplus$ is the string concatenation operator.

$\mathbf{ID} : \mathbf{\Omega} \longrightarrow \alpha^*$

$ID(A)$ returns the string that represents the id of agent $A \in \Omega$.

Appendix B

Verification of the Needham-Schroeder Public Key Protocol

The Needham-Schroeder Public Key Protocol (NSPK) protocol is introduced in Section 1.2.3. It is a mutual authentication protocol. After the second step, entity A should be confident that it is communicating with B. After the last step, entity B should be confident that it is communicating with A.

We follow the technique introduced in Chapter 4. First, we construct a model that does not include the intruder. Then, we add the intruder. We try two different entity configurations. In the first one, we model the behaviour of entities in a session where A initiates the protocol to mutually authenticate B. In the second configuration, we model the behaviour in a session where A initiates the protocol to mutually authenticate I. We discover no flaws in the first configuration, however, there is a flaw in the second configuration allowing I to pose as A to B. This flaw was discovered by Lowe in 1995 [Low95], fourteen years after the publication of the NSPK protocol [NS78].

B.1 The Model with no Intruder

Figure B.1 shows the top-level model of the NSPK protocol, along with the CPN ML declarations.

Note the definition of the ciphertext colour set C. All ciphertexts in the NSPK protocol can be represented using the ordered pair $(p1, p2, k)$, where $p1$ and $p2 \in IN = \{A, B, N_A, N_B, X\}$, $k \in K$ (the set of keys), and X denotes the empty field. For instance, the ciphertext of the first message is represented by the colour (A, Na, Kbp), whereas the ciphertext of the last message is represented by the colour (Nb, X, Kbp). Thus, C is defined as $IN \times IN \times K$.

Figures B.2 and B.3 show the CPN models of entity A and B, respectively. Figure B.4 shows the hierarchy page.

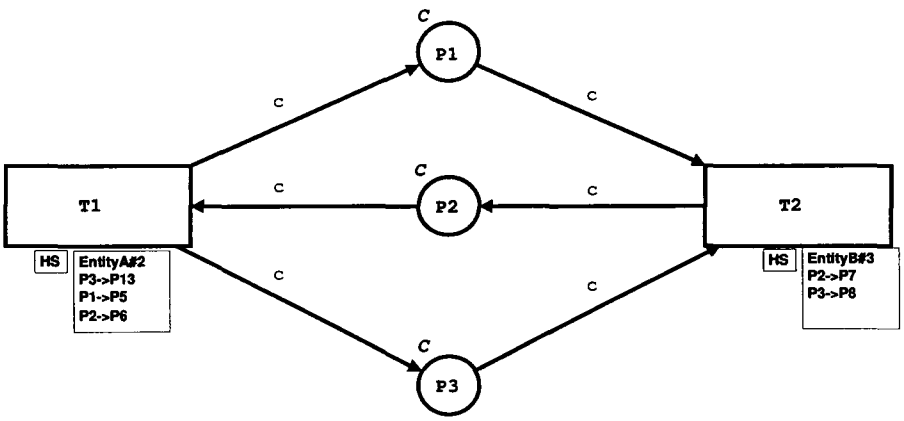
B.2 The Model with an Intruder- A Initiates a Session with B

In this section, we construct the model of the NSPK protocol with an intruder, as outlined in chapter 4. It models the behaviour of the protocol entities in a setting where A initiates a session with B.

Figure B.5 shows the top level model of the NSPK protocol with an intruder. Figure B.6 contains the CPN ML declarations. Figures B.7 and B.8 show the CPN models of entities A and B, respectively.

The intruder subpage is shown in Figure B.9. The intruder subprocess is shown in Figure B.10. As shown in Figure B.10, the DB is set initially to $\{A, B, In, K_I, K_A^+, K_B^+\}$. Figure B.11 shows the hierarchy page.

One of the requirements that this protocol aims to satisfy is the authentication of A to B. In terms of CPN markings, this translates into the requirement that place $P12$ in $EntityB$ (Figure B.8) never has a token with a colour other than A.




```
color T = with A | B | Na | Nb | Kap | Kapr | Kbp | Kbpr | X;
color I = subset T with [A,B];
color K = subset T with [Kap, Kapr, Kbp, Kbpr];
color N = subset T with [Na, Nb];
color IN = subset T with [A,B,Na,Nb,X];
color C = product IN * IN * K;
color E = with e;
var c:C;
var i:I;
var n1,n2,n: N;
var k1,k2,k:K;
fun DecryptionKey(k:K):K = case k of Kap => Kapr | Kapr => Kap | Kbp =>
Kbpr | Kbpr => Kbp;
fun PublicKey(i:I):K = case i of A => Kap | B => Kbp;
```

Figure B.1: The NSPK model with no intruder (with declarations)

