修 士 論 文

セキュリティプロトコルの代数モデルに基づく形式化

指導教官　二木 厚吉

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

金城 直貴

2001 年 2 月 15 日

要 旨

近年、ネットワークセキュリティに対する関心が高まっており、その安全性の検証が重要となっている。特に、ネットワーク上での商取引や選挙への応用が期待されているセキュリティプロトコルの検証においては、形式手法が有効であると考えられている。

そこで本研究では、代数仕様言語 CafeOBJ を用いてセキュリティプロトコルの仕様記述を行い、その仕様の検討を行う。形式手法によるセキュリティプロトコルの検証に期待がなされ、いくつかの形式化が提案されている。その中から、G.Denker らによる代数仕様言語 Maude による Configuration を用いたセキュリティプロトコルの仕様記述と萩谷らが提案したネットワークのシステム状態を定義した形式化を取り上げ、認証プロトコルの一例である Needham-Shoroeder Public-Key Protocol を例題として、それぞれの形式化を書き換え論理による仕様、振舞仕様で記述し、セキュリティプロトコルの形式化技法として報告する。

目 次

第 1 章

はじめに

近年、インターネットに代表される広域情報ネットワークの普及に伴い、多くの通信プロトコルが考案された。考案されたプロトコルには、通信内容を暗号化し、通信者間の秘密通信を目的とするセキュリティプロトコルがある。セキュリティプロトコルの特徴を適用することで、ネットワーク上での安全な商取引や選挙などの実現が期待されている。

このように、情報ネットワークが大規模化、複雑化し、多彩なサービスが提供される一方、ネットワーク上において、発信者から特定の受信者に誤りのないデータをどのようにして確実に伝えるか、ネットワーク上での安全性をいかにして確保するかといった、ネットワークセキュリティへの関心が高まっている。セキュリティプロトコルは、ネットワーク上での安全な通信の実現が期待されたプロトコルであるが、仮にその安全性に欠陥があり、セキュリティ要件が満たされなかった場合、その適用が期待される領域を考えると、経済的、社会的な損失と危機は計り知れない。そのため、セキュリティプロトコルがそのセキュリティ要件を満たし、安全であることの保証が必要とされる。

一般に、ソフトウェアの開発段階で作成される仕様は、記述に漏れがなく、正確かつ、実現可能で検証やテストができることが要求される。仕様はソフトウェアのテストにかかるコストを軽減するだけではなく、最終的な品質も大きく左右するものであり、その信頼性は重要である。仕様記述の技法に形式仕様がある。形式仕様は、仕様の記述に形式的な記法を用いて記述された仕様である。そのため、仕様段階でシステムの整合性を数学的に検証することができ、検証の容易性、信頼性の点で他の手法よりも優れている。

セキュリティプロトコルを実用化する場合、あらかじめ、その安全性を保証する必要がある。しかし、それはしばしば人間の直感に頼っていた。形式手法による検証によって、良く知られたプロトコルの不具合が報告され、セキュリティプロトコルの検証方法として、形式手法への期待が高まっている。

本稿では、はじめにネットワークセキュリティについての説明を行い、ネットワーク上における危険性と必要とされるセキュリティ要件を明らかにし、これらの問題に対処するためのプロトコルである、セキュリティプロトコルについて説明する。次に、形式手法および、本研究でセキュリティプロトコルの仕様記述に用いる代数仕様言語 CafeOBJ について説明する。また、セキュリティプロトコルの形式化事例の中から、Denker らの Configuration を用いた仕様記述と、萩谷らのネットワークのシステム状態を定義した形式化に注目し、その枠組みについて説明する。

そして、これらの形式化に基づき、認証プロトコルの一例である Needham-Schroeder Public-Key Protocol を例題として取り上げ、CafeOBJ によって、それぞれ、書き換え論理による仕様記述、振舞仕様による記述というアプローチを取り、セキュリティプロトコルの形式化技法として報告する。

第 2 章

ネットワークセキュリティ

この章では、ネットワーク上の危険性を電子商取引を例に解説し、安全性を確保するためのセキュリティ要件を明らかにする。また、通信の安全性を確保するためのセキュリティティプロトコルについて説明する。

2.1 ネットワーク上の危険性

ネットワーク上で様々なサービスが受けられるようになり、インターネットが我々の生活基盤として浸透するにつれ、その安全性が重視されるようになった。いわゆる、インターネット上の電子商取引はもちろんのこと、個人間におけるメール等の私的な情報のやりとりにおいても、安全性が重要視されている。

では、ネットワーク上の安全性とは何か。従来からある情報セキュリティの捉え方で、情報を守るという着目のほかに、その情報が作成者のつくった通りの内容であるかという真正性や、情報が複数の当事者の手によって処理される場合の、処理に対する責任追求性なども重要でなる。そこで、ネットワークに潜む危険性を明らかにし、安全性について考える。

本研究で考慮する、ネットワーク上において考えられる危険性は、大きく以下の 3 つに分けられる。

2.1.1 なりすまし

A 氏は B 通販のオンラインショップを利用しようと試みた。(図 2.1)

A 氏は B 通販の商品が気に入り、注文をしようと B 通販の注文フォームに希望商品名とその数を入力し、送信した。もちろん、A 氏は B 通販との取引を行っているつもりだっ

たが、実際にはB通販とは何の関係もないC氏と通信を行っていた。

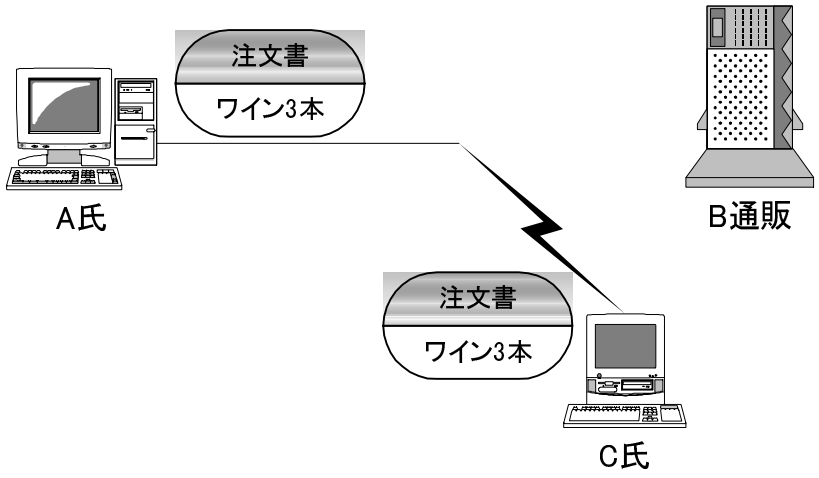

図 2.1: なりすましによる危険性

C氏はB通販とは何の関わりもないのにも関わらず、あたかもB通販であるかのようにみせかけ、取引を行っている。これは、**なりすまし**という行為である。A氏は何の疑いもなく通信を終了したが、残念ながら注文した品物は届かない。そればかりか、支払いのために相手にカード情報を伝えた場合、カード情報は不正に利用されてしまうだろう。これは、A氏の被害ばかりではなく、B通販にとっても信用に関わる問題となる。

2.1.2 改竄

A氏はB通販のオンラインショップを利用した買物を試みた。(図 2.2)

購入する商品が決まり、その商品をB通販が用意した注文フォームに入力し、送信した。

C氏はA氏が注文フォームで送信した情報を通信路上で一部差し替えてB通販に送信している。**情報の改竄**である。

A氏には、自分が希望したものとは異なる商品や注文数が届けられる。A氏は届いた商品とその数に驚き、返品を求めるかも知れない。また、2度とB通販を利用しないだろう。注文通りに誠実に対応したB通販は、代金を回収できないリスクを背負うことになる。

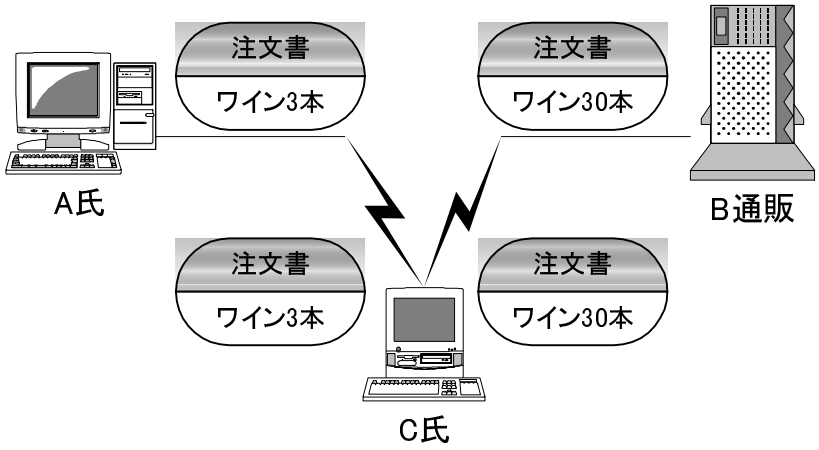

図 2.2: 原文の改竄による危険性

2.1.3 盗聴

A 氏は B 通販での注文を終え、代金の支払いを行うことになった。(図 2.3)

A 氏はカードによる支払いを希望するため、カードの暗証番号を入力し、B 通販に送信する。

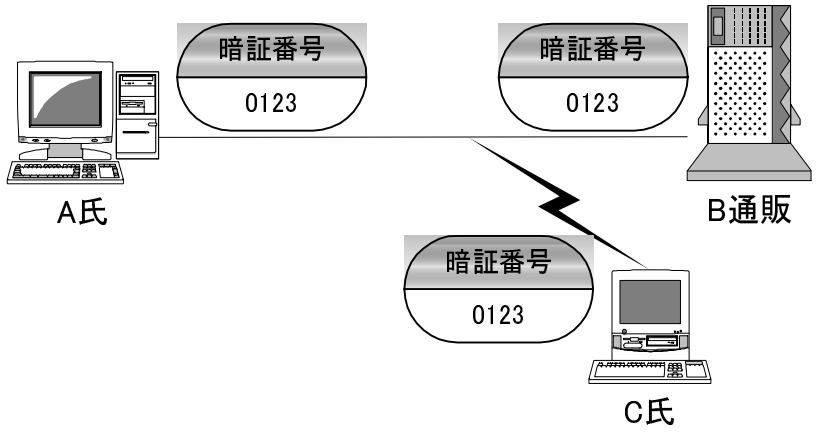

図 2.3: 情報の盗聴による危険性

A 氏のカードの暗証番号は、通信路上に流れる情報を観察している C 氏に見られた。**盗聴**である。

C 氏が盗聴によって知り得たカード情報を用いて、オンラインショッピング等で商品を

購入すれば、カードの不正利用が簡単に成立する。そして後日、A 氏の下へ、身に覚えのない請求書が届くことになる。B 通販側は正式な方法でカードの処理を行っているにも関わらず、いらぬ嫌疑が掛けられ、信用を落としかねない。

これらの不正行為は、現存の対面取引や電話による通信販売でも可能であろうが、大掛かりな仕掛けや体制が必要となる。しかし、ネットワーク上で試みる場合は、インターネット・サービス・プロバイダーの周辺など、さまざまな情報が集中するであろう絶好のポイントが存在し、そこに比較的簡単な仕掛けを用意することで不正行為が可能となる。

2.2 セキュリティ要件とセキュリティ確保

ネットワークの安全性を確保するためには、危険性を把握し、それにきちんと対処することである。つまり、2.1で挙げたネットワーク上の危険性を踏まえると次の 3 つの行為への対処が必要となる。

- なりすましへの対処
- 改竄への対処
- 盗聴への対処

このような不正行為を未然に防ぐことができれば、その通信システムは安全性の高いシステムだということが想像できる。これら、安全性を確保するために必要な要件を**セキュリティ要件**という。

情報セキュリティを考える場合には、いくつかの異なる観点があり、セキュリティ要件もそのシステムによって異なる。通信システム全体を対象とし、詳細にネットワークセキュリティを考えた場合、上記の要件のみでは必ずしも十分ではない。しかし、通信のみを考えた場合、上記要件を満たすことができれば、かなり安全だと考えられる。

現在、安全な通信を確保する方法として、**暗号化通信**が提案され、実現されている。さらに、暗号化通信に用いられる暗号と暗号技術を安全に利用するための要素技術として、共通鍵暗号方式[1]と公開鍵暗号方式[2]、ハッシュ関数[3]、乱数、識別などが研究されている。

暗号化通信やその要素技術を用いて、ネットワーク上のなりすましや情報の改竄、盗聴といった不正行為に対抗する。具体的な方法として、秘密通信とデジタル署名がある。

[1]暗号化と復号化に同じ鍵を用いる暗号方式。

[2]対になる 2 つの鍵を使ってデータの暗号化・復号化を行う暗号方式。秘密鍵で暗号化されたデータは対応する公開鍵で、公開鍵で暗号化されたデータは対応する秘密鍵でしか復号できない。

[3]与えられた原文から固定長の疑似乱数を生成する演算手法。

2.2.1 秘密通信

通信路上のメッセージを暗号化して送受信することで、通信を行う当事者以外の人に対する秘密通信を可能にする。暗号化には、共通鍵暗号方式と公開鍵暗号方式がある。

暗号文を平文に戻せるのは、鍵の情報を知っている人に限られ、それ以外の人はその暗号文の内容を知ることはできない。つまり、メッセージが盗聴されても、そのメッセージの内容を知ることができないため、不正利用ができなくなる。

さらに、ハッシュ関数を用いることで、通信路上で改竄が行われても検知することが可能である。メッセージを暗号化する前に、そのハッシュ値[4]を取り、暗号化したメッセージに付加して送信する。相手がメッセージを受信し、復号した際に、その平文のハッシュ値と付加されている値を比較することで、メッセージが経路の途中で改竄されていないか確認することができる。この場合、メッセージを比較する前に用いるハッシュ関数を決め、通信者は供に知っていなければならない。

2.2.2 デジタル署名

公開鍵暗号方式では秘密鍵は各人ごとに秘密に設定しているので、秘密鍵による暗号化で生成された情報は本人だけがつくり得る情報であり、デジタルな世界における署名・捺印相当の機能として利用される。これを**デジタル署名**という。

デジタル署名に併せて、本人を証明する電子的証明書を提示することで、本人認証が可能となり、なりすましに対抗することができる。電子的証明書とは、ある信用のおける第3者である認証局により発行されるもので、あらかじめ登録しておく必要がある。

認証局の発行する電子的証明書は登録された人がデジタル署名に使う公開鍵の持ち主であることを証明するものであり、デジタル署名に併せてこの電子的証明書を示すようにすれば、そのデジタル署名ができる人はその電子的証明書を発行した認証局に登録された人に相違ないことを確信する根拠となるものである。

デジタル署名を利用する本人認証機能は、ソフトだけで構成できることと、ネットワークとの親和性があることから、ネットワーク上での本人認証機能として利用されている。

2.3 セキュリティプロトコル

セキュリティプロトコルとは、ネットワーク上の通信者間でやり取りされるメッセージを暗号化し、通信者間の秘密通信の機能を持つ通信プロトコルである。セキュリティプロ

[4]ハッシュ関数により生成された値。

トコルは、ただ秘密通信を可能とするだけでなく、他者が通信に参加できないという秘密通信の利点をもって、両者の相互認証を目的とするものもある。

本人認証と秘密通信は、オープンなネットワーク上での、意図する特定の人との内密な会話を可能とする。この効果を期待し、現在セキュリティプロトコルは電子商取引や選挙などへの適用が期待され、標準化作業が進められている。

次に、セキュリティプロトコルの一例として、通信者相互の認証を目的とした公開鍵暗号方式プロトコル、Needham-Schroeder Public-Key Protocol (本稿中では、以下 NSPK Protocol と表記) を紹介する。

NSPK Protocol は、公開鍵暗号方式を用いたプロトコルを直感的に理解しやすいことから、公開鍵プロトコルの例題としてよく利用される。

2.3.1 Needham-Schroeder Public-Key Protocol

NSPK Protocol は、1978 年、R.Needham と M.Schroeder によって、公開鍵暗号方式を用いた、認証を目的とするプロトコルの一例として発表された [2]。プロトコルの手順は、全部で 7 ステップからなる。(図 2.4)

$$
\begin{aligned}
&1.\quad A \to S \quad : \quad A, B \\
&2.\quad S \to A \quad : \quad \{K_B, B\}_{K_S^{-1}} \\
&3.\quad A \to B \quad : \quad \{N_A, A\}_{K_B} \\
&4.\quad B \to S \quad : \quad B, A \\
&5.\quad S \to B \quad : \quad \{K_A, A\}_{K_S^{-1}} \\
&6.\quad B \to A \quad : \quad \{N_A, N_B\}_{K_A} \\
&7.\quad A \to B \quad : \quad \{N_B\}_{K_B}
\end{aligned}
$$

図 2.4: Needham-Schroeder Public-Key Protocol

ネットワーク上には、通信を行う A、B と公開鍵を管理する S が存在する。これら通信に参加可能な存在を主体 (principal) と呼ぶ。S の公開鍵は通信に参加しうる全ての主体が所持している。つまり、S の秘密鍵で暗号化されたメッセージは、全ての主体が復号可能で、その内容を確認することが可能である。

では、図 2.4を参照しながら、NSPK Protocol の手順を具体的に追う。

- STEP1

$$1.\ A \to S : A, B$$

B との通信を希望する A が、B の公開鍵を要求するメッセージ (A, B) を S に送信する (このメッセージを $message1$ と呼び、以下、各ステップで作成されるメッセージを同様に呼称していく)。

- STEP2

$$2.\ S \to A : \{K_B, B\}_{K_S^{-1}}$$

$message1$ を受信した S は、B の公開鍵 (K_B) と B の ID(B) を S の秘密鍵 (K_S^{-1}) で暗号化したメッセージ $(\{K_B, B\}_{K_S^{-1}})$ を A に送る $(message2)$。

- STEP3

$$3.\ A \to B : \{N_A, A\}_{K_B}$$

$message2$ を受信し、S の公開鍵で復号した A は B の公開鍵を手に入れた。それから、今回の B との通信に用いるノンス[5](N_A) と自分の ID(A) を B の公開鍵で暗号化 $(\{N_A, A\}_{K_B})$ し、B に送信する $(message3)$。

- STEP4

$$4.\ B \to S : B, A$$

$message3$ を受信し、自分の秘密鍵で復号した B はメッセージ中の ID(A) を確認し、S に A の公開鍵を要求する $(message4)$。

- STEP5

$$5.\ S \to B : \{K_A, A\}_{K_S^{-1}}$$

$message4$ を受け取った S は、A の公開鍵 (K_A) と A の ID を S の秘密鍵で暗号化したメッセージ $(\{K_A, A\}_{K_S^{-1}})$ を B に送る $(message5)$。

[5]ノンスに関しては 5.1で説明する

- STEP6

$$6.\ B \rightarrow A : \{N_A, N_B\}_{K_A}$$

$message5$を受信し、S の公開鍵で復号した B は A の公開鍵を手に入れた。そして、B は $message3$ 中のノンス (N_A) と、今回の通信に用いる自分のノンス (N_B) を生成し、これらノンスを A の公開鍵で暗号化し $(\{N_A, N_B\}_{K_A})$、A に送信する $(message6)$。

- STEP7

$$message7. A \rightarrow B : \{N_B\}_{K_B}$$

$message6$ を受信し、自分の秘密鍵で復号した A は $message6$ 中に、先に送った $message3$ に用いたノンス (N_A) が含まれていることを確認した。$messege3$ は、B の公開鍵で暗号化しているため、原文に含まれる N_A を抜き出すことができるのは、B の秘密鍵を持つ主体のみでり、B の秘密鍵を所持する主体は、B 以外には存在しない。すなわち、$message6$ が A に届いたことで、A はネットワーク上に B が存在し、B と通信を行っていることを確信できる。そして、A は $message6$ から、通信相手 B のノンス N_B を抜き出し、B の公開鍵で暗号化し $(\{N_B\}_{K_B})$、B に送信する $(message7)$。

