國立交通大學

資訊科學系

碩 士 論 文

安全規約分析系統

A System for Deriving Vulnerabilities of Security Protocols

研 究 生：蔡柏良

指導教授：楊　武　教授

中 華 民 國　九 十 四　年 六 月

安全規約分析系統

A System for Deriving Vulnerabilities of Security Protocols

研 究 生：蔡柏良　　Student：Po-Lian Tsai

指導教授：楊　武　　Advisor：Wuu Yang

國 立 交 通 大 學

資 訊 科 學 系

碩 士 論 文

A Thesis

Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

安全規約分析系統

學生：蔡柏良　　指導教授：楊 武 博士

國立交通大學資訊科學研究所

摘 要

由於在目前的電腦網路架構下，網路使用者並沒有一個好的機制可以認證對方的身分，因此一個安全認證規約就不可或缺，所以它的安全性和正確性就一直受到注意，在過去的文獻中，許許多多分析的方法都被提出，而其中以邏輯系統分析的方式比較完整且比較正式，但是這些邏輯系統不是實作上較困難就是需要非常有經驗的人才可以了解，因此我們在本論文中，提出一個可以有效找出一個安全認證規約弱點的系統，且實作上是非常容易的，我們將會介紹如何實作所提出的安全規約分析系統在 Prolog 的環境下，並且提出分析的方法，而這個安全規約分析系統將對發展一個安全規約的設計者有很大的幫助。

A System for Discovering Vulnerabilities of Security Protocols

Student: Po-Lian Tsai **Advisors: Dr. Wuu Yang**

Department of Computer and Information Science

National Chiao Tung University

Abstract

A security protocol plays an important role in network environment in that under current network environment, no one can make sure whom he is “talking” to is whom he was intending to talk to. The soundness of a security protocol has been drawing many spotlights since past years. Many logic systems have been proposed in plenty of papers. However, they are either hard to implement or need lots of experience. We propose an efficient system for inspecting the correctness of security protocols and its implementation, which we adopt prolog language as our derivation engine. Then we suggest detection methods to reveal the potential vulnerabilities of a security protocol. We believe that the proposed system will be helpful in testing a security protocol.

誌謝

首先，我要感謝我的指導教授，楊武博士，在他細心的教導下，才會有這篇論文的誕生，真的真的感謝我的指導教授，感謝他仔細閱讀每次我所修改的初稿，感謝他每次給我寶貴意見，感謝他對我的信任，感謝他給我一個美好的研究生活，謝謝您，老師。

接著，我要謝謝實驗室的學長，同學和學弟們，在這一段時間裡，謝謝你們的照顧和指導，還有同進退的室友們，在這二年中，陪伴在我身邊最久的室友們，一起歡笑，一起度過難關，你們將是我這二年中最難忘的朋友，再見了，我的朋友。

最後，我要感謝我所愛的家人，我父親，母親和弟弟，你們是我最大的精神支柱，沒有你們，我無法走下去，我只想說，我愛你們。

目錄

圖目錄

第一章 序言

在此章節中，我們首先介紹我們的研究動機爲何，然後說明本篇論文的目標爲何，在最後一節中，我們描述此論文的架構。

1.1 研究動機

在現今的社會中，電腦網路的發明不僅帶來生活上的方便，對各產業行爲模式上的改變也是不可忽視的，甚至產生新興行業，如 E-Commerce，但電腦網路的世界是數位虛擬的世界，也就是說，在這樣一個虛擬世界中，人與人的溝通不像現實世界中以面對面的模式出現，然而這也是電腦網路的特性之一，再說，任何在網路世界裡傳送的資料都可以任意被截取或複製，且在網路規約協定裡，無法明確地辨識訊息發送者的特性使任何人都可以輕易地僞造訊息欺騙它人，因此如何在虛擬網路世界中認證彼此的分身已成爲現今不可或缺的。一個認證規約是提供認證機制，當一個認證規約不安全時，它可能被蓄意欺騙它人的組織拿來當作欺騙工具，因此造成反效果，所以一個認證規約的安全性是非常重要的。

1.2 研究目標

在過去幾年內，很多人花了很多時間設計出很多認證規約，可是大多數在事後被證實出有弱點存在，改進的版本也是如此，而這些出現在認證規約裡的弱點通常是因爲設計者不知道一個安全認證規約應有的要素爲何，因此就有人提出如何設計一個安全認證規約或一個安全認證規約應有的特性，但是他們所提的方法通常只對有經驗的人才有幫助；相反地，有人就提出安全認證規約的分析方法，這些方法大多以邏輯系統的方式呈現，如 BAN logic[1]和 Floyd-Hoare style logic [10, 17]，利用邏輯分析推導認證規約的安全性，但這樣一個邏輯系統推導過程需要人的介入，且對一個沒有經驗的人就沒辦法正確地定義所提出認證規約的邏輯假設，並且在[13]中，Nessett 指出 BAN logic 能力的限制，BAN logic 並不是適用於每個認證規約，如 Neumann 和 Stubblebine 在[14]中所提出的認證規約就以 BAN logic 證明所提的認證規約是安全的，但它卻有弱點存在，Nessett 也指出 BAN logic 並不是一個認證規約，而是一個 Key distributed authentication protocol 的邏輯系統，所以他就用安全規約來總稱認證規約和提供其它服務的規約 (如金

鑰建立規約)，因此我們就用安全規約來總稱任何規約。如果要用程式設計出像 BAN logic 那樣的系統，是不容易的，因爲那樣邏輯系統都以抽象爲主，並且如果使用者邏輯假設錯誤，就會使分析錯誤，因此我們的目標是設計出一個可以在電腦跑的一個安全規約分析系統。

我們知道許許多多的安全規約被發現有弱點存在，然而當我們不考慮那些以密碼學角度分析時（如 Chiphertext only, Known plaintext, Chosen plaintext, Chosen chiphertext and Chosen text attacks)，一般攻擊的分類有 Man in the middle attack，Parallel session attack， Type flaw attack，Duplicate session attack 和 Reflection attack 等等，我們仔細的想想，在這些攻擊的過程中，所有攻擊者傳給所要攻擊對象的訊息都是符合攻擊對象在一個規約中所期待的一連串訊息，而我們也知道當要認證一個人時，我們需要他提出一個證據是我們相信只有他所宣稱的人才擁有的，例如某甲向某乙說："我是甲"，乙就說："那你可以提出一個我相信只有甲知道的事或物嗎？"，這樣的事或物可能是一種暗號或證件，而在一個安全規約中就以安全金鑰的擁有性來表示一個人的身分，如私人的加密金鑰，因此認證一個人就可能用雙方之前就建立的安全金鑰(如暗號)，或者是藉由雙方都信任的第三者(如認證機關)，所以一個安全規約通常都在交換訊息中用這樣一個安全金鑰來加密某些訊息來表示自己爲所宣稱的人。在上面所提的攻擊分類中，一個攻擊者並不知道任何安全金鑰除了自己的安全金鑰以外，如果一個安全規約有弱點，那表示攻擊者就可能需要用所宣稱那個人的安全金鑰來欺騙攻擊對象，如何做到呢？其實仔細看看上述每一個攻擊分類的實例，不難發現當需要僞造攻擊者自己沒辦法產生的加密訊息時，其實都是用之前收到相似的訊息或執行另一次規約所得到相似的訊息，而這是規約設計者沒有預料到的事(訊息相似度是以接收者的角度看兩個訊息格式是否一樣)，再進一步地想想看，那些攻擊者自己沒辦法產生的加密訊息其實都是經過規約中的某些步驟得到的，而這些步驟就是說明在規約中，當經過某些訊息傳送後，就會產生相對應的加密訊息，因此我們可以歸納出一個特性：攻擊的方式通常利用規約中相似的訊息相互取代或利用他人(包括攻擊對象本身)產生想要的相似訊息。相似訊息的取代通常是利用比較“容易”產生的相似訊息取代“不容易”產生的相似訊息(容易或不容易是以攻擊者所擁有的資訊決定，也就是是否知道加密的金鑰)，除此之外，相似訊息也可以藉由其它 runs 中得到的(一個 run 表示執行一次安全規約)。在此我們先定義何爲相似訊息：

兩個訊息是否爲相似訊息是必需以接收者的觀點來看，也就是說，根據接收者目前所擁有的資訊，當這兩個訊息都能符合接收者所預期要收到的訊息時，兩個訊息爲相似訊息。

在定義中，“接收者所預期”是指當接收者在規約中的某一步驟時，他期待收到一個符合規約中規定的訊息出現，而此訊息可能要有幾個元素或在某一個元素是必需是接收者之前產生的 Nonce 等等。接著我們把剛才所提的特性稱爲“相似訊息攻擊方式”，其定義如下：

如果一個安全規約存在一個弱點，而在此弱點的攻擊方式中所有攻擊者沒辦法自行產生的加密訊息(使用攻擊者不知道的金鑰加密之加密訊息)都是從規約訊息交換中的相似訊息取得(可能爲之前的相似訊息，或是執行其它 runs 時得到的相似訊息)，我們稱這樣的攻擊方式爲相似訊息攻擊方式。

所以我們的安全規約分析系統主要是建立一個安全規約中加密訊息的產生方式，然後分析者可藉由這個系統詢問是否可得到一個想要的加密訊息(如當攻擊時，攻擊者沒辦法自行產生的加密訊息)，如果系統回答 true 的話且發現產生此加密訊息並不是如規約中所規定的步驟得到，這表示此加密訊息可藉由其它步驟獲得，因此並不如設計者所預期的，所以就可能有弱點存在，進而分析，就能推導出攻擊的步驟，在 4.1 中，我們會紹介如何找出一個相似訊息攻擊。

1.3 論文架構

在第二章中，我們將介紹從觀察安全規約中所得到的結果，然後推導出我們的安全規約分析系統，並且介紹系統的架構，接著用一個範例說明，最後討論所提的系統能力範圍爲何。在第三章中，我們說明如何在 Prolog 的環境中實作所提出的安全規約分析系統，在此之前我們也簡單地介紹 Prolog Logic Language。第四章中，我們用三個安全規約測試所提出的安全規約分析系統，進而分析系統的能力，最後，我們在第五章做個結論和提出日後的目標。

第二章 安全規約分析系統設計

本章中，我們將介紹如何設計一個安全規約的安全分析系統，在提出我們的安全規約分析系統前，我們先討論我們所看到的安全規約和網路特性之關係，進而推導出四個觀察到的結論，經由這些結論，我們導出一個安全規約分析系統，因爲此安全分析系統會將安全規約轉換成 Prolog language，再藉由電腦高速處理的能力，我們就可以快速找出一個安全規約的弱點。

一個安全規約可以用文字說明，但通常都會附圖加以補充，或只有圖形描述，不管是文字或圖形，都以訊息交換的方式說明一個安全規約的程序，如 A -> B: M 表示參與者 A 送出訊息 M 給參與者 B，我們稱這樣一個動作爲一個 communication action，而一個安全規約是由一連串的 communication actions 構成的，訊息 M 可以有多個訊息存在，如圖 2.1，發送者 A 傳送給接收者 B 一個加密訊息{A, B, {A, B, Na}Kbs}Kas 和一個未加密訊息 A。圖 2.2 圖是 Neumann 和 Stubblebine[14]所提出的一個安全規約，它的設計目的在於雙方的認證(利用 Na 和 Nb 當作 challenges)和建立一把安全金鑰(Kab)，在本論文中，我們將用這個安全規約當作我們的範例。依慣例，我們定義一個 run 爲執行一次安全規約，一個 communication action 爲一個訊息交換動作，如圖 2.2 就有四個 communication actions。

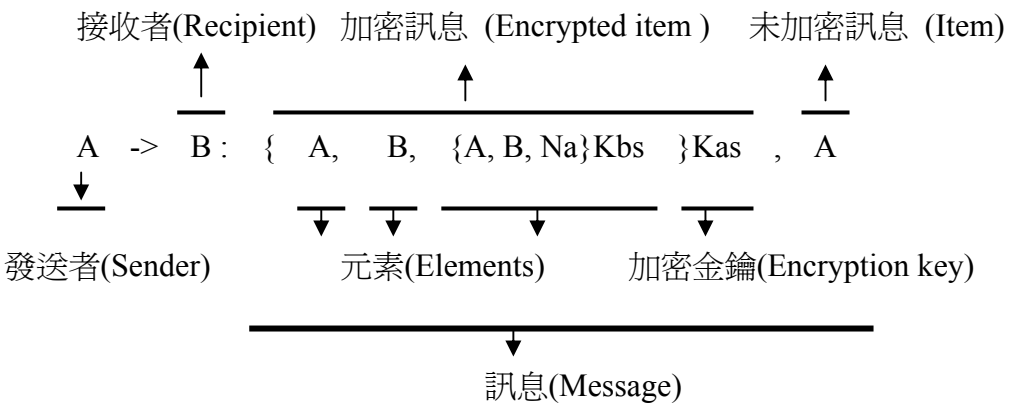


圖 2.1 Message notation

PROTOCOL:

A->B : A, Na

B->S : B, { A, Na, Tb }Kbs, Nb

S->A : { B, Na, Kab, Tb }Kas , { A, Kab, Tb }Kbs, Nb

A->B : { A, Kab, Tb }Kbs, {Nb}Kab

圖 2.2 Neumann Stubblebine security protocol

2.1 安全規約的觀察

在目前的 TCP/IP 架構下，一個封包的 source IP 是說明這個封包的來源，當這個封包在網路上傳送時，任何經過的伺服器都有能力更改 source IP 的欄位，並且任何一個人都可以發送一個不屬於自己 source IP 的封包給任何人，也就是說，一個封包的 source IP 並沒有任何保護的機制，因此對於接收到封包的人而言，就不能完全依賴 source IP 的欄位去認定封包的來源，而在安全規約的描述中，並無這樣的考慮，但是在分析時，我們就必需以真實環境的特性去驗證規約的安全性。一個安全規約的描述通常如圖 2.2，圖 2.2 是 Neumann 和 Stubblebine 在[14]所提出的一個安全規約，他們用簡潔地方式說明一下這個安全規約如何運作和一些事前的必要條件，很多提出安全規約的人也是用相同的方式，但實際上這樣的描述方式是不足夠的。下文中我們可以看到爲什麼那樣的敘述方式是有問題的，並提出四個觀察。

如我們所知，在電腦網路的世界和真實的世界是不一樣的，當程式設計師在撰寫一個安全規約的程式時，他所需要的資訊不只是每一個步驟的作法，還需要更仔細的說明，例如訊息之間的關係，以圖 2.2 爲例，圖中的描述如下，當 A 和 B 要建立一個安全通訊管道時，A 送出他的 Identity 和一個 Nonce，當 B 收到這樣一個訊息時，他根據規約中的規定送出相對應的訊息給一台信任的伺服器 S，當每個步驟都正確執行完時，A 就認定 B 是他一開始所想要建立連線的人，並且雙方共有一把安全金鑰，對 B 而言，他也有相同的認知。不過，實際上，這樣的一個規約只有說明在那一步時要傳送什麼訊息的行爲，而收到訊息時和送出相對應訊息之前所必需完成的動作有什麼，這樣的動作並未很明確地表現在規約裡，甚至收到訊息和之前收到的訊息是否有需要檢驗的動作也沒有詳盡的說明，例如說，時間郵戳的檢查，當 B 在第四步時所收到的加密訊息中的時間郵戳 Tb，