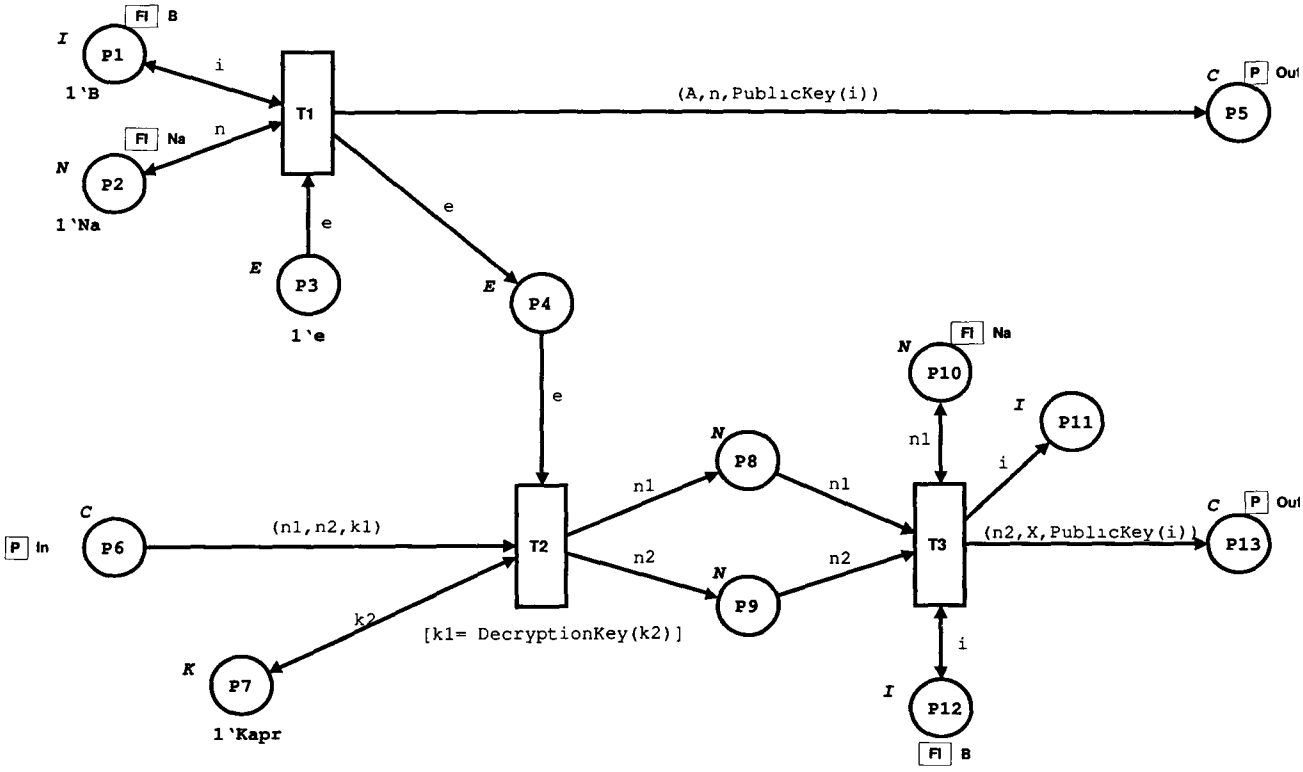


Figure B.2: Page *EntityA*

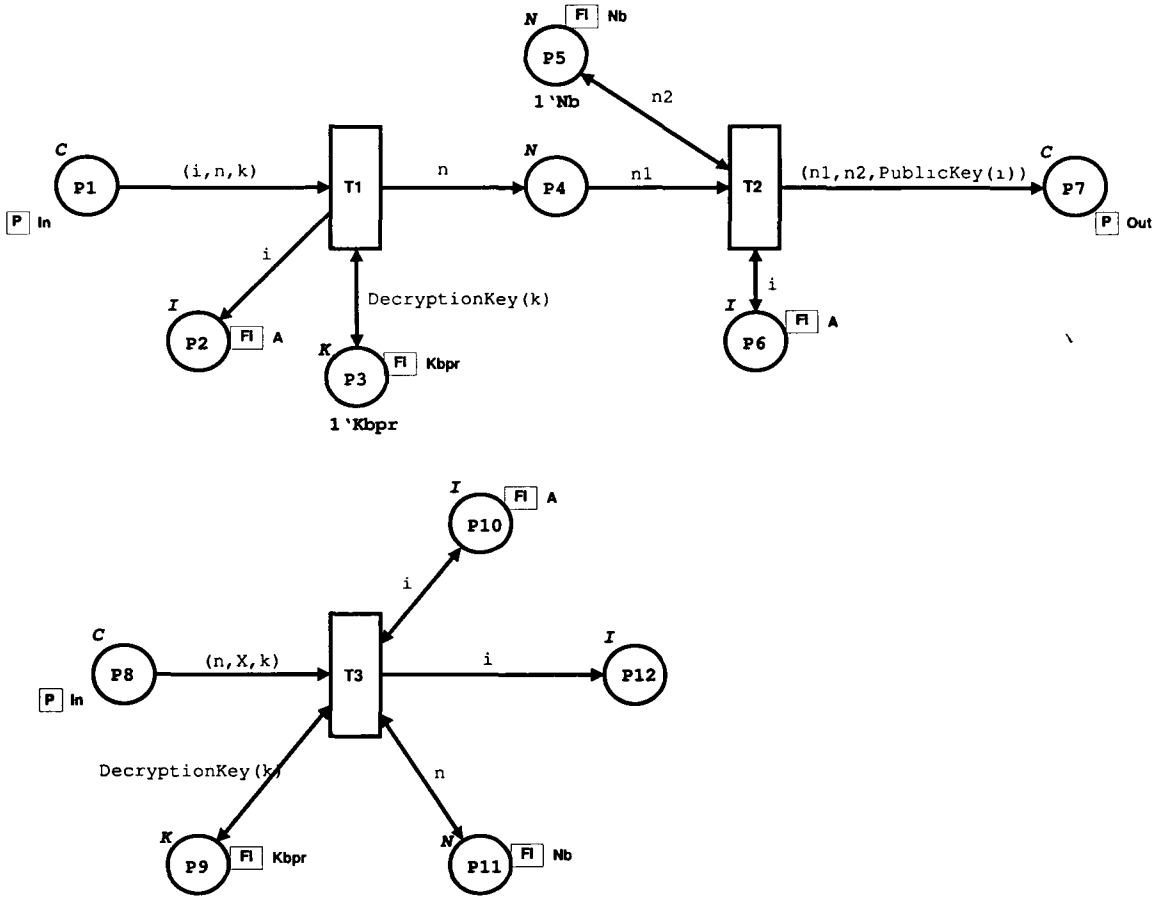


Figure B.3: Page *EntityB*

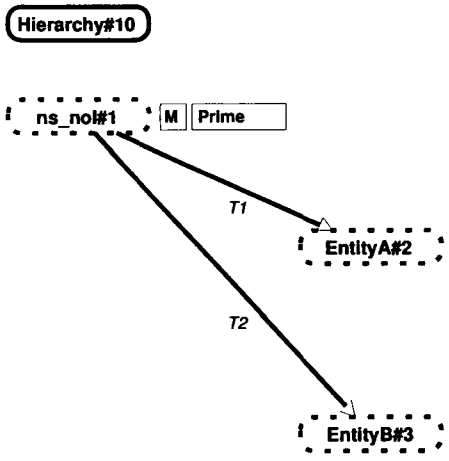


Figure B.4: The hierarchy page

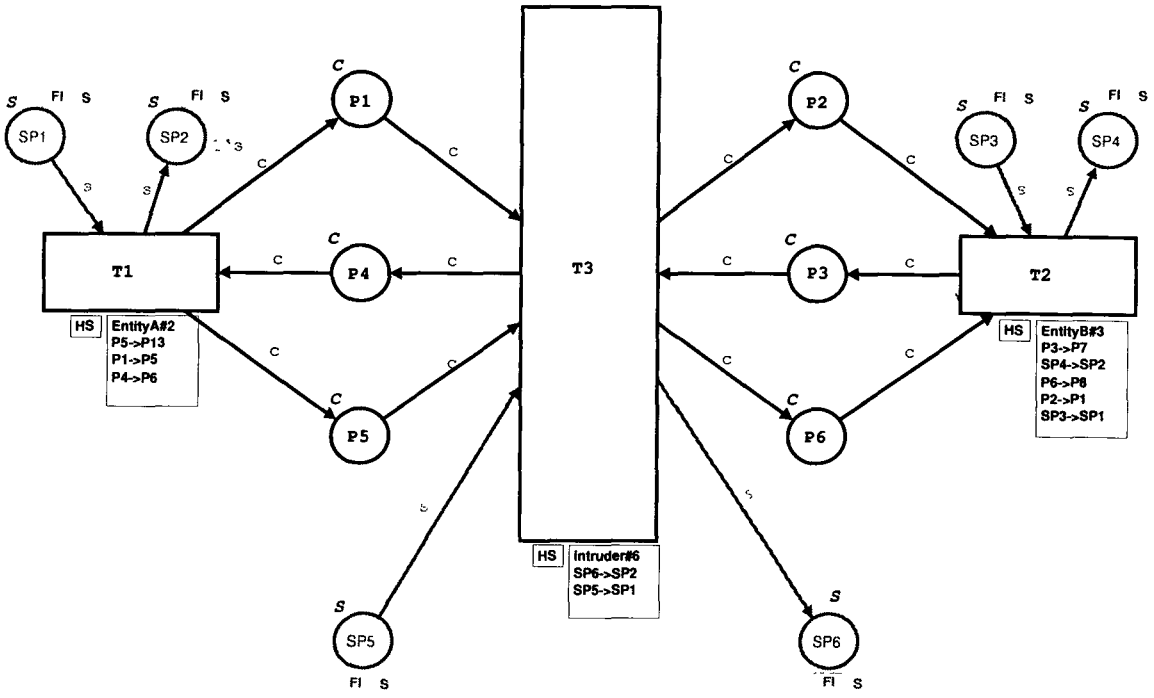


Figure B.5: The NSPK top-level model with an intruder

```
color T = with A | B | In | Na | Nb | Kap | Kapr | Kbp | Kbpr | Ki | X;
color I = subset T with [A,B,In];
color K = subset T with [Kap, Kapr, Kbp, Kbpr,Ki];
color N = subset T with [Na, Nb];
color IN = subset T with [A,B,In,Na,Nb,X];
color C = product IN * IN * K;
color E = with e;
var c:C;
var i:I;
var n1,n2,n: N;
var k1,k2,k:K;
var t1,t2:IN;
fun DecryptionKey(k:K):K = case k of Kap => Kapr | Kapr => Kap | Kbp =>
Kbpr | Kbpr => Kbp | Ki => Ki;
fun PublicKey(i:I):K = case i of A => Kap | B => Kbp | In => Ki;
color DB = union cC:C + cIN:IN + cK:K;
color S=with s;
```