$message7$を受信し、自分の秘密鍵で復号したBは、メッセージが先に送った$message6$で用いた自分のノンス (N_B) と同じであることを確認した。これで、A が B との通信を確信したのと同様に、B も A との通信を確信できる。

両者がお互いに通信相手を確信することで、相互の認証が行われる。

この一連の流れを、シーケンスチャートで表す。(図 2.5)

NSPK Protocol では、鍵を管理する主体に要求すれば、公開鍵の配布が誰にでも行われることと、過去に同じ主体と通信を行い、その際に鍵の配布が行われていることを加味することで、全ての主体の公開鍵が周知であるものとし、公開鍵の管理を行う主体 S を省略して議論されることが多々ある。これは、A と B の S への通信相手の公開鍵のリクエストと、S の A と B への公開鍵の配布に関する 4 ステップを省略し、A と B の通信に着目することを意味し、残る 3 ステップで NSPK Protocol の議論を行う。(図 2.6)

鍵配布が省略された NSPK Protocol のシーケンスチャートは図 2.7である。

以降、本研究でも、NSPK Protocol を論ずる際には、鍵配布を省略した形の 3 ステップの NSPK Protocol を扱う。

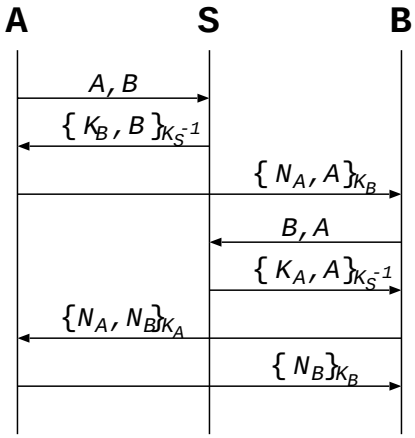

図 2.5: NSPK Protocol シーケンスチャート

3. $A \rightarrow B \ : \ \{N_A, A\}_{K_B}$
6. $B \rightarrow A \ : \ \{N_A, N_B\}_{K_A}$
7. $A \rightarrow B \ : \ \{N_B\}_{K_B}$

図 2.6: 鍵配布が省略された NSPK Protocol

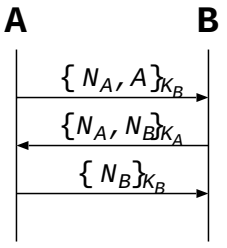

図 2.7: 鍵配布が省略された NSPK Protocol シーケンスチャート

2.3.2 Lowe's Attack

1995 年、Gavin Lowe により、NSPK Protocol に欠陥があることが発表された [3][4]。これは、A が C に通信を開始し、C が A から受信したメッセージ内容をそのまま転用して B との通信を開始すると、意図した認証が行われないことを指摘するものである。(図 2.8)

この Lowe が発表したプロトコル手順を **Lowe's Attack** と呼ぶ。

$$
\begin{array}{lll}
1. & A \to C & : \{N_A, A\}_{K_C} \\
1'. & C \to B & : \{N_A, A\}_{K_B} \\
2'. & B \to C(A) & : \{N_A, N_B\}_{K_A} \\
2. & C \to A & : \{N_A, N_B\}_{K_A} \\
3. & A \to C & : \{N_B\}_{K_C} \\
3'. & C(A) \to B & : \{N_B\}_{K_B}
\end{array}
$$

図 2.8: Lowe's Attack

Lowe's Attack は、主体が 2 つの通信を同時に行い、一方の通信相手に対し、なりすまし[6]をしたまま通信を完了し、認証が正しく行われないことを表している。

では、図 2.8を参照しながら、Lowe's Attack の手順を具体的に追う。

- STEP1

  $$1.\ A \to C : \{N_A, A\}_{K_C}$$

  A は C と通信を開始する。自分のノンスと ID を含むメッセージを C の公開鍵で暗号化し、送信する。($message1$)

- STEP1'

  $$1'.\ C \to B : \{N_A, A\}_{K_B}$$

  $message1$ を受信し、復号した C は、その復号した message1 をそのまま B の公開鍵で暗号化し、送信する。($message1'$)

[6] この場合、B は A と通信を行っているつもりだが、実際は C との通信を完了している。つまり、C は他者に知られることなく、A になりすました。

- STEP2'

$$2'.\ B \to C : \{N_A, N_B\}_{K_A}$$

$message1'$ を受信し、復号した B はメッセージ中に、A の ID とノンスを確認し、$message1'$ 中のノンスと生成した自身のノンスを A の公開鍵で暗号化し、送信する。($message2'$)

この際、$message1'$ 中に A の ID が含まれていることから、B は通信相手を A と判断する。

ここで送信された $message2'$ は、C に横取りされるが、メッセージは A の公開鍵によって暗号化されているため、内容を確認することはできない。

- STEP2

$$2.\ C \to A : \{N_A, N_B\}_{K_A}$$

$message2'$ を横取りすることで、B との通信の STEP2 を完了した形となった C は、$message2'$ には手を加えず[7]、暗号文のまま A に送信する。($message2$)

- STEP3

$$3.\ A \to C : \{N_B\}_{K_C}$$

$message2$ を受信し、復号した A は、メッセージ中に $message1$ で使用したノンスが含まれていることを確認した。ここで、通信を始めた A にとっては、通信相手は C であり、これは $message2$ 中に C の公開鍵で暗号化した $message1$ 中のノンスが含まれていることで、A には疑いようがない。そして、A は $message2$ 中に含まれる通信相手のノンスを C の公開鍵で暗号化し、送信する。

この時点で、A は C の存在を確認し、認証が行われた。

- STEP3'

$$3'.\ C \to B : \{N_B\}_{K_B}$$

[7]暗号文を復号することができないため、手を加えることができないという表現が正確かもしれない

$message3$ を受信し、復号した C は、A との相互の認証が完了する。また、そのメッセージを B の公開鍵で暗号化し、送信する。($messege3'$)

$messege3'$ を受信し、復号した B は、メッセージが自分が用いたノンスと同じであることを確認し、A の存在を確認、認証ということになるが、実際は、B の通信相手は C であり、意図した結果とは異なる。

Lowe's Attack では、$message2'$ を導入することで、C は A ばかりでなく、B との通信においても NSPK Protocol の手順通りに通信が完了したことを表しているが、C をプロトコルの手順によらず、どのようなメッセージも受信可能で、さらに、受信したメッセージの書式に応じた処理 (手順の進行) をする主体であるとすると、$message2'$ の手順は必要がなくなり、Lowe's Attack は図 2.9で表すことができる。

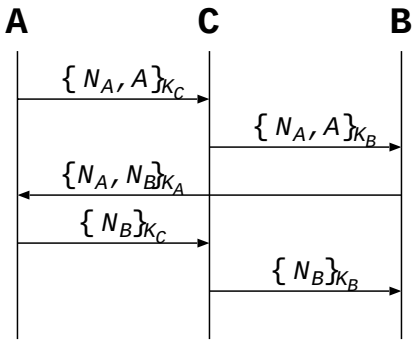

図 2.9: Lowe's Attack シーケンスチャート

通信路上で行われる不正行為が、通信を行う当事者以外の他者からの盗聴、改竄、なりすましのみと仮定した場合、NSPK Protocol は、暗号が破られない限り、その送受信されるメッセージの内容は当事者以外にはわからないことから、不正行為が成立しない、安全なプロトコルであると考えられる。しかし、通信相手が秘密情報を他人に漏らし、利用するような者である場合、不正行為が成立してしまう。

Lowe's Attack の発表は、暗号が破られなくても、主体とプロトコルの手順や、そのメッセージのフォーマットよっては意図した目的が達成されないことを示唆するものと捉えることができる。

2.3.3 改訂版 Needham-Schroeder Public-Key Protocol

Lowe は、Lowe's Attack で発見した不具合を修正するために、NSPK Protocol への改良も同時に発表している。(図 2.10)

$$
\begin{aligned}
1.\quad & A \to B \quad : \{N_A, A\}_{K_B} \\
2.\quad & B \to A \quad : \{N_A, N_B, B\}_{K_A} \\
3.\quad & A \to B \quad : \{N_B\}_{K_B}
\end{aligned}
$$

図 2.10: Needham-Schroeder Public-Key Protocol 改良版

NSPK Protocol の改訂版は、$message2$ に送信者の ID を付加することである。この改良によって、Lowe's Attack による不都合は回避される。

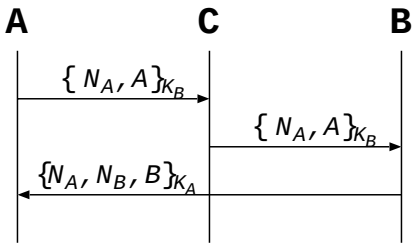

図 2.11: 改訂版 NSPK Protocol への Lowe's Attack

改訂版 NSPK Protocol への Lowe's Attack を図 2.11に表す。

改訂版 NSPK Protocol へ、Lowe's Attack を試みた場合、A は $message2'$ を受信した際に、$message2'$ 中の ID を確認すると、通信を行っているはずの C の ID ではなく、B の ID を確認し、この通信の不正を検知し、中止することができる。これで、Lowe's Attack は失敗し、C の B に対する A としてのなりすましも成立しない。

第 3 章

形式仕様と代数仕様言語 CafeOBJ

本章では、システムを形式的に記述するための形式仕様と、仕様記述に用いる代数仕様言語 CafeOBJ の構文について説明する。

3.1 形式仕様

仕様とは、システムやその構成要素に対する利用者の要求や、システムの設計、振る舞い、またその他の特性を定義するものである。仕様には、抜けがなく、正確で、検証やテストが可能であること、また、実現可能であることが求められ、さらに分かりやすさが要求される。

形式仕様とは、仕様記述に関して、文法・意味が形式的に定義された言語を用いて表現された仕様のことである。ここで形式的とは、数学的な厳密性を意味する。自然言語を用いて記述された仕様は、可読性が高い反面、曖昧性、冗長性、整合正等の点で優れているとはいえない。一方、形式的な仕様では、厳密性、無矛盾性の点で非常に優れており、システムの信頼性の向上に極めて効果が高い。システム開発の上流工程での欠陥により引き起こされるコストは、システムの規模が大きく、信頼性が求められるものであればあるほど大きくなる。このような信頼性が求められるシステムに形式仕様を採用することは有効な方法である。

3.2 代数仕様言語 CafeOBJ

一般に、代数は、台集合と呼ばれるその代数を構成する対象の集合とその上に定義されたいくつかの関数の組で表される。データの集合上にいくつかの操作が定義されたデータ

型の基本構造は、数学における代数の基本構造と一致している。ここで、データ型とその抽象化である抽象データ型の数学モデルを**代数モデル**と呼ぶ。

このような抽象データ型の構造、機能、特性などを代数モデルに基づいて記述するための手法を**代数的仕様記述法**といい、代数的仕様記述法を用いて記述された仕様を**代数仕様**という。

CafeOBJ は、代数仕様言語 OBJ の流れを汲む実行可能な仕様言語である。CafeOBJ は複数の論理の組み合わせを仕様の意味定義の基礎とする特徴がある。この複数の論理とは、**順序ソート代数** (order sorted algebra)、**書き換え論理** (rewriting logic)、**隠蔽代数** (hidden algebra) を指す。

3.2.1 抽象データ型と等式仕様

抽象データ型とは、内部データ型とその型が提供する操作 (演算) のリストとその特性を組にして定義したものである。またデータ型とは、同じ性質を持つ値からなる集合に対する名前である。抽象データ型は、その使用者に対して型名の参照と提供している操作の呼び出しのみを許し、使用上必要な情報のみをインターフェースとして公開する。そのため、内部の実現方法については外部から完全に隠蔽される。このような情報隠蔽をカプセル化という。

代数仕様では、指標という概念を用いて、抽象データ型のインターフェースを表すことができる。

では例として、自然数を CafeOBJ で記述する。

```
mod! NAT{
  [ Nat ]

  op  0 : -> Nat
  op  s_ : Nat -> Nat
  op  _+_ : Nat Nat -> Nat

  vars M N : Nat

  eq M + 0 = M .
  eq M + s N = s(M + N) .
}
```

CafeOBJ では、仕様はモジュールを単位として記述する。モジュールの宣言は、mod、mod!、mod*のいずれかで行うが、抽象データ型の仕様を記述するときには、mod!を用いる。mod!の後に続く NAT はこのモジュールの名前である。[ ] で囲まれた Nat がソート (sort) 名であり、型を表す。予約語 op は演算記号の宣言を表す。op の後に、演算記号、:、アリティ (arity:引数ソート)、->、コアリティ (coarity:結果ソート) の順で宣言する。このアリティとコアリティの組をランク (rank) と呼ぶ。また、予約語 var で変数 (vars で複数の変数) を宣言する。var の後に変数名、:、ソート名の順で宣言する。予約語 eq で等式を宣言している。eq の後に、それぞれ左辺、右辺と呼ばれる 2 つの項を、=を挟む形で宣言する。1 つの等式毎に. で宣言の終わりを表す必要がある。また、条件付き等式を表す予約語として ceq がある。

ここで、ソートはその抽象データ型で使用される内部データ型を、演算子の集合属は、その抽象データ型が持つ演算を表す。それぞれの演算記号の集合に付けられたランクは、その集合に含まれた演算記号に対する抽象データ型の演算の引数と返り値の情報を表している。代数仕様では、各演算の働きを等式を用いて記述する。

3.2.2 抽象状態機械と振舞仕様

抽象データの仕様を記述するための手法として考案された代数的仕様記述法は、近年、**抽象状態機械** (abstract state machine) の仕様を記述するための手法として拡張がなされた。

抽象状態機械とは、システムの取り得る状態の空間と、その上に状態遷移の規則が定義された計算の代数モデルをいう。

代数仕様で抽象状態機械を表現する場合、**隠蔽指標**という特別な指標と、それに関する等式を用いて、抽象状態機械の仕様を記述することができる。

では、例としてスタックを CafeOBJ で表す。

```
mod* STACK{
  protecting(NAT)
  *[ Stk ]*

  op  null : -> Stk           -- initialize
  bop put : Nat Stk -> Stk    -- method
  bop pop_ : Stk -> Stk       -- method
  bop top_ : Stk -> Nat       -- observation
```

```
  var S : Stk
  var N : Nat

  eq  top(put(N, S)) = N .
  beq pop(put(N, S)) = S .
}
```

CafeOBJ では、抽象状態機械の仕様に対して、その仕様に対おうする隠蔽代数を意味として与えることができる。隠蔽代数で表される抽象状態機械は、操作 (method) によって、その状態を変更し、状態は観測 (observation) を通してしか、知ることができない。それぞれ、操作に関する演算を**操作演算**、観測に関する演算を**観測演算**といい、これらをまとめて**振舞演算**という。隠蔽代数による仕様には、抽象状態機械の性質を記述する隠れた部分 (hidden sort:**隠蔽ソート**) と、観測値 (データ) の集合を記述する見える部分 (visible sort:可視ソート) が存在する。

モジュール名 STACK において、protecting(NAT) はモジュール名 NAT の輸入を意味する。CafeOBJ では、組み込みのモジュールや既に定義されたモジュールを他のモジュールに輸入して再利用が可能である。モジュールを輸入するための予約語として、protecting、extending、using が用意され、それぞれ、輸入先のモジュールでどのような意味を持つかという点で異なっている。*[ ]*で囲まれた Stk が隠蔽ソートを表し、スタックの状態集合を意味する。隠蔽ソートにおける予約語 op は、**隠蔽定数**の宣言を表し、null はスタックの初期状態を意味する。予約語 bop で、振舞演算を宣言する。演算 put、pop は、スタックの状態を変更する操作を、演算 top は、スタックが一番上に積んでいるデータの値を示す属性である。なお、仕様中の -- は、コメントを表す予約語であり、-- の他に**、-->、**> があり、行中で使用されると、それら以降の文字列はコメントである。

隠蔽指標と等式の集合によって記述される仕様を**振舞仕様**という。

振舞仕様で記述されたあるシステムの状態は、観測によってしか知ることができない。従って、あるシステムに対して、ある状態が別の状態と等しいというのは、観測される値全てを比較しても区別をつけられないことである。

ここで、あるシステムの状態を □ とし、状態に対する操作を $m_1, \ldots, m_i$、観測を o とすると、この時、$o\Box$、$om_1\Box$ のような、操作で状態を変更した後、観測でその状態を観測する振舞演算列が、観測手段となる。この振舞演算列のことを**観測文脈** (context) と呼ぶ。観測文脈によるシステム状態上の等価関係 $\sim$ を図 3.1のように定義する。

$$s, t \in SystemState$$
$$s \sim t \;\; iff \;\; \wedge_{oc \in AllOc} \, (oc[s] = oc[s'])$$

図 3.1: 振舞等価

ここで、s、t はシステムの状態を表し、$AllOc$ は観測文脈の全体集合を表す。図 3.1は、全ての観測手段 (観測文脈) を通してみたとき、等式が成り立つ関係であることを意味する。このような関係 $\sim$ を、**振舞等価**という。

スタックの仕様内に記述している等式、beq は振舞等式の宣言である。これは、左辺と右辺が振舞等価であることを意味する。

第 4 章

セキュリティプロトコル形式化の枠組み

Lowe によって、NSPK Protocol の欠陥が発表されたこともあり、形式手法を用いたセキュリティプロトコルの検証の有効性が期待されている。それにともない、セキュリティプロトコルの形式化として、いくつか提案がなされている。

4.1 Configuration を用いたプロトコルの形式化

G.Denker らは、セキュリティプロトコルを形式化し、代数仕様言語 Maude によって記述した [5]。Denker らは、通信プロトコルなどの並行分散システムを表現し、分析するために、並行オブジェクト指向モデルを用いている。

並行オブジェクト指向モデルの世界は、図 4.1のように、オブジェクトとメッセージからなる世界である。

並行オブジェクト指向モデルには、次のような特徴がある。

- オブジェクトは計算能力、記憶能力、通信能力を持つ自立的な存在である。
- オブジェクトは一度に一つのメッセージを処理する。
- オブジェクトはそのシステム内で唯一に識別されるような ID を持つ。
- メッセージの送受信は非同期に行われる。
- オブジェクトはメッセージを受け取ると、自分自身の状態を変化させて、0 個以上のメッセージを生成するか、0 個以上のオブジェクトを生成するか、その両方を行う。

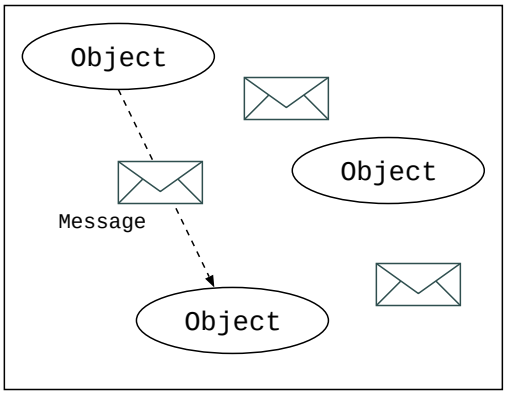

図 4.1: 並行オブジェクト指向モデル

並行オブジェクト指向モデルを形式化するために、**Configuration** というデータ構造を定義している。Configuration は **Object** と **Message** を要素に持つマルチセットであり、並行オブジェクトの計算を Configuration を書き換え論理に基づく遷移規則で遷移させ、表現している。

Configuration の遷移は、Object が Message を受信した時に起こり、図 4.2 のように定義している。

$$
\begin{aligned}
n, m, i, k &\in Nat \\
O, O', Q &\in Object \\
M, M' &\in Message
\end{aligned}
$$

$$
M_1 \ldots M_m O_1 \ldots O_n \rightarrow O'_1 \ldots O'_n Q_1 \ldots Q_i M'_1 \ldots M'_k
$$

図 4.2: Configuration の遷移

図 4.2の意味は以下の通りである。

- Message:$M_1 \ldots M_m$ は Object:$O_1 \ldots O_n$ に受信され、消費される。
- Message を受信した Object:$O_1 \cdots O_n$ は、クラスや属性が変更された上で O' となる。
- 新しい Object:$Q_1 \cdots Q_i$ が生成される。