是否要和在第二步時所產生的時間郵戳一致？如果 B 忽略了這個檢查的重要性，下列情況將會發生 (S 是一台信任的伺服器，Z 是攻擊者)。

(i1) A->B: A, Na

(i2) B->S: B, {A, Na, Tb}Kbs, Nb

(i3) S->(A)Z: {B,Na,Kab,Tb}Kas , {A,Kab,Tb}Kbs, Nb #Z intercepts this message

(ii1) (A)Z->B: A, Na #Z pretends A

(ii2) B->S: B, {A, Na, Tb'}Kbs, Nb'

(ii3) S->(A)Z: {B,Na,Kab',Tb'}Kas , {A,Kab',Tb'}Kbs, Nb' #Z intercepts this message

(i3) (S)Z->A: {B,Na,Kab',Tb'}Kas , {A,Kab',Tb'}Kbs, Nb

(i4) A->B: {A,Kab',Tb'}Kbs, {Nb}Kab'

這樣的攻擊方法似乎並沒有違反當初 Neumann 和 Stubblebine 設計此安全規約的目的，雙方彼此認證，也建立了一把安全金鑰，然而，雙方所建立的安全金鑰是 Kab'，但伺服器在建立這把安全金鑰時，是要給 B 和攻擊者 Z 之間的安全通訊使用，如今成爲 A 和 B 共有的一把金鑰，或許有人會爭議說，這樣的攻擊方式對攻擊者 Z 也沒有利益，而且 A 和 B 也建立了一把安全金鑰，但仔細想想，這個攻擊方式的確違反了設計者所期望一個成功的 run 所必需的條件，所以對一位程式設計師而言，這樣未在規約中呈現的隱喻行爲必需被明朗化，對一個安全規約分析師也是如此。

觀察一：一個安全規約隱含著未被明確說明的行爲。

一個安全規約只規範如何完成認證的行爲和訊息中所要的資訊，因此在規約中的符號相當於電腦程式中的變數一樣。如圖 2.2 中, A，B，Na，Tb，Nb 和 Kab 都是變數，簡單地分析一下 Na 和 Nb，Na 是 A 送出的 Challenge，而 Nb 是 B 送出的 Challenge，個別只對產生他們的 A 和 B 有意義，因此 Na 和 Nb 並無任何的相關性存在，也就是說，Na 和 Nb 可以爲相同的值，但並不是每一個在規約中的符號都可以爲任意值，首先，A 和 B 可以是任何人，但不可以是同一個人，理由是非常直覺的，A 不會相信攻擊者聲稱自已是 A 而要和 A 認證，可是在範

例中，S 是必需要和 A 與 B 個別有一把安全金鑰，如 Kas 和 Kbs，並且 Kas 只有 S 和 A 共有，Kbs 只有 S 和 B 知道，因此 S 必需是一個被 A 和 B 所信任的伺服器。如果 Na 和 Kab 碰巧爲相同值時，那下面列的情況也就有可能發生，而 Kab 就可以從第一個訊息中得到了。

A->B: A, NK
B->S: B, {A, NK, Tb}Kbs, Nb
S->A: {B, NK, NK, Tb}Kas , {A, NK,Tb}Kbs, Nb
A->B: {A, NK, Tb}Kbs, {Nb}NK

觀察二：一個安全規約只規範如何完成認證的行爲和訊息中所要的資訊，訊息中的符號相當於電腦程式中的變數。

如我們之前所提到的，在電腦網路裡，一個訊息的傳送者並不一定就是訊息的發送者，根據這個認知，一個安全規約所描述的只是一連串訊息的收發動作，如圖 2.2 中，對 A 所描述的行爲如下，當 A 要和 B 建立安全連線時，A 送出他的 Identity 和一個 Nonce，然後等待接收訊息，這個訊息必需合乎規約的規範，等他收到一個所期望的訊息後，他便送出相對應的訊息，然後完成建立連線，此連線的資料是依據交換訊息中變數的值而定，所以在範例中，對一個發起建立連線者，如 A，所要執行的步驟如下:

步驟一：送出自己的 Identity, A 和一個 Nonce, Na
步驟二：收到 {對方的 Identity, Na, K, T}Kas ，M 和 N (K, T, M, N 爲變數)
步驟三：送出 M 和{N}K

在步驟二中，因爲發起建立連線者 A，並不知道對方和伺服器的安全金鑰 Kbs，所以對他而言，第二個訊息{A,Kab,Tb}Kbs 只是一個變數 M，而第三個訊息是 B 產生的 Nonce，所以也是一個變數 N，因此一個安全規約是以概觀的角度來描述一連串交換訊息的動作，但這樣的概觀方式和個別參與者對訊息接受和發送的觀點是有所不同的，所以當在分析一個安全規約或程式設計時，安全規約的概觀會困惑分析師和程式設計師，然而以個別參與者的觀點去分析和設計將更容易看出

個別參與者的行爲。

觀察三：一個安全規約是由數個子規約所構成的，每個子規約代表個別參與者和伺服器的行爲。

我們引用在[15]所說的一段話:

> “The spy is allowed to perform any of these actions at any time, subject only to the limitation that the messages it introduces must be feasible to produce from its current state of knowledge (initial and what it has learned since) under the rules of the cryptographic systems in use.”

這段話所描述的內容是非常直覺的，一個攻擊者在任何時間都能竊聽和中斷參與者之間或參與者和伺服器之間的連線，修改或僞造訊息，但其修改或僞造訊息必須合乎攻擊者能力範圍內；相對地，如果以觀察三的觀點切入，任何參與者所產生的訊息也是根據他現有的資訊產生出來的，換句話說，任何參與者在一個安全規約中所產生的訊息不管是加密或未加密都是有依據的，在分析一個安全規約時，我們重視的是加密訊息，原因非常明顯，一個攻擊者可以很容易地僞造一個未加密的訊息，但就不一定能僞造一個加密的訊息。

觀察四：每一個加密訊息都有一個因果關係。

在下一節中，我們將利用四個觀察結果設計出一個安全規約分析系統，此系統將會把一個安全規約轉換成 Prolog 的語言，因此我們就可以藉由電腦高速處理的能力來發現安全規約中的弱點。

2.2 系統架構

在介紹我們提出的安全規約分析系統前，我們先定義在我們系統裡何謂一個有弱點的安全規約，當一個攻擊者可以利用它欺騙它人時，或它可以產生或執行不如設計者所預期的情況出現時，我們說這個安全規約是有弱點。

從上一節中的四個觀察中，我們推導如下：從觀察三中，我們可以將一個安全規約以個別參與者和伺服器的觀點分爲多個子規約，而個別的子規約描述相對

應的參與者或伺服器的一連串收發訊息動作，因此從觀察四中，我們就得知，任何參與者或伺服器在發送任何加密訊息時，此加密訊息是依據他們原有的資訊和之前收到訊息中的資訊，再從觀察一中，我們得知此關係是隱含的，未被明確地表現在規約上，因此在分析時，我們就必需將它明確地列出 (這些隱含的關係是以發送此加密訊息者的觀點爲出發)，在我們的系統裡，我們稱這樣的關係爲 rule，最後從觀察二中，我們知道當我們建立任何加密訊息和他們個別依據資訊的關係時，此關係並不是對特定參與者才適用，也就是說，任何人只要在規約中扮演這個角色，且所收到訊息之間的關係符合要求，那就會產生相對應的加密訊息，所以我們用“rule”說明加密訊息和他們個別依據資訊的關係並不特定於任何人，任何攻擊者都可以利用這些 rules 僞造相對應的加密訊息只要他們能符合 rules 的要求。以圖 2.2 的 B 爲例，B 的收發行爲如下：

步驟一：收到 A, Na
步驟二：送出 B, {A, Na, Tb}Kbs, Nb
步驟三：收到 {A,Kab,Tb}Kbs, {Nb}Kab

當 B 在第步驟二送出{A, Na, Tb}Kbs 時，這個加密訊息中的 A 和 Na 是從步驟一得到的，因此他們的關係如下（“:-”表示依據）：

{A, Na, Tb}Kbs :- 收到 A 和 Na (r1)

但這樣的條件是不足夠的，因爲在{A, Na, Tb}Kbs 中，Tb 是 B 所產生的 Timestamp，所以 B 必需產生一個 Timestamp：

{A, Na, Tb}Kbs :- 收到 A 和 Na，並且 Tb 是 B 產生的 Timestamp (r2)

上面的描述對加密訊息{A, Na, Tb}Kbs 和他依據資訊之關係只適用特定的參與者 B，因爲 Kbs 和 B 之間的關係並未明確地顯示出來，因此正確的關係如下：

{A, Na, Tb}Kbs :- 收到 A 和 Na，並且 Tb 是 B 產生的 Timestamp, B 必須有金鑰 Kbs (r3)

上面的描述就是一個 rule，它適用於任何參與者扮演 B 的角色。所以當攻擊者 Z 需要僞造一個訊息是用 Kas 加密時，並且訊息內容只有三個欄位，第一個和第二個欄位分別爲 X 和 Y，第三個欄位可以爲任意值，因此攻擊者 Z 就可以利用這個 rule 造出所要的訊息，如下：

(X)Z->A: X, Y
A->(S)Z: A, {X, Y, Ta}Kas, Na

我們所提出安全規約分析系統的架構主要有三個原素構成，此三個元件爲 basic facts，rules 和 available predicates，首先，我們介紹何謂 basic facts，每一個安全規約在被執行前，提出規約者都有一些前提必要的假設，然後以這些前提的假設爲基礎，用正式或非正式的方法證明他們所提出的規約是安全且正確，因此我們在設計一個分析系統時，系統也要把這些前提的假設考慮進去，所以我們用 basic facts 來表示安全規約中的前提假設，表示的方式有三種：public(X)，key(A, K)和 generate(A, N)，注意在這裡 X，A，K 和 N 爲變數。public(A)是說 A 是公開的，大家都可以得到的，如每一位參與者的 Identity 都是公開的，基本上，因爲電腦網路的特性，每一個在規約中的訊息都是公開的，而加密的訊息和未加密的訊息只差別於知道加密金鑰的人才能知道加密訊息中的內容。第二個 key(A, K)是用來描述參與者和伺服器所知道的金鑰，例如參與者 A 和伺服器 S 事前有建立一把安全金鑰 Kas，在我們的系統裡，就以 key(A, Kas)和 key(S, Kas)來表示這個前提假設。最後，generate(A, N)是用來表示當在執行一個安全規約時，每一位參與者在這個 run 中所要產生的東西，如圖 2.2 中，A 要產生一個 Nonce Na，B 要產生一個 Nonce Nb 和一個 Timestamp Tb，而伺服器 S 要產生一把金鑰 Kab，我們以下列的 generate 來表示這些要求：generate(A, Na)，generate(B, Nb)，generate(B, Tb)和 generate(S, Kab)。對一個安全規約的前提假設，我們已經提出轉換的方式，接下來我們將在下一節中說明如何運用因果關係將一個安全規約中所描述的訊息交換機制轉換成系統的 rules，而將以集合 BF 表示一個安全規約的前提假設，在我們的系統裡，每一個在規約中的加密訊息都有一個相對應的 rule，每一個 rule 說明了如何產生相對應的加密訊息，當然有些 rules 會依賴其它 rules，如圖 2.2，在第三個 communication action 時，S 送出兩個加密訊息，這兩個加密訊息的 rule 都是依據上一步所收到加密訊息，在系統裡，這樣 rules 和 rules

之間的相依性我們就用 available predicates 來表示，下一節我們將對 available predicates 進一步說明。在這裡我們必要注意到一件事，如果一個安全規約規定第一個 communication action 有加密訊息，在我們的系統裡，對此加密訊息我們並無建立任何相對應的 rule,因爲我們是以一個攻擊者的角度去分析一個安全規約的弱點，對攻擊者而言，這個加密訊息只有當發起建立連線者在執行一個 run 時，才能得到的。

2.3 Rule 的建立

在說明如何建立加密訊息的 rules 前，我們有一個很自然的假設：每一位參與者和伺服器都會老老實實地根據安全規約中的 communication actions 去執行，以觀察三的結果來說，每一位參與者和伺服器都會根據自己的子規約所規定的收發順序去執行，所以說，每一位參與者和伺服器在一個 run 中，如果收到符合規約中所期望的訊息，就會往下一步執行，送出相對應的訊息，然後再次等待收到下一個符合規約中所期望的訊息出現，就這樣接收和傳送，一步一步地完成自己的子規約，最後依據交換訊息中變數的值結束。在這個假設之下，對每一個加密訊息 X 內的元素處理，以發送者的觀點去推導個別元素的來源，最後列出元素之間的隱含關係和元素與來源之間的隱含關係 (這些隱含關係是用 basic facts 的 key 和 generate 表示)，然後我們就得到一個 rule (如圖 2.3 的表示方式) 說明產生此加密訊息 X 所需要的訊息爲何以及其訊息之間必需有何關連。

maybeforged(X) :- C1, C2 … , Cn (for $1 \le i \le n$, Ci 爲一個 basic fact 或 available predicate)

圖 2.3 The format of a rule

在我們的系統裡，我們用圖 2.3 的方式來表示一個 rule，其說明爲：當 C1, C2,…,Cn 都符合時，攻擊者就可以得到加密訊息 X；如果加密訊息 X 會依賴另一個加密訊息 Y，我們就以 available(Y)來表示，因爲 Y 可能是任何人都可獲得或是可藉由另一個 rule 得到，因此 available predicates 爲下(此 W 爲變數)：

available(W) :- public(W).

available(W) :- maybeforged(W).