Figure B.6: Declarations used in the NSPK model with an intruder

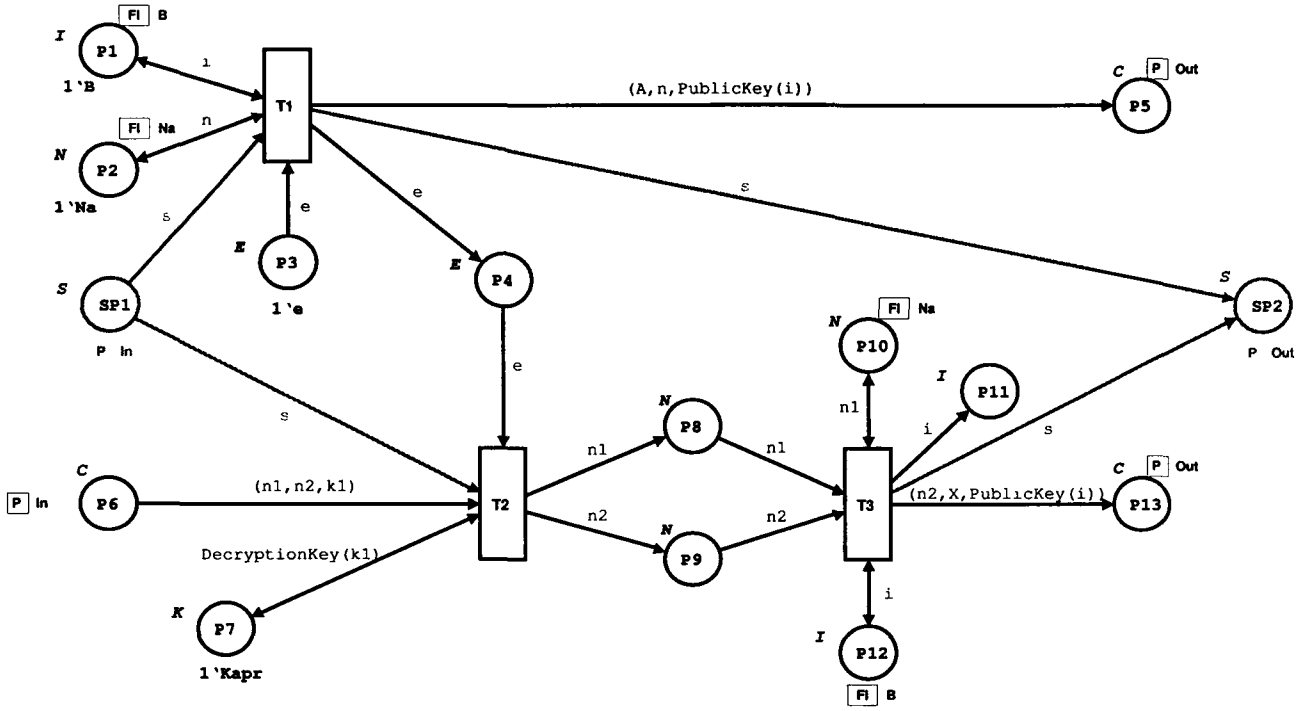


Figure B.7: Page *EntityA* in the NSPK model with an intruder

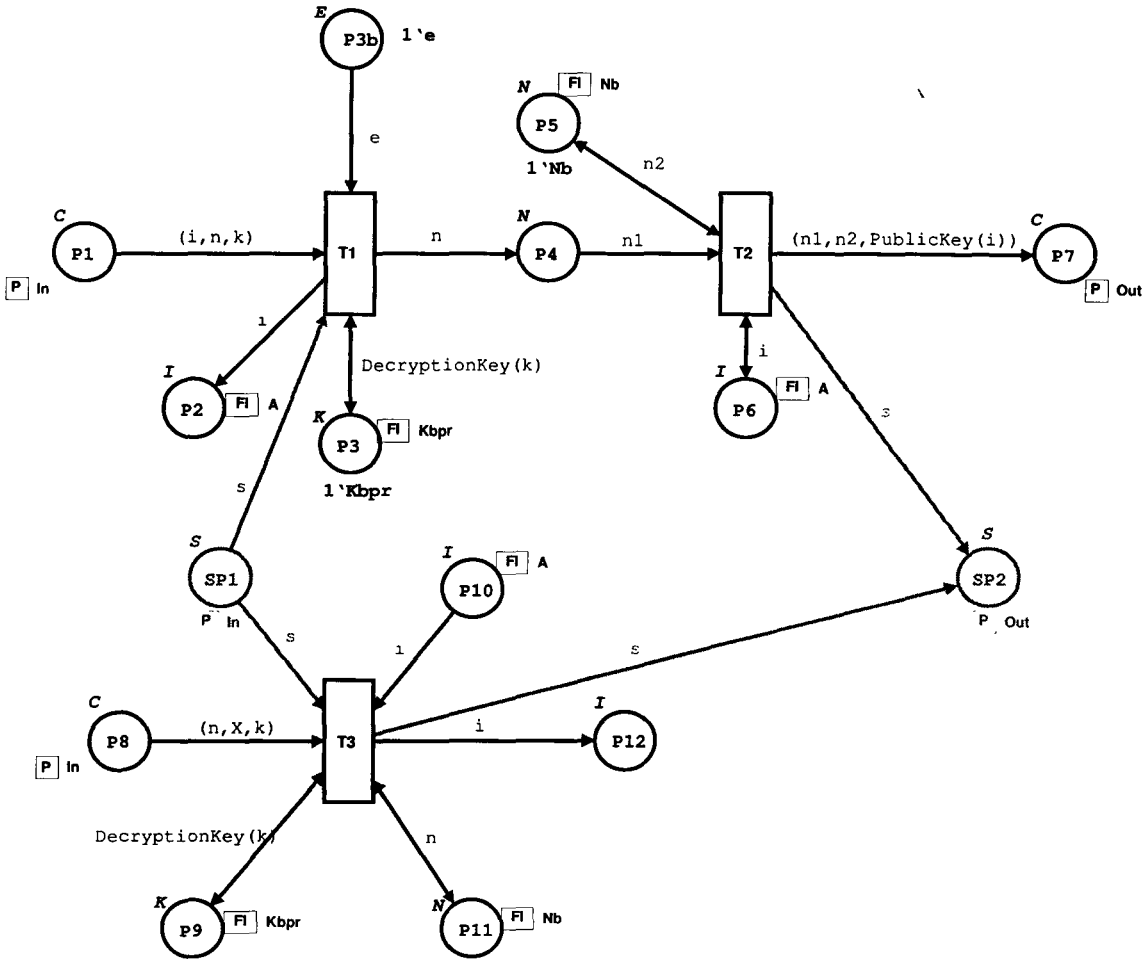


Figure B.8: Page *EntityB* in the NSPK model with an intruder

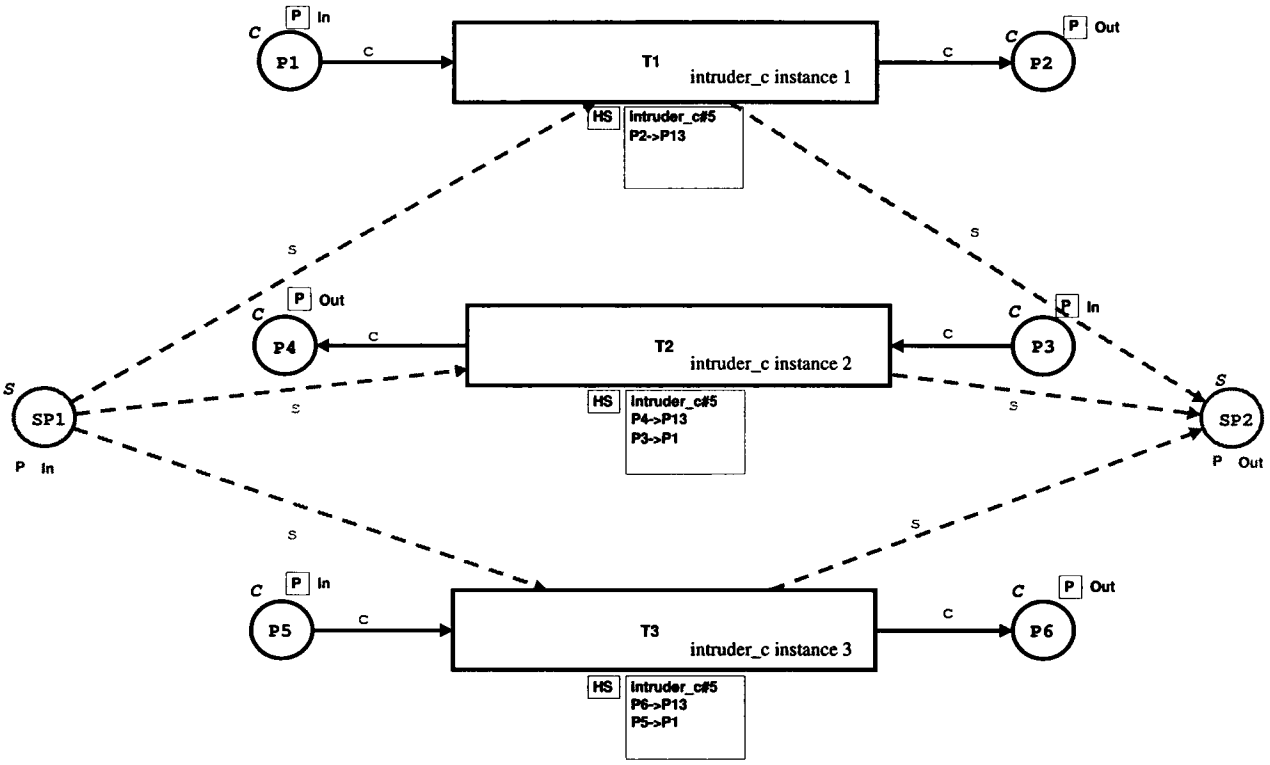


Figure B.9: The *intruder* page

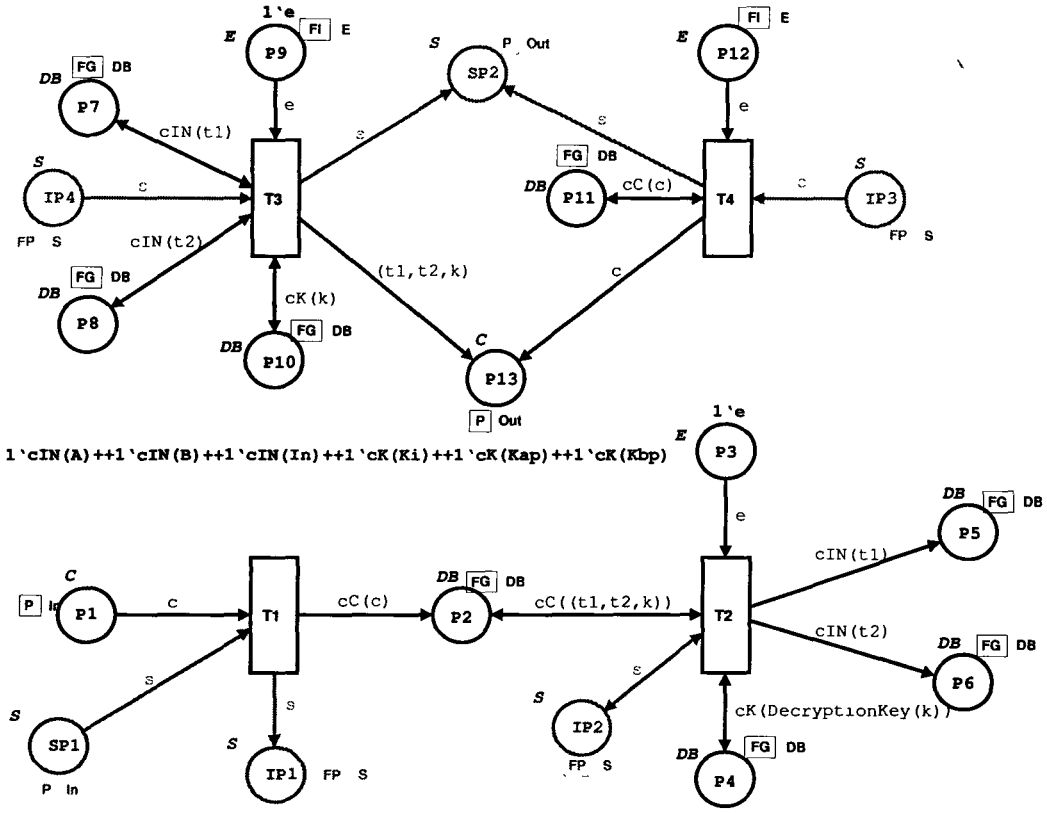


Figure B.10: Page *intruder_c*

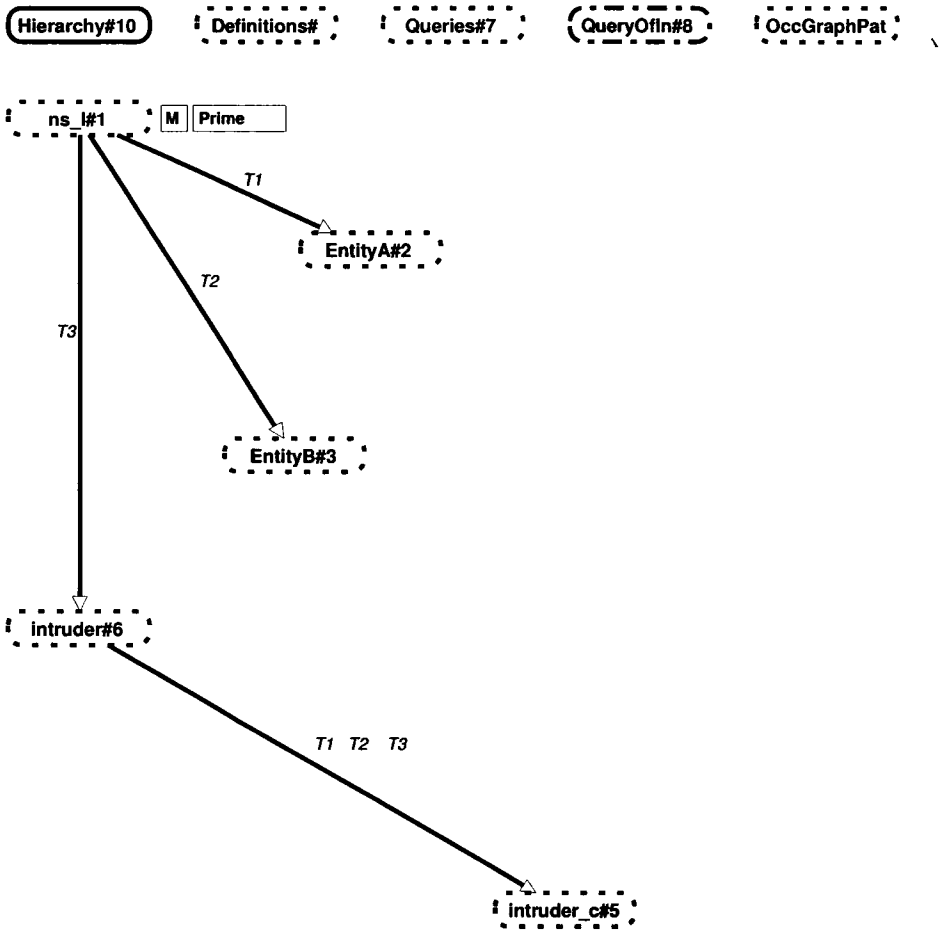


Figure B.11: The hierarchy page of the NSPK model with an intruder

The following CPN ML function returns all nodes of the occurrence graph where this requirement is violated:

```
fun AuthViolation1():Node list
= PredAllNodes(fn n =>
cf(B,Mark.EntityB'P12 1 n) > 0
orelse