- 新しい Message:$M'_1 \cdots M'_k$ が送信される。

図 4.2以外にも、Configuration の遷移には、Message を受信した Object が消滅したり、新しい Object、新しい Message の生成が無い場合もある。

Denker らが通信プロトコルのような並行分散システムを分析、表現するために並行オブジェクトモデルを用いたことは前述した。ここで、並行オブジェクトモデルと通信プロトコルの対応について述べると、通信プロトコルが適用されるネットワークは Configuration に、通信を行う主体は Configuration 上の Object に、主体が送受信するメッセージは Configuration 上の Message になる。さらに、プロトコルの各ステップについては、各ステップを Configuration の遷移規則として定義することで、プロトコルの進行を表現している。

Configuration による通信プロトコルの形式化は以下のようになる。

- Object

  Object(Obj) は、ObjectID(OId) と ClassID(CId) と属性 ($attr = v$) の組からなる。

  $$< OId : CId | attr_1 = v_1, \cdots, attr_n = v_n > \in Obj$$

  Object は、ObjectID によってシステム内で唯一に識別される。ClassID は任意の文字列であり、属性は 0 個以上存在し、属性名 ($attr$) と属性値 (v) からなる。

- Message

  Message($Message$) は、送信元の ObjectID と送信先の ObjectID、メッセージの内容 (Msg) からなる。

  $$msg(OId_1, OId_2, Msg) \in Message$$

  メッセージの内容はプロトコルの各ステップのフォーマットに従う。

- Protocol

  プロトコルの進行は、Configuration($Conf$) の遷移規則 ($Conf_1 \rightarrow Conf_2$) で表す。Configuration は Object と Message を元とするマルチセットである。

$$msg(OId_2, OId_1, Msg_1) < (OId_1 : CId_1 | attr_1 = v_1, \cdots, attr_n = v_n > \rightarrow$$
$$< (OId_1 : CId_1 | attr_1 = v_1^{'}, \cdots, attr_n = v_n^{'} > msg(O_2, O_1, Msg_2)$$

$$msg(OId_2, OId_1, Msg_1) < OId_1 : CId_1 | attr_1 = v_1, \cdots, attr_n = v_n >,$$
$$< OId_1 : CId_1 | attr_1 = v_1^{'}, \cdots, attr_n = v_n^{'} > msg(O_2, O_1, Msg_2) \in Conf$$

プロトコルの進行とは、Object が Message を受信し、その返信を行うことである。

Denker らによる形式化では、Object と Message の組み合わせによる書き換え規則を定義することで、通信プロトコルを記述する。

4.2 ネットワークの状態を定義したプロトコルの形式化

萩谷らは、ネットワーク・プロトコルを形式化するために、次のような枠組みを提案 [6] している。

まず、ネットワーク全体の状態を、通信を行う主体の状態の集合 S と、ネットワーク上を流れるメッセージの集合 M とし、メッセージと主体の状態の集合を $< M, S >$ と表す。また、この $< M, S >$ を**システム状態**と呼称する。なお、主体はそれぞれの識別子 p と、その状態 s の組 $p : s$ とし、メッセージのフォーマットと主体の状態はプロトコルに依存する。また、M にも S にも現れていない数の集合を $nonce(M, S)$ と定義し、$nonce(M, S)$ の元をノンスと呼称する。

萩谷らの枠組みでは、プロトコルの各ステップを、2 つのシステム状態間の遷移としている。

ここで、システム状態を $< M, S >$ としたときの、萩谷らの形式化によるプロトコルの実行を説明する。

- まず、主体 (p) とその状態 (s) の組 $(p : s)$ を選ぶ。$(p : s \in S)$
- 主体はメッセージ (m) を受け取る。$(m \in M)$
- ノンス (n) を見つける。$(n \in nonce(M, S))$
- 命題 $R(p, s, m, n, s', m')$ を証明する。R は、プロトコル・ステップを特徴づける述語である。

- 命題が証明される (真である) と、プロトコルが進行する。

命題の s' は遷移後の主体 p の状態を表し、m' はプロトコルステップによって送信されるメッセージを表している。命題が証明された後、システム状態 $< M, S >$ は次のように遷移する。

$$< M, S > \ \rightarrow \ < M \cup m', (S - \{p : s\}) \cup \{p : s'\} >$$

図 4.3: システム状態の遷移

図 4.3は、システム状態 $< M, S >$ だったシステムが、プロトコルのステップにより、M は、主体 $(p : s)$ がメッセージ (m') を送信することで M の元が増え、主体 (p) の状態 s が遷移後 s' に変わったため、遷移前の主体が S の元から除去され、遷移後の p の状態である $p : s'$ が追加されたことを意味する。

プロトコルの最初のステップによっては、主体の状態の存在を仮定しない。このようなステップでは、最初の条件 $p : s \in S$ を除去することができる。また、同様にプロトコルのステップによっては、ノンスを必要としなかったり、メッセージを送信、または受信しない場合があり、各ステップによって、それぞれ必要とする条件が異なる。

萩谷らは、システム状態間の遷移関係 $\rightarrow$ を次のようにまとめている。

$$\frac{p : s \in S \ \ m \in M \ \ n \in nonce(M, S) \ \ R(p, s, m, n, s', m')}{< M, S > \ \rightarrow \ < M \cup \{m'\}, (S - \{p : s\}) \cup \{p : s'\} >}$$

つまり、プロトコルの進行は、主体とその状態を選び $(p : s \in S)$、受信するメッセージがシステムに存在するなら $(m \in M)$、ノンスを見つけ $(n \in nonce(M, S))$、命題を証明する $(R(p, s, m, n, s', m'))$ ことで、システム状態 $(< M, S >)$ が遷移 $(< M \cup \{m'\}, (S - \{p : s\}) \cup \{p : s'\} >)$ することで表す。

さらに、萩谷らはシステム状態 $< M, S >$ から侵入者 (Denker らの枠組みでいう Spy) が生成できるメッセージの集合 $closure(M, S)$ を定義している。$closure(M, S)$ は M のメッセージを分解し、$nonce(M, S)$ のノンスを導入し、それらを合成してできるメッセージの全体を表す。

萩谷らによる形式化では、システムの状態遷移としてプロトコルを定義している。プロトコルステップに関する命題を定義し、ネットワークに存在する主体の状態によってメッセージを受信し、その命題を証明することでプロトコルが進行する。

第 5 章

NSPK Protocol の記述

4章で紹介したセキュリティプロトコルの形式化の枠組みを用いて、NSPK Protocol を代数仕様言語 CafeOBJ によって記述する。

5.1 メッセージの定義

セキュリティプロトコルにおいて、主体がやり取りするメッセージには、主体の名前、ノンス、暗号化・復号化に用いる鍵、これらを組み合わせたメッセージ、ハッシュされたメッセージ、暗号化されたメッセージなどが存在する。

それぞれのメッセージを CafeOBJ を用いて定義する。

- Agent

  Agent とは、通信に参加する主体の名前を表す。また、名前だけではなく、本人の電子証明的な意味として用いられる場合もある。

  CafeOBJ による定義は以下の通りである。

```
mod! AGENT {
  [ Friend Spy Server < Agent ]

  op nobody : -> Agent .
}
```

  これは、ネットワーク上に Agent が存在し、Agent は通信を行う際に不正行為を行

なわない Friend、不正行為を行う Spy、そして、鍵の管理を行う Server を要素として持つ集合であることを意味する。

本論文では以降、主体の区別として、Friend、Spy、Server という呼称を用いる。

- Nonce

  Nonce とは、他者が推測不可能なランダムに生成される値を表す[1]。メッセージが Nonce を含むことで、そのメッセージの単一性を保証し、ひいては、その通信を他の通信と区別したり、通信相手の認証を行うことが可能となる。

  CafeOBJ による定義は以下の通りである。

```
mod! NONCE {
  protecting(AGENT + INT)
  [ Int < Nonce ]

  op nonce : -> Nonce
  op n : Agent Int -> Nonce
}
```

  実際に、通信で用いられる Nonce は数値であり、その値に Agent の情報は持たないことが考えられるが、本稿における形式化の枠組みでは、仕様を実行した際の可読性を高めるため、アリティに Agent を含めて定義する。

- Key

  Key はメッセージの暗号化に用いる鍵を表す。鍵の種類には、秘密鍵 (Sec-Key)、公開鍵 (Pub-Key)、共通鍵 (Com-Key) が存在する。鍵はメッセージを暗号化するだけではなく、鍵交換を行う際にメッセージ中に含まれることも考えられる。

  CafeOBJ による定義は以下の通りである。

```
mod! KEYS {
  protecting(AGENT)
  [ Sec-Key Pub-Key Com-Key < Key ]
```

[1]萩谷らの定義したノンスと比較すると曖昧であるが、この表現を用いさせてもらう [7]

```
  op seck : Agent -> Sec-Key
  op pubk : Agent -> Pub-Key
  op comk : Agent Agent -> Com-Key {comm}

  op keypair : Key -> Key

  vars A B : Agent .

  eq keypair(seck(A)) = pubk(A) .
  eq keypair(pubk(A)) = seck(A) .
  eq keypair(comk(A, B)) = comk(A, B) .
}
```

鍵から、その対となる鍵を導くために、keypair を定義する。これは、メッセージを暗号化した鍵から、そのメッセージを復号するために必要な鍵を導くために用いる。

さらに、これらのメッセージから、以下のメッセージを帰納的に定義する。

- pair

  pair は、メッセージの組み合わせを意味する。メッセージの要素が組み合わされたものもまた、メッセージである。

- hash

  hash は、ハッシュしたメッセージを表す。

- crypt

  crypt は、暗号化されたメッセージを表す。

  CafeOBJ による定義は以下の通りである。

```
mod! PROTOCOL-MESSAGE {
  protecting(AGENT + NONCE + KEYS)
  [ Agent Nonce Key < Msg ]

  op empty-msg : -> Msg
```

```
  op _,_ : Msg Msg -> Msg {assoc}
  op hash : Msg -> Msg
  op crypt : Key Msg -> Msg
}
```

定義において、empty-msg は、空のメッセージを意味する。

メッセージを組み合わせる pair は、_,_で定義する。pair は、メッセージが複数個の組み合わせが考えられるため、さらに、assoc で結合則を満たすを定義する。

ネットワーク上で送受信されるメッセージのフォーマットは、プロトコルに依存し、そのフォーマットは上記の定義をもとにして表現することが可能である。

5.2 NSPK Protocol の定義

NSPK Protocol の記述に際して、次のことを前提とする。

- 暗号化されたメッセージは、対応する鍵でしか復号されない。

  現実には、暗号を破る方法が研究され、いくつか成功例が報告されているが、本研究では、正式な手続きをふまなければ復号することができない暗号化技術を仮定している。本研究では、暗号が完全であっても、プロトコルの手順によって得られる不具合を問題とする。

- メッセージの送信元を特定することはできない。

  メッセージを受信する際に、送信元の確認はできないこととし、自分に対して送られたメッセージは、どんなメッセージでも受信することを仮定する。

- Spy は Friend としても動作することができ、ネットワーク上で区別することができない。

  ネットワーク上で、不正な行為を行う主体が存在することは前述したが、実際に通信を行う際に、そのような主体を区別することはできない。そのため、誰もが Spy と通信する可能性があり、また Spy と通信していることはわからない。Spy は Friend の動作と、それ以外の動作が可能な主体である。

- Spy はプロトコルの手順によらない

  Spy はプロトコルのフォーマットに従ったメッセージなら、プロトコルの手順を無視して、いつでも送受信可能である。

5.2.1 Configuration を用いた NSPK Protocol の記述

Denker らの行った、Configuration を用いたセキュリティプロトコルの形式化をもとにして、代数仕様言語 CafeOBJ の書き換え論理による、NSPK Protocol の仕様記述を行う。

Configuration は Object と Message からなる。まず、Object と Message の定義と記述を説明し、Configuration と、その遷移規則について説明する。

Object の定義

```
mod! MY-OBJECT {
  protecting(OID + CID + ATTRIBUTES)
  [ MyObject ]

  op <(_:_)|_> : OId CId Attributes -> MyObject
}
```

セキュリティプロトコルにおける Object は、通信を行う主体を表す。通信者を識別するために必要な ObjectID(OId) は、Agent を部分集合とし、ClassID(CId) は、主体が不正行為をしない Friends、不正行為を行う Spies、鍵の管理を行う Server のいずれに属するかを記述、さらに Object の属性 (Attributes) には、所持している鍵 (keys)、正常に通信が完了した際の通信履歴 (ec)、実行中の通信履歴 (role) などを定義する。

Message の定義

```
mod! MSG{
  protecting(MESSAGE)
  protecting(MESSAGES)

  op msg : Agent Agent Msg -> Message
}
```

Message は、プロトコルに従って送受信されるメッセージを表す。送信元と送信先は、Agent で定義し、メッセージの内容は PROTOCOL-MESSAGE(Msg) で定義する。

Msg は各プロトコルで決められたフォーマットに従う。

Configuration の定義

```
mod! CONFIGURATION {
  protecting(MY-OBJECT + MESSAGE)
  [ MyObject Message < Configuration ]

  op conf : -> Configuration
  op (_ _) : Configuration Configuration -> Configuration {assoc comm id: conf}
}
```

Configuration は、Object と Message を要素として持つ集合で表す。Configuration の初期状態を conf で定義し、Configuration 上には、複数のオブジェクトとメッセージが存在し、それらは順番がなく可換であるため、結合則と交換則が成立する。

書き換え規則

Configuration の遷移、すなわちプロトコルの進行は Configuration の書き換え規則で表す。ネットワーク上には通信を行う主体として、Friend、Spy、Server が存在し、それぞれメッセージを受信した際に異なる動作をする。

- Friend のメッセージの送受信

  Friend は、プロトコルの正当なメッセージを受信すると、メッセージの内容に即して属性を変化させ、返信を行う。(図 5.1)

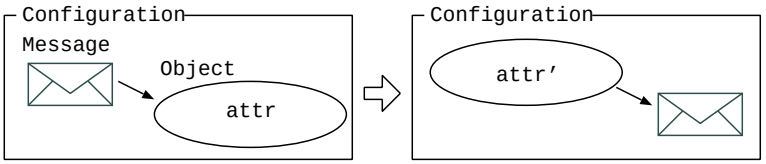

図 5.1: Friend のメッセージ送受信

次に、Friend のメッセージ送受信の一例として、ObjectID が B の Friend が $message2$ を受信し、$message3$ を送信する際の規則を記述する。

```
ctrans [RecieveMsg2SendMsg3] :
       <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI),
                     (ec = EC), (X)>
       msg(HI, A, crypt(K, (NI, NR)))(CONF)
```

```
    => <(A : Friends)|(keys = MyK), (rolei = RI),
                      (ec = ecom(i, N, WHO, NR) U EC), (X)>
       msg(A, WHO, crypt(pubk(WHO), NR))(CONF)
if (keypair(K) ==  MyK) and (N == NI) .

ctrans [RecieveErrorMsg2] :
       <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI),
                      (X)>
       msg(HI, A, crypt(K, (NI, NR)))(CONF)
    => <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI),
                      (X)>(CONF)
if (keypair(K) =/= MyK) or (N =/= NI) .
```

$message2$ を受信するにあたり、B は $message1$ を送信していなければならない。NSPK Protocol の $message2$ は、$B \rightarrow A : \{Na, Nb\}_{K_A}$ であり、本研究で定義したメッセージに従うと、msg(HI, A, crypt(K, (NI, NR))) として記述される。規則中の WHO と HI はともに Agent であり、NI と NR は Nonce である。送信元が特定できないという前提から、$message2$ の送信元 HI と $message1$ を送信した相手 (run(N, WHO, empty-msg)) である WHO が一致するかはわからない。

B が message1 を正当なメッセージと判断するための条件は、メッセージを復号できることと、受信する $message2$ 中に先に送信した $message1$ 中の Nonce が含まれていることである。この条件を満たせば、B はメッセージを受信し、$message1$ を送信した相手に向けて $message3$ を生成、送信し、通信を完了する (ecom(i, N, WHO, NR)) こととなる [ReceiveMsg2SendMsg3]。メッセージが不当である場合は、そのメッセージを無視する (返信をしない)[RecieveErrorMsg2]。

- Spy のメッセージの受信

  Spy は、プロトコルの手順によらず、どんなメッセージでも受信することが可能である。Spy がメッセージを受信するケースとして、Friend のようにプロトコルの手順に従った通信を行う場合と、メッセージがプロトコルのフォーマットに従ってさえいれば、手順を無視して受信する場合が考えられる。(図 5.2)

  次に、Spy のメッセージ受信の一例として、ObjectID が S の Spy が messsage2 を受信する際の規則を記述する。

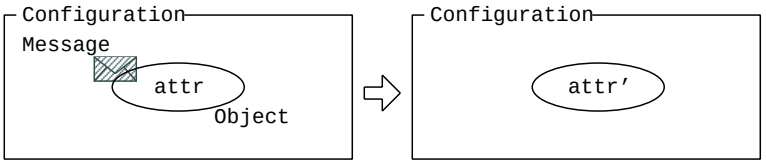

図 5.2: Spy のメッセージ受信

```
ctrans [AcceptMsg2] :
       <(S : Spies)|(ncs =  MSET1), (keys = MyK), (rolei = RI), (X)>
       msg(WHO, S, crypt(K, (NI, NR)))(CONF)
    => <(S : Spies)|(ncs = NR U NI U MSET1), (keys = MyK),
                    (rolei = run(NI, nobody, NR) U RI), (X)>(CONF)
if (WHO =/= S) and (keypair(K) == MyK) .

ctrans [AcceptMsg2LikeFriend] :
       <(S : Spies)|(ncs =  MSET1), (keys = MyK),
                    (rolei = run(N, HI, empty-msg) U RI), (X)>
       msg(WHO, S, crypt(K, (NI, NR)))(CONF)
    => <(S : Spies)|(ncs = NR U NI U MSET1), (keys = MyK),
                    (rolei = run(N, HI, NR) U RI), (X)>(CONF)
if (WHO =/= S) and (keypair(K) == MyK) and (NI == N) .

ctrans [AcceptCryptMsg2] :
       <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
       msg(WHO, S, crypt(K, (NI, NR)))(CONF)
    => <(S : Spies)|(msgs = crypt(K, (NI, NR)) U MSET2), (keys = MyK),
                    (X)>(CONF)
if (WHO =/= S) and (keypair(K) =/= MyK) .
```

Spy はメッセージを受信した際、条件反射的な返信をしない。Spy は受信したメッセージが復号できた場合、新しい通信が開始されたものとみなす場合 [AcceptMsg2] と実行途中の通信かどうかを確認して Friend と同じ振る舞いをする場合 [AcceptMsg2LikeFriend] と、受信するメッセージの内容に応じて情報を蓄積する。受信した

メッセージを復号できなかった場合は、その暗号メッセージの原文を保存する [AcceptCryptMsg2]。

- Spy のメッセージの送信

  Spy は、プロトコルの手順によらず、どんなメッセージでも送信することが可能である。Spy がメッセージを送信するケースとして、Friend のようにプロトコルの手順に従った通信を行う場合と、手順を無視して、プロトコルのメッセージフォーマットに従ったメッセージを送信する場合が考えられる。(図 5.3)

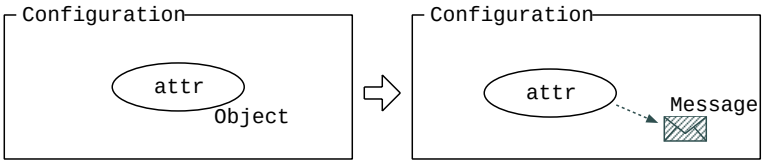

図 5.3: Spy のメッセージ送信

次に、Spy のメッセージ送信の一例として、ObjectID が S の Spy が $messsage3$ を送信する際の規則を記述する。

```
ctrans [FakeMsg3] :
       <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
       <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>(CONF)
    => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
       <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>
       msg(S, WHO, crypt(pubk(WHO), N))(CONF)
if (WHO =/= S) and (NR == N) .

ctrans [MakeMsg3] :
       <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                    (ec = EC), (rolei = run(NI, WHO, empty-msg) U RI), (X)>
       <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>(CONF)
    => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                    (ec = ecom(i, NI, WHO, N) U EC), (rolei = RI), (X)>
       <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>
```

```
        msg(S, WHO, crypt(pubk(WHO), N))(CONF)
  if (WHO =/= S) and (NR == N) .

  ctrans [MakeMsg3LikeFriend] :
        <(S : Spies)|(ec = EC), (rolei = run(NI, WHO, NR) U RI), (X)>(CONF)
     => <(S : Spies)|(ec = ecom(i, NI, WHO, NR) U EC), (rolei = RI), (X)>
        msg(S, WHO, crypt(pubk(WHO), NR))(CONF)
  if (WHO =/= S) .
```