以前一節的(r3)爲例，在我們的系統裡，就以下面方式來表示：

maybeforged({A, Na, Tb}Kbs) :- public(A), public(Na), generate(B, Tb), key(B, Kbs). (r4)

我們用下面的五個 Procedures 來建立加密訊息的 rules，這五個 Procedures 分別爲 COLLECT，CONTAIN，DERIVE，PRINTRULE 和 MAIN，在介紹每一個 Procedure 功能爲何前，我們定義 BF 爲 basic facts 的假設集合，且所有出現在 Procedures 中的邏輯關係(and 和 or)爲 Short-circuit Boolean Expression。

首先，COLLECT Procedure 有三個輸入欄位，第一個爲要收集元素的訊息 M，第二個爲要存放元素的集合 Q，而最後爲安全金鑰的集合 KQ。這個 Procedure 的功能爲收集一個未加密或加密訊息 M 中的元素到元素集合 Q 且依據金鑰集合 KQ 決定是否可收集加密訊息 M 內的元素或把整個加密訊息 M 當作一個元素，如果加密訊息 M 中有加密訊息，其一樣依據金鑰集合 KQ 決定是否繼續收集此加密訊息內的元素或把它當成一個元素收集。圖 2.4 爲 COLLECT Procedure，下面我們說明圖 2.4 的正確性：

如果訊息 M 爲一個未加密訊息，訊息 M 就會在第四行時被加入元素集合 Q 裡，然後結束，回傳元素集合 Q，而元素集合 Q 也只有訊息 M 一個元素，所以是正確的；反之，如果訊息 M 是一個用金鑰 K 加密的訊息，但如果金鑰 K 不在金鑰集合 KQ 裡，訊息 M 就會被視爲一個單一的元素，一樣在第四行時被加入元素集合 Q 裡，然後結束，所以一個加密訊息的金鑰不在金鑰集合 KQ 裡，此加密訊息內的元素就不會被收集；可是，如果金鑰 K 在金鑰集合 KQ 裡，我們就再次利用 COLLECT Procedure 收集每一個在訊息 M 裡的元素 Y，如果元素 Y 是一個用 K'加密的訊息且 K' 在集合金鑰 KQ 裡，我們就又一次呼叫 COLLECT Procedure 收集每一個在 Y 裡的元素，因爲每一個元素最後一定是一個未加密訊息或是一個加密訊息，但加密金鑰不存在金鑰集合 KQ 裡，因此第三行並不會產生無限遞迴且會正確地依據集合 KQ 來收集每一個元素，所以 COLLECT Procedure 就能正確地依據金鑰集合 K 收集每一個在訊息 M 中的元素到元素集合 Q 中。

下列爲一個範例：
COLLECT({A, {B, {C, {D}Kas}Kcs}Kbs}Kas, Q, {Kas, Kbs})
其輸出的 Q 爲：
{"A", "B", "{C, {D}Kas}Kcs" }

```
COLLECT (M, Q, KQ)
1 if M is encrypted with key K and K ∈ KQ
2     then for each element Y in M
3               do Q ← COLLECT(Y, Q, K);
4     else Q ← Q ∪ M;
5 return Q;
```

圖 2.4 COLLECT procedure

接著我們介紹 CONTAIN Procedure，CONTAIN Procedure 有四個輸入，分別爲訊息 X，訊息 M，和二個安全金鑰的集合 KQ1 和 KQ2。CONTAIN Procedure 的功能爲判斷訊息 M 是否爲訊息 X 或著訊息 M 是否有包含訊息 X，如果回傳 true 的話，表示訊息 M 爲訊息 X 或訊息 M 包含訊息 X；相反，如果回傳 false 的話，表示訊息 M 不爲訊息 X 且訊息 M 不包含訊息 X。當回傳的值爲 true 時，從(加密)訊息 M 最外層一直到包含訊息 X 的那一層中所有經過的安全金鑰都會被收集到金鑰集合 KQ1，而其它出現在(加密)訊息 M 中的安全金鑰就加入金鑰集合 KQ2，也就是說，如果 TotalKQ 爲所有出現在(加密)訊息 M 的安全金鑰集合，TotalKQ ⊆ KQ1 ∪ KQ2，換句話說，如果一開始 KQ1 = Ø 且 KQ2 = Ø，然後回傳值爲 true 時，KQ2 = TotalKQ – KQ1。圖 2.5 爲 COTAIN Procedure，下面我們說明其正確性：

在第一行時，我們宣告二個空集合，分別爲 TQ1 和 TQ2，接著在第二行時，我們設定一個 flag 爲 false，這個 flag 決定 CONTAIN Procedure 回傳時的值爲 true 或 false，在第三行中，如果訊息 M 等於訊息 X，我們就設 flag 爲 true，然後結束，此時 KQ1 和 KQ2 沒有變，所以是正確的；如果訊息 M 不等於訊息 X 且不爲一個加密訊息(第五行的判斷)，就直接結束，此時 flag 沒有變，還是爲 false，且 KQ1 和 KQ2 也沒變，所以是正確的；如果訊息 M 不等於訊息 X，但爲一個加密訊息，因此第六行到第十四行就會被執行，在第七行中，我們用 CONTAIN Procedure 來判斷訊息 X 是否在加密訊息 M 中的元素 Y 裡，如果有的話，也就是呼叫 CONTAIN Procedure 回傳的值爲 true，那我們把 flag 設爲 true，表示加密訊息 M 有包含訊息 X，且連集 KQ1 和 TQ1；如果是相反的話，表示元素 Y 沒有包含訊息 X，flag 就保持不變，但連集 KQ2 和 TQ2。注意，因爲一個加密訊息最後一定由未加密訊息組成，所以第七行並不會產生無限遞迴。當從第六行到第十行的迴圈結束時，我們就可以從 flag 知道訊息 M 是否有包含訊息 X，如果 flag 爲 true，表示訊息 M 包含訊息 X，所以我們就連集 KQ1 和訊息 M 的加密金鑰 K；相反，如果 flag 爲 false 的話，表示訊息 M 不包含訊息 X，因此連集 KQ2 和加密金鑰 K，所以 CONTAIN Procedure 能正確地判斷出是否訊息 M 等於訊息 X 或訊息 M 包含訊息 X，且集合

KQ1 和 KQ2 會正確地存放相對應的安全金鑰。

下列爲一範例：
CONTAIN({a}Kbs, {b, {a}Kbs, {{c}Kcs}Kbs}Kas, KQ1 ← Ø, KQ2 ← Ø)
其輸出的爲 true，KQ1 爲{Kas}，KQ2 爲{Kbs, Kcs}。

```
CONTAIN(X, M, KQ1, KQ2)
1   TQ1← Ø , TQ2 ← Ø;
2   flag ← false;
3   if X = M
4      then flag ← true;
5      else if M is encrypted with key K
6              then for each element Y in M
7                      do if CONTAIN(X, Y, TQ1, TQ2) = true
8                           then flag ← true;
9                                KQ1 ← KQ1 ∪ TQ1;
10                          else KQ2 ←KQ2 ∪ TQ2;
11                 if flag = true
13                    then KQ1 ←KQ1 ∪ K;
14                    else KQ2 ←KQ2 ∪ K;
15 return flag;
```

圖 2.5 CONTAIN procedure

下面我們介紹第三個 Procedure，DERIVE Procedure 有五個輸入，分別爲訊息 X，訊息陣列 SM，元素集合 Q，條件集合 P 和字串 S。S 爲訊息 X 的發送者，而其訊息陣列 SM 爲 S 在發送訊息 X 之前所有收到的訊息，其 SM[*length*[SM]]爲最近收到的訊息。DERIVE Procedure 的功能爲從訊息陣列 SM 中找出訊息 X，如果存在的話，最後回傳 true；反之如果不存在的話，就回傳 false，注意，尋找的方法是從訊息陣列 SM 的最後一個元素(S 最近收到的)找回到第一個元素，並且以 S 的觀點出發(依據假設集合 BF 中對 S 的 key 假設)。每一個在訊息陣列 SM 中的元素爲一個安全規約中以 S 爲接收者的 communication action，所以一個訊息陣列 SM 中的元素可以有多個訊息存在。如果 DERIVE Procedure 回傳 true 的話，表示訊息 X 出現在某一個 SM 的元素中(communication action, SM[I] for some I)，元素集合 Q 就會收集每一個包含或等於訊息 X 的訊息 M 中的每一個元素，其包括 S 知道在訊息 M 中的金鑰；另外，如果回傳 true 且訊息 M 爲未加密訊息(等於訊息 X)，加入 public(M)到條件集合 P，說明 M 是一個未加密訊息；反之如果是加密訊息(包含訊息 X)，加入 available(M)到條件集合 P，說明 M 是一個加密訊息。圖 2.6 爲 DERIVE Procedure，下面我們說明其正確性：

在第一行中，我們先設 flag 爲 false，然後，在從第二行到第二十五行迴圈中，檢查每一個在訊息陣列 SM 中的元素 SM[I]，如果都沒有找到訊息 X，在第二十六行時，就會回傳 false。在第三行到第二十三行中，我們對每一個元素 SM[I]中的訊息 M 看看是否訊息 M 等於訊息 X 或包含訊息 X，在第五行時，我們判斷訊息 M 是否爲訊息 X，如果是的話，flag 設爲 true，加入 public(M)到條件集合 P(說明訊息 X 是從未加密訊息 M 得到的)，且加入訊息 M 到元素集合 Q 裡，因爲 flag 被設爲 true，所以當檢查完其它 SM[I]中的訊息後，在第二十五行就會回傳 true，因此就不會繼續檢查 SM[I-1]，所以是正確的；相反，如果訊息 M 不等於訊息 X，且爲一個未加密訊息，那第九行的判斷就不成立，所以就檢查下一個在 SM[I]的訊息，正確；如果在第九行中，訊息 M 爲一個加密訊息，且從 CONTAIN Procedure 中知道到訊息 M 包含訊息 X 的話，也就是 CONTAIN Procedure 回傳 true，那我們就設先 flag 爲 true，因爲金鑰集合 KQ1 含有從加密訊息 M 最外層一直到包含訊息 X 的那一層中所有經過的安全金鑰，所以在第十二行到第十五行要檢查金鑰集合 KQ1 中的每一個元素 K1 是否爲 S 在前提假設下就知道，還是 S 可以從之前收到的訊息中找到(第十三行)，如果前提假設下，S 不知道 K1，且從之前收到的訊息中也找不到，我們就把 flag 設爲 false，表示訊息 X 並不是從這個加密訊息 M 中得到的，注意在第十三行呼叫 DERIVE Procedure 中的第二個欄位爲 SM – M，這表示保留所有訊息陣列 SM 中的每一個訊息除了訊息 M，因爲我們要找 S 是否知道金鑰集合 KQ1 中的每一個元素 K1，所以我們把訊息 M 排除，如果有任何一個在金鑰集合 KQ1 中的元素 K1 因 key(S, K1)不存在前提假設集合 BF 中且從 S 之前接到的訊息中也找不到，表示 S 不知道金鑰 K1，也就是說，訊息 X 並不是從加密訊息 M 中得到的，所以第十四行就設 flag 爲 false，因此就繼續檢查下一個 SM[I]中的訊息；如果每一個在金鑰集合 KQ1 中的元素 K1 有 key(S, K1)存在前提假設集合 BF 中或可以從 S 之前接到的訊息中找到，那表示訊息 X 是從加密訊息 M 中得到的，因此 flag 就保持 true，所以第十六行到第二十三行就會被執行，第十七行中我們把 available(M)到加入條件集合 P 裡，說明訊息 X 是從加密訊息 M 中獲得的，在第十八行到第二十二行中，我們針對金鑰集合 KQ2 中的元素 K2 作檢查，因爲金鑰集合 KQ2 存放的金鑰對於從加密訊息 M 中得到訊息 X 無關，但以 S 的觀點來看，如果 S 知道金鑰集合 KQ2 中的元素 K2，表示 S 知道在加密訊息 M 中用金鑰 K2 加密的訊息(可能有隱含關係存在)，所以必需收集相關的元素和條件分別存在元素集合 Q 和條件集合 P 裡，最後收集加密訊息 M 中的元素。注意，我們是以 communication action 爲處理單位，也就是說，在訊息陣列 SM 的元素 SM[I]中，如果有一訊息 Y(未加密訊息)等於訊息 X 或訊息 Y(加密訊息)包含訊息 X，我們就不處理下一個陣列 SM 的元素 SM[I-1]，但在元素 SM[I]中的其它訊息 Y’還是

會被處理，因爲其它訊息 Y'也有可能等於或包含訊息 X(訊息 X 可能不只依賴訊息 Y，或許也有依賴在同一個 communication action 中的訊息 Y')。如果以 S 爲觀點出發且訊息 X 無法從訊息陣列 SM 中找到，DERIVE Procedure 能正確回答 false；反之如果以 S 爲觀點出發且在一個 SM[I]中有找到訊息 X，那元素集合 Q 和條件集合 P 就會存放相關的元素和條件。

```
DERIVE(X, SM, Q, P, S)
1  flag ← false;
2  for I ← length[SM] downto 1
3      do for each item M ∈ SM[I]
4          do KQ1←Ø , KQ2 ← Ø;
5             if X = M
6                then flag ← true;
7                     P ← P ∪ "public(M)";
8                     Q ← Q ∪ M;
9                else if M is encrypted and CONTAIN(X, M, KQ1, KQ2) = true
10                    then TQ1←Ø, TP1←Ø ;
11                        flag ← true;
12                        for each K1 ∈ KQ1
13                            do if key(S, K1) ∉ BF and
                                    DERIVE(K1, SM - M, TQ1, TP1, S) = false
14                               then flag ← false;
15                                    break;
16                        if flag ← true
17                          then Q ← Q U TQ1, P ← P ∪ TP1 ∪ "available(M)";
18                               for each K2 ∈ KQ2
19                                   do TQ2←Ø , TP2 ← Ø;
20                                      if key(S, K2) ∈ BF or
                                           DERIVE(K2, SM - M, TQ2, TP2, S) = true
21                                         then KQ1 ← KQ1 U K2;
22                                              Q ← Q ∪ TQ2, P ← P ∪ TP2;
23                                  Q ← COLLECT(M, Q, KQ1) , Q ← Q U KQ1;
24         if flag = true
25            then return true;
26 return false;
```

圖 2.6 DERIVE procedure

PRINTRULE Procedure 有四個輸入，分別爲訊息 X，元素集合 Q，條件集合 P 和字串 S。這個 Procedure 的功能必需要列印出產生訊息 X 時所必需符合的條件爲何，其條件爲條件集合 P 加上以 S 的觀點出發之所有隱含關係(依據假設集合 BF 中的假設)，其隱含關係以 generate 和 key 表示，也就是說，如果 generate(S, I) ∈ BF for I ∈ Q，我們就加入 generate(S, I)爲必要的條件，如果 key(S, I) ∈ BF for I ∈ Q，我們就加入 key(S, I)爲必要的條件，如果 key(S, J), key(I, J) ∈ BF for I , J ∈ Q，我們就加入 key(I, J) 爲必要的條件。圖 2.7 爲 PRINTRULE Procedure，下面說明其正確性：