cf(In,Mark.EntityB'P12 1 n)>0);
```

The full occurrence graph has 128 nodes and 127 arcs. Executing $AuthViolation1$ results in an empty list. Thus, our model of the NSPK protocol that involves A initiating a session with B does not violate the authentication requirement stated earlier.

B.3 The Model with an Intruder- A Initiates a Session with I

In this section, we model a setting where A initiates a session with I. Thus, A aims to mutually authenticate I.

In $EntityA$, place $P1$ contains the token whose colour represents the agent to be authenticated. Thus, by using the token In instead of B in Figure B.7, A initiates a session with I.

In this setting, A should not be accepted by B as an authenticating agent. Thus, one of the requirements of this CPN model is that a token with colour A should never reach place $P12$ in $EntityB$. The following CPN ML function returns all nodes of the occurrence graph where this requirement is violated.

```
fun AuthViolation2():Node list
= PredAllNodes(fn n=>
cf(A,Mark.EntityB'P12 1 n) > 0);
```

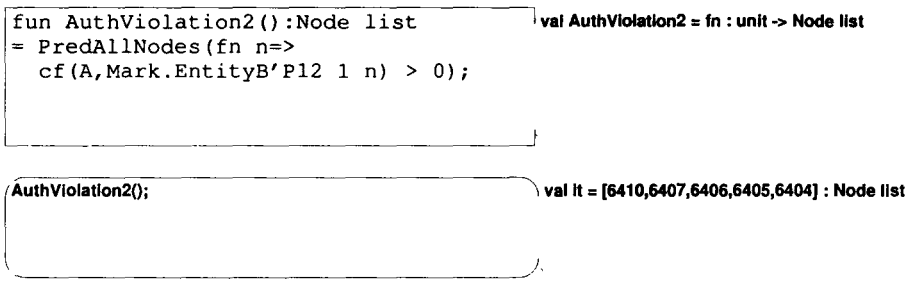


Figure B.12: The result of executing AuthViolation2

The resulting occurrence graph has 6410 nodes and 6685 arcs. Figure B.12 shows the result of executing AuthViolation2. One of the nodes returned by AuthViolation2 is node 6410. Figure B.13 shows a path of the occurrence graph from the initial marking to this insecure marking. The corresponding occurrence sequence is as follows:

EntityA		T1	$n = Na, i = In$
intruder_c	1	T1	$c = (A, Na, Ki)$
intruder_c	1	T2	$t1 = A, t2 = Na, k = Ki$
intruder_c	1	T3	$t1 = A, t2 = Na, k = Kbp$
EntityB		T1	$i = A, n = Na, k = Kbp$
EntityB		T2	$n1 = Na, n2 = Nb, i = A$
intruder_c	2	T1	$c = (Na, Nb, Kap)$
intruder_c	2	T4	$c = (Na, Nb, Kap)$
EntityA		T2	$n1 = Na, n2 = Nb, k1 = Kap$
EntityA		T3	$n1 = Na, n2 = Nb, i = In$
intruder_c	3	T1	$c = (Nb, X, Ki)$
intruder_c	2	T2	$t1 = Nb, t2 = X, k = Ki$
intruder_c	3	T2	$t1 = Nb, t2 = X, k = Ki$
intruder_c	3	T3	$t1 = Nb, t2 = X, k = Kbp$
EntityB		T3	$n = Nb, i = A, k = Kbp$

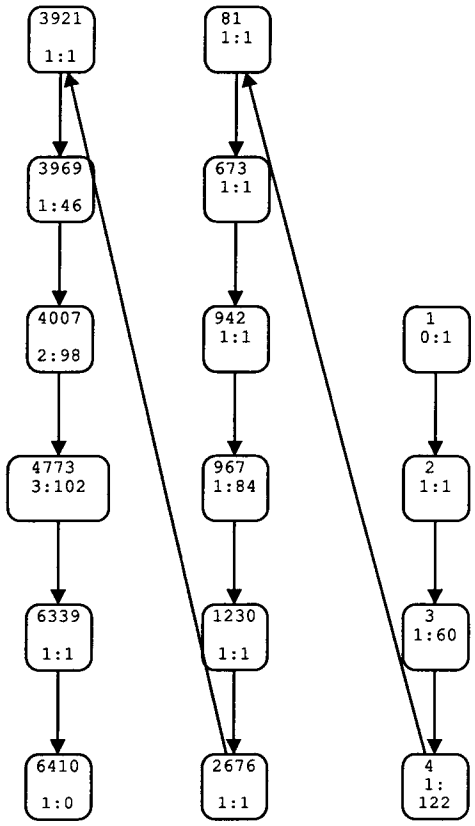


Figure B.13: A path from the initial marking to the insecure marking 6410

Note the reachability of a token A to place $P12$ in the last step. This attack is stated in a high level description as follows:

1.($I1$) $Send(A, I, E(ID(A) \oplus N_A, K_I))$

2.($II1$) $Send(I, B, E(ID(A) \oplus N_A, K_B^+)), where\ PosesAs(I, A)$

3.($II2$) $Send(B, A, E(N_A \oplus N_B, K_A^+)), where\ PosesAs(I, A)$

4.($I2$) $Send(I, A, E(N_A \oplus N_B, K_A^+))$

5.($I3$) $Send(A, I, E(N_B, K_I))$

6.($II3$) $Send(I, B, E(N_B, K_B^+)), where\ PosesAs(I, A)$

Bibliography

[AD98] H. Alla and R. David. A modeling and analysis tool for discrete event systems – continuous Petri net. *Performance Evaluation*, 33(3):175–199, 1998.

[Aur95] T. Aura. Modelling the Needham-Schröder authentication protocol with high level Petri nets. Technical Report B14, Helsinki University of Technology, Digital Systems Laboratory, Otaniemi, Finland, September 1995. http://research.microsoft.com/users/tuomaura/Publications/HUT-TCS-B14.pdf [20/02/2004].

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[Bas97] A. Basyouni. Analysis of wireless cryptographic protocols. Master's thesis, Queen's University, Kingston, Ontario, 1997.

[BBD01] D.D. Burdescu, M. Brezovan, and D.B.Marghitu. High level Petri nets and rule based systems for discrete event system modelling. *International Journal of Smart Engineering System Design*, 3(2):91–97, 2001.

[BGH+91] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. *Lecture Notes in Computer Science*, 576:44–60, 1991.

[BM94] C. Boyd and W. Mao. On a limitation of BAN logic. *Lecture Notes in Computer Science*, 765:240–247, 1994.

[BT89] N. Behki and S.E. Tavares. An integrated approach to protocol design. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 244–248, Victoria, BC, Canada, 1989.

[BT97] 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.

[CJ96] J. Clark and J. Jacob. Attacking authentication protocols. *High Integrity Systems*, 1(5):465–474, August 1996.

[CJ97] 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 [17/02/2004].

[CMAFE03] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks (extended abstract). In *Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, April 2003. http://www.ub.utwente.nl/webdocs/ctit/1/000000b2.pdf [16/02/2004].

[Col04a] Coloured Petri Nets at the University of Aarhus. Examples of Design/CPN models, 2004. http://www.daimi.au.dk/designCPN/exam/ [20/ 02/2004].

[Col04b] Coloured Petri Nets at the University of Aarhus. Examples of industrial use of CP-nets, 2004. http://www.daimi.au.dk/CPnets/intro/example_indu.html [20/02/2004].

[Col04c] Coloured Petri Nets at the University of Aarhus. History, 2004. http://www.daimi.au.dk/designCPN/history.html [28/09/2004].

[Col04d] Coloured Petri Nets at the University of Aarhus. Occurrence graphs with equivalence classes, 2004. http://www.daimi.au.dk/designCPN/libs/ogequiv/ [21/10/2004].

[Col04e] Coloured Petri Nets at the University of Aarhus. What is Design/CPN?, 2004. http://www.daimi.au.dk/designCPN/what.html [28/09/2004].

[CPN04] CPN Group at the University of Aarhus. Main page, 2004. http://www.daimi.au.dk/CPnets/cpngroup.html [28/09/2004].

[CW01] F. Crazzolara and G. Winskel. Petri nets in cryptographic protocols. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*, San Francisco, CA, April 2001.

[DJ02] Z. Ding and C. Jiang. Temporal Petri nets model of concurrent systems. *Computer Systems Science and Engineering*, 17(6):353–358, November 2002.