Spy はメッセージの送信方法をいくつか持つ。メッセージを受信待ちしている Friend に向けて、過去に通信路上に流れたメッセージで偽造したフォーマットに従ったメッセージを送信したり [FakeMsg3][MakeMsg3]、実行途中の通信があることを確認して、Friend と同じように、プロトコルの手順に従ったメッセージを送信する場合がある [MakeMsg2LikeFriend]。

- メッセージの横取り・盗聴

  Spy のメッセージ受信方法として、横取りと盗聴がある図 (5.4)。これは、メッセージの送信先が自身の ObjectID と一致しない場合でも受信することで表現する。

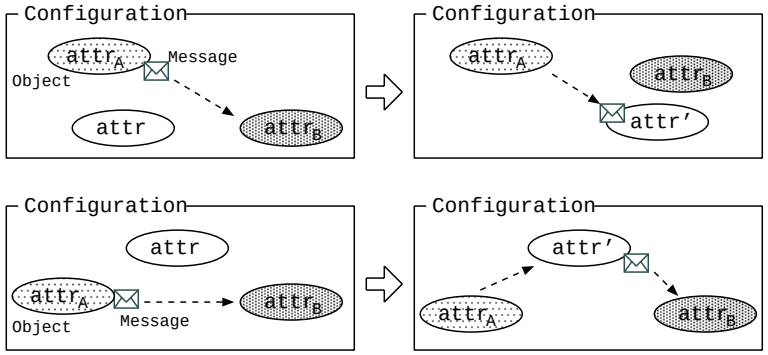

図 5.4: Spy のメッセージ横取り・盗聴

次に、Spy のメッセージ横取り・盗聴の一例として、ObjectID が S の Spy が messsage3 を横取りする際、盗聴する際の規則を記述する。

まず、メッセージの横取りである。Spy は自身宛てではないメッセージを受信し、そのメッセージが復号できる際は、メッセージの内容に応じてその情報を保存し [InterceptMsg3]、復号できない場合は暗号化された原文を保存する [GetMsg3]。

```
ctrans [InterceptMsg3] :
       <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
    => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>
       (CONF)
if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

ctrans [GetMsg3] :
       <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
    => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>(CONF)
if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .
```

次に、メッセージの盗聴である。Spy は自身宛てではないメッセージを受信し、そのメッセージが復号できる際は、メッセージの内容に応じてその情報を保存し [OHknowMsg3]、復号できない場合は暗号化された原文を保存する [OHMsg3]。

```
ctrans [OHknowMsg3] :
       <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
    => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

ctrans [OHMsg3] :
       <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
    => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>
       msg(HI, WHO, crypt(K, N))(CONF)
if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .
```

メッセージの横取りと盗聴の違いは、メッセージを受信した際に、そのメッセージのコピーを送信するか、しないかの違いである。送信しない場合が横取りで、送信する場合が盗聴である。

- 認証の成功となりすましの検知

  主体は属性として通信履歴を持つ。通信履歴は、プロトコルが最後のステップを終了した際に追加され、通信履歴には自身の役割と通信相手、その通信で用いられた自身と通信相手のノンスが記録されている。

  この通信履歴をもとに、認証が正常に行われたこととなりすましの検知の両方が可能である。認証は、通信者相互がその通信のために生成したノンスを共有することで成立する。ノンスを共有し、通信が終了した際に相手を正常に認識していれば、意図した認証が行われており、相手を誤認識していればなりすましが行われたことを意味する。

  Friend と Friend を例として、認証が正常に行われた場合の記述である。

```
ctrans [AuthFF] :
        <(A : Friends)|(ec = ecom(i, NI, HI, NR) U EC), (X)>
        <(B : Friends)|(ec = ecom(r, NR, WHO, NI) U EC'), (X')>(CONF)
    => AuthFriendFriend
if (HI == B) and (WHO == A) .
```

  正常な認証は、Configuration 上の 2 つの Object を選択し、その Object の属性値によって確認することが可能である。Object の選択方法は、属性の中の通信履歴の集合 (ec) 中に、ノンスを共有 (NI、NR) する通信履歴 (ecom) を持ち、その通信において、$message1$ を送信した (i) のか、受信した (r) のかという役割が異なる Object を選択する。ここで、両 Object が通信相手を誤認識していないか確認する。上記の例だと、A は ObjectID が HI である Object と通信を行い、B は WHO と行っている。HI が B と WHO が A と一致すれば、通信相手を間違えることなく、意図した認証が行われたことになる。

  次に、なりすましの検知である。

  Friend に対してなりすましがなされた場合は、次のように記述される。

```
ctrans [WhoMasq] :
        <(A : Friends)|(ec = ecom(i, NR, HI, NI) U EC), (X)>
        <(B : Friends)|(ec = ecom(r, NI, WHO, NR) U EC'), (X')>(CONF)
    => WhoMasq
if  HI =/= B or WHO =/= A .
```

通信履歴を確認する点においては、なりすましの検知も認証の確認と同じである。なりすましは、ノンスを共有した 2 つの Object のどちらかが、別の相手と通信を終了したことで検知する。上記の例だと、$message1$ を送信 (i) した A は HI と通信を行い、$message2$ を受信 (r) した B は、WHO と通信を行った。HI が B とは違う ObjectID、もしくは WHO が A とは違う ObjectID の場合、HI が B への、もしくは WHO が A へのなりすましが行われたことを意味する。

Configuration を用いた NSPK Protocol の定義は、これらの書き換え規則を不足無く網羅することで、正当なものとなる。

通信に参加する主体はメッセージの送受信でその属性を変更する。Spy は属性と振る舞いによっていくつか書き換え規則があり、プロトコルのステップ数が多くなると、その属性と振る舞いは複雑となり、各々の規則を網羅し記述することが困難になると予想できる。

5.2.2 ネットワークの状態を定義した NSPK Protocol の記述

萩谷らが提案したネットワークの状態の遷移によるセキュリティプロトコルの形式化をもとに、代数仕様言語 CafeOBJ での振る舞い仕様による、NSPK Protocol の記述を行う。

システム状態は、ネットワーク上のメッセージの集合 M と主体の集合 S で定義される。まず、メッセージの集合と主体とその集合の定義と記述を説明し、システム状態とその遷移について説明する。

メッセージの集合

ネットワーク上に流れるメッセージは、プロトコルのフォーマットに依存する。メッセージを構成する要素は、5.1で述べた。では、メッセージの集合を CafeOBJ によって、定義する。

```
mod! MESSAGES {
  protecting(EX-BOOL + PROTOCOL-MESSAGE)

  op empty : -> Msg
  op _in_ : Msg Msg -> Bool
  op _U_ : Msg Msg -> Msg {assoc comm id: empty-msg}
}
```

上記は、メッセージの集合の指標部分である。empty は、空の集合を表し、in はそのメッセージが集合に含まれるかを Bool 値で返す。U が集合へのメッセージの追加である。これら指標に関する等式を定義する。

主体の集合

主体は、自身の通信の状況を表す State を持つ、State の値によって特定のメッセージの送受信が可能である。

CafeOBJ による主体の定義は、次の通りである。

```
mod! PRINCIPAL {
  protecting(PID + SID + STATES)
  [ Principal ]

  op peopleless : -> Principal
  op (_:_) : PId States -> Principal
}
```

主体は、PId によって区別される。主体の状態は State で保持される。では、次に主体の集合の指標部分である。

```
mod! PRINCIPALS {
  protecting(EX-BOOL + PRINCIPAL)
  protecting(MESSAGES)
  [ Agent < PId ]
  [ Msg Int < SV ]

  op _in_ : PId Principal -> Bool

  op _U_ : Principal Principal -> Principal {assoc comm id: peopleless}
  op _-_ : Principal Principal -> Principal

  op getnotes : PId Principal -> Msg
  op getstates : PId Principal -> SVs
}
```

PId は、Agent を部分集合とする。State の値として、Msg と Int を取る。in は主体が集合に含まれているかを表し、U は集合への追加を、-が集合からの削除を表す。getnote、getstates ともに、主体の保持する状態を取り出す。特に、getstates が通信の状況を保持する State である。

ネットワーク状態の遷移

ネットワーク状態の遷移は、プロトコル・ステップを特徴づける命題を証明することで、行われる。主体 A と主体 B の NSPK Protocol による通信でのネットワーク状態間の遷移関係は、図 5.5の規則によって定義される。

$$\frac{m \in closure(M,S)}{< M,S > \ \rightarrow \ < M \cup \{m\}, S >} \quad \text{(i)}$$

$$\frac{n_A \in nonce(M,S) \quad R(n_A, s_0(B,n_A), (n_A,A)_{pubk_B})}{< M,S > \ \rightarrow \ < M \cup \{(n_A,A)_{pubk_B}\}, S \cup \{A : s_0(B,n_A) > } \quad \text{(ii)}$$

$$\frac{m_0(n_A,A)_{pubk_B} \in M \quad n_B \in nonce(M,S) \quad R(m_0(n_A,A)_{pubk_B}, n_B, s_1(A,n_A,n_B), m_1(n_A,n_B)_{pubk_A})}{< M,S > \ \rightarrow \ < M \cup \{m_1(n_A,n_B))\}, S \cup \{B : s_1(A,n_A,n_B)\} >} \quad \text{(iii)}$$

$$\frac{A : s_0(B,n_A) \in S \quad m_1(n_A,n_B) \in M \quad R(A, s_0(B,n_A), m_1(n_A,n_B), s_2(B,n_A,n_B), m_2(n_B)_{pubk_B})}{< M,S > \ \rightarrow \ < M \cup \{m_2(n_B)\}, (S - \{A : s_0(B,n_A)\}) \cup \{A : s_2(B,n_A,n_B)\} >} \quad \text{(iv)}$$

$$\frac{B : s_1(A,n_A,n_B) \in S \quad m_2(n_B) \in M \quad R(B, s_1(A,n_A,n_B), m_2(n_B), s_3(A,n_A,n_B))}{< M,S > \ \rightarrow \ < M, (S - \{B : s_1(A,n_A,n_B)\}) \cup \{B : s_3(A,n_A,n_B)\} >} \quad \text{(v)}$$

図 5.5: NSPK Protocol のネットワーク状態遷移

図 5.5中の s_1、s_2、s_3 は状態の構成子を m_1、m_2、m_3 はメッセージの構成子を表し、$pubk_A$ は A の公開鍵を表す。

(i) は、侵入者が合成可能なメッセージをネットワーク上に流すことを意味する規則である。$closure(M,S)$ は侵入者が合成可能なメッセージの集合であり、侵入者は、自身の状態やネットワークの状況に依存せず、いつでもメッセージを流すことが可能である。

(ii) は、主体が $message1$ を送信するための規則である。萩谷らは、プロトコルの最初のいくつかのステップにおいて、主体の状態を仮定していない。これは、主体はどんな状態にあってもプロトコルの開始が可能であることを意味する。通信の開始時にノンスを見つけ、主体の状態を変更し、$messege1$ をネットワーク上に流す。

(iii) は、主体が $message1$ を受信し、$message2$ を送信するための規則である。このステップにおいても主体の状態を仮定せず、主体はどんな状態にあっても NSPK Protocol の $message1$ を受信することができる。ネットワーク上を流れる $message1$ を受信し、ノ

ンスを見つけ、$message1$ を受信する主体の状態を変更し、$message2$ をネットワーク上に流す。

(iv) は、主体が $message2$ を受信し、$message3$ を送信するための規則である。$message2$ を受信する主体は (ii) において、$message1$ を送信していなければならず、それは、$message2$ を受信する主体の状態で確認することができる。$message1$ を送信した主体は、ネットワークに流れる $message2$ を受信し、状態を変更し、$message3$ をネットワーク上に流す。主体の状態の変更は、それ以前の状態をシステムから除去し、新しい状態をシステムに追加することで行われる。

(v) は、主体が $message3$ を受信するための規則である。(iv) と同様に、$message3$ を受信する主体の状態を確認する。$message3$ を受信した主体は状態を変更し、NSPK Protocol を終了する。

本研究では、萩谷らの定義を拡張して記述する。定義の拡張は以下の点である。

- 侵入者の区分を除去して、主体を扱う。

  全ての主体を侵入者と同等とみなし、プロトコルの手順を行うものとする。萩谷らは、侵入者が合成可能なメッセージの集合として $closure(M, S)$ を定義したが、これを全ての主体に適用できるようにする。

- プロトコルステップに関する規則を詳細化し、6 つの規則を定義する。

  NSPK Protocol のステップは 3 つであり、最後のメッセージを受信することを保証することを含めた、萩谷らの 4 つのプロトコルステップに関する定義はごく自然なものである。しかし、本研究では、システム状態の遷移をより詳細に確認するために、各プロトコルステップでの主体のメッセージの送信、受信に関する 6 つの規則の定義を行う。

図 5.6に、拡張した NSPK Protocol の遷移規則の定義を表す。

遷移規則を主体の送信、受信に各々に関して定義し、詳細化したため、主体の状態が増えている。

では、ネットワークを抽象状態機械とみなし、拡張した NSPK Protocol の遷移規則を用いて、システムの振舞仕様を記述する。記述するネットワークの振舞いを図 5.7で表す。

ネットワークの状態は protocol-step によって遷移し、protocol-step は、ネットワークの状態 (NETWORK) とプロトコルの各々のステップを特徴づける命題 (R) によって定義される。また、命題は主体 (Principal) と受信するメッセージ (Message)、ノンス (Nonce)

$$\frac{m \in closure(M,S)}{< M,S > \ \rightarrow \ < M \cup \{m\}, S >} \tag{5.0}$$

$$\frac{n_A \in nonce(M,S) \quad R(n_A, s_0(B,n_A), (n_A,A)_{pubk_B})}{< M,S > \ \rightarrow \ < M \cup \{(n_A,A)_{pubk_B}\}, S \cup \{A : s_0(B,n_A) >} \tag{5.1}$$

$$\frac{m_0(n_A,A)_{pubk_B} \in M \quad R(m_0(n_A,A)_{pubk_B}, s_1(A,n_A))}{< M,S > \ \rightarrow \ < M, S \cup \{B : s_1(A,n_A)\} >} \tag{5.2}$$

$$\frac{B : s_1(A,n_A) \in S \quad n_B \in nonce(M,S) \quad R(B, s_1(A,n_A), n_B, s_2(A,n_A,n_B), m_1(n_A,n_B)_{pubk_A})}{< M,S > \ \rightarrow \ < M \cup \{m_1(n_A,n_B))\}, (S - \{A : s_1(B,n_A)\}) \cup \{B : s_2(A,n_A,n_B)\} >} \tag{5.3}$$

$$\frac{A : s_0(B,n_A) \in S \quad m_1(n_A,n_B) \in M \quad R(A, s_0(B,n_A), m_1(n_A,n_B), s_3(B,n_A,n_B)}{< M,S > \ \rightarrow \ < M, (S - \{A : s_0(B,n_A)\}) \cup \{A : s_3(B,n_A,n_B)\} >} \tag{5.4}$$

$$\frac{A : s_3(B,n_A,n_B) \in S \quad R(A, s_3(B,n_{A,n_B}), s_2(B,n_A,n_B), m_2(n_B)_{pubk_B})}{< M,S > \ \rightarrow \ < M \cup \{m_2(n_B)\}, (S - \{A : s_3(B,n_A,n_B)\}) \cup \{A : s_4(B,n_A,n_B)\} >} \tag{5.5}$$

$$\frac{B : s_2(A,n_A,n_B) \in S \quad m_2(n_B) \in M \quad R(B, s_2(A,n_A,n_B), m_2(n_B), s_5(A,n_A,n_B))}{< M,S > \ \rightarrow \ < M, (S - \{B : s_2(A,n_A,n_B)\}) \cup \{B : s_5(A,n_A,n_B)\} >} \tag{5.6}$$

図 5.6: NSPK Protocol のシステム状態遷移の拡張

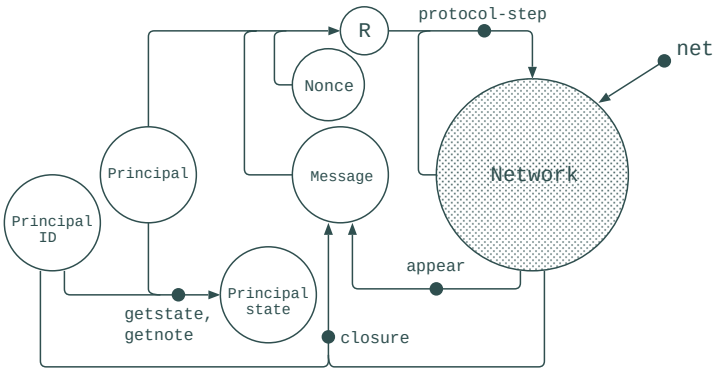

図 5.7: ネットワークの ADJ 図

によって定義される。次に、メッセージの送信、メッセージの受信に関するネットワークの振る舞いを記述する。

メッセージの送信

メッセージの送信は、主体の選択、ノンスの選択に関しての等式で表現する。
メッセージの送信の一例として、主体 B が message2 を送信する際の規則を記述する。

```
ceq nspk2(< M, P >, B, nspkR1(A, NI), NR,
              nspkS2(A, NI, NR), crypt(pubk(A), (NI, NR))) =
      < crypt(pubk(A), (NI, NR)) U M,
      (B : (states = add(nspkS2(B, NI, NR), del(nspkR1(A, NI),
              getstates(B, P)))), (notes = NR U getnotes(B, P))) U
      (P - (B : (states = getstates(B, P)), (notes = getnotes(B, P)))) >
if (B in P) and (nspkR1(A, NI) in getstates(B, P)) and nonce?(NR, < M, P >) .

ceq nspk2(< M, P >, B, nspkR1(A, NI), NR,
              nspkS2(A, NI, NR), crypt(pubk(B), (NI, NR))) = < M, P >
if not(B in P) or not(nspkR1(A, NI) in getstates(B, P)) or
              not(nonce?(NR, < M, P >)) .
```

<M,P>はネットワークの状態を表すものであり、主体 B の状態が nspkR1(A, NI) であるときに、ノンス (NR) を見つけ、状態を変更 (nspkS2(A, NI, NR)) し、メッセージ crypt(pubk(A), (NI, NR)) を送信する。この結果、ネットワークの状態遷移が起こる。ここで、主体の状態を表す nspkR1(A, NI) は、$message1$ を主体 A から、ノンス NI で受信したことを意味し、無作為に選択した NR はノンスであるかわからない。主体が状態を変更した nspkS2(A, NI, NR) は、$message2$ を送信したことを意味する。

ネットワーク上にメッセージを送信する主体が存在しない (not(B in P))、主体の状態が $message2$ を送信する状態にない (not(nspkR1(A, NI) in getstates(B, P)))、または、ノンスが見つからない (not(nonce?(NR, < M, P >)))、いずれかの場合にはネットワークの状態は遷移しない。

プロトコルの最初のステップはメッセージの送信から始まり、そのステップは主体の状態に関係なく、どの主体もメッセージの送信が可能である。また、送信するメッセージによってはノンスを必要としない場合がある。