隱含關係的處理在第一行到第九行，對元素集合 Q 中的每一個元素 I 判斷是否有 generate(S, I)或 key(S, I)存在於假設集合 BF 中(第二行和第五行)或是否元素集合 Q 中有另一元素 J 符合 key(S, J), key(I, J) ∈ BF (第八行)，如果其中任何一個條件符合我們就加入相對應的條件 generate(S, I)，key(S, I)或 key(I, J)到條件集合 P，注意，在第四行的 Continue 會直接跳到第一行的迴圈，執行下一個在元素集合 Q 中的元素，那是因爲在前提假設下，所有 key 的關係是在執行一個 run 前所要有的，但 generate 的關係是在執行一個 run 中才發生的，所以如果 generate(S, I) ∈ BF for I ∈ Q，就不會有 key(S, I)。最後，在第十一行到第十二行列出所有產生訊息 X 的條件，也就是所有存在條件集合 P 的元素(格式如圖 2.3)，因此符合上面對 PRINTRULE Procedure 的要求。

```
PRINTRULE(X, Q, P, S)
1   for each I ∈ Q
2       do if generate(S, I) ∈ BF
3             then P ← P ∪"generate(S, I)";
4                  Continue;
5             else if key(S, I) ∈ BF
6                    then P ← P ∪"key(S, I)";
7           for each J ∈ Q
8               do if key(S, J) ∈ BF and key(I, J) ∈ BF
9                     then P ← P ∪"key(I, J)";
10 print "maybeforged (X) :- ";
11 for each V ∈ P
12     do print V;
```

圖 2.7 PRINTRULE procedure

最後，我們介紹主要的 MAIN Procedure，其輸入有二個欄位，一個存放一個安全規約的 communication action 陣列 CA 和一個存放 basic facts 的假設集合 BF。MAIN Procedure 將會對所有規約中的每一個加密訊息建立一個 rule，除了出現在第一個 communication action 中的加密訊息(圖 2.8 爲 MAIN Procedure)。而我們定義何謂一個正確的 rule，一個正確的 rule 爲當所需要滿足的條件都成立時，此 rule 就會產生相對應的訊息(如圖 2.3 的 X)，然而如何建立一個正確的 rule 就是我們要討論的問題，首先，我們想一想，在一個安全規約中，第一個 communication action 的發送者在一個規約開始前，他並沒有事先收到任何訊息，因此任何第一個 communication action 中的訊息是發送者依據自己擁有的資訊產生的，所以沒有任何人可以主動地要求第一個 communication action 的發送者開始執行一個 run，也就是說，如果有一個加密訊息出現在第一個 communication action 中，其加密訊息並不依賴外界的影響，所以在 MAIN Procedure 中的第四行和第五行，我們把出現在第一個 communication action 中的訊息以 public 的方式加到假設集合 BF 中，接下來想想第二個 communication action，如果第二個 communication action 的發送者是第一個 communication action 的接收者的話，那表示第二個 communication action 中的訊息產生是因爲收到第一個 communication action 的訊息且收到的訊息是合乎規約的規範，如圖 2.2 中的參與者 B 在第二個 communication action 中的加密訊息{A, Na, Tb}Kbs，其中 A 和 Na 是依據第一個 communication action 中的 A 和 Na，所以說，如果我們在第一個 communication action 中傳送 C 和{A, Na}Kcs 給參與者 B 的話，而參與者 B 就會在第二個 communication action 送出 C，{C, {A, Na}Kcs, Tb}Kbs 和 Nb，其中 Tb 和 Nb 爲參與者 B 自己產生的，所以說，除了第一個 communication action 的訊息外的加密訊息都有一個相對應的 rule 存在(注意，我們只討論只有一個 initiator 的安全規約，不討論 multiple initiators)，所以在 MAIN Procedure 中的第六行到第二十三行中，我們對每一個在陣列 CA 的元素做處理，因爲我們需要記錄每一位參與者之前所收到的訊息，所以我們有一個陣列 Message 來存放每一位參與者所收到的訊息，如 Message[B]表示 B 到目前爲止所有收到的訊息，而 Message[B][1]表示 B 第一次收到的訊息，反之，Message[B][*length*[Message[B]]]表示 B 最近收到的訊息，注意，此訊息是表示一個 communication action 中的所有訊息，如在 MAIN Procedure 中的第三行和第十行。如我們之前所提的，一個 rule 並非只適用於特定的參與者，所以說，我們要使它一般化，如上例中{A, Na, Tb}Kbs 的 rule，我

們要說明 key(B, Kbs)的關係是必需存在的，才能使它適用於任何參與者，所以當要建立這個 rule 時，元素 A，Na，Tb 和 Kbs 就會被加入元素集合 Q 裡，而 public(A)和 public(Na)加入條件集合 P 裡，最後，從元素集合 Q 中找出隱含關係(如 generate(B, Tb), key(B, Kbs))，將他們加入條件集合 P 裡，然後列出所如圖 2.3 的 rule。從第十一行到第二十三行中，我們針對每一個在 CA[I]中的訊息 X 處理，以發送者 S 的觀點出發，也就是依據假設集合 BF 中對發送者 S 的 key 和 generate 假設爲基礎，如果訊息 X 爲一個未加密訊息，在第十三行的判斷就不成立，因此就直接跳到第二十三行執行，把未加密訊息 X 以 public 的方式加入到假設集合 BF(注意第二十三行中，如果訊息 X 是加密訊息，表示發送者 S 是轉送之前收到的訊息，因此不處理)，如果訊息 X 是一個以金鑰 K 加密的訊息且 key(S, K)存在假設集合 BF 中或可以從發送者 S 之前收到的訊息 Message[S]中找到，我們就知道訊息 X 是發送者 S 產生的，注意，如果 key(S, K)存在假設集合 BF 中，我們就不會檢查是否可以從發送者 S 之前收到的訊息 Message[S]中找金鑰 K(Short-circuit Boolean Expression)；反之如果 key(S, K)不存在假設集合 BF 中，我們才需要檢查是否可以從發送者 S 之前收到的訊息 Message[S]中找到金鑰 K，如果這兩個條件沒有一個成立的話，表示加密訊息 X 不是由發送者 S 產生的，而是 S 轉送出來的，所以就跳到第二十三行；相反，如果其中只要有一個成立，表示發送者 S 有能力解開加密訊息 X，所以加密訊息 X 是發送者 S 產生的，因此第十五行到第二十二行中，我們去找在加密訊息 X 中的每一個元素 Y 是否可以從發送者 S 之前收到的訊息中找到(第十七行)或是發送者 S 自己產生的(第十九行)，如果都不是的話，表示發送者 S 並不知道元素 Y，所以規約在設計上是有問題的；如果都沒問題時，最後第二十二行就會被執行，列出產生加密訊息 X 的條件爲何，也就是建立加密訊息 X 的 rule，所以如果 MAIN Procedure 最後回傳 true 時，表示每一個在陣列 CA 中的加密訊息(除了在 SM[1]中)都會有一個相對應的 rule，說明產生此加密訊息的條件爲何；反之如果沒有回傳 true 時，表示第二十一行會被執行，說明所給的安全規約在設計上有問題。

```
MAIN(CA, BF)
1   P ← Ø, Q ← Ø, TP ← Ø, TQ ← Ø;
2   R ← receiver of CA[1];
3   Message[R][1] ← CA[1];
4   for each V in CA[1]
5       do BF ← BF ∪ public(V);
6   for I ← 2 to length[CA]
7       do S ← sender of CA[I];
8          R ← receiver of CA[I];
9          J ← length[Message[R]];
10         Message[R][J+1] ← CA[I];
11         for each item X in CA[I]
12             do Q ← Ø, P ← Ø, TQ ← Ø, TP ← Ø;
13                if X is encrypted with key K and
                     (key(S, K) ∈ BF or DERIVE(K, Message[S], TQ, TP, S) = true)
14                then Q ← Q ∪ TQ ∪ K , P ←P ∪ TP;
15                     for each element Y in X
16                         do TQ ← Ø, TP ← Ø;
17                            if Y = S or
                                 DERIVE(Y, Message[S], TQ, TP, S) = true
18                               then Q ← Q ∪ TQ, P ← P ∪ TP;
19                            else if generate(S, Y) ∈ BF
20                                    then P ←P ∪ "generate(S, Y)";
21                                 else error "The source of Y is either not
                                               found from all messages
                                               received by S or not generated
                                               by S in X of CA[I]";
22                     PRINTRULE(X, Q, P, S);
23                else if X is not encrypted then BF ← BF ∪ public(X);
24  return true
```

圖 2.8 MAIN procedure

2.4 範例

以上一節所提出轉換的方法，我們將圖 2.2 Neumann 和 Stubblebine 所提出的安全規約轉換出其相對應的 rules。首先，我們將 Neumann 和 Stubblebine 所提出的前提假設轉換成我們系統的格式，如圖 2.9，此為假設集合 BF，而陣列 CA 存放著安全規約中的 communication actions，CA[1]為第一個 communication action，而 *length*[CA]為四，因為安全規約中有四個 communication actions。

key(A, Kas)

key(S, Kas)

key(B, Kbs)

key(S, Kbs)

generate(A, Na)

generate(B, Nb)

generate(B, Tb)

generate(S, Kab)

圖 2.9 Initial assumption of Neumann Stubblebine security protocol

我們呼叫 MAIN procedure，把 Neumann Stubblebine security protocol 的 CA 和 BF 當作輸入參數，在第三行中，把第一個 communication action 的訊息(CA[1])存在 Message[B][1]中，也就是把 A 和 Na 存在 Message[B][1]中，然後在第四行和第五行中，把 public(A)和 public(Na)加到假設集合 BF 中，接著在第六行中，因為 *length*[CA]等於四，所以迴圈中的變數 I 會從二加到四，也就是說，從第二個 communication action CA[2]處理到第四個 communication action CA[4]，在變數 I 等於二時，我們知道發送者為 B，接收者為 S，所以第十行就會把第二個 communication action 的訊息(CA[2])存在 Message[S][1]中，然後，在第十一行的迴圈中，對每一個在 CA[2]的訊息作處理，也就是訊息 B，{A, Na, Tb}Kbs 和 Nb，首先，我們先看 B 這個訊息，在第十三行中，因為訊息 B 不是一個加密訊息，所以在第二十三行中把 public(B)加到假設集合 BF，從原來的規約中，我們知道這個訊息 B 是未加密的，所以上面的作動是正確的，接下來處理{A, Na, Tb}Kbs，因為這個訊息為加密訊息且 key(B, Kbs)存在假設集合 BF 裡，加入 Kbs 到元素集合 Q，在第十五行中，我們要對每一個在加密訊息{A, Na, Tb}Kbs 中的元素做檢查，首先看 A 這個元素，第十七行就會呼叫 DERIVE Procedure 尋找在 Message[B]中(也就是發送者 B 之前所收到的訊息)是否有元素 A 存在，我們發現元素 A 存在 Message[B][1]，所以在第十八行時，就把元素 A(TQ 內的元素)加入元素集合

Q 裡，把 public(A)(TP 內的元素)加入條件集合 P 裡，接著處理{A, Na, Tb}Kbs 中的元素 Na，其結果一樣，在 Message[B][1]中發現元素 Na，因此把元素 Na 加入元素集合 Q，public(Na)加入條件集合 P 裡，然後處理{A, Na, Tb}Kbs 中的元素 Tb，第十七行就會呼叫 DERIVE Procedure，但發現元素 Tb 不存在 Message[B]的任何訊息裡(TQ 和 TP 爲 Ø)，但在第十九行中，generate(B, Tb)出現在假設集合 BF 中，所以加入 generate(B, Tb)到條件集合 P 中，最後結束第十五行的迴圈，然後執行第二十二行，呼叫 PRINTRULE Procedure，目前的元素集合 Q 爲{Kbs, A, Na}，而條件集合 P 爲{“public(A”, “public(Na)”)，所以在 PRINTRULE Procedure 中就會列出如上節中的(r4)，其如下面所列(注意，rule 中的符號都是變數)：

maybeforged({A, Na, Tb}Kbs) :- public(A), public(Na), key(B, Kbs), generate(B, Tb).

最後在第二個 communication action 中，還剩下 Nb，第十三行中，因爲 Nb 是一個未加密訊息，所以在第二十三行中把 public(Nb)加到假設集合 BF，我們已經處理過第二個 communication action 中的每一個訊息，接我們看第三個 communication action，其發送者爲 S，接收者爲 A，因此在第十行中，把第三個 communication action 的訊息(CA[3])存在 Message[A][1]中，然後在第十一行的迴圈中，對每一個在 CA[3]的訊息處理，也就是{B,Na,Kab,Tb}Kas，{A,Kab,Tb}Kbs 和 Nb，先看加密訊息{B,Na,Kab,Tb}Kas，在第十三行中，key(S, Kas)存在假設集合 BF 中，所以成立，就把 Kas 加入元素集合 Q 中，然後檢查每一個在加密訊息{B,Na,Kab,Tb}Kas 中的元素，在第十七行中，B 存在 Message[S][1]中且爲未加密息訊，因此第十八行中，就把 B 存放在元素集合 Q 裡，條件集合 P 存放 public(B)，而下一個元素 Na 存在 Message[S][1]中的{A, Na, Tb}Kbs，所以在第十八行後，元素集合 Q 爲{Kas, B, A, Na, Tb, Kbs}，條件集合 P 爲{“public(B)”, “available({A, Na, Tb}Kbs)”}，接著處理元素 Kab，在第十七行中元素 Kab 並沒有出現在 Message[S]中，但在十九行時，generate(S, Kab)存在假設集合 BF 中，所以加 generate(S, Kab)到條件集合 P 中，最後元素 Tb 存在 Message[S][1]中的{A, Na, Tb}Kbs 中(第十七行)，所以元素集合 Q 和條件集合 P 並沒有變，然後在第二十二行呼叫 PRINTRULE Procedure 時，其元素集合 Q 爲{Kas, B, A, Na, Tb, Kbs}，而條件集合 P 爲{“public(B)”, “available({A, Na, Tb}Kbs)”, “generate(S, Kab”}，因此產生加密訊息{B,Na,Kab,Tb}Kas 的 rule 如下：

maybeforged({B,Na,Kab,Tb}Kas) :- public(B), available({A,Na,Tb}Kbs), generate(S, Kab), key(S, Kas), key(S, Kbs), key(B, Kbs), key(A, Kas).

接著，我們處理第三個 communication action 中的{A,Kab,Tb}Kbs，在第十三行中，key(S, Kbs)存在假設集合 BF 裡，就以存放 Kbs 到元素集合 Q 中，然後在第十七行中，{A,Kab,Tb}Kbs 中的元素 A 存在 Message[S][1]中的{A, Na, Tb}Kbs，

因此第十八行後的元素集合 Q 爲{Kbs, A, Na, Tb}，而條件集合 P 爲{“available({A, Na, Tb}Kbs)”}，{A,Kab,Tb}Kbs 中的元素 Kab 並不存在 Message[S]中，但在第十九行，generate(S, Kab)存在假設集合 BF 裡，加入 generate(S, Kab)到條件集合 P 中，最後在{A,Kab,Tb}Kbs 中的元素 Tb，也是存在 Message[S][1]，因此元素集合 Q 和條件集合 P 沒有變，在第二十二行中，元素集合 Q 爲{Kbs, A, Na, Tb}，而條件集合 P 爲{“available({A, Na, Tb}Kbs)”, “generate(S, Kab)”}，因此產生加密訊息{A,Kab,Tb}Kbs 的 rule 如下：

maybeforged({A,Kab,Tb}Kbs) :- available({A,Na,Tb}Kbs)), generate(S, Kab), key(S, Kbs).