[Doy96] E. Doyle. Automated security analysis of cryptographic protocols using coloured Petri net specifications. Master's thesis, Queen's University, Kingston, Ontario, 1996.

[DTM95] E. Doyle, S. E. Tavares, and H. Meijer. Automated security analysis of cryptographic protocols using colored Petri net specifications. In *Workshop on Selected Areas in Cryptography, SAC '95 Workshop Record*, pages 35–48, May 1995.

[Edw98] K. Edwards. Cryptographic protocol specification and analysis using coloured Petri nets and Java. Master's thesis, Queen's University, Kingston, Ontario, Canada, 1998.

[ETM98] K. Edwards, S. Tavares, and H. Meijer. A Java tool for specification and analysis of cryptographic protocols using coloured Petri nets. In *19th Biennial Symposium on Communications*, pages 403–407, Queen's University, Kingston, Ontario, May 1998.

[Fid01] C.J. Fidge. A survey of verification techniques for security protocols. Technical Report 01-22, Software Verification Research Centre, The University of Queensland, July 2001.

[FKK96] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. Technical Report 3.02, The Internet Engineering Task Force (IETF), 1996. http://wp.netscape.com/eng/ssl3/draft302.txt [16/02/2004].

[Fus87] H. Fuss. Numerical simulations with place/transactor nets. In Voss, K., Genrich, H.J., and Rozenberg,G., editors, *Concurrency and Nets - Advances in Petri Nets*, pages 187–199. Springer-Verlag, Berlin, 1987.

[GL81] H.J. Genrich and K. Lautenbach. System modeling with high-level Petri nets. *Theoretical Computer Science*, 13(1):109–136, 1981.

[GNY90] 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.

[GV03] C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications.* Springer-Verlag, 2003.

[HC98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Technical Report 2409, The Internet Engineering Task Force (IETF), 1998. http://www.ietf.org/rfc/rfc2409.txt [16/02/2004].

[Hel78] M. Hellman. An overview of public key cryptography. *IEEE Communications Society Magazine*, 16(6):24–32, November 1978.

[Hel98] 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. http://www.tml.hut.fi/Opinnot/Tik-110.501/1998/papers/13finitestate/finitestate.htm.

[HG99] J. Thayer J. Herzog and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[HLS00] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society, Los Alamitos, CA, USA, July 2000.

[ISO89] *ISO - 8807 - Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, 1989. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16258 [06/10/2004].

[Jen81] K. Jensen. Coloured Petri nets and the Invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[Jen96a] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts. Springer-Verlag, 2nd edition, 1996.

[Jen96b] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 2: Analysis Methods. Springer-Verlag, 2nd edition, 1996.

[Jen96c] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 3: Practical Use. Springer-Verlag, 2nd edition, 1996.

[Jen97] 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.

[Kah97] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Simon & Schuster Inc, 1997.

[KCJ98] 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.

[Kem89] R. Kemmerer. Analzying encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[KV98] L. M. Kristensen and A. Valmari. Finding stubborn sets of coloured Petri nets without unfolding. *Lecture Notes In Computer Science*, 1420:104–123, 1998.

[Lab95] RSA Laboratories. *Answers to Frequently Asked Questions about Today's Cryptography, Version 3.0*, 1995. http://lib.ua.ac.be/ibw/PDF/lab_faq.pdf [16/02/2004].

[LFK91] S. Leu, E. Fernandez, and T. Khoshgoftaar. Fault-tolerant software reliability modeling using Petri nets. *Microelectronics and Reliability*, 31(4):645–667, 1991.

[LL97] G. Lee and J. Lee. Petri net based models for specification and analysis of cryptographic protocols. *Journal of Systems and Software*, 37:141–159, 1997.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, February 1995.

[Low02] G. Lowe. Analyzing protocols subject to guessing attacks. In *Workshop on Issues in the Theory of Security (WITS'02)*, January 2002. http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Papers/guessing.ps [17/02/2004].

[LR97] 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.

[MB93] 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.

[Mea94] 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.

[Mea96] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, Feb 1996.

[Met93a] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*, Version 2.0, 1993. http://www.daimi.au.dk/designCPN/man/Reference/Reference.All.pdf [29/09/2004].

[Met93b] Meta Software Corporation. *Design/CPN Tutorial for X-Windows*, Version 2.0, 1993. http://www.daimi.au.dk/designCPN/man/Tutorial/ Tutorial.All.pdf [29/09/2004].

[Met96] Meta Software Corporation. *Design/CPN Occurrence Graph Manual*, Version 3.0, 1996. http://www.daimi.au.dk/designCPN/man/Misc/ OccGraph.All.pdf [29/09/2004].

[Met04] Meta Software Corporation. Main page, 2004. http://www.metasoftware.com/ [28/09/2004].

[MF85] G. Memmi and A. Finkel. An introduction to FIFO nets - Monogeneous nets: A subclass of FIFO nets. *Theoretical Computer Science*, 35(2–3):191–214, February 1985.

[MN90] M. Ajmone Marsan and F. Neri. Modelling and analysis of telecommunication protocols using Petri nets. In *Modelling the Innovation, Communications, Automation,and Information Systems. Proceedings of the IFIP TC 7 Conference, 1990, Rome, Italy*, pages 9–20. North-Holland, 1990.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[Nie92] B. Nieh. Modelling and analysis of cryptographic protocols using Petri

net formalism. Master's thesis, Queen's University, Kingston, Ontario, Canada, 1992.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[NS93] B. Neuman and S. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, April 1993.

[NT92] B. Nieh and S. Tavares. Modelling and analyzing cryptographic protocols using Petri net. In *Advances in Cryptology-ASIACRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 275–295. Springer, 1992.

[NT94] B. Clifford Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[Pat97] S. Patel. Weaknesses of North American wireless authentication protocol. In *IEEE Personal Communications Magazine*, pages 40–44, June 1997.

[Pau96] L. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[PC92] Y. Papelis and T. Casavant. Specification and analysis of parallel/distributed software and systems by Petri nets with transition

enabling functions. *IEEE Transactions on Software Engineering*, 18(3):252–261, March 1992.

[Pet04] Petri Nets World. A classification of Petri nets, 2004. http://www.daimi.au.dk/PetriNets/classification/ [20/02/2004].

[RH93] 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 [17/02/2004].

[Ros04] G. Rose. Authentication and security in mobile phones, 2004. http://www.qualcomm.com.au/PublicationsDocs/AUUG99AuthSec.pdf [16/02/2004].

[RPD95] A. El Rhalibi, F. Prunet, and C. Durante. From modelling using function charts for control systems to analysis using Petri nets. In *MASCOTS '95, Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems, January 10-18, 1995, Durham, North Carolina, USA*. IEEE Computer Society, 1995.