メッセージの受信

メッセージの受信は、主体の選択、メッセージの受信に関しての等式で表現する。
メッセージの受信の一例として、主体 A が $message2$ を受信する際の規則を記述する。

```
ceq nspk3(< M, P >, A, nspkS1(B, NI), crypt(pubk(A), (NI, NR)),
              nspkR2(B, NI, NR)) =
      < M, (A : (states = add(nspkR2(B, NI, NR), del(nspkS1(B, NI),
              getstates(A, P)))), (notes = NR U getnotes(A, P))) U
      (P - (A : (states = getstates(A, P)), (notes = getnotes(A, P)))) >
if (A in P) and (nspkS1(B, NI) in getstates(A, P)) and
              (crypt(pubk(A), (NI, NR)) in M) .

ceq nspk3(< M, P >, A, nspkS1(B, NI), crypt(pubk(A), (NI, NR)),
              nspkR2(B, NI, NR)) = < M, P >
if not(A in P) or not(nspkS1(B, NI) in getstates(A, P)) or
              not(crypt(pubk(A), (NI, NR)) in M) .
```

メッセージの送信と同様に、<M,P>はネットワークの状態を表すものであり、主体 A の状態が nspkS1(B, NI) であるときに、メッセージ crypt(pubk(A), (NI, NR)) を受信し、状態を変更 nspkR2(B, NI, NR) して、ネットワークの遷移が行われる。主体の状態 nspkS1(B, NI) は、$message1$ を主体 B にノンス NI で送信したこと意味し、メッセージ crypt(pubk(A), (NI, NR)) は受信するメッセージである。状態を変更した nspkR2(B, NI, NR) は、$message2$ を受信したことを意味する。

また、ネットワーク上にメッセージを受信する主体が存在しない、主体の状態が $message2$ を受信する状態にない、または、ネットワーク上に $message2$ が存在しない、いずれかの場合はネットワークは遷移しない。

プロトコルで最初に送信されたメッセージを受信する際は、主体の状態に関係なく、どのような主体も受信が可能である。

メッセージの合成

メッセージの合成は closure(A,<M,P>) で表現する。主体 A がネットワーク上のメッセージと自らが保持するメッセージを利用して生成可能なメッセージを表す。一例として、ネットワーク上にある暗号化されたメッセージから生成可能なメッセージを CafeOBJ で定義する。

```
  ceq closure(A, < crypt(K, X) U M, P >) =
crypt(K, X) U closure(A, < X U M, P >)
      if (keypair(K) in closure(A, < M, P >)) .
  ceq closure(A, < crypt(K, X) U M, P >) =
crypt(K, X) U closure(A, < M, P >)
      if not(keypair(K) in closure(A, < M, P >)) .
```

はじめの等式は、復号に必要な鍵を所持している場合である。その場合は暗号文の原文、およびその復号した内容が、closure に追加される。復号できない場合が次の等式であり、生成可能なメッセージは暗号文の原文のみである。closure は自身の保持するメッセージも含まれるため、鍵の有無が closure で判断することができる。また、自身は鍵を所持していなくても、ネットワーク上のメッセージ中に含まれているかも知れない。そのような場合もまた closure で判断される。

closure に関する等式を、ネットワーク上に流れる全てのメッセージに関して定義する。

5.3 改訂版 NSPK Protocol の記述

改訂版 NSPK Protocol の変更点は、NSPK Protocol の STEP2 で生成する message2 に送信者の ID を付加することである。改訂版 NSPK Protocol の仕様記述は、NSPK Protocol の仕様を変更する形で行う。

5.3.1 書き換え論理による仕様の変更

NSPK Protocol の仕様を変更し、改訂版 NSPK Protocol に対応させる。

Configuration を用いた仕様が改訂によって影響をうける部分は、変更された個所である $message2$ に関連する遷移規則の全てである。

変更する遷移規則は次の通りである。

- Friend の message1 受信後の messag2 送信に関する規則
- Friend の message2 受信後の message3 送信に関する規則
- Spy の message2 送信に関する規則
- Spy の message2 受信に関する規則

変更例として、正しいフォーマットの $message2$ を受信した際の Spy が Friend と同様の振る舞いをする場合の遷移規則を挙げる。

NSPK Protocol の message2 の受信 : Spy

```
ctrans [AcceptMsg2LikeFriend] :
       <(S : Spies)|(ncs =  MSET1), (keys = MyK),
                    (rolei = run(N, HI, empty-msg) U RI), (X)>
       msg(WHO, S, crypt(K, (NI, NR)))(CONF)
    => <(S : Spies)|(ncs = NR U NI U MSET1), (keys = MyK),
                    (rolei = run(N, HI, NR) U RI), (X)>(CONF)
if (WHO =/= S) and (keypair(K) == MyK) and (NI == N) .
```

改訂版 NSPK Protocol の message2 の受信 : Spy

```
ctrans [AcceptMsg2LikeFriend] :
       <(S : Spies)|(ncs =  MSET1), (agents = MSET2), (keys = MyK),
                    (rolei = run(N, HI, empty-msg) U RI), (X)>
       msg(WHO, S, crypt(K, (NI, NR, W)))(CONF)
    => <(S : Spies)|(ncs = NR U NI U MSET1), (agents = W U MSET2),
                    (keys = MyK), (rolei = run(N, HI, NR) U RI), (X)>(CONF)
if (WHO =/= S) and (keypair(K) == MyK) and (NI == N) and (NI == W) .
```

Configuration を用いた仕様記述では、Object の属性値がその遷移規則に大きく影響を与える。NSPK Protocol 改訂版による Agent の属性値に与える影響はそれほど大きくなく、ほとんどその直接の変更個所である $message2$ のフォーマットの変更のみですむ。

5.3.2 振舞仕様の変更

書き換え規則による仕様と同様に、振舞仕様でも改訂個所を NSPK Protocol の仕様に反映させる。

振舞仕様では、全てのプロトコルステップはネットワークに対する操作演算で定義する。プロトコルに変更があった場合は、操作演算とその操作に関する等式を変更する。

変更する等式は次の通りである。

- 主体の message2 の送信に関する等式

- 主体の message2 の受信に関する等式

変更例として、正しいフォーマットの $message2$ を主体が受信する場合のシステム遷移を挙げる。

NSPK Protocol の message2 の受信

```
ceq nspk3(< M, P >, A, nspkS1(X, NI), crypt(pubk(A), (NI, NR)),
              nspkR2(X, NI, NR)) =
      < M, (A : (states = add(nspkR2(X, NI, NR), del(nspkS1(X, NI),
              getstates(A, P)))), (notes = NR U getnotes(A, P))) U
      (P - (A : (states = getstates(A, P)), (notes = getnotes(A, P)))) >
if (A in P) and (nspkS1(X, NI) in getstates(A, P)) and
              (crypt(pubk(A), (NI, NR)) in M) .
```

改訂版 NSPK Protocl の messege2 の受信

```
ceq nspk3(< M, P >, A, nspkS1(B, NI), crypt(pubk(A), (NI, NR, Y)),
              nspkR2(B, NI, NR)) =
      < M, (A : (states = add(nspkR2(B, NI, NR), del(nspkS1(B, NI),
              getstates(A, P)))), (notes = NR U getnotes(A, P))) U
      (P - (A : (states = getstates(A, P)), (notes = getnotes(A, P)))) >
if (A in P) and (nspkS1(B, NI) in getstates(A, P)) and
              (crypt(pubk(A), (NI, NR, Y)) in M) and (Y == B).
```

振舞仕様による記述では、プロトコルに変更があった際には操作演算とその関連する等式の変更で対応する。

5.4 仕様の実行例と実行結果

CafeOBJ によって仕様を記述することの利点の 1 つとして、その仕様が実行可能であることが挙げられる。本研究では、NSPK Protocol とその改訂版の仕様記述を行うにあたって、書き換え論理による仕様と振舞仕様の 2 つのアプローチを行った。

それぞれの仕様に関して、その実行例と実行結果を紹介する。

5.4.1 書き換え論理による仕様

書き換え論理による仕様の特徴は、仕様にある初期状態を与えることで、特定の状態に到達可能であるか確認、検知可能であることである。では、本研究で記述した仕様を実行する。

実行例[2]は、Configuration 上に、alice という名の Friend、bob という名の Friend、spy という名の Spy が存在する。alice は、spy が Spy であるとは知らず、spy と通信を始める。つまり、この実行例は Lowe's Attack の再現を狙いとする。

NSPK Protocol とその改訂版の仕様を読み込み、初期状態の設定を行う。初期状態の設定は、モジュール名 SIMULATE1 および、SIMULATE2 内で行うものとする。

```
trans NSPK-FSF
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)),
                         (rolei = no-run), (roler = no-run),
                         (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)),
                       (rolei = no-run), (roler = no-run),
                       (dcom = empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)),
                     (rolei = no-run), (roler = no-run), (dcom = empty-msg),
                     (cnt = 1), (ncs = empty-msg), (msgs = empty-msg),
                     (agents = bob U spy U empty-msg)> .
```

上記の NSPK-FSF は、モジュール SIMULATE1 に定義した NSPK Protocol 仕様内の Configuration のある 1 つの状態である。また同様に、SIMULATE2 内に NSPK Protocol 改訂版における状態、NSPK-FSF-RV を定義する。NSPK-FSF-RV の初期状態は NSPK-FSF と同じである。

では、実行方法を表す。

```
exec in SIMULATE1 : NSPK-FSF .
exec in SIMULATE2 : NSPK-FSF-RV .
```

実行方法は、あるモジュール内 (ここでは、SIMULATE1、SIMULATE2) の書き換えを行いたい状態 (NSPK-FSF および、NSPK-FSF-RV) を指定して、実行 (exec) する。

実行結果は、それぞれ次のようになった。

[2]詳細の実行方法は、付録を参考にされたい

```
-- execute in SIMULATE1 : NSPK-FSF
AuthFriendSpy : Configuration
(0.000 sec for parse, 2999 rewrites(7.340 sec), 4830 matches)

-- execute in SIMULATE2 : NSPK-RV-FSF
AuthFriendSpy : Configuration
(0.000 sec for parse, 3002 rewrites(12.620 sec), 4829 matches)
```

それぞれ、Friend と Spy の認証が正常に行われたことを表す。

通信プロトコルのような並行分散システムの性質を把握するのが困難である一因として、その処理の流れが複数存在することにある。

上記の NSPK-FSF、NSPK-RV-FSF の結果において、書き換えのトレースを確認すると、spy が $message2$ の送信を Friend として振る舞っていることがわかる。そこで、Lowe's Attack を再現するために、Spy の振る舞いから、Friend として振る舞う遷移規則を外して実行する。

実行結果は次の通りであった。

```
-- execute in SIMULATE1 : NSPK-FSF
WhoMasq : Configuration
(0.000 sec for parse, 939 rewrites(2.370 sec), 1773 matches)

-- execute in SIMULATE2 : NSPK-RV-FSF
< ( alice : Friends ) | ((keys = seck(alice)),
                         (rolei = run(n(alice, 1), spy, empty-msg)),
                         (dcom = empty-msg), (cnt = 2),
                         (ec = no-ec), (roler = no-run))>
< ( bob : Friends ) | ((keys = seck(bob)), (rolei = no-run),
                       (roler = run(n(bob, 1), alice, n(alice, 1))),
                       (cnt = 2), (ec =  no-ec), (dcom = empty-msg))>
< ( spy : Spies ) | ((ncs = n(alice,1)) , (agents = (alice U bob U spy)),
                     (rolei = run(n(alice, 1), bob, empty-msg)),
                     (roler = run(empty-msg, alice, n(alice, 1))),
                     (keys = seck(spy)), (ec = no-ec), (dcom = empty-msg),
                     (cnt = 1), (msgs = empty-msg))> : Configuration
```

```
(0.010 sec for parse, 316 rewrites(1.770 sec), 744 matches)
```

NSPK Protocol では、なりすましが行われたことが結果として現れ、改訂版では通信が途中で終了するという結果が得られた。

処理の流れが複数存在し、その処理が非決定的に選択される、並行分散システムのようなシステムの仕様を記述し、シミュレートする場合、その性質、結果の全てを得られることが望ましい[3]。

5.4.2 振舞仕様

振舞仕様によって記述した仕様は、実行する者がプロトコルを 1 ステップずつ、明示的に行う必要があり、逐次的に行われる。

実行例は、ネットワークの状態が< empty-msg , peopleless >である際に、主体 alice が主体 bob に向けて通信を開始した例である。

```
-- reduce in %SIMULATE1 : nspk0(< empty-msg , peopleless >, alice,
                                n(alice, 0),nspkS1(bob, n(alice, 0)),
                                crypt(pubk(bob), n(alice, 0), alice))
< crypt(pubk(bob),n(alice,0),alice),
   (alice : ((states =  add(nspkS1(bob, n(alice, 0)), nothing)),
             (notes = (seck(alice) U n(alice, 0))))) > : NW
(0.030 sec for parse, 8 rewrites(0.000 sec), 18 matches)

-- reduce in %SIMULATE1 : closure?(alice,n(alice,0),
< crypt(pubk(bob), n(alice, 0), alice),
   (alice : ((states = add(nspkS1(bob, n(alice, 0)), nothing)),
             (notes = (seck(alice) U n(alice, 0))))) >)
true : Bool
(0.110 sec for parse, 71 rewrites(0.030 sec), 219 matches)

-- reduce in %SIMULATE1 : closure?(bob,n(alice,0),
< crypt(pubk(bob), n(alice, 0), alice),
```

[3]G.Denker らが Maude によって記述した仕様 [5] は、Configuration の遷移規則に対して、幅優先探索法を導入し、この問題に対処している

```
   (alice : ((states = add(nspkS1(bob, n(alice, 0)), nothing)),
            (notes = (seck(alice) U n(alice, 0))))) >)
false : Bool
(0.100 sec for parse, 43 rewrites(0.020 sec), 134 matches)
```

alice が生成した n(alice,0) は、ノンスだったので無事 $message1$ をネットワークに流すことができた。そのため、ネットワークのメッセージの集合に追加され、主体の集合に alice も追加された。$message1$ が送信された時点で、n(alice,0) を知りうる主体を closure? で確認することができる。alice は知っているが、bob は $message1$ をまだ受信していないので、わからない。

記述した振舞仕様はこのように逐次実行する。実行後、closure?で生成可能なメッセージを確認することで、可能なプロトコルステップを判断することができる。

Lowe's Attack をシミュレートする際は、プロトコルステップと closure?の手続きを繰り返し、メッセージが生成可能かを確認し、シーケンスをトレースする。

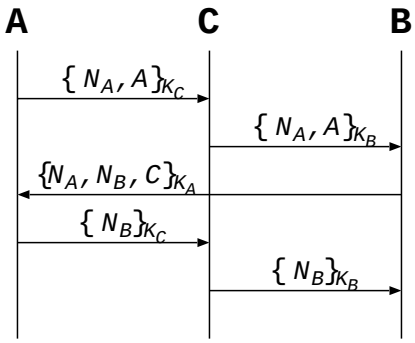

図 5.8: NSPK Protocol 改訂版への Lowe's Attack

図 5.8は、NSPK Protocol 改訂版において、Lowe's Attack を試みた際、通信が無事終了することを表している。図 5.8は A と C、C と B の 2 つの通信を表している。主体の状態を無視して、図 5.8を見ると、A と C の通信では、両者の認証が成立し、C と B の通信では、C は A に、B は C になりすましを行い、認証が成立するという、奇妙な結果が得られる[4]。これは、認証という意味では、問題のある結果であるが、被害者は誰もいない。

[4]書き換え論理による仕様で、このような結果が得られなかったのは、Friend と Spy と Friend の Configuration を設定したからである。この仕様では、主体の生成可能な全メッセージについて考えるので、全主体が Spy のように振る舞うことが可能である。

第 6 章

まとめと今後の課題

本研究では、セキュリティプロトコルを記述するための形式化の枠組みとして、Denker らの行った Configuration を用いた形式化と、萩谷らの提案したネットワークのシステム状態を定義した 2 つの形式化事例を紹介し、それぞれを、代数仕様言語 CafeOBJ によって書き換え規則による仕様、振舞仕様によって記述した。記述した仕様について考察を行い、今後の課題について述べる。

6.1 書き換え論理による仕様

書き換え論理による仕様記述では、Configuration を定義し、その状態遷移を書き換え規則で記述している。Configuration は、通信プロトコルなどの並行分散システムを表現するための並行オブジェクト指向モデルを形式化するために定義され、そのモデルは直感的に理解しやすい。そのため、Configuration を用いたセキュリティプロトコルの記述は、直感的に理解しやすいものとなっており、その可読性も高いものとなっている。記述した仕様は書き換え規則により自動的に状態遷移が行われる。その状態遷移の追跡はシーケンスチャートに自然に対応させることができるため、遷移の様子を把握することは容易である。さらに、書き換え論理による仕様では、ある目的の状態を規則の中に盛り込むことで、その状態を取り出すことができるため、認証の成立や攻撃を検知することが可能である。このようなことから、書き換え規則による仕様は、非常に強力なシミュレータとなる。

しかしながら、書き換え規則による状態遷移の記述は、一つ一つの状態を想定し、その各々の状態に関して全ての規則を用意する。その規則を漏れなく記述することで仕様の正しさとなるのだが、全ての遷移規則を記述することは困難であり、記述した規則が全てで

あることを証明することもまた困難である。このことから、セキュリティプロトコルの安全性を書き換え論理による仕様で考えた場合、プロトコルをシミュレートした結果、攻撃が発見されなかったとしても、そのプロトコルの安全性を保証するものではない。

書き換え規則で記述された仕様は、仕様に変更があった場合、その変更個所によって影響を受ける全ての規則を抽出し、修正を加える必要がある。書き換え規則は、主体の属性に起因するものも多く、仕様が大きくなるほど属性の変更は複雑なものになることが予想される。このことから、仕様の変更に関連する規則の抽出は、その直接の変更点のみならず、そこから起因する規則などにも影響を与えることが考えられ、その抽出作業は困難なものになることが予想できる。

6.2 振舞仕様

一方、振舞仕様によって記述した仕様は、現時点では、シミュレータ、検証系としても、ツールとして力不足である。プロトコルの進行の確認は、逐次的に行われ、人の手入力を必要とする。これは、プロトコルのような非決定的な処理の流れがある場合は、大変な作業である。しかし、振舞仕様による記述では、安全性検証の期待ができるのではないかと考える。セキュリティプロトコルの安全性を保証するには、ある性質[1]を満たすことで正当なものにする。ネットワークの状態は無限状態である。この無限状態において、その性質が成り立つことを証明するには、書き換え規則による状態探索では困難である。そのため、抽象モデル検査の検証方法である帰納法を用いて検証ができないかと考える。ネットワークに対するプロトコルのステップによる帰納法を適用することで、証明可能なのではないかと考える。しかしながら、これには証明すべき性質を形式化する必要がある。NSPK Protocol のノンスは通信を行っている以外の者に知られることがないという性質は、特殊な主体を仮定することで反証された。しかし、Lowe はそのような主体がいても、$message2$ に ID を付加することで問題はないことを述べた。このことにより、ノンスが知られた際も対処できるプロトコルとなった。NSPK Protocol 改訂版は、ノンスは通信者以外に知られた場合でも、正当な認証を保証するものとなっている。このような性質を形式化するような、もしくは、安全性を表す性質を明らかにするという課題がある。

両仕様ともに再利用性に関しては、プロトコル部分の遷移規則をモジュール化しているため問題は少ないと考える。例えば、本研究で作成した仕様を NSPK Protocol 以外のプ

[1]NSPK Protocol では、生成したノンスを知っているのは自身と通信相手以外に存在しないことが認証を保証する性質であった。

ロトコルに適用したいと考えた場合、プロトコル部分のモジュールを差し換えることで可能である。

書き換え論理による仕様は、プロトコルのチェックツールとして有効であろうが、遷移規則の抜け落ちなどが心配である。

6.3 今後の課題

今後の課題として、プロトコルの安全性、もしくはそれに関連する性質を形式化し、仕様に検証の枠組みを取り入れた、セキュリティプロトコル検証系の記述。本研究で記述した振舞仕様へのプロトコルステップを自動化するためのスクリプトの記述。また、書き換え論理による仕様をシミュレータとしてより、強力なものとするために、書き換え規則の適用戦略を盛り込む必要がある。また、NSPK Protocol 以外のセキュリティプロトコルへの適用を行うことが挙げられる。