最後處理在第三個 communication action 中的 Nb，因爲 Nb 是一個未加密訊息，如同第二個 communication action 中的 Nb，在第二十三行中把 public(Nb)加到假設集合 BF(假設集合 BF 並沒有變)。最後是第四個 communication action，其發送者爲 A，接收者爲 B，因此在第十行中，把第四個 communication action 的訊息(CA[4])存在 Message[B][2]中，接著處理加密訊息{A,Kab,Tb}Kbs 和{Nb}Kab，先看{A,Kab,Tb}Kbs，在第十三行中，因爲 key(A, Kbs)不存在假設集合 BF 中且在 Message[A]中也找不到加密金鑰 Kbs，所以跳到第二十三行，不處理{A,Kab,Tb}Kbs，表示加密訊息{A,Kab,Tb}Kbs 是發送者 A 轉送出去的。我們接著看加密訊息{Nb}Kab， 在第十三行中，雖然 key(A, Kab)不在假設集合 BF 中，但卻可以在 Message[A][1]中找到加密金鑰 Kab，因此在第十四行後，元素集合 Q 爲{Kab, Kas, B, Na, Tb}，條件集合 P 爲{“available({B,Na,Kab,Tb}Kas)”}，所以在第十五行中，處理{Nb}Kab 中的元素 Nb，在第十七行中，我們在 Message[A][1]中找到元素 Nb，且爲一個未加密訊息，因此在第十八行中，加入 public(Nb)到條件集合 P 裡，最後第二十二行呼叫 DERIVE Procedure，其元素集合 Q 爲{Kab, Kas, B, Na, Tb}，條件集合 P 爲{“available({B,Na,Kab,Tb}Kas)”, “public(Nb)”}，因此產生加密訊息{Nb}Kab 的 rule 如下：

maybeforged({Nb}Kab)) :- key(A, Kas), generate(A, Na), public(Nb), available({B,Na,Kab,Tb}Kas)

因此 Neumann Stubblebine security protocol 就有下面四個 rules：

1) maybeforged({A, Na, Tb}Kbs) :- public(A), public(Na), key(B, Kbs), generate(B, Tb).

2) maybeforged({B,Na,Kab,Tb}Kas) :- public(B), available({A,Na,Tb}Kbs), generate(S, Kab), key(S, Kas), key(S, Kbs), key(B, Kbs), key(A, Kas).

3) maybeforged({A,Kab,Tb}Kbs) :- available({A,Na,Tb}Kbs)), generate(S, Kab), key(S, Kbs).

4) maybeforged({Nb}Kab)) :- key(A, Kas), generate(A, Na), public(Nb), available({B,Na,Kab,Tb}Kas)

假設集合 BF 最後爲下列：

key(a, kas)

key(s, kas)

key(b, kbs)

key(s, kbs)

generate(a, na)

generate(b, nb)

generate(b, tb)

generate(s, kab)

public(a)

public(na)

public(b)

public(nb)

注意，在最後的假設集合 BF 裡，我們以小寫開頭表示一個 run 的 instance，上面的 BF 爲參與者 A 和 B 執行一次 Neumann Stubblebine 的安全規約。我們簡單地看一下上面所建的 rule 是否正確，我們推導在第三個 communication action 中的第一個加密訊息的產生，也就是{B, Na, Kab, Tb}Kas 的產生，其從 rule 2 中再推導出 rule 1 就得到正確規約中的方法，如下(小寫爲 constant，大寫爲變數)：

2) maybeforged({b, na, kab, tb}kas) :- public(b), available({A, na, tb}kbs), generate(S, kab), key(S, kas), key(S, kbs), key(b, kbs), key(A, kas).
available({A, na, tb}kbs) => 1) maybeforged({A, na, tb}kbs) :- public(A), public(na), key(B, kbs), generate(B, tb).

上面中的 available({A, na, tb}kbs)可以用 rule 1 推導，如果變數 A，B 和 S 分別爲值 a，b 和 s 的話，那上面每一個 public，generate 和 key 都可在 BF 中找到，所以{b, na, kab, tb}kas 的產生可從 rule 2 和 rule 1 得到。問如何產生一個加密訊息{a, K, tb}kbs (K 爲變數)？我們可以從 rule 3 再推導 rule 1 得到(原本規約中的方

式)或直接從 rule 1 得到(訊息的相似性)，其 rule 1 爲：

1) maybeforged({a, K, tb}kbs) :- public(a), public(K), key(B, kbs), generate(B, tb).

只要變數 B 爲 b，變數 K 就可以是任意值。因此在這裡我們很清楚地看出 rule 是說明如何依據 BF 中的資料判斷所要的加密訊息是否可以得到，所以如果我們在 BF 中加入多個 run 的資料，那我們就可以分析在多個 run 下有可能發生的弱點了，如在附錄一中的範例 Adapted Andrew protocol。

2.5 系統能力

我們的系統是藉由加密訊息之間的相似性找出所有可能產生一加密訊息的方式，而這是因爲我們在建立 rules 時是以發送者的觀點出發，所以有些表面上是不相似的加密訊息實際上是相似的(如第 4.2 節中 Woo And Lam security protocol 的分析)，所以當我們詢問一個加密訊息時，只要有符合我們加密訊息格式的 rules 都會被詢問，且當 rule 和 rule 之間有依賴性時，我們用 available predicates 來表示，而 available predicates 會先看看是否有一個 public 的訊息可以符合目前要求的訊息，也就是說，我們是否可以用一個已知的訊息取代，如果沒有的話，available predicates 就會問是否有一個 rule 可以符合目前要求的訊息，這時，available predicates 並沒有問特定某一個 rule，換句話說，如果有兩個 rules 都和目前要求的訊息相似，available predicates 就會問這兩個 rules 是否可以符合目前要求的訊息，而這就是我們所提到的訊息相似性。我們系統的能力如下：如果在一個安全規約的正常前提假設下，此安全規約存在一個相似訊息攻擊方式，我們的系統就可以找出此攻擊方式。我們證明在我們的系統下，這樣一個攻擊中所有攻擊者沒辦法自行產生的加密訊息 M 都可以正確地被找出來，也就是說，當我們詢問每一個加密訊息 M 時，系統就會回答 true 且攻擊方式中產生此加密的路徑(步驟)也可以正確地找出來，注意，產生加密訊息 M 的路徑也許有很多(因爲訊息的相似性)，但攻擊方式中產生加密訊息 M 的路徑爲其中之一，我們利用矛盾法證明，如果有一個加密訊息 M 爲攻擊者沒辦法自行產生的加密訊息，且產生的加密訊息 M 的路徑爲 P，當我們詢問是否可僞造加密訊息 M 時，系統回答 false 或回答 true 但沒有一個路徑爲 P，因此我們證明系統可以推導出路徑 P，所以系統會回答 true 且有一路徑爲 P。

令加密訊息集合F爲攻擊方式中攻擊者無法自行產生的加密訊息，假設有一個加密訊息M ∈ F，加密訊息M是由路徑P = M_1->M_2->…->M_n->M的步驟中得到的(M_i ∈ F for 2 ≤ i ≤ n，M_1必爲未加密訊息或攻擊者可以自行產生的，這是因爲在正常的假設下，攻擊者不知道任何安全金鑰除了他自己擁有的以外)；我們用反推的方式推導出路徑P，首先因爲加密訊息M的格式是符合規約中某一個加密訊息的格式(相似性)，而系統也對每一個加密訊息建立一個rule，所以有一個rule可以產生加密訊息M，這是因爲加密訊息M是攻擊者在攻擊中所傳送的訊息，因此加密訊息M必符合規約中的某一個加密訊息格式，所以有一個rule存在是可以產生加密訊息M，接著我們知道在路徑P中的M_n->M表示必需得到加密訊息M_n才能得到加密訊息M，也就是說，產生加密訊息M時需要加密訊息M_n(訊息之間的依賴性)，而我們知道系統中也有一個rule可以產生加密訊息M_n(因爲加密訊息M_n的格式也必符合規約中某一個加密訊息的格式)且系統用available predicates來表示訊息之間的依賴性(詢問是否有一個public的訊息或有一個rule可以符合目前的要求)，所以在某一個可以產生加密訊息M的rule中的available predicates就能問到一個產生加密訊息M_n的rule，同理，產生加密訊息M_n的rule中的available predicates就能問到產生加密訊息M_{n-1}的rule，當在推導到M_2時，因爲M_1->M_2，所以系統中也會有存在一個rule可以產生加密訊息M_2，且我們知道M_1爲未加密訊息，表示某一個可以產生加密訊息M_2的rule也沒有依賴任何其它的rules，所以說，當在詢問是否可得到加密訊息M時，系統會找到某一個產生加密訊息M的rule，然後一個產生加密訊息M_n的rule，接著一個產生加密訊息M_{n-1}的rule，就這樣一直找下去，最後找到某一個產生加密訊息M_2的rule，且此rule就會結束(不依賴其它的rules)，因此系統就回答true且尋找的路徑和P一樣，這和我們的假設相反，證明我們的系統可以正確地找出每一個攻擊中所有攻擊者沒辦法自行產生的加密訊息。

以圖 2.2 的 Neumann Stubblebine security protocol 爲例，其攻擊方式爲：

(i1) (A)Z->B: A, Na

(i2) B->(S)Z: B, { A, Na, Tb }Kbs, Nb

(i4) (A)Z->B: { A, Na, Tb }Kbs, { Nb }Na

從攻擊方式中，我們知到攻擊者無法自行產生的加密訊息爲{A, Na, Tb}Kbs (在第(i4)步驟)，而產生此加密訊息的路徑爲 “A, Na”->{A, Na, Tb}Kbs ((i1)->(i2))，因此當我們詢問{A, Na, Tb}Kbs時，系統必需回答true且有一路徑爲“A, Na”->{A, Na, Tb}Kbs，下面爲上一節中對 Neumann Stubblebine security protocol 所建立的假設集合和四個 rules(小寫爲 constant，大寫爲變數)：

key(a, kas)

key(s, kas)

key(b, kbs)

key(s, kbs)

generate(a, na)

generate(b, nb)

generate(b, tb)

generate(s, kab)

public(a)

public(na)

public(b)

public(nb)

1) maybeforged({A, Na, Tb}Kbs) :- public(A), public(Na), key(B, Kbs), generate(B, Tb).

2) maybeforged({B,Na,Kab,Tb}Kas) :- public(B), available({A,Na,Tb}Kbs), generate(S, Kab), key(S, Kas), key(S, Kbs), key(B, Kbs), key(A, Kas).

3) maybeforged({A,Kab,Tb}Kbs) :- available({A,Na,Tb}Kbs)), generate(S, Kab), key(S, Kbs).

4) maybeforged({Nb}Kab)) :- key(A, Kas), generate(A, Na), public(Nb), available({B,Na,Kab,Tb}Kas)

從 rule 1)中，系統可以正確地推導如下：

1) maybeforged({a, na, tb}kbs) :- public(a), public(na), key(B, kbs), generate(B, tb).

我們知道變數 B 可爲參與者 b，所以這個 rule 是成立的，且這個 rule 是在第二個 communication action 時建立的，也就是說它的路徑爲“A, Na”->{A, Na, Tb}Kbs，如同攻擊方式中的(i1)和(i2)，因系統就正確地回答 true 且找出路徑。在第 4.2 節和附錄一中有其它規約的分析，從這些分析中，可以再次看出我們系統的能力。

第三章 系統實作

在本章中，我們將介紹如何實作上一章節所提出的安全規約分析系統，我們以 Prolog 語言來實作我們的系統，藉由 Prolog 的 Backtracking 特性來分析一個安全規約的弱點，因此我們先介紹 Prolog 的基本語法，然後再說明如何把上一章節所提出的安全規約分析系統轉成 Prolog，根據設計出來的使用者介面，使用者可以利用詢問的方式分析安全規約的安全性。

3.1 Prolog 環境

Prolog 是一種邏輯語言，當使用者載入一個資料庫到 Prolog 的環境後，使用者就可以像一般資料庫系統一樣下詢問指令，系統就可以找出符合條件的資料，但 Prolog 有別於一般的資料庫系統，除了資料庫本身要符合 Prolog 的語法外，使用者也可以自行定義詢問的規則，而每個規則中也可以含有其它規則，因此它的彈性大於一般資料庫系統，所以在上一章節所提到的 basic facts 和 rules 就會用 Prolog 的語法以資料庫和規則的方式呈現。

首先，在 Prolog 中，一個 atom 就如同 C 語言中的 constant 一樣，但一個 atom 的命名方式必需以小寫的字母爲開頭 (如 kab, kAb, kAB)，而一個變數則必需以大寫爲開頭 (如 Kab,KAb, KaB)。在 Prolog 中，資料庫以 facts 的方法表示，一個 fact 的表示方法如圖 3.1，它可以只是一個 atom，或加上參數值，並以“.”作爲結束符號，注意，參數值也是 atoms，而且每個 atom 之間必需以“,”隔開。一個規則以“:- ”符號表示，表示方法如圖 3.2，在“:- ”左手邊爲規則的名字，名字後可接參數值 (可爲 atom 或變數)，如同 C 語言中的 function，但規則的名字是一個 atom，所以必需符合 atom 的命名方式，在“:-“的右手邊爲規則的內容，可以是其它規則或 facts，每個規則或 facts 之間可用“,”或“;”隔開，“,”表示邏輯上的 AND，相反地，“;”表示爲 OR，但最後必需以點“.”作爲結束符號，例如 positive(X):-integer(X), >(X, 0).，當使用者詢問 positive 這個規則時，並傳入一個值，如果傳入的值是正整數，規則顯示 true，否則 false。在 Prolog 中，“->”表示 if 的意思，如果左邊的結果是 true，就執行右邊的條件，且可以配合“;”做出 else 的效果。最後，我們介紹 List Structure，在 Prolog 裡，一個 List 是由一對“[]”所表示，List 內可以有許多個 atoms(以“,”隔開)或一個都沒有 (Empty List)，當然 List 內也可

以有 List，在我們的實作中，我們將會以 List 來表示一個加密訊息。

atom_name[(atoms)].