[RS01] P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach.* Addison-Wesley, 2001.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algoritms, and Source Code in C.* John Wiley, 2nd edition, 1996.

[Sec04a] Security Protocols Open Repository, 2004. http://www.lsv.ens-cachan.fr/ spore/index.html [17/02/2004].

[Sec04b] Security Protocols Open Repository. TMN protocol, 2004. http://www.lsv.ens-cachan.fr/spore/tmn.html#attack4 [17/02/2004].

[Sha99] Y. Shao. Specification and analysis of Internet cryptographic protocols using a Petri net modeler. Master's thesis, Queen's University, Kingston, Ontario, Canada, 1999.

[Sin99] S. Singh. *The Code Book: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography.* Doubleday, 1st edition, 1999.

[Sta94] D. M. Stal. Backward state analysis of cryptographic protocols using coloured Petri nets. Master's thesis, Queen's University, Kingston, Ontario, Canada, 1994.

[Sti02] D. Stinson. *Cryptography: Theory and Practice.* CRC Press, 2nd edition, 2002.

[STM94] 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.

[Syv94] P. Syverson. A taxonomy of replay attacks. In *Computer Security Foundations Workshop VII.* IEEE Computer Society Press, 1994.

[The04a] The Computer Security Division (CSD). Cryptographic Toolkit, 2004. http://csrc.nist.gov/CryptoToolkit/index.html [28/04/2004].

[The04b] The Computer Security Division (CSD). Mission, 2004. http://csrc.nist.gov/mission.html [28/04/2004].

[TMN90] M. Tatebayashi, N. Matsuzaki, and D. Newman. Key distribution protocol for digital mobile communication systems. *Advances in Cryptology–CRYPTO'89 Proceedings, Lecture Notes in Computer Science*, 435:324–334, 1990.

[TZ97] S. E. Tavares and W. Zhao. An analysis of MSAT security protocols using coloured Petri nets. Technical report, Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, 1997.

[US 99] US Department of of Commerce/National Institute of Standards and Technology. *Data Encryption Standard(DES)*, 1999. Federal Information Processing Standards Publication (FIPS PUB) 46-3.

[Var90] V. Varadharajan. Petri net based modelling of information flow security requirements. In *Proceedings of the Computer Security Foundations Workshop III, 1990, Franconia, NH, USA*, pages 51–61, Piscataway, NJ, USA, 1990. IEEE Service Center.

[Var91] 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.

[WH94] H. Wabnig and G. Haring. Petri net performance models of parallel systems-methodology and case study. In *PARLE '94, Proceedings of the Sixth International Parallel Architectures and Languages,July 4-8, 1994, Athens, Greece*, volume 817 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 1994.

[Zha97] W. Zhao. Efficient analysis of cryptographic protocols in wireless communication systems. Master's thesis, Queen's University, Kingston, Ontario, 1997.

[Zub98] W. Zuberek. Timed Petri nets and performance evaluation of systems. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC'98), 11-14 October 1998, San Diego, USA*, pages 278–283, 1998.

Index

Aarhus University, 42
Acceptance Check Step (ACS), 52
arcs, 23
    weight, 23
atomic fields, 61
authentication and key exchange protocols, 8

BAN logic, 15, 16
binding element, 34
    enabled, 35

cipher, 3
ciphertext, 2
colour set, 33
CP-nets, 29
CPN, 29
CPN formal definition, 33
CPN group, 42
CPN ML, 29
CPN Simulator, 29
CPTN-analyzer, 58
CPTN-language, 58
cryptanalysis, 2
cryptanalyst, 2
cryptographic algorithm, 3
cryptographic algorithms
    public key, 3
    symmetric, 3
cryptographic protocol, 4
    flaws, 5
cryptographic protocols
    IKE, 8
    Kerberos, 8
    Needham-Shroeder, 8
    Newman-Stubblebine, 8
    Otway-Rees, 8
    SSL,(Secure Socket Layer Protocol), 8
    TMN, 8
    Wide-Mouth Frog, 8
    Yahalom, 8
Cryptographic Timed Petri Net(CPTN), 58
cryptography, 2
CSD, 4
CSP algebra, 16

Data Encryption Standard, DES, 3

deadlock, 24
decryption, 2
Design/CPN, 29
    CPN Editor, 29
direct subpage, 40
direct superpage, 40
directly reachable markings, 36
dynamic behaviour, 23

encryption, 2

finite occurrence sequence, 36
formal methods, 16
    methods based on algebra, 16
    methods based on logic, 16
    methods based on state machines, 16
fusion set, 40
    global, 40
    instance, 41
    page, 41

GNY logic, 16

hierarchical net inscriptions box, 39

Inajo, 16
incidence matrix equation, 52
infinite occurrence sequence, 36
inhibitor arc, 51
initial marking, 23
input port, 39
Inscriptions, 27
intruder, 9
    actions, 9
    attack strategies, 10
      algebraic, 11
      forging of keys, 10
      guessing, 11
      interleave , 10
      man-in-the-middle, 10
      oracle, 10
      replay, 10
      type, 11

LOTOS, 47

MAO logic, 16
marking, 23
Matrix Analysis Step (MAS), 52
message sequence diagram, 6
Meta Software Corporation, 42
ML, 29
module, 47
multi-set, 33

Needham-Schroeder Public Key (NSPK), 14
Needham-Schroeder Secret Key protocol(NSSK), 5
NIST, 4

nonce, 6
NRL Analyzer, 16

occurrence graph, 26
Occurrence Graph Tool, 29
OG queries, 43
OG tool
    branching criteria, 43
    stop options, 43
open addressing scheme, 9
output port, 39

page, 36
page hierarchy, 39
page instance, 38
Performance Tool, 29
Petri net, 22
    definition, 23
Petri Net Objects(PNO), 47
Petri nets
    coloured, 27
    formal definition, 23
    high level, 27
    predicate-transition, 27
Petri nets Jensen, 29
places, 23
plaintext, 2
port, 39
port assignment, 39
prime page, 40
PROD, 56
Prolog, 51

reachability tree, 26
reachable marking, 36
RSA, 3
rule of transition enabling and firing, 24

security classes, 46
security policy, 46
security properties, 7
    authentication, 7
    confidentiality, 7
    entity authentication, 15
    integrity, 7
    non-repudiation, 8
    secrecy, 7
simulation
    automatic, 42
    interactive, 42
socket, 39
standards, 4
state, 23
state invariants, 26
steganography, 2
step, 35
    enabled, 35
    occurrence, 36
substitution transition, 38
super page, 38

TCP/IP, 9
TMN protocol, 11
token, 23
    colours, 27
    types, 27
transformation rules, 26
transition
    enabled, 24, 35
    firing, 24
transitions, 23

Varadharajan, 46
vulnerable data set (VDS), 52