謝辞

本研究をご指導いただいた、北陸先端科学技術大学院大学の二木厚吉教授、渡部卓雄助教授、緒方和博助手、森彰助手、天野憲樹助手に深く感謝致します。また、研究に関する議論に付き合っていただいた言語設計学講座の皆様に感謝致します。

付録 A

組み込み BOOL 拡張版の仕様

```
** ********************************************************** **
** FILE : ex-bool.mod                                        **
** CONTENTS: formalize extended bool                         **
** AUTHOR: Naoki Kinjo                                       **
** ********************************************************** **

** 組み込みの BOOL を拡張する
mod! EX-BOOL {
  protecting(BOOL)
  [ Bool < Prop ]

  op _&_ : Prop Prop -> Prop {assoc comm idem idr: true prec 40}        ** and
  op _|_ : Prop Prop -> Prop {assoc comm idr: false prec 41}            ** xor

  op _v_ : Prop Prop -> Prop {assoc prec 42}                    ** or
  op ~_ : Prop -> Prop {prec 30}                                ** not
  op _->_ : Prop Prop -> Prop {prec 45}                         ** implies
  op _<->_ : Prop Prop -> Prop {assoc prec 45}                  ** iff

  vars P Q R : Prop

  eq P & false = false .
  eq P | P = false .
  eq P & (Q | R) = (P & Q) | (P & R) .

  eq P v Q = (P & Q) | P | Q .
  eq ~ P = P | true .
  eq P -> Q = (P & Q) | P | true .
  eq P <-> Q = P | Q | true .
}
```

付録 B

メッセージの仕様

```
** ********************************************************** **
** FILE : message.mod                                         **
** CONTENTS: formalize all messages in Protocol               **
** AUTHOR: Naoki Kinjo                                        **
** ********************************************************** **

-- プロトコル中の登場人物、識別子
mod! AGENT {
  [ Friend Spy Server < Agent ]

  op nobody : -> Agent          ** Agent の初期状態
}


-- nonce は、Agent 識別子と整数値で一意性を表す
mod! NONCE {
  protecting(AGENT + INT)
  [ Nonce ]

  op nonce : -> Nonce           ** Nonce の初期状態
  op n : Agent Int -> Nonce     ** Nonce の定義：n(Agent 識別子, 整数値)
}


-- メッセージの暗号化に用いる鍵 (公開鍵、秘密鍵、共通鍵)
mod! KEYS {
  protecting(AGENT)
  [ Sec-Key Pub-Key Com-Key < Key ]

  op seck : Agent -> Sec-Key                ** 秘密鍵
  op pubk : Agent -> Pub-Key                ** 公開鍵
  op comk : Agent Agent -> Com-Key {comm}   ** 共通鍵

  op keypair : Key -> Key            ** 鍵から、その対となる鍵を導く

  vars A B : Agent

  eq keypair(seck(A)) = pubk(A) .
  eq keypair(pubk(A)) = seck(A) .
  eq keypair(comk(A, B)) = comk(A, B) .
```

```
}

-- 通信ネットワーク上に流れるメッセージ
mod! PROTOCOL-MESSAGE {
  protecting(AGENT + NONCE + KEYS)
  [ Agent Nonce Key < Msg ]

  op empty-msg : -> Msg                  ** 空のメッセージ
  op _,_ : Msg Msg -> Msg {assoc}        ** メッセージの組
  op hash : Msg -> Msg                   ** ハッシュメッセージ
  op crypt : Key Msg -> Msg              ** 暗号メッセージ
}
```

付録 C

書き換え論理による仕様記述

a Configuration

```
** ********************************************************** **
** FILE : configuration.mod                                   **
** CONTENTS: formalize Configuration                          **
** AUTHOR: Naoki Kinjo                                        **
** ********************************************************** **

-- オブジェクトの識別子
mod! OID {
  [ OId ]
}


-- オブジェクトのクラス識別子
mod! CID {
  [ CId ]
}


-- オブジェクトの属性識別子
mod! AID {
  [ AId ]
}


-- オブジェクトの属性値
mod!  ATTR-VALUE {
  [ AttrValue ]
}


-- オブジェクトの属性の定義
mod! ATTRIBUTES {
  protecting(AID)
  protecting(ATTR-VALUE)
  [ Attribute < Attributes ]

  op no-attr : -> Attributes                    ** 属性の初期状態
```

```
  op (_=_) : AId AttrValue  -> Attribute      ** 属性の定義
** 属性群の定義
  op (_,_) : Attributes Attributes -> Attributes {assoc comm id: no-attr}
}


-- オブジェクトの定義
mod! MY-OBJECT {
  protecting(OID + CID + ATTRIBUTES)
  [ MyObject ]

  op <(_:_)|_> : OId CId Attributes -> MyObject      ** オブジェクトの定義
}


-- メッセージの定義
mod! MESSAGE {
  [ Message ]
}


-- Configuration の定義
mod! CONFIGURATION {
  protecting(MY-OBJECT + MESSAGE)
  [ MyObject Message < Configuration ]

  op conf : -> Configuration      ** Configuration の初期状態
** Configuration の定義
  op (_ _) : Configuration Configuration -> Configuration {assoc comm id: conf}
}
```

b 主体の属性

```
** ********************************************************** **
** FILE : attribute.mod                                       **
** CONTENTS: formalize object's attribute                     **
** AUTHOR: Naoki Kinjo                                        **
** ********************************************************** **

in ex-bool.mod
in message.mod
in configuration.mod

-- その通信の始動者 or 応答者
mod! ROLE{
  [ Role ]
  op i : -> Role      ** 始動者 (initiator)
  op r : -> Role      ** 応答者 (responder)
}


-- メッセージの集合
mod! MESSAGES{
  protecting(EX-BOOL)
  protecting(PROTOCOL-MESSAGE)

  op _U_ : Msg Msg -> Msg {assoc comm id: empty-msg}      ** メッセージ群
  op _in_ : Msg Msg -> Bool                               ** メッセージの有無
```

```
  vars E E1 E2 : Msg

  eq  E U E = E .

  eq  E in E1 U E2 = (E in E1) v (E in E2) .
  eq  empty-msg in E = true .
  eq  E in E  = true .
  eq  E in E1 = false .
}

-- 実行中プロトコルの通信相手と使用ノンス
mod! RUN{
  protecting(PROTOCOL-MESSAGE)
  [ Run ]

  op run : Msg Agent Msg -> Run             ** 実行中の通信
}

-- 実行中プロトコル群
mod! RUNS{
  protecting(EX-BOOL + RUN)

  op no-run : -> Run
  op _U_ : Run Run -> Run {assoc comm id: no-run}
  op _in_ : Run Run -> Bool
  op connect? : Agent Run -> Bool ** この Agent と通信中か?
  op appear? : Msg Run -> Bool          ** 以前の通信に現れたメッセージか?

  vars R R1 R2 : Run .
  vars A A' : Agent .
  vars N M M' : Msg .

  eq  R U R = R .

  eq  R in R1 U R2 = (R in R1) v (R in R2) .
  eq  no-run in R = true .
  eq  R in R  = true .
  eq  R in R1 = false .

  eq  connect?(A, no-run) = false .
  ceq connect?(A, run(N, A', M) U R) = true      if (A == A') .
  ceq connect?(A, run(N, A', M) U R) = connect?(A, R)      if (A =/= A') .
  ceq connect?(A, run(N, A', M)) = true            if (A == A') .
  ceq connect?(A, run(N, A', M)) = false      if (A =/= A') .

  eq  appear?(M, no-run) = false .
  ceq appear?(M, run(N, A, M') U R) = true
      if (M == N) or (M == M') .
  ceq appear?(M, run(N, A, M') U R) = appear?(M, R)
      if (M =/= N) and (M =/= M') .
  ceq appear?(M, run(N, A, M')) = true      if (M == N) or (M == M') .
  ceq appear?(M, run(N, A, M')) = false      if (M =/= N) and (M =/= M') .
}

-- 正常終了した通信 -自分の役割と通信に用いられたノンス-
mod! ESTABCOM{
  protecting(ROLE + NONCE + AGENT)
```

```
  [ EstabCom ]

  op ecom : Role Nonce Agent Nonce -> EstabCom      ** 通信履歴
}


-- 正常終了した通信の履歴
mod! ESTABCOMS{
  protecting(EX-BOOL + ESTABCOM)

  op no-ec : -> EstabCom
  op _U_ : EstabCom EstabCom -> EstabCom {assoc comm id: no-ec}
  op _in_ : EstabCom EstabCom -> Bool

  vars E E1 E2 : EstabCom

  eq  E U E = E .

  eq  E in E1 U E2 = (E in E1) v (E in E2) .
  eq  no-ec in E = true .
  eq  E in E  = true .
  eq  E in E1 = false .
}


-- プロトコル中のオブジェクト識別子は Agent 識別子で表す
mod! PROTOCOL-OID{
  protecting(AGENT)
  protecting(OID)
  [ Agent < OId ]
}


-- Agent の持つ属性群
mod! PROTOCOL-AID{
  protecting(AID)

  ops ec keys rolei roler dcom cnt ncs msgs agents : -> AId
}


-- Agent の持つ属性値群
mod! PROTOCOL-ATTR-VALUE{
  protecting(ATTR-VALUE)
  protecting(ROLE + MESSAGES + RUNS + ESTABCOMS + INT)
  [ Role Msg Run EstabCom Int < AttrValue ]
}


-- ネットワーク上に流れるメッセージ形式
mod! MSG{
  protecting(MESSAGE)
  protecting(MESSAGES)

  op msg : Agent Agent Msg -> Message
}
```

c NSPK Protocol の仕様

```
** ************************************************************ **
** FILE : nspk.mod                                              **
** CONTENTS: Specification of Needham-Schroeder Public Key Protocol **
** AUTHOR: Naoki Kinjo                                          **
** ************************************************************ **

-- -------------------------------------------------------------
-- NSPK Protcol
-- -------------------------------------------------------------
mod! NSPK {
  protecting(CONFIGURATION)
  protecting(PROTOCOL-OID + PROTOCOL-AID + PROTOCOL-ATTR-VALUE + MSG)

  op  nspkcom : Agent -> Msg
  ops Friends Spies : -> CId
  ops AuthFriendFriend AuthFriendSpy AuthSpyFriend : -> Configuration
  ops SpyMasqInit SpyMasqResp : -> Configuration
  op  WhoMasq : -> Configuration

  vars A B : Friend                          ** 通信者
  var  S : Spy                               ** 不正行為をする通信者
  vars WHO HI W : Agent                      ** 特定不能な通信者
  vars N NI NR NI' NR' : Nonce               ** Nonce
  vars MyK K : Key                           ** 所持している鍵
  vars MSET MSET1 MSET2 MSET3 : Msg
  vars EC EC' : EstabCom
  vars RI RR : Run
  vars I R : Role
  vars CNT CNT' : Int
  var  CONF : Configuration
  vars X X' : Attributes

** ************************************************************ **
**                          Friends' rule                       **
** ************************************************************ **
-- Send Message1
  trans [SendMsg1] :
        <(A : Friends)|(rolei = RI), (dcom = nspkcom(WHO) U MSET), (cnt = CNT), (X)>
        (CONF)
     => <(A : Friends)|(rolei = run(n(A, CNT), WHO, empty-msg) U RI),
                       (dcom = MSET), (cnt = (CNT + 1)), (X)>
        msg(A, WHO, crypt(pubk(WHO), (n(A, CNT), A)))(CONF) .

  ctrans [ReceiveMsg1SendMsg2] :
         <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (cnt = CNT), (X)>
         msg(WHO, B, crypt(K, (N, HI)))(CONF)
      => <(B : Friends)|(keys = MyK), (rolei = RI),
                        (roler = run(n(B, CNT), HI, N) U RR), (cnt = (CNT + 1)), (X)>
         msg(B, HI, crypt(pubk(HI), (N, n(B, CNT))))(CONF)
  if (keypair(K) == MyK) and not(appear?(N, RR)) and not(appear?(N, RI)) .

  ctrans [RecieveErrorMsg1] :
         <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (X)>
         msg(WHO, B, crypt(K, (N, HI)))(CONF)
      => <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (X)>(CONF)
  if (keypair(K) =/= MyK) or appear?(N, RR) or appear?(N, RI) .

-- Recieve Message2 and Send Message3
```

```
  ctrans [RecieveMsg2SendMsg3] :
         <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI),
                        (ec = EC), (X)>
         msg(HI, A, crypt(K, (NI, NR)))(CONF)
      => <(A : Friends)|(keys = MyK), (rolei = RI),
                         (ec = ecom(i, N, WHO, NR) U EC), (X)>
         msg(A, WHO, crypt(pubk(WHO), NR))(CONF)
  if (keypair(K) ==  MyK) and (N == NI) .

  ctrans [RecieveErrorMsg2] :
         <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI), (X)>
         msg(HI, A, crypt(K, (NI, NR)))(CONF)
      => <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI), (X)>
           (CONF)
  if (keypair(K) =/= MyK) or (N =/= NI) .

-- Message3 Recieve
  ctrans [RecieveMsg3] :
         <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (ec = EC), (X)>
          msg(WHO, B, crypt(K, N))(CONF)
      => <(B : Friends)|(keys = MyK), (roler = RR),
                        (ec = ecom(r, NR, HI, NI) U EC), (X)>(CONF)
  if (keypair(K) == MyK) and (NR == N) .

  ctrans [RecieveMsg3Error] :
         <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (X)>
         msg(WHO, B, crypt(K, N))(CONF)
      => <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (X)>(CONF)
  if (keypair(K) =/= MyK) or (NR =/= N) .

** ********************************************************************* **
**                              Spies' rule                              **
** ********************************************************************* **
** Spies make a message according by protocol message format.
-- Spies Begin NSPK-Protocol .
  ctrans [FakeMsg1] :
         <(S : Spies)|(ncs = N U MSET1), (agents = HI U WHO U MSET2),
                      (rolei = RI), (roler = RR), (X)>
         <(WHO : Friends)|(X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = HI U WHO U MSET2),
                      (rolei = run(N, WHO, empty-msg) U RI), (roler = RR), (X)>
         <(WHO : Friends)|(X')>
         msg(S, WHO, crypt(pubk(WHO), (N, HI)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (HI =/= WHO) and
       not(connect?(WHO, RI)) and not(connect?(WHO, RR)) .

  trans [MakeMsg1LikeFriend] :
        <(S : Spies)|(ncs = MSET1), (agents = MSET2), (rolei = RI),
                     (dcom = nspkcom(WHO) U MSET3), (cnt = CNT), (X)>(CONF)
     => <(S : Spies)|(ncs = n(S, CNT) U MSET1), (agents = WHO U MSET2),
                     (rolei = run(n(S, CNT), WHO, empty-msg) U RI),
                     (dcom = MSET3), (cnt = (CNT + 1)), (X)>
        msg(S, WHO, crypt(pubk(WHO), (n(S, CNT), S)))(CONF) .

-- Spies Make Message2
  ctrans [FakeMsg2] :
         <(S : Spies)|(ncs = NI U NR U MSET1), (agents = WHO U MSET2),
                      (roler = RR), (X)>
         <(WHO : Friends)|(rolei = run(N, HI, empty-msg) U RI), (X')>(CONF)
```

```
      => <(S : Spies)|(ncs = NI U NR U MSET1), (agents = WHO U MSET2),
                      (roler = run(NR, WHO, NI) U RR), (X)>
         <(WHO : Friends)|(rolei = run(N, HI, empty-msg) U RI),(X')>
         msg(S, WHO, crypt(pubk(WHO), (NI, NR)))(CONF)
  if (WHO =/= S) and (N == NI) .

  ctrans [MakeMsg2LikeFriend] :
         <(S : Spies)|(ncs = MSET1), (cnt = CNT),
                      (roler = run(empty-msg, HI, N) U RR), (X)>(CONF)
      => <(S : Spies)|(ncs = n(S, CNT) U MSET1), (cnt = (CNT + 1)),
                      (roler = run(n(S, CNT), HI, N) U RR), (X)>
         msg(S, HI, crypt(pubk(HI), (N, n(S, CNT))))(CONF)
  if (HI =/= S) .

-- Spies Make Message3
  ctrans [FakeMsg3] :
         <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>
         msg(S, WHO, crypt(pubk(WHO), N))(CONF)
  if (WHO =/= S) and (NR == N) .

  ctrans [MakeMsg3] :
         <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                      (ec = EC), (rolei = run(NI, WHO, empty-msg) U RI), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                      (ec = ecom(i, NI, WHO, N) U EC), (rolei = RI), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>
         msg(S, WHO, crypt(pubk(WHO), N))(CONF)
  if (WHO =/= S) and (NR == N) .

  ctrans [MakeMsg3LikeFriend] :
         <(S : Spies)|(ec = EC), (rolei = run(NI, WHO, NR) U RI), (X)>(CONF)
      => <(S : Spies)|(ec = ecom(i, NI, WHO, NR) U EC), (rolei = RI), (X)>
         msg(S, WHO, crypt(pubk(WHO), NR))(CONF)
  if (WHO =/= S) .

** Spies accept all message .
-- SpiesAcceptEveryMessage1
  ctrans [AcceptMsg1LikeFriend] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (roler = RR), (X)>
         msg(WHO, S, crypt(K, (N, HI)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = HI U MSET3), (keys = MyK),
                      (roler = run(empty-msg, HI, N) U RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [AcceptCryptMsg1] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, (N, A)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (N, A)) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

-- SpiesAcceptEveryMessage2
  ctrans [AcceptMsg2] :
         <(S : Spies)|(ncs =  MSET1), (keys = MyK), (rolei = RI), (X)>
         msg(WHO, S, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(ncs = NR U NI U MSET1), (keys = MyK),
```

```
                      (rolei = run(NI, nobody, NR) U RI), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [AcceptMsg2LikeFriend] :
         <(S : Spies)|(ncs =  MSET1), (keys = MyK),
                      (rolei = run(N, HI, empty-msg) U RI), (X)>
         msg(WHO, S, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(ncs = NR U NI U MSET1), (keys = MyK),
                      (rolei = run(N, HI, NR) U RI), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) and (NI == N) .

  ctrans [AcceptCryptMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR)) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

-- SpiesAcceptEveryMessage3
  ctrans [AcceptMsg3Auth] :
         <(S : Spies)|(ec = EC), (ncs =  MSET1), (keys = MyK), (roler = RR), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(ec = ecom(r, N, nobody, nonce) U EC), (ncs = N U MSET1),
                      (keys = MyK), (roler = RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) and not(appear?(N, RR)) .

  ctrans [AcceptMsg3LikeFriend] :
         <(S : Spies)|(ec = EC), (ncs = MSET1), (keys = MyK),
                      (roler = run(NR, HI, NI) U RR), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(ec = ecom(r, NR, HI, NI) U EC), (ncs = N U MSET1),
                      (keys = MyK), (roler = RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) and (NR == N).

  ctrans [AcceptCryptMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

** define intercept message
-- Spies Intercept Message1
  ctrans [InterceptMsg1] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = W U MSET3), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [GetMsg1] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (N, W)) U MSET2), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- Spies Intercept Message2
  ctrans [InterceptMsg2] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(ncs = NI U NR U MSET1), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) == MyK) .
```

```
  ctrans [GetMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR)) U MSET2),(keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) =/= MyK) .

-- Spies Intercept Message3
  ctrans [InterceptMsg3] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [GetMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

** define Spies' Overhear
--  Spies Overhear Message1
  ctrans [OHknowMsg1] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = W U MSET3), (keys = MyK), (X)>
           msg(HI, WHO, crypt(K, (N, W)))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [OHMsg1] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (N, A)) U MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- Spies Overhear Message2
  ctrans [OHknowMsg2] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(ncs = NI U NR U MSET1), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) == MyK) .

  ctrans [OHMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR)) U MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) =/= MyK) .

-- Spies Overhear Message3
  ctrans [OHknowMsg3] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .
```

```
  ctrans [OHMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- Spies Replay Message
**  ctrans <(S : Spies)|(msgs = M U MSET1), (agents = A U MSET2), (X)>(CONF)
**       => <(S : Spies)|(msgs = M U MSET1), (agents = A U MSET2), (X)>
**          msg(S, A, M)(CONF)
**     if (M =/= empty-msg) and (A =/= S) .

-- detection some situations
  ctrans [AuthFF] :
         <(A : Friends)|(ec = ecom(i, NI, HI, NR) U EC), (X)>
         <(B : Friends)|(ec = ecom(r, NR, WHO, NI) U EC'), (X')>(CONF)
      => AuthFriendFriend
  if (HI == B) and (WHO == A) .

  ctrans [AuthFS] :
         <(A : Friends)|(ec = ecom(i, NI, HI, NR) U EC), (X)>
         <(S : Spies)|(ec = ecom(r, NR, WHO, NI) U EC'), (X')>(CONF)
      => AuthFriendSpy
  if (HI == S) and (WHO == A) .

  ctrans [AuthSF] :
         <(S : Spies)|(ec = ecom(i, NI, HI, NR) U EC'), (X')>
         <(A : Friends)|(ec = ecom(r, NR, WHO, NI) U EC), (X)>(CONF)
      => AuthSpyFriend
  if (WHO == S) and (HI == A) .

  ctrans [SpyMasqResp] :
         <(A : Friends)|(ec = ecom(i, NI, WHO, NR) U EC), (X)>
         <(S : Spies)|(ec = ecom(r, NR, A, NI) U EC'), (X')>(CONF)
      => SpyMasqResp
  if WHO =/= S .

  ctrans [SpyMasqInit] :
         <(A : Friends)|(ec = ecom(r, NR, WHO, NI) U EC), (X)>
         <(S : Spies)|(ec = ecom(i, NI, A, NR) U EC'), (X')>(CONF)
      => SpyMasqInit
  if WHO =/= S .

  ctrans [WhoMasq] :
         <(A : Friends)|(ec = ecom(i, NR, HI, NI) U EC), (X)>
         <(B : Friends)|(ec = ecom(r, NI, WHO, NR) U EC'), (X')>(CONF)
      => WhoMasq
  if (HI =/= B) or (WHO =/= A) .
}
```