圖 3.1 Expression of a fact in Prolog

predicate_name[(atoms or variables)] :- predicates

圖 3.2 Expression of a rule in Prolog

3.2 輸入規格

在這節中，我們說明輸入的格式，圖 3.3 爲圖 2.2 安全規約的輸入格式，第一行爲“ASSUMPTIONS:”，表示接下來的資料爲一個安全規約的前提假設，前提假設以 key 和 generate 表示，key 說明在執行安全規約前參與者之間和參與者與伺服器之間的安全金鑰關係，而 generate 表示在執行安全規約時，每一位參與者和伺服器所需要產生的東西(如 Nonce)，因此要分析一個安全規約是否有 Parallel Attack 或 Multiple Session Attack 時，只要用 generate 表示多個 run，就可能分析出在一個 run 不會發生的弱點。在前提假設的條件後，我們以“PROCOTOL:”表示下面的資料爲一個安全規約的內容，我們依據圖 2.1 作爲安全規約的格式，以“A->B: M”表示參與者 A 傳送 M 給參與者 B，M 可以爲多個訊息不管是加密或未加密訊息，但訊息之間以“,”隔開，加密訊息表示的方式爲“{E}K”，K 爲加密金鑰，E 爲加密訊息內的元素，相同地，各元素之間以“,”隔開且可爲一加密或未加密訊息。注意，在輸入的資料中，字母的大小寫並不影響輸出的結果，因爲在我們設計的程式中，所有輸入的資料都會被轉成大寫，後然輸出時，就依 Prolog 的格式轉成相對應的大寫或小寫(basic facts 爲小寫，而 rules 爲大寫)。如果輸入的資料有符號“#”，從“#”到換行符號之間的資料爲註解。圖 3.4 爲我們輸入格式規格語法，其中 BL，BR，L_PAREN 和 R_PAREN 分別爲“{”，“}”，“(”和“)”。

ASSUMPTIONS:

key(A, Kas).

key(B, Kbs).

key(S, Kas).

key(S, Kbs).

generate(A, Na).

generate(B, Nb).

generate(B, Tb).

generate(S, Kab).

PROCOTOL:

A->B: A, Na

B->S: B, { A, Na, Tb }Kbs, Nb

S->A: { B, Na, Kab, Tb }Kas , { A, Kab, Tb }Kbs, Nb

A->B: { A, Kab, Tb }Kbs, { Nb }Kab

圖 3.3 Input format of Neumann Stubblebine security protocol

start: ASSUMPTION COLON assumMultiple body ;

assumMultiple: Δ | assumMultiple assum ;

assum : ID L_PAREN ID COMMA ID R_PAREN DOT ;

body : PROTOCOL COLON contentMultiple ;

contentMultiple : Δ | contentMultiple statement ;

statement : ID ARROW ID COLON message ;

message : item | message COMMA item ;

item : ID | BL message BR keyid ;

keyid: ID ;

圖 3.4 Input format syntax

[[element, …, element], key]

圖 3.5 List structure for encrypted item in Prolog

1)maybeforged({A, Na, Tb}Kbs) :- key(B, Kbs), public(A), public(Na), generate(B, Tb).

2)maybeforged({B,Na,Kab,Tb}Kas) :- key(S, Kas), key(S, Kbs), public(B), generate(S, Kab), key(B, Kbs), key(A, Kas), available({A,Na,Tb}Kbs).

圖 3.6 Rules for the second and third encrypted items of Neumann Stubblebine security protocol

3.3 Rule 轉換

上一章節中，我們已經介紹我們的安全規約分析系統，其主要有三個部分：basic facts，rules 和 available predicates，在這一節中，我們將說明如何把系統中的三個元件轉成 Prolog 的格式。我們實作在第二章中如何建立每一個加密訊息的 rules，實作 MAIN，COTAIN，COLLECT，DERIVE 和 PRINTRULE Procedures 只是程式設計，因此我們不討論如何用 C，Java，或其它程式語言撰寫這四個 Procedures，但是我們重視的是在執行完這些 Procedures 後所產生的輸出：basic facts，rules 和 available predicates。首先是 basic facts，基本上，basic facts 已經是合乎 Prolog 的語法了，basic facts 是從使用者輸入得到的，且從上一節輸入格式的規定中得知，basic facts 已經是 Prolog 的 facts，惟一需要修改的是把 basic facts 轉成小寫，這樣才符合 Prolog 中 atom 的定義，從使用者的輸入中，我們已經有一個 facts 的資料庫。現在我們需要建立加密訊息的 rules，從第二章中我們知道每一個加密訊息在輸入的安全規約中，經過 PRINTRULE Procedure 就會產生相對應的 rule (第一個 communication action 除外)，但那樣的格式並不符合 Prolog 的語法，因爲 Prolog 中沒有“{”和“}”，所以我們必需將原來加密訊息的格式轉成 Prolog 中的 List Structure，如圖 3.5，用二層 List 來表示一個加密訊息，外層 List 的第一個欄位爲另一個存放所有元素的 List，第二個欄位則爲加密的金鑰，如果有一個加密訊息裡含有一個加密訊息，其表示方式爲[[[[elements, …, elements], key2]], key1]，當加密訊息經過轉換 List 的格式後，其格式已經符合 Prolog 的要求，但我們需要額外的修改，才能有效地找出一個安全規約的弱點，首先，如圖

3.6 中的 2)，僞造加密訊息{B,Na,Kab,Tb}Kas 需要加密訊息{A,Na,Tb}Kbs，但在詢問僞造加密訊息{B,Na,Kab,Tb}Kas 時，如果 Prolog 回答 true 時，表示加密訊息{B,Na,Kab,Tb}Kas 是可以被僞造的，但僞造時要經過其它的 rules (如在這個例子中的{A,Na,Tb}Kbs)並沒有被顯示出來，因此我們需要建立一條 derivation path，所以修改 maybeforged(M)爲 maybeforged(M,RR,LL)，其中 M 爲加密訊息，RR 爲臨時的 derivation path，LL 爲最後成功時的 derivation path，相對地，在每一個 maybeforged 中的 available 就必須負責建立臨時的 derivation path 的責任，因此修改成 available(N, [[S, CA-nth], M | RR], LL)，M 爲 maybeforged 中的加密訊息，N 爲加密訊息 M 所依賴的加密訊息，S 爲 M 的發送者，CA 爲目前第幾個 communication action，nth 爲目前 communication action 的第幾個訊息，如圖 3.6 中的 2)就修改成如下：

maybeforged([[B,NA,KAB,TB],KAS], RR, LL) :- key(S, Kas), key(S, Kbs), public(B), generate(S, Kab), key(B, Kbs), key(A, Kas), available([[A,NA,TB],KBS], [[S,3-1], [[B,NA,KAB,TB],KAS]|RR], LL).

接著，因爲 available 可能會用相同的 rule 去詢找所需要的加密訊息 N，爲了避免無限迴圈發生，我們在每一個 available 的詢問前加入\+member(M, RR)，\+和 member 爲 Prolog 內建的規則，\+爲邏輯的 NOT，如果 Element 是 List 中的一個元素，member(List, Element)爲 true。修改後如下：

maybeforged([[B,NA,KAB,TB],KAS], RR, LL) :- key(S, Kas), key(S, Kbs), public(B), generate(S, Kab), key(B, Kbs), key(A, Kas), \+member([[B,NA,KAB,TB],KAS], RR), available([[A,NA,TB],KBS], [[S,3-1], [[B,NA,KAB,TB],KAS]|RR], LL).

注意，上面這個 rule 是由伺服器發送出來的加密訊息，因此此加密訊息只有伺服器才能送出的，因此再次修改如下，其中 same(X, Y)表示如果 Y 或 X 是一變數而另一個爲值，變數那個就會等於值，注意 same(S,s)，S 爲一變數代表此加密訊息的發送者，而 s 爲值，表示伺服器。注意，因爲 same 這個規則不是 Prolog 內建，所以在輸出時，我們必需加“same(X,X).”到輸出的檔案裡。

maybeforged([[B,NA,KAB,TB],KAS], RR, LL) :- same(S,s), key(S, Kas), key(S, Kbs), public(B), generate(S, Kab), key(B, Kbs), key(A, Kas), \+member([[B,NA,KAB,TB],KAS], RR), available([[A,NA,TB],KBS], [[S,3-1], [[B,NA,KAB,TB],KAS]|RR], LL).

最後我們修改 public，如果一個 rule 中沒有 available，那表示此 rule 並沒有依賴其它 rules，接著如果此 rule 有 public，但 public 中的參數爲一變數，表示此變數可以爲任意值，因此我們改變原有的 public(X)爲(var(X) -> same(X, '_$'); public(X))，其中$爲任意數值，var 爲 Prolog 內建的規則，如果 X 爲變數，var(X)爲 true，此用意是使 public 的值不會因爲詢問時所用的值使結果爲特定的值，也就是說，所以如果在詢問後的結果中看到"_"開頭的，表示其變數可爲任意值 (如 2.2 中的範例)，因此圖 3.6 的 1)修改如下：

maybeforged([[A,NA,TB],KBS], RR, LL) :- key(B, KBS), (var(A) -> same(A, '_0'); public(A)), (var(NA) -> same(NA, '_1'); public(NA)), generate(B, TB), \+member([[A,NA,TB],KBS], RR), same([[B, 2-2], [[A,NA,TB],KBS]| RR], LL).

因此從第二章 PRINTRULE Procedure 得到的 rule，需要做下面四個修改動作：

修改一：修改 maybeforged(M)和 available(N)分別爲 maybeforged(M,RR,LL)和 available(N, [[S, CA-nth], M | RR], LL)，S 爲訊息 M 的發送者，CA 爲目前第幾個 communication action，nth 爲目前 communication action 的第幾個加密訊息。

修改二：在 available(N, [[S, CA-nth], M | RR], LL)之前加入\+member(M, RR)。

修改三：如果訊息 M 的發送者爲伺服器，加入 same(S,s)到條件裡。注意，在我們的實作中，我們以 s 表示爲何服器。

修改四：如果 rule 中有 public 的條件但沒有 available，以(var(X) -> same(X, '_$'); public(X))取代原有的 public(X)，$爲任意數值，並加入\+member(M, RR), same([[S, CA-nth], M| RR], LL).

最後，因爲 available 並不會因安全規約的改變而有所變化，但爲了配合 maybeforged 的修改，因此必需由

available(W) :- public(W).

available(W) :- maybeforged(W).

修改成

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

3.4 使用者介面

上一節中，我們已經介紹如何修改加密訊息的 rules，在輸出檔中除了有 facts 以外，也如入了 rules 和 available predicates，但是我們還必需提供使用者介面，以便使用者藉由使用者介面可以分析安全規約的安全性，因此我們加入下面的規則到輸出檔中：

forge(A, B):- available(A, [], B).

如果當使用者詢問 forge(A, B)且系統回 true 的話，表示一個攻擊者就可以經由路徑 B 得到訊息 A；注意 B 必需爲一變數，A 可以是變數，atom 或變數和 atom 的組合來表示一個未加密或加密訊息，加密訊息就必需符合 List Structure 的結構，如 3.1 節中介紹，下面爲一個簡單的範例，詢問加密訊息{{A,KAB,TB},KBS}。

forge([[a,kab,tb],kbs], PATH)

第四章 系統測試和安全規約分析

4.1 分析方法

在這一節中，我們介紹四種分析方法，藉由使用者介面 forge 去找出一個安全規約的弱點，下面列出四種方法：

1.如果安全規約有建立安全金鑰的機制時，詢問安全金鑰，如”forge(kab, PATH)”, 如果系統回 true 的話，表示此安全金鑰可以從 PATH 中得到，因此此安全規約就有這個弱點。

2.如果安全規約有建立安全金鑰的機制時，詢問每一個含有安全金鑰的加密訊息，但使安全金鑰的位置爲變數，如”forge([[a, KAB, tb], kbs], PATH)”， 如果系統回 true 的話且變數 KAB 不是所要建立的安全金鑰 (如 kab)，表示此訊息經由 PATH 產生，可以代取帶原來的訊息，因此使所要建立的安全金鑰爲其它值。

3.詢問在安全規約中的每一個加密訊息，如果有一個加密訊息所詢問出來的 PATH 不是安全規約中所預期的路徑，表示此安全規約可能有弱點存在，經由分析 PATH 中的資料，去推斷可能的弱點。

4.詢問每一個攻擊者要欺騙某一個參與者所需要的加密訊息，如果每一個加密訊息都可以僞造的話，攻擊者就可以藉由每一個所得到的 PATHs 去欺騙參與者，注意，如果攻擊者只是在雙方參與者之間做正常 relay 的作動話，此方式並不是一個攻擊。

在進入下一節時，我們解釋如果一個 forge 的詢問成功的話，相對在 PATH 中的資訊爲何，其如圖 4.1。

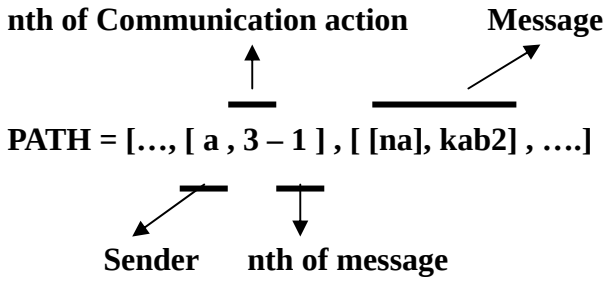


圖 4.1 Meaning of information in PATH

4.2 系統測試

在這節中，我們以 Neumann Stubblebine，Woo And Lam 和 Andrew Secure RPC 三個 Security Protocols 爲實驗範例 (在附錄一中例出其它範例)，爲了方便解說，我們在分析時，會直接找出弱點。首先，我們先看 Neumann Stubblebine Security Protocol，圖 3.3 爲此安全規約的輸入格式，經過程式轉換後，下列爲輸出的內容：

maybeforged([[A,NA,TB],KBS], RR, LL) :- key(B, KBS), generate(B, TB), (var(A) -> same(A, '_0'); public(A)), (var(NA) -> same(NA, '_1'); public(NA)), \+same(B, A), \+member([[A,NA,TB],KBS], RR), same([[B, 2-2], [[A,NA,TB],KBS]| RR], LL).

maybeforged([[B,NA,KAB,TB],KAS], RR, LL) :- same(S,s), key(A, KAS), key(B, KBS), key(S, KAS), key(S, KBS), generate(S, KAB), public(B), \+member([[B,NA,KAB,TB],KAS], RR), available([[A,NA,TB],KBS], [[S,3-1], [[B,NA,KAB,TB],KAS]|RR], LL), \+same(S, A), \+same(S, B).

maybeforged([[A,KAB,TB],KBS], RR, LL) :- same(S,s), key(S, KBS), generate(S, KAB), \+member([[A,KAB,TB],KBS], RR), available([[A,NA,TB],KBS], [[S,3-2], [[A,KAB,TB],KBS]|RR], LL), \+same(S, A).

maybeforged([[NB],KAB], RR, LL) :- key(A, KAS), generate(A, NA), public(NB), \+member([[NB],KAB], RR), available([[B,NA,KAB,TB],KAS], [[A,4-2], [[NB],KAB]|RR], LL), \+same(A, B).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(s,kas).

key(a,kas).

key(s,kbs).

key(b,kbs).

generate(a,na).

generate(b,nb).

generate(b,tb).

generate(s,kab).

public(a).

public(na).

public(b).

public(nb).

public(nb).

在 Prolog 的環境中，我們用 Prolog 內建的指令 consult 將轉換出來的檔案載入，用法如下：

?- consult(‘filename.pl’).

假如我們要攻擊 B 時，我們就必需傳送 B 在安全規約所有接收到的訊息，其爲第一個和第四個 communication actions，在第一個 communication action 中的所有訊息都是未加密訊息，所以我們可以很容易僞造，但在第四個 communication action 中的所有訊息都爲加密訊息，因此我們就必需想辦法僞造，注意在第四個 communication action 中的第二個加密訊息爲{Nb}Kab，其 Nb 爲公開的，而 Kab 是從第四個 communication action 中的第一個加密訊息中得到的，因此我們用上一節中所提出的第二種分析方法，詢問 A 在最後一步送出的加密訊息，如下：

?- forge([[a,K,tb],kbs],L).

其有一結果爲：

K = '_1'

L = [[b,2-2],[[a,'_1',tb],kbs]]

從結果中，我們發現變數 K 可以爲任意值，且此加密訊息可從 B 在第二 communication action 中的第二個加密訊息得到，因此攻擊方式如下：

(i1) (A)Z->B: A, Na

(i2) B->(S)Z: B, { A, Na, Tb }Kbs, Nb

(i4) (A)Z->B: { A, Na, Tb }Kbs, { Nb }Na

接下來我們分析 Woo And Lam Security Protocol，其輸入內容爲下：

ASSUMPTION:

key(S,Kas).

key(A,Kas).

key(S,Kbs).

key(B,Kbs).

generate(B,Nb).

PROTOCOL:

A->B: A

B->A: B, Nb

A->B: { A, B, Nb }Kas

B->S: { A, B, { A, B, Nb }Kas }Kbs

S->B: { A, B, Nb }Kbs

經過程式轉換後的輸出爲：

maybeforged([[A,B,NB],KAS], RR, LL) :- key(A, KAS), (var(B) -> same(B, '_0'); public(B)), (var(NB) -> same(NB, '_1'); public(NB)), \+same(A, B), \+member([[A,B,NB],KAS], RR), same([[A, 3-1], [[A,B,NB],KAS]| RR], LL).

maybeforged([[A,B,LABEL_0],KBS], RR, LL) :- key(B, KBS), (var(A) -> same(A, '_0'); public(A)), (var(LABEL_0) -> same(LABEL_0, '_1'); public(LABEL_0)), \+same(B, A), \+member([[A,B,LABEL_0],KBS], RR), same([[B, 4-1], [[A,B,LABEL_0],KBS]| RR], LL).

maybeforged([[A,B,NB],KBS], RR, LL) :- same(S,s), key(A, KAS), key(B, KBS), key(S, KAS), key(S, KBS), \+member([[A,B,NB],KBS], RR), available([[A,B,[[A,B,NB],KAS]],KBS], [[S,5-1], [[A,B,NB],KBS]|RR], LL), \+same(S, B), \+same(S, A).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(s,kas).

key(a,kas).

key(s,kbs).

key(b,kbs).

generate(b,nb).

public(b).

public(nb).

public(a).

載入 Prolog 的環境後，假如我們要攻擊 B，如上例一樣，必需傳送給 B 在安全規約所有接收的訊息，B 在安全規約的第一個，第三個和第五個 communication actions 分別收到 A，M 和{A, B, Nb}Kbs，注意，第三個 communication action 收到的訊息是一個用 Kas 加密的訊息，所以對 B 而言，它只是一個看不懂且不需要了解的加密訊息 (如之前所說的，分析時，必需以個別參與者的觀點看待一個安全規約)，所以我們只要能僞造{A, B, Nb}Kbs，我們就可以用 A 的身分去欺騙 B，因此我們詢問 {A, B, Nb}Kbs：

?- forge([[a,b,nb],kbs], PATH).

其有一結果如下：

PAHT = [[b,4-1],[[a,b,nb],kbs]]

我們可以從 B 的第四個 communication action 中得到此加密訊息，其攻擊方式爲下：

(i1) (A)Z->B: A

(i2) B->(A)Z: B, Nb

(i3) (A)Z->B: Nb

(i4) B->(S)Z: { A, B, Nb }Kbs

(i5) (S)Z->B: { A, B, Nb }Kbs

最後我們分析 Andrew Secure RPC Security Protocol，其輸入內容爲下：

ASSUMPTION:

key(A, Kab).

key(B, Kab).

generate(B, Kab1).

generate(A, Na).

generate(B, Nb).

generate(B, Nb1).

PROTOCOL:

A->B: A, { Na }Kab

B->A: { Na, Nb }Kab

A->B: { Nb }Kab

B->A: { Kab1, Nb1 }Kab

轉換後的輸出如下：

maybeforged([[NA,NB],KAB], RR, LL) :- key(B, KAB), generate(B, NB),
\+member([[NA,NB],KAB], RR), available([[NA],KAB], [[B,2-1],
[[NA,NB],KAB]|RR], LL).

maybeforged([[NB],KAB], RR, LL) :- key(A, KAB), generate(A, NA),
\+member([[NB],KAB], RR), available([[NA,NB],KAB], [[A,3-1], [[NB],KAB]|RR],
LL).

maybeforged([[KAB1,NB1],KAB], RR, LL) :- key(B, KAB), generate(B, KAB1),
generate(B, NB1), \+member([[KAB1,NB1],KAB], RR), available([[NB],KAB],
[[B,4-1], [[KAB1,NB1],KAB]|RR], LL).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR],
LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(a,kab).

key(b,kab).

generate(b,kab1).

generate(a,na).

generate(b,nb).

generate(b,nb1).

public([[na],kab]).

public(a).

從這個安全規約中，我們可看出此安全規約是利用之前所建立的安全金鑰(Kab)建立一把新的金鑰(Kab1)，我們用上一節中所提出的第二種分析方法詢問一下：

?- forge([[K,nb1],kab],PATH).

其有一結果如下：

K = na

PATH = [[public,[[na],kab]],[b,2-1],[[na,nb1],kab]]

從結果中我們知道，如果我們得到一個訊息{Na}Kab，然後送給 B，B 就在第二個 communication action 中產生{Na ,Nb1}Kab，因此我們就可以在第四個 communication action 時把此加密訊息傳送給 A，使 A 誤以爲 Na 爲新產生的安全金鑰，或許有人會質疑，在規約中，B 在第二個和第四個 communication actions 分別產生 Nb 和 Nb1，而此二個 Nonce 是不一樣的，但是上述的攻擊方式是用同一個 Nonce Nb1，注意到，在規約中，第四個 communication action 所產生的 Nb1 對 A 是沒有意義的，因爲 A 是第一次收到 Nb1，所以並沒有那樣的問題存在，其攻擊方式如下：

(i1) A->B: A, { Na }Kab

(i2) B->A: { Na, Nb }Kab

(i3) A->B: { Nb }Kab

(i4) B->(A)Z: { Kab1, Nb1 }Kab

(i4) (B)Z->A: { Na, Nb }Kab

上面的攻擊方式對攻擊者而言，似乎沒有利益，但是上面的 run 執行完時，B 認爲所建立新的安全金鑰爲 Kab1，但 A 認爲是 Na，因此雙方的認知不同。

第五章 結論和未來展望

本文中，我們已經提出一個安全規約分析系統可以正確地找出利用訊息相似特性產生的攻擊方式，此分析系統主要由三個元件構成：basic facts，rules 和 available predicates，在我們的系統裡，因爲 basic facts 是每次執行安全規約時所需要的事前資料，而 rules 是說明一個安全規約內的加密訊息如何產生，因此它與個別的 run 是無關的，所以我們就可以藉由增加 basic facts（也就是資料庫），使它擁有多個 run 時所需要的事前資料，因此我們在分析上，就有多個 run 同時執行的情況產生，此外我們系統的 scalability 也很高，如在上文中，我們只有用 key 來表示參與者之間和參與者與伺服器之間的安全金鑰關係，這樣的方法同常是用在使用 symmetric key 的安全規約，但只要些微的改變就可以適用於 asymmetric key 的安全規約，如將 key 修改成 private_key 和 public_key，並且在建立 rules 也把 key 修改成相對的 private_key 或 public_key。在表示隱含的關係時，我們用 key 和 generate 這兩個關係，如果有其它類型的安全規約有這兩個關係無法含蓋的隱含關係存在或需要更明確的關係，而擴充隱含關係也可以很容易地加入系統中，進而更明確地說明不同類型的需要，所以我們的系統可以提供不同類型的安全規約分析。相同地，我們目前只針對單一個安全規約分析其安全性，但在系統中，一個安全規約分析中的加密訊息都用一個相對應的 rule 表示，而 rule 中的依賴性用 availablep predicate 來表示，且 available predicate 會詢問是否要求的訊息可以獲得，因此我們可以分析當多個安全規約一起存在時，是否因此有弱點存在，例如在一個安全規約中的某一個加密訊息可能很容易從另一個安全規約中的某一個加密訊息得到，也就是訊息相似性。

在我們的實作中，我們可以看出詢問時，有些結果看似不合理，例如在 Nonce 的位置出現一個 Timestamp，此狀況的產生是因爲我們只用 generate 來表示參與者和參與者所產生的東西其之間關係，如果把 generate 的關係細分爲 Nonce 和 Timestamp 兩種，那在分析時，把 Nonce 拿來當 Timestamp 的可能性就不見了，因此有些潛在的弱點也就無法發現，如上節中對 Neumann Stubblebine Security Protocol 的攻擊方法，把 Nonce 拿來當所建立的安全金鑰 Key，所以說，越嚴密的的條件，越不容易找出潛在的弱點，越寬鬆的條件，出現越多不可能發生的況狀，因此如何達得更有效地找出潛在的弱點成爲以後的目標。

參考文獻

[1] M. Burrows, M. Abadi, and R. Needham, “A logic of authentication”, *ACM Trans. Computer Systems*, Vol. 8, No. 1, pp. 18-36, 1990.

[2] J. Clark and J. Jacob, “A survey of authentication protocol literature: Version 1.0”, 1997. http://www-users.cs.york.ac.uk/~jac/under the link \Security Protocols Review.

[3] W. Diffie and M.E. Hellman, “New directions in cryptography”, *IEEE Trans. on Information Theory*, IT-22, pp. 644-654, November 1976.

[4] D. Gries, *The Science of Programming*, Springer-Verlag, New York, 1981.

R.S. Pressman, *Software Engineering*, 3rd ed., McGraw-Hill, New York, 1992.

[5] C.A.R. Hoare, “An axiomatic basic for computer programming”, *Communications of ACM*, Vol. 12, No. 10, pp. 576-580, October 1969.

[6] R.E. Hodel, *An Introduction to Mathematical Logic*, PWS Publishing, Boston, 1995.

[7] P. Janson, G. Tsudik and M. Yung, “Scalability and flexibility in authentication services: The KryptoKinght Approach”, *Proceedings of INFOCOM'97*, Vol. 2, pp. 725 -736, 1997.

[8] A. Kehne, J. Schonwalder, and H. Langendorfer, “A nounce-based protocol for multiple authentication”, *AMC Operating Systems Review*, Vol. 26, No. 4, pp. 84-89, October 1992.

[9] J. Kelly, *The Essence of Logic*, Prentice-Hall, New York, 1997.

[10] C. Laferriere and R. Charland, “Authentication and authorization techniques in distributed systems”, *Proceedings of IEEE 1993 International Carnahan Conference on Security Technology*, pp. 164-170, 1993.

[11] M. Naor, “Deniable ring authentication”, Crypto 2002, August 2002.

[12] R.M. Needham and M.D. Schroeder, “Using encryption for authentication in large networks of computers”, *Communications of ACM*, Vol. 21, No. 12, pp. 993-999, December 1978.

[13] D. Nessett, “A critique of the Burrows, Abadi and Needham logic”, *ACM Operating Systems Review*, pp.35-38, April 1990.

[14] B.C. Neuman and S.G. Stubblebine, “A note on the use of the timestamps as nounces”, *ACM Operating Systems Review*, Vol. 27, No. 2, pp. 10-14, April 1993.

[15] B. Clifford Neuman and Theodore Ts'o, “Kerberos: An authentication service for computer network”, *IEEE Communications Magazine*, Vol. 32, No. 9, pp. 33-38, September 1994.

[16] A.W. Roscoe, “Intensional specifications of security protocols”, *Proceedings of 9th IEEE Computer Security Foundations Workshop,* pp. 28-38,1996.

[17] S.P. Shieh and W.H. Yang, “An authentication and key distribution system for open network systems”, *ACM Operating Systems Review*, Vol. 30, No. 2, pp. 32-41, 1996.

[18] J.G. Steiner, C. Neuman, and J.I. Schiller, “An authentication service for open network systems”, *Proceedings of the Winter 1988 USENIX Conference*, pp. 191-202, 1988.

[19] A.A. Stolyar, “*Introduction to Elementary Mathematical Logic”*, Dover, New York, 1970.

[20] W. Yang (2005), “Uncovering Attacks On Security Protocols”, in *Proceedings of International Conf. Information Technology and Applications*, Sydney, Australia, pp. 4-7, July 2005.

附錄一

下面我們列出三個安全規約的輸入內容，轉換後的輸出，分析結果以及攻擊方式，三個安全規約爲：Adapted Andrew protocol，Denning-Sacco shared key 和 BAN simplified version of Yahalom。

Adapted Andrew protocol：

輸入內容 (注意，給兩個 run 的資料)：

ASSUMPTION:

key(A,Kab).

key(B,Kab).

generate(B,Kab1).

generate(A,Na).

generate(B,Nb).

generate(A,Kab2).

generate(A,Na1).

generate(B,Nb1).

PROTOCOL:

A->B: A, Na

B->A: { Na, Kab1 }Kab

A->B: { Na }Kab1

B->A: Nb

轉換後的輸出：

maybeforged([[NA,KAB1],KAB], RR, LL) :- key(B, KAB), generate(B, KAB1), (var(NA) -> same(NA, '_0'); public(NA)), \+member([[NA,KAB1],KAB], RR), same([[B, 2-1], [[NA,KAB1],KAB]| RR], LL).

maybeforged([[NA],KAB1], RR, LL) :- key(A, KAB), generate(A, NA), \+member([[NA],KAB1], RR), available([[NA,KAB1],KAB], [[A,3-1], [[NA],KAB1]|RR], LL).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(a,kab).

key(b,kab).

generate(b,kab1).

generate(a,na).

generate(b,nb).

generate(a,kab2).

generate(a,na1).

generate(b,nb1).

public(na).

public(a).

分析結果，當要攻擊 B 時，我們要想辦法偽造一個加密訊息{Na}Kab1：

?- forge([[na],K], PATH).