d NSPK Protocol 改訂版の仕様

```
** ************************************************************ **
** FILE : nspk-rv.mod                                           **
** CONTENTS: Specification of Needham-Schroeder Public Key Protocol **
**                                                 ReVision     **
** AUTHOR: Naoki Kinjo                                          **
```

```
** ************************************************************ **

-- ------------------------------------------------------------
-- NSPK-RV Protcol
-- ------------------------------------------------------------
mod! NSPK-RV {
  protecting(CONFIGURATION)
  protecting(PROTOCOL-OID + PROTOCOL-AID + PROTOCOL-ATTR-VALUE + MSG)

  op  nspkcom : Agent -> Msg
  ops Friends Spies : -> CId
  ops AuthFriendFriend AuthFriendSpy AuthSpyFriend : -> Configuration
  ops SpyMasqInit SpyMasqResp : -> Configuration
  op  WhoMasq : -> Configuration

  vars A B : Friend                              ** 通信者
  var  S : Spy                                   ** 不正行為をする通信者
  vars WHO HI W : Agent                          ** 特定不能な通信者
  vars N NI NR NI' NR' : Nonce                   ** Nonce
  vars MyK K : Key                               ** 所持している鍵
  vars MSET MSET1 MSET2 MSET3 : Msg
  vars EC EC' : EstabCom
  vars RI RR : Run
  vars I R : Role
  vars CNT CNT' : Int
  var  CONF : Configuration
  vars X X' : Attributes

** ************************************************************ **
**                          Friends' rule                       **
** ************************************************************ **
-- Send Message1
  trans [SendMsg1] :
        <(A : Friends)|(rolei = RI), (dcom = nspkcom(WHO) U MSET),
                       (cnt = CNT), (X)>(CONF)
     => <(A : Friends)|(rolei = run(n(A, CNT), WHO, empty-msg) U RI),
                       (dcom = MSET), (cnt = (CNT + 1)), (X)>
        msg(A, WHO, crypt(pubk(WHO), (n(A, CNT), A)))(CONF) .

  ctrans [ReceiveMsg1SendMsg2] :
         <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (cnt = CNT), (X)>
         msg(WHO, B, crypt(K, (N, HI)))(CONF)
      => <(B : Friends)|(keys = MyK), (rolei = RI),
                        (roler = run(n(B, CNT), HI, N) U RR), (cnt = (CNT + 1)), (X)>
         msg(B, HI, crypt(pubk(HI), (N, n(B, CNT), B)))(CONF)
  if (keypair(K) == MyK) and not(appear?(N, RR)) and not(appear?(N, RI)) .

  ctrans [RecieveErrorMsg1] :
         <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (X)>
         msg(WHO, B, crypt(K, (N, HI)))(CONF)
      => <(B : Friends)|(keys = MyK), (rolei = RI), (roler = RR), (X)>(CONF)
  if (keypair(K) =/= MyK) or appear?(N, RR) or appear?(N, RI) .

-- Recieve Message2 and Send Message3
  ctrans [RecieveMsg2SendMsg3] :
         <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI),
                        (ec = EC), (X)>
         msg(W, A, crypt(K, (NI, NR, HI)))(CONF)
      => <(A : Friends)|(keys = MyK), (rolei = RI), (ec = ecom(i, N, WHO, NR) U EC), (X)>
         msg(A, WHO, crypt(pubk(WHO), NR))(CONF)
  if (WHO == HI) and (keypair(K) ==  MyK) and (N == NI) .
```

```
  ctrans [RecieveErrorMsg2] :
         <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI), (X)>
         msg(W, A, crypt(K, (NI, NR, HI)))(CONF)
      => <(A : Friends)|(keys = MyK), (rolei = run(N, WHO, empty-msg) U RI), (X)>(CONF)
  if (WHO =/= HI) or (keypair(K) =/= MyK) or (N =/= NI) .

-- Message3 Recieve
  ctrans [RecieveMsg3] :
         <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (ec = EC), (X)>
          msg(WHO, B, crypt(K, N))(CONF)
      => <(B : Friends)|(keys = MyK), (roler = RR),
                        (ec = ecom(r, NR, HI, NI) U EC), (X)>(CONF)
  if (keypair(K) == MyK) and (NR == N) .

  ctrans [RecieveMsg3Error] :
         <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (X)>
          msg(WHO, B, crypt(K, N))(CONF)
      => <(B : Friends)|(keys = MyK), (roler = run(NR, HI, NI) U RR), (X)>(CONF)
  if (keypair(K) =/= MyK) or (NR =/= N) .

** ********************************************************************* **
**                             Spies' rule                               **
** ********************************************************************* **
** Spies make a message according by protocol message format.
-- Spies Begin NSPK-Protocol .
  ctrans [FakeMsg1] :
         <(S : Spies)|(ncs = N U MSET1), (agents = HI U WHO U MSET2),
                      (rolei = RI), (roler = RR), (X)>
         <(WHO : Friends)|(X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = HI U WHO U MSET2),
                      (rolei = run(N, WHO, empty-msg) U RI), (roler = RR), (X)>
         <(WHO : Friends)|(X')>
         msg(S, WHO, crypt(pubk(WHO), (N, HI)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (HI =/= WHO) and
       not(connect?(WHO, RI)) and not(connect?(WHO, RR)) .

  trans [MakeMsg1LikeFriend] :
        <(S : Spies)|(ncs = MSET1), (agents = MSET2), (rolei = RI),
                     (dcom = nspkcom(WHO) U MSET3), (cnt = CNT), (X)>(CONF)
     => <(S : Spies)|(ncs = n(S, CNT) U MSET1), (agents = WHO U MSET2),
                     (rolei = run(n(S, CNT), WHO, empty-msg) U RI),
                     (dcom = MSET3), (cnt = (CNT + 1)), (X)>
        msg(S, WHO, crypt(pubk(WHO), (n(S, CNT), S)))(CONF) .

-- Spies Make Message2
  ctrans [FakeMsg2] :
         <(S : Spies)|(ncs = NI U NR U MSET1), (agents = W U WHO U MSET2),
                      (roler = RR), (X)>
         <(WHO : Friends)|(rolei = run(N, HI, empty-msg) U RI), (X')>(CONF)
      => <(S : Spies)|(ncs = NI U NR U MSET1), (agents = W U WHO U MSET2),
                      (roler = run(NR, WHO, NI) U RR), (X)>
         <(WHO : Friends)|(rolei = run(N, HI, empty-msg) U RI),(X')>
         msg(S, WHO, crypt(pubk(WHO), (NI, NR, W)))(CONF)
  if (WHO =/= S) and (N == NI) and (W =/= WHO) .

  ctrans [MakeMsg2LikeFriend] :
         <(S : Spies)|(ncs = MSET1), (cnt = CNT),
                      (roler = run(empty-msg, HI, N) U RR), (X)>(CONF)
```

```
      => <(S : Spies)|(ncs = n(S, CNT) U MSET1), (cnt = (CNT + 1)),
                      (roler = run(n(S, CNT), HI, N) U RR), (X)>
         msg(S, HI, crypt(pubk(HI), (N, n(S, CNT), S)))(CONF)
  if (HI =/= S) .

-- Spies Make Message3
  ctrans [FakeMsg3] :
         <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RR), (X')>
         msg(S, WHO, crypt(pubk(WHO), N))(CONF)
  if (WHO =/= S) and (NR == N) .

  ctrans [MakeMsg3] :
         <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                      (ec = EC), (rolei = run(NI, WHO, empty-msg) U RI), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = WHO U MSET2),
                      (ec = ecom(i, NI, WHO, N) U EC), (rolei = RI), (X)>
         <(WHO : Friends)|(roler = run(NR, HI, NI) U RI), (X')>
         msg(S, WHO, crypt(pubk(WHO), N))(CONF)
  if (WHO =/= S) and (NR == N) .

  ctrans [MakeMsg3LikeFriend] :
         <(S : Spies)|(ec = EC), (rolei = run(NI, WHO, NR) U RI), (X)>(CONF)
      => <(S : Spies)|(ec = ecom(i, NI, WHO, NR) U EC), (rolei = RI), (X)>
         msg(S, WHO, crypt(pubk(WHO), NR))(CONF)
  if (WHO =/= S) .

** Spies accept all message .
-- SpiesAcceptEveryMessage1
  ctrans [AcceptMsg1LikeFriend] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (roler = RR), (X)>
         msg(WHO, S, crypt(K, (N, HI)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = HI U MSET3), (keys = MyK),
                      (roler = run(empty-msg, HI, N) U RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [AcceptCryptMsg1] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, (N, A)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (N, A)) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

-- SpiesAcceptEveryMessage2
  ctrans [AcceptMsg2] :
         <(S : Spies)|(ncs =  MSET1), (agents = MSET2), (keys = MyK), (rolei = RI), (X)>
         msg(WHO, S, crypt(K, (NI, NR, HI)))(CONF)
      => <(S : Spies)|(ncs = NR U NI U MSET1), (agents = HI U MSET2), (keys = MyK),
                      (rolei = run(NI, nobody, NR) U RI), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [AcceptMsg2LikeFriend] :
         <(S : Spies)|(ncs =  MSET1), (agents = MSET2), (keys = MyK),
                      (rolei = run(N, HI, empty-msg) U RI), (X)>
         msg(WHO, S, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(ncs = NR U NI U MSET1), (agents = W U MSET2),
                      (keys = MyK), (rolei = run(N, HI, NR) U RI), (X)>(CONF)
```

```
  if (WHO =/= S) and (keypair(K) == MyK) and (NI == N) and (NI == W) .

  ctrans [AcceptCryptMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR, W)) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

-- SpiesAcceptEveryMessage3
  ctrans [AcceptMsg3Auth] :
         <(S : Spies)|(ec = EC), (ncs =  MSET1), (keys = MyK), (roler = RR), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(ec = ecom(r, N, nobody, nonce) U EC),
                      (ncs = N U MSET1), (keys = MyK), (roler = RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) and not(appear?(N, RR)) .

  ctrans [AcceptMsg3LikeFriend] :
         <(S : Spies)|(ec = EC), (ncs = MSET1), (keys = MyK),
                      (roler = run(NR, HI, NI) U RR), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(ec = ecom(r, NR, HI, NI) U EC), (ncs = N U MSET1),
                      (keys = MyK), (roler = RR), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) == MyK) and (NR == N).

  ctrans [AcceptCryptMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, S, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (keypair(K) =/= MyK) .

** define intercept message
-- Spies Intercept Message1
  ctrans [InterceptMsg1] :
         <(S : Spies)|(ncs = MSET1), (msgs = MSET2), (agents = MSET3),
                      (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (msgs = MSET2), (agents = W U MSET3),
                      (keys = MyK), (X)>(CONF)
  if (A =/= S) and (B =/= S) and (keypair(K) == MyK) .

  ctrans [GetMsg1] :
         <(S : Spies)|(ncs = MSET1), (msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(ncs = MSET1), (msgs = crypt(K, (N, W)) U MSET2),
                      (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- Spies Intercept Message2
  ctrans [InterceptMsg2] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(ncs = NR U NI U MSET1), (agents = W U MSET2),
                      (keys = MyK), (X)>(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) == MyK) .

  ctrans [GetMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR, W)) U MSET2), (keys = MyK), (X)>(CONF)
```

```
  if (WHO =/= S) and (HI =/= S) and (keypair(K) =/= MyK) .

-- Spies Intercept Message3
  ctrans [InterceptMsg3] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [GetMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

** define Spies' Overhear
--  Spies Overhear Message1
  ctrans [OHknowMsg1] :
         <(S : Spies)|(ncs = MSET1), (agents = MSET3), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (agents = W U MSET3), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [OHMsg1] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (N, W)) U MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, (N, W)))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- Spies Overhear Message2
  ctrans [OHknowMsg2] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(ncs = NI U NR U MSET1), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) == MyK) .

  ctrans [OHMsg2] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
      => <(S : Spies)|(msgs = crypt(K, (NI, NR, W)) U MSET2), (keys = MyK), (X)>
         msg(WHO, HI, crypt(K, (NI, NR, W)))(CONF)
  if (WHO =/= S) and (HI =/= S) and (keypair(K) =/= MyK) .

-- Spies Overhear Message3
  ctrans [OHknowMsg3] :
         <(S : Spies)|(ncs = MSET1), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(ncs = N U MSET1), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
  if (HI =/= S) and (WHO =/= S) and (keypair(K) == MyK) .

  ctrans [OHMsg3] :
         <(S : Spies)|(msgs = MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
      => <(S : Spies)|(msgs = crypt(K, N) U MSET2), (keys = MyK), (X)>
         msg(HI, WHO, crypt(K, N))(CONF)
```

```
  if (HI =/= S) and (WHO =/= S) and (keypair(K) =/= MyK) .

-- detection of some situations
  ctrans [AuthFF] :
         <(A : Friends)|(ec = ecom(i, NI, HI, NR) U EC), (X)>
         <(B : Friends)|(ec = ecom(r, NR, WHO, NI) U EC'), (X')>(CONF)
      => AuthFriendFriend
  if (HI == B) and (WHO == A) .

  ctrans [AuthFS] :
         <(A : Friends)|(ec = ecom(i, NI, HI, NR) U EC), (X)>
         <(S : Spies)|(ec = ecom(r, NR, WHO, NI) U EC'), (X')>(CONF)
      => AuthFriendSpy
  if (HI == S) and (WHO == A) .

  ctrans [AuthSF] :
         <(S : Spies)|(ec = ecom(i, NI, HI, NR) U EC'), (X')>
         <(A : Friends)|(ec = ecom(r, NR, WHO, NI) U EC), (X)>(CONF)
      => AuthSpyFriend
  if (WHO == S) and (HI == A) .

  ctrans [SpyMasqResp] :
         <(A : Friends)|(ec = ecom(i, NI, WHO, NR) U EC), (X)>
         <(S : Spies)|(ec = ecom(r, NR, A, NI) U EC'), (X')>(CONF)
      => SpyMasqResp
  if WHO =/= S .

  ctrans [SpyMasqInit] :
         <(A : Friends)|(ec = ecom(r, NR, WHO, NI) U EC), (X)>
         <(S : Spies)|(ec = ecom(i, NI, A, NR) U EC'), (X')>(CONF)
      => SpyMasqInit
  if WHO =/= S .

  ctrans [WhoMasq] :
         <(A : Friends)|(ec = ecom(i, NR, HI, NI) U EC), (X)>
         <(B : Friends)|(ec = ecom(r, NI, WHO, NR) U EC'), (X')>(CONF)
      => WhoMasq
  if HI =/= B or WHO =/= A .
}
```

d 実行例

```
** ************************************************************ **
** FILE : simulate.mod                                          **
** CONTENTS: Simulation of Needham-Schroeder Public-Key Protocol    **
** AUTHOR: Naoki Kinjo                                          **
** ************************************************************ **

in attribute.mod
in nspk.mod
in nspk-rv.mod

mod SIMULATE1 {
protecting(NSPK)

op  NSPK-FF : -> Configuration          ** Friend と Friend の通信
op  NSPK-FS : -> Configuration          ** Friend と Spy の通信
op  NSPK-FFS : -> Configuration         ** Friend と Friend の通信 (Spy が傍観)
```

```
op  NSPK-FSF : -> Configuration        ** Friend と Spy の通信 (Friend が傍観)

ops alice bob : -> Friend
ops spy : -> Spy

trans NSPK-FF
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(bob) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)), (rolei = no-run),
                       (roler = no-run), (dcom = empty-msg), (cnt = 1)> .

trans NSPK-FS
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)), (rolei = no-run),
                     (roler = no-run), (dcom = empty-msg), (cnt = 1),
                     (ncs = empty-msg), (msgs = empty-msg), (agents = empty-msg)> .

trans NSPK-FFS
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(bob) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)), (rolei = no-run),
                       (roler = no-run), (dcom = empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)), (rolei = no-run), (roler = no-run),
                     (dcom = empty-msg), (cnt = 1), (ncs = empty-msg),
                     (msgs = empty-msg), (agents = bob U spy U empty-msg)> .

trans NSPK-FSF
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)), (rolei = no-run),
                       (roler = no-run), (dcom = empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)), (rolei = no-run), (roler = no-run),
                     (dcom = empty-msg), (cnt = 1), (ncs = empty-msg),
                     (msgs = empty-msg), (agents = bob U spy U empty-msg)> .
}

exec in SIMULATE1 : NSPK-FF .
exec in SIMULATE1 : NSPK-FS .
exec in SIMULATE1 : NSPK-FFS .
exec in SIMULATE1 : NSPK-FSF .

mod SIMULATE2 {
protecting(NSPK-RV)

op  NSPK-RV-FF : -> Configuration
op  NSPK-RV-FS : -> Configuration
op  NSPK-RV-FFS : -> Configuration
op  NSPK-RV-FSF : -> Configuration
op  NSPK-RV-FSS : -> Configuration

ops alice bob : -> Friend
ops spy : -> Spy

trans NSPK-RV-FF
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(bob) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)), (rolei = no-run),
                       (roler = no-run), (dcom = empty-msg), (cnt = 1)> .
```

```
trans NSPK-RV-FS
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)), (rolei = no-run),
                         (roler = no-run), (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)), (rolei = no-run), (roler = no-run),
                     (dcom = empty-msg), (cnt = 1), (ncs = empty-msg),
                     (msgs = empty-msg), (agents = empty-msg)> .

trans NSPK-RV-FFS
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)),
                         (rolei = no-run), (roler = no-run),
                         (dcom = nspkcom(bob) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)),
                       (rolei = no-run), (roler = no-run),
                       (dcom = empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)),
                     (rolei = no-run), (roler = no-run), (dcom = empty-msg),
                     (cnt = 1), (ncs = empty-msg), (msgs = empty-msg),
                     (agents = bob U spy U empty-msg)> .

trans NSPK-RV-FSF
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)),
                         (rolei = no-run), (roler = no-run),
                         (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(bob : Friends)|(ec = no-ec), (keys = seck(bob)),
                       (rolei = no-run), (roler = no-run),
                       (dcom = empty-msg), (cnt = 1)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)),
                     (rolei = no-run), (roler = no-run), (dcom = empty-msg),
                     (cnt = 1), (ncs = empty-msg), (msgs = empty-msg),
                     (agents = bob U spy U empty-msg)> .

trans NSPK-RV-FSS
   => <(alice : Friends)|(ec = no-ec), (keys = seck(alice)),
                         (rolei = no-run), (roler = no-run),
                         (dcom = nspkcom(spy) U empty-msg), (cnt = 1)>
      <(spy2 : Spies)|(ec = no-ec), (keys = seck(spy)),
                     (rolei = no-run), (roler = no-run), (dcom = empty-msg),
                     (cnt = 1), (ncs = empty-msg), (msgs = empty-msg),
                     (agents = spy U spy2 U empty-msg)>
      <(spy : Spies)|(ec = no-ec), (keys = seck(spy)),
                     (rolei = no-run), (roler = no-run), (dcom = empty-msg),
                     (cnt = 1), (ncs = empty-msg), (msgs = empty-msg),
                     (agents = spy2 U spy U empty-msg)> .
}

exec in SIMULATE2 : NSPK-RV-FF .
exec in SIMULATE2 : NSPK-RV-FS .
exec in SIMULATE2 : NSPK-RV-FFS .
exec in SIMULATE2 : NSPK-RV-FSF .
exec in SIMULATE2 : NSPK-RV-FSS .
```

付録 D

振舞仕様による仕様記述

a 主体

```
** ********************************************************************* **
** FILE : prinsipal.mod                                                  **
** CONTENTS: Specification of Principal                                  **
** AUTHOR: Naoki Kinjo                                                   **
** ********************************************************************* **

-- 主体の識別子
mod! PID {
  [ PId ]
}


-- 状態を表す識別子
mod! SID {
  [ SId ]
}


-- 状態値
mod!  STATE-VALUE {
  [ SV < SVs ]

  op  nothing : -> SVs              ** 状態値の初期値
  op  _in_ : SV SVs -> Bool         ** 状態値の有無
  ops add del : SV SVs -> SVs       ** 状態の追加・削除

  vars V V' : SV
  var  S : SVs

  eq  V in nothing = false .
  ceq V in add(V', S) = true         if V == V' .
  ceq V in add(V', S) = false         if V =/= V' .

  eq  del(V, nothing) = nothing .
  ceq del(V, add(V', S)) = S         if V == V' .
  ceq del(V, add(V', S)) = add(V', del(V, S))         if V =/= V' .
```

```
}

-- 主体の状態の定義
mod! STATES {
  protecting(SID + STATE-VALUE)
  [ State < States ]

  op init-state : -> States         ** 初期状態
  op (_=_) : SId SVs  -> State      ** 状態は識別子と状態値で定義する
  op (_,_) : States States -> States {assoc comm id: init-state} ** 主体の状態群

  ops states notes : -> SId         ** 状態の識別子
}


-- 主体の定義
mod! PRINCIPAL {
  protecting(PID + SID + STATES)
  [ Principal ]

  op peopleless : -> Principal              ** 主体の初期値
  op (_:_) : PId States -> Principal        ** 主体の定義
}
```

b ネットワークの仕様

```
** ********************************************************************* **
** FILE : network.mod                                                    **
** CONTENTS: Specification of Network State                              **
** AUTHOR: Naoki Kinjo                                                   **
** ********************************************************************* **

in ex-bool.mod
in message.mod
in prinsipal.mod

-- メッセージの集合
mod MESSAGES {
  protecting(EX-BOOL + PROTOCOL-MESSAGE)

  op _in_ : Msg Msg -> Bool         ** メッセージの有無
  op _U_ : Msg Msg -> Msg {assoc comm id: empty-msg} ** メッセージの追加

  var  A : Agent
  var  N : Nonce
  var  K : Key
  vars M M' M1 M2 : Msg

  eq  empty-msg in M = true .
  eq  M in empty-msg = false .
  eq  M in A = (M == A) .
  eq  M in N = (M == N) .
  eq  M in K = (M == K) .
  eq  M in crypt(K, M') = (M == crypt(K, M')) .
  eq  M in (M1, M2) = (M == (M1, M2)) .
  eq  M in hash(M') = (M == hash(M')) .
  eq  M in (A U M') = (M == A) v (M in M') .
```

```
  eq  M in (N U M') = (M == N) v (M in M') .
  eq  M in (K U M') = (M == K) v (M in M') .
  eq  M in (crypt(K, M') U M1) = (M == crypt(K, M')) v (M in M1) .
  eq  M in ((M1, M2) U M') = (M == (M1, M2)) v (M in M') .
  eq  M in (hash(M') U M1) = (M == hash(M')) v (M in M1) .
  ceq M in M' = true        if M == M' .
  ceq M in M' = false        if M =/= M' .
}


-- 主体の集合
mod! PRINCIPALS {
  protecting(EX-BOOL + PRINCIPAL)
  protecting(MESSAGES)
  [ Agent < PId ]
  [ Msg Int < SV ]

  op _in_ : PId Principal -> Bool        ** 主体の有無
** 主体の追加
  op _U_ : Principal Principal -> Principal {assoc comm id: peopleless}
** 主体の削除
  op _-_ : Principal Principal -> Principal

** 主体からの状態抽出
  op getnotes : PId Principal -> Msg        ** 主体の既知情報の抽出
  op getstates : PId Principal -> SVs       ** 主体の状態の抽出

  vars A B C : PId
  vars S S' : States
  vars P P' P1 P2 : Principal
  var  M : Msg
  var  V : SVs

  eq  nobody in P = true .
  eq  A in peopleless = false .
  eq  A in (B : S) = (A == B) .
  eq  A in ((B : S) U P) = (A in (B : S)) v (A in P) .

  eq  P - peopleless = P .
  eq  peopleless - P = peopleless .
  ceq (A : S) - (B : S') = peopleless        if (A == B) and (S == S') .
  ceq (A : S) - (B : S') = (A : S)        if (A =/= B) or (S =/= S') .
  eq  (A : S) - ((B : S') U P) = ((A : S) - (B : S')) - P2 .
  ceq ((A : S) U P) - (B : S') = P        if (A == B) and (S == S') .
  ceq ((A : S) U P) - (B : S') = (A : S) U (P - (B : S'))   if (A =/= B) or (S =/= S') .
  eq  ((A : S) U P1) - ((B : S') U P2) = (((A : S) U P1) - (B : S')) - P2 .

  eq  getnotes(A, peopleless) = empty-msg .
  ceq getnotes(A, (B : (notes = M), S)) = M        if A == B .
  ceq getnotes(A, (B : (notes = M), S)) = empty-msg        if A =/= B .
  ceq getnotes(A, (B : (notes = M), S) U P) = M        if A == B .
  ceq getnotes(A, (B : (notes = M), S) U P) = getnotes(A, P)        if A =/= B .

  eq  getstates(A, peopleless) = nothing .
  ceq getstates(A, (B : (states = V), S)) = V        if A == B .
  ceq getstates(A, (B : (states = V), S)) = nothing        if A =/= B .
  ceq getstates(A, (B : (states = V), S) U P) = V        if A == B .
  ceq getstates(A, (B : (states = V), S) U P) = getstates(A, P)        if A =/= B .
}
```

```
-- ネットワークの定義
mod* NETWORK {
  protecting(MESSAGES + PRINCIPALS)
  *[ NW ]*

  op net : -> NW                          ** ネットワークの初期状態

  bop getmsg : NW -> Msg                  ** ネットワーク上のメッセージ
  bop getpri : NW -> Principal            ** ネットワーク上の主体
  bop closure : PId NW -> Msg             ** 主体が捏造可能なメッセージ
  bop closure? : PId Msg NW -> Bool       ** 主体はあるメッセージを捏造可能か

  op analz : PId Msg -> Msg               ** 主体が可視のメッセージ
  op appear : Msg -> Msg                  ** ネットワーク上に流れたメッセージ
  op nonce? : Msg Msg -> Bool             ** ネットワーク上に存在しないメッセージか

  var  NET : NW
  var  A : PId
  var  S : States
  var  K : Key
  vars P P' : Principal
  var  Z : Agent
  var  N : Nonce
  vars X Y M : Msg

  eq  getmsg(net) = empty-msg .

  eq  getpri(net) = peopleless .

  eq  closure(A, NET) = getnotes(A, getpri(NET)) U analz(A, getmsg(NET)) .

  eq  analz(A, empty-msg) = empty-msg .
  ceq analz(A, crypt(K, X) U M) = crypt(K, X) U analz(A, X U M)
      if (keypair(K) in analz(A, M)) .
  ceq analz(A, crypt(K, X) U M) = crypt(K, X) U analz(A, M)
      if not(keypair(K) in analz(A, M)) .
  eq  analz(A, (X, Y) U M) = (X, Y) U analz(A, X U Y U M) .
  eq  analz(A, hash(X) U M) = hash(X) U analz(A, M) .
  eq  analz(A, Z U M) = Z U analz(A, M) .
  eq  analz(A, N U M) = N U analz(A, M) .
  eq  analz(A, K U M) = K U analz(A, M) .
  eq  analz(A, X U M) = X U analz(A, M) .
  eq  analz(A, X) = X U analz(A, empty-msg) .

  eq  closure?(A, crypt(K, X), NET) = (closure?(A, K, NET) and closure?(A, X, NET))
        or (crypt(K, X) in closure(A, NET)) or nonce?(crypt(K, X), getmsg(NET)) .
  eq  closure?(A, (X, Y), NET) = (closure?(A, X, NET) and closure?(A, Y, NET))
        or ((X, Y) in closure(A, NET)) or nonce?((X, Y), getmsg(NET)) .
  eq  closure?(A, Z, NET) = true .
  eq  closure?(A, pubk(Z), NET) = true .
  eq  closure?(A, X, NET) = (X in closure(A, NET)) or nonce?(X, getmsg(NET)).

  eq  appear(empty-msg) = empty-msg .
  eq  appear(crypt(K, X) U M) = crypt(K, X) U appear(X U M) .
  eq  appear((X, Y) U M) = (X, Y) U appear(X U Y U M) .
  eq  appear(hash(X) U M) = hash(X) U appear(X U M) .
  eq  appear(Z U M) = Z U appear(M) .
  eq  appear(N U M) = N U appear(M) .
```

```
  eq  appear(K U M) = K U appear(M) .
  eq  appear(X U M) = X U appear(M) .
  eq  appear(X) = X U appear(empty-msg) .

  eq  nonce?(X, M) = not(X in appear(M)) .
}
```

c NSPK Protocol の仕様

```
** ************************************************************ **
** FILE : nspk.mod                                              **
** CONTENTS: Specification of Needham-Schroeder Public Key Protocol **
** AUTHOR: Naoki Kinjo                                          **
** ************************************************************ **

in network.mod

-- -------------------------------------------------------------
-- NSPK Protcol
-- -------------------------------------------------------------
mod NSPK {
  using(NETWORK)

  bop make : NW PId Msg -> NW                    ** メッセージの捏造
  bop nspk0 : NW PId Nonce SV Msg -> NW          ** NSPK のステップ 1
  bop nspk1 : NW PId Msg SV -> NW                ** NSPK のステップ 2
  bop nspk2 : NW PId SV Nonce SV Msg -> NW       ** NSPK のステップ 3
  bop nspk3 : NW PId SV Msg SV -> NW             ** NSPK のステップ 4
  bop nspk4 : NW PId SV SV Msg -> NW             ** NSPK のステップ 5
  bop nspk5 : NW PId SV Msg SV -> NW             ** NSPK のステップ 6

** 主体の状態値の定義：主体が通信のどのステップにあるかの値を定義
  ops nspkS1 nspkR1 : PId Nonce -> SV
  ops nspkS2 nspkR2 nspkS3 nspkR3 : PId Nonce Nonce -> SV

  var  NET : NW
  vars A B X : Agent
  var  S : States
  vars NI NR : Nonce
  var  M : Msg
  var  P : Principal

** ネットワーク上のメッセージを抽出する
-- Message Make
  ceq getmsg(make(NET, A, M)) = M U getmsg(NET) if closure?(A, M, NET) .

-- Message1 send
  ceq getmsg(nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, X)))) =
        crypt(pubk(B), (NI, X)) U getmsg(NET)
  if nonce?(NI, getmsg(NET)) .

-- Message1 receive
  eq  getmsg(nspk1(NET, B, crypt(pubk(B), (NI, X)), nspkR1(X, NI))) = getmsg(NET) .

-- Message2 send
  ceq getmsg(nspk2(NET, B, nspkR1(X, NI), NR, nspkS2(X, NI, NR),
        crypt(pubk(X), (NI, NR)))) = crypt(pubk(X), (NI, NR)) U getmsg(NET)
  if nonce?(NR, getmsg(NET)) and (nspkR1(X, NI) in getstates(B, getpri(NET))) .
```

```
-- Message2 receive
  eq  getmsg(nspk3(NET, A, nspkS1(X, NI), crypt(pubk(A), (NI, NR)), nspkR2(X, NI, NR))) =
        getmsg(NET) .

-- Message3 send
  ceq getmsg(nspk4(NET, A, nspkR2(X, NI, NR), nspkS3(X, NI, NR), crypt(pubk(X), NR))) =
        crypt(pubk(X), NR) U getmsg(NET)
  if (nspkR2(X, NI, NR) in getstates(A, getpri(NET))) .

-- Message3 receive
  eq  getmsg(nspk5(NET, B, nspkS2(X, NI, NR), crypt(pubk(B), NR), nspkR3(X, NI, NR))) =
        getmsg(NET).


** ネットワーク上の主体を抽出する
-- Message Make
  eq  getpri(make(NET, A, M)) = getpri(NET) .

-- Message1 send
  eq  getpri(nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, X)))) =
        (A : (states = add(nspkS1(B, NI), nothing)), (notes = seck(A) U NI)) U
          getpri(NET) .

-- Message1 receive
  ceq getpri(nspk1(NET, B, crypt(pubk(B), (NI, X)), nspkR1(X, NI))) =
        (B : (states = add(nspkR1(X, NI), nothing)), (notes = seck(B) U NI)) U
          getpri(NET)
  if (crypt(pubk(B), (NI, X)) in getmsg(NET)) .

-- Message2 send
  ceq getpri(nspk2(NET, B, nspkR1(X, NI), NR, nspkS2(X, NI, NR),
                                                crypt(pubk(X), (NI, NR)))) =
        (B : (states = add(nspkS2(X, NI, NR), del(nspkR1(X, NI),
          getstates(B, getpri(NET))))), (notes = NR U getnotes(B, getpri(NET)))) U
        (getpri(NET) - (B : (states = getstates(B, getpri(NET))),
                                      (notes = getnotes(B, getpri(NET))))
  if nonce?(NR, getmsg(NET)) and (nspkR1(X, NI) in getstates(B, getpri(NET))) .

-- Message2 receive
  ceq getpri(nspk3(NET, A, nspkS1(X, NI), crypt(pubk(A), (NI, NR)), nspkR2(X, NI, NR))) =
        (A : (states = add(nspkR2(X, NI, NR), del(nspkS1(X, NI),
           getstates(A, getpri(NET))))), (notes = NR U getnotes(A, getpri(NET)))) U
        (getpri(NET) - (A : (states = getstates(A, getpri(NET))),
                                       (notes = getnotes(A, getpri(NET))))
  if (nspkS1(X, NI) in getstates(A, getpri(NET))) and
                                  (crypt(pubk(A), (NI, NR)) in getmsg(NET)) .

-- Message3 send
  ceq getpri(nspk4(NET, A, nspkR2(X, NI, NR), nspkS3(X, NI, NR), crypt(pubk(X), NR))) =
        (A : (states = add(nspkS3(X, NI, NR), del(nspkR2(X, NI, NR),
           getstates(A, getpri(NET))))), (notes = getnotes(A, getpri(NET)))) U
        (getpri(NET) - (A : (states = getstates(A, getpri(NET))),
                                       (notes = getnotes(A, getpri(NET))))
  if (nspkR2(X, NI, NR) in getstates(A, getpri(NET))) .

-- Message3 receive
  ceq getpri(nspk5(NET, B, nspkS2(X, NI, NR), crypt(pubk(B), NR), nspkR3(X, NI, NR))) =
        (B : (states = add(nspkR3(X, NI, NR), del(nspkS2(X, NI, NR),
```

```
          getstates(B, getpri(NET))))), (notes = getnotes(B, getpri(NET)))) U
        (getpri(NET) - (B : (states = getstates(B, getpri(NET))),
                                    (notes = getnotes(B, getpri(NET)))))
  if (nspkS2(X, NI, NR) in getstates(B, getpri(NET))) and
                                     (crypt(pubk(B), NR) in getmsg(NET)) .
}
```

d NSPK Protocol 改訂版の仕様

```
** ************************************************************ **
** FILE : nspk-rv.mod                                           **
** CONTENTS: Specification of Needham-Schroeder Public Key Protocol **
**                                                  ReVision    **
** AUTHOR: Naoki Kinjo                                          **
** ************************************************************ **

-- -------------------------------------------------------------
-- NSPK-RV Protcol
-- -------------------------------------------------------------
mod! NSPK-RV {
  using(NETWORK)

  bop make : NW PId Msg -> NW                    ** メッセージの捏造
  bop nspk0 : NW PId Nonce SV Msg -> NW          ** NSPK のステップ 1
  bop nspk1 : NW PId Msg SV -> NW                ** NSPK のステップ 2
  bop nspk2 : NW PId SV Nonce SV Msg -> NW       ** NSPK のステップ 3
  bop nspk3 : NW PId SV Msg SV -> NW             ** NSPK のステップ 4
  bop nspk4 : NW PId SV SV Msg -> NW             ** NSPK のステップ 5
  bop nspk5 : NW PId SV Msg SV -> NW             ** NSPK のステップ 6

  vars A B X Y : Agent
  var  S : States
  vars NI NR : Nonce
  var  M : Msg
  var  P : Principal

** ネットワーク上のメッセージを抽出する
-- Message Make
  ceq getmsg(make(NET, A, M)) = M U getmsg(NET) if closure?(A, M, NET) .

-- Message1 send
  ceq getmsg(nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, X)))) =
        crypt(pubk(B), (NI, X)) U getmsg(NET)
  if nonce?(NI, getmsg(NET)) .

-- Message1 receive
  eq  getmsg(nspk1(NET, B, crypt(pubk(B), (NI, X)), nspkR1(X, NI))) = getmsg(NET) .

-- Message2 send
  ceq getmsg(nspk2(NET, B, nspkR1(X, NI), NR, nspkS2(X, NI, NR),
                                              crypt(pubk(X), (NI, NR, B)))) =
        crypt(pubk(X), (NI, NR, B)) U getmsg(NET)
  if nonce?(NR, getmsg(NET)) and (nspkR1(X, NI) in getstates(B, getpri(NET))) .

-- Message2 receive
  eq  getmsg(nspk3(NET, A, nspkS1(X, NI), crypt(pubk(A), (NI, NR, B)),
                                                    nspkR2(X, NI, NR))) =
```

```
        getmsg(NET) .

-- Message3 send
  ceq getmsg(nspk4(NET, A, nspkR2(X, NI, NR), nspkS3(X, NI, NR), crypt(pubk(X), NR))) =
        crypt(pubk(X), NR) U getmsg(NET)
  if (nspkR2(X, NI, NR) in getstates(A, getpri(NET))) .

-- Message3 receive
  eq  getmsg(nspk5(NET, B, nspkS2(X, NI, NR), crypt(pubk(B), NR), nspkR3(X, NI, NR))) =
       getmsg(NET).


** ネットワーク上の主体を抽出する
-- Message Make
  eq  getpri(make(NET, A, M)) = getpri(NET) .

-- Message1 send
  eq  getpri(nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, X)))) =
        (A : (states = add(nspkS1(B, NI), nothing)), (notes = seck(A) U NI)) U
          getpri(NET) .

-- Message1 receive
  ceq getpri(nspk1(NET, B, crypt(pubk(B), (NI, X)), nspkR1(X, NI))) =
        (B : (states = add(nspkR1(X, NI), nothing)), (notes = seck(B) U NI)) U
          getpri