K = kab2

PATH = [[a,2-1],[[na,kab2],kab],[a,3-1],[[na],kab2]] (Principal B does not involve)

攻擊方式：

i1 A->(B)Z : A, Na

ii1 (B)Z->A : B, Na

ii2 A->(B)Z : { Na, Kab2 }Kab #([a,2-1])

i2 (B)Z->A : { Na, Kab2 }Kab

i3 A->(B)Z : { Na }Kab2 #([a,3-1])

ii3 (B)Z->A : { Na }Kab2

i4 (B)Z->A : Nz

ii4 A->(B)Z : Na2

Denning-Sacco shared key：

輸入內容：

ASSUMPTION:

key(A,Kas).

key(B,Kbs).

key(S,Kas).

key(S,Kbs).

generate(S,Kab).

generate(S,T).

PROTOCOL:

A->S: A, B

S->A: {B, Kab, T,{Kab,A,T}Kbs}Kas

A->B: {Kab,A,T}Kbs

轉換後的輸出：

maybeforged([[B,KAB,T,[[KAB,A,T],KBS]],KAS], RR, LL) :- same(S,s), key(A, KAS), key(B, KBS), key(S, KAS), key(S, KBS), generate(S, KAB), generate(S, T), (var(A) -> same(A, '_0'); public(A)), (var(B) -> same(B, '_1'); public(B)), \+same(S, A), \+same(S, B), \+member([[B,KAB,T,[[KAB,A,T],KBS]],KAS], RR), same([[S, 2-1], [[B,KAB,T,[[KAB,A,T],KBS]],KAS]| RR], LL).

maybeforged(LABEL_0, RR, LL) :- key(A, KAS), \+member(LABEL_0, RR), available([[B,KAB,T,LABEL_0],KAS], [[A,3-1], LABEL_0|RR], LL).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(a,kas).

key(b,kbs).

key(s,kas).

key(s,kbs).

generate(s,kab).

generate(s,t).

generate(s,kab1).

generate(s,t1).

public(b).

public(a).

分析結果，當要攻擊 B 時，我需要在第三個 communication action 時傳送 {Kab,A,T}Kbs 給 B：

?- forge([[K,a,t],kbs], PATH).

K = kab

PATH = [[s,2-1],[[b,kab,t,[[kab,a,t],kbs]],kas],[a,3-1],[[kab,a,t],kbs]] (Principal B does not involve)

攻擊方式：

i1 A->S: A, B

i2 S->A: { B, Kab, T, { Kab, A, T }Kbs }Kas #([s,2-1])

i3 A->B: { Kab, A, T }Kbs #([a,3-1])

ii1 A->S: A, B

ii2 S->A: { B, Kab1, T1, { Kab1, A, T1 }Kbs }Kas

ii3 A->(B)Z: { Kab1, A, T1 }Kbs

ii3 (A)Z->B: { Kab, A, T }Kbs

BAN simplified version of Yahalom：

輸入內容：

ASSUMPTION:

key(A,Kas).

key(B,Kbs).

key(S,Kas).

key(S,Kbs).

generate(A,Na).

generate(B,Nb).

generate(S,Kab).

PROTOCOL:

A->B: A, Na

B->S: B, Nb, {A, Na }Kbs

S->A: Nb, {B, Kab, Na}Kas, {A, Kab, Nb}Kbs

A->B: {A, Kab, Nb}Kbs, {Nb}Kab

轉換後的輸出：

maybeforged([[A,NA],KBS], RR, LL) :- key(B, KBS), (var(A) -> same(A, '_0'); public(A)), (var(NA) -> same(NA, '_1'); public(NA)), \+same(B, A), \+member([[A,NA], KBS], RR), same(([[B, 2-3], [[A,NA],KBS]| RR], LL).

maybeforged([[B,KAB,NA],KAS], RR, LL) :- same(S, s), key(A, KAS), key(B, KBS), key(S, KAS), key(S, KBS), generate(S, KAB), public(B), \+member([[B,KAB,NA],KAS], RR), available([[A,NA],KBS], [[S, 3-2], [[B,KAB,NA],KAS]|RR], LL), \+same(S, A), \+same(S, B).

maybeforged([[A,KAB,NB],KBS], RR, LL) :- same(S, s), key(S, KBS), generate(S, KAB), public(NB), \+member([[A,KAB,NB],KBS], RR), available([[A,NA],KBS], [[S,3-3], [[A,KAB,NB],KBS]|RR], LL), \+same(S, A).

maybeforged([[NB],KAB], RR, LL) :- key(A, KAS), generate(A, NA), public(NB), \+member([[NB], KAB], RR), available([[B,KAB,NA],KAS], [[A,4-2], [[NB],KAB]|RR], LL), \+same(A, B).

available(Y, RR, LL) :- current_predicate(public/1), public(Y),same([[public, Y]|RR], LL).

available(Y, RR, LL) :- maybeforged(Y, RR, LL).

forge(A, B):- available(A, [], B).

same(X, X).

key(a,kas).

key(b,kbs).

key(s,kas).

key(s,kbs).

generate(a,na).

generate(b,nb).

generate(s,kab).

public(a).

public(na).

public(b).

public(nb).

分析結果，當要假冒 B 欺騙 A 時，我們需要在第三個 communication action 時送出 Nb 和{B, Kab, Na}Kas 和{A, Kab, Nb}Kbs，因爲 A 不知道金鑰 Kbs，所以我們只要想辦僞造{B, Kab, Na}Kas：

?- forge([[b, kab, na],kas], PATH).

PATH = [[a, 2-3], [[b, '_1'],kas], [s,3-3], [[b,kab,na],kas]]

攻擊方式：

i1 A -> (B)Z : A, Na

ii2 (B)Z -> A : B, Na

ii2 A -> (S)Z : A, Na', { B, Na }Kas #([a, 2-3])

iii2 (A)Z -> S : A, Na, { B, Na }Kas

iii3 S -> (B)Z : Na, { A, Kab, Na }Kbs, { B, Kab, Na }Kas #([s, 3-3])

i3 (S)Z -> A : Nz, { B, Kab, Na }Kas, { A, Kab, Na }Kbs

i4 A -> (B)Z : { A, Kab, Na }Kbs, { Nz }Kab

附錄二

這裡我們介紹三個安全規約，這三個安全規約都有弱點存在，且是我們系統能力範圍外的攻擊方式，也就是我們系統無法找到的弱點，這是因爲這些攻擊方式且不符合我們在 2.5 節所提的相似訊息攻擊方式。首先，附圖 2.1 爲 Kao Chow Authentication protocol v.1 [2]，在假設攻擊者知道的 Kab 情況下，其攻擊方式爲：

(i1) A -> S : A, B, Na

(i2) S -> B : { A, B, Na, Kab }Kas, { A, B, Na, Kab }Kbs

(i3) B -> A : { A, B, Na, Kab }Kas, { Na }Kab, Nb

(i4) A -> B : { Nb }Kab

(ii1) omitted

(ii2) (S)Z -> B : { A, B, Na, Kab }Kas, { A, B, Na, Kab }Kbs

(ii3) B -> (A)Z : { A, B, Na, Kab }Kas, { Na }Kab, N'b

(ii4) (A)Z -> B : { N'b }Kab

上述的攻擊方式主要是因爲攻擊者知道先前的 Kab，所以可以假冒 A 的名義再一次和 B 建立一條以金鑰 Kab 爲加密金鑰的管道，但在我們的系統中，我們並沒有這樣的前提假設，也就是說，攻擊者在前提下並不知道 Kab，因爲假設集合 BF 是存放設計者認爲正常執行一個安全規約前所必要的條件，所以我們的系統並不包含因其它假設而出現弱點的攻擊方式。

A -> S : A, B, Na

S -> B : { A, B, Na, Kab }Kas, { A, B, Na, Kab }Kbs

B -> A : { A, B, Na, Kab }Kas, { Na }Kab, Nb

A -> B : { Nb }Kab

附圖 2.1 Kao Chow Authentication protocol v.1

接下來我們看另一個例子，附圖 2.2 爲 Otway Rees security protocol [2]，其攻擊方式如下：

(i1) A -> B : M, A, B, { Na, M, A, B }Kas

(i2) B -> S : M, A, B, { Na, M, A, B }Kas , { Nb, M, A, B }Kbs

(i3) S -> B : M, { Na, Kab }Kas, { Nb, Kab }Kbs

(i4) B -> (A)Z : M, { Na, Kab }Kas

(i4) (B)Z -> A : M, { Na, M, A, B }Kas

上面的攻擊方式是利用在第一個 communication action 中的{ Na, M, A, B } Kas 取代在第二個 communication action 中的{ Na, Kab } Kas，使 A 認爲“M,A,B”爲所建立的安全金鑰 Kab，但是在我們的系統中，如果要攻擊 A 時，我們就要詢問 A 所必需收到的加密訊息是否都可從 rules 中得到，因此我們會詢問是否可以得到加密訊息{Na, Kab}Kas，但系統在尋找過程時，並不會找到加密訊息{Na, M, A, B}Kas，因爲這兩個加密訊息的格式是不一樣(不相似的訊息)，所以說，我們的系統並不包含因用不相似訊息而出現弱點的攻擊方式。附圖 2.3 爲 Woo And Lam security protocol，我們利用這個安全規約說明何謂 相似訊息，看看在第四個 communication action 中的{A, B, {A, B, Nb}Kas}Kbs 和第五個 communication action 中的{A, B, Nb}Kbs，從表面上看來，這兩個加密訊息是不相似的，但仔細看看規約時，發現這兩個加密訊息是相似的，因爲當我們在建立 rule 時，是以發送者的觀點出發，也就是說，在第四個 communication action 的發送者爲 B，所以以 B 的觀點爲主，{A, B, {A, B, Nb}Kas}Kbs 被識爲{A, B, W}Kbs，其中 W 是 B 在第三個 communication action 中收到的訊息，因此{A, B, {A, B, Nb}Kas}Kbs 和{A, B, Nb}Kbs 是相似的，在第 4.2 節中，我們分析這個安全規約和說明攻擊方式，而攻擊方式就是利用這兩個加密訊息的相似特性。

A -> B : M, A, B, { Na, M, A, B } Kas

B -> S : M, A, B, { Na, M, A, B } Kas , { Nb, M, A, B } Kbs

S -> B : M, { Na, Kab } Kas, { Nb, Kab } Kbs

B -> A : M, { Na, Kab } Kas

附圖 2.2 Otway Rees security protocol

A->B : A

B->A : B, Nb

A->B : { A, B, Nb }Kas

B->S : { A, B, { A,B,Nb }Kas }Kbs

S->B : { A, B, Nb }Kbs

附圖 2.3 Woo And Lam security protocol

A -> B : A, Na

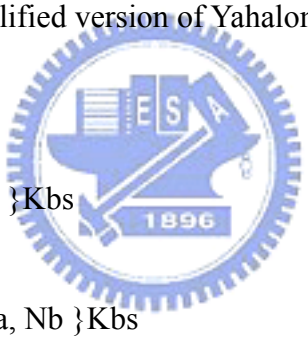
B -> S : B, Nb, { A, Na }Kbs

S -> A : Nb, { B, Kab, Na }Kas, { A, Kab, Nb }Kbs

A -> B : { A, Kab, Nb }Kbs, { Nb }Kab

附圖 2.4 BAN simplified version of Yahalom security protocol

最後，附圖 2.4 爲 BAN simplified version of Yahalom [2]，其攻擊方式爲：



(i1) A -> B : A, Na

(i2) B -> (S)Z : B, Nb, { A, Na }Kbs

(ii1) (A)Z -> B : A, Na, Nb

(ii2) B -> (S)Z : B, Nb, { A, Na, Nb }Kbs

(i3) omitted

(i4) (A)Z -> B : { A, Na, Nb }Kbs, { Nb }Na

從上面的攻擊方式中，我們可以很容易知道此攻擊方式是利用 B 在第二個 communication action 時，會把從第一個 communication action 中的所有訊息用安全金鑰 Kbs 加密，所以攻擊者在第二個 run 的第一個 communication action 中傳送 A, Na, Nb 給 B，因此攻擊者就可以得到{A, Na, Nb}Kbs，而此加密訊息剛好和第四個 communication action 中的{A, Kab, Nb}Kbs 相似(Kab 爲 Na)，而這個攻擊方式和上面討論 Otway Rees security protocol 時的攻擊方式是類似的，在第一個 communication action 中使 B 把 A, Na, Nb 識爲 A, “Na, Nb”，但是當我們在建立第二個 communication action 中{A, Nb}Kbs 的 rule 時，其 A 和 Nb 是 public，也就是說可以得到一個訊息爲{A, “Na, Nb”}Kbs，但這個加密訊息只有兩個元

素，和{A, Kab, Nb}Kbs(三個元素)是不相似的。另外一種攻擊方法如下：

(i1) A -> (B)Z : A, Na

(ii1) (B)Z -> A : B, Na

(ii2) A -> (S)Z : A, Na', { B, Na }Kas

(iii1) omitted

(iii2) (A)Z -> S : A, Na, { B, Na }Kas

(iii3) S -> (B)Z : Na, { A, Kab, Na }Kbs, { B, Kab, Na }Kas

(i2) omitted

(i3) (S)Z -> A : Nz, { B, Kab, Na }Kas, { A, Kab, Na }Kbs

(i4) A -> (B)Z : { A, Kab, Na }Kbs, { Nz }Kab

上面的攻擊方法就如同我們所提到利用訊息相似特性所產生的攻擊，這個攻擊是利用在第三個 communication action 中的兩個加密訊息{B, Kab, Na}Kas 和{A, Kab, Nb}Kbs 的相似格式所產生的，如果要以 B 的身分欺騙 A 時(上述的攻擊)，我們就要產生在第三個 communication action 中的訊息，也就是 Nb，{B, Kab, Na}Kas 和{A, Kab, Nb}Kbs，前提假設下，A 不知道安全金鑰 Kbs，所以{A, Kab, Nb}Kbs 對 A 來說只是一個數變，因此我們只想辦法產生加密訊息{B, Kab, Na}Kas，我們知道第三個 communication action 的發送者爲 S，所以 S 在第二個 communication action 中必需先收到 B，Nb 和{A, Na}Kbs，也就是說，第三個 communication action 中加密訊息{B, Kab, Na}Kas 的加密金鑰 Kas 是依據{A, Na}Kbs 中的 A，但金鑰 Kbs 只有 B 和 S 知道，可是第三個 communication action 中加密訊息{A, Kab, Nb}Kbs 的加密金鑰 Kbs 依據第二個 communication action 中的 B(未加密訊息)，所以我們可以藉由第三個 communication action 中的第二個加密訊息的 rule 產生我們要的{B, Kab, Na}Kas，如上述攻擊方式中的 iii3，而在 iii2 中的{B, Na}Kas 也可以用第二個 communication action 中加密訊息的 rule 推導出，如 ii2 中所表示，所以在我們的系統中就可以找出這個攻擊方法(利用相似特性的攻擊方式)，如附錄一中對此安全規約的分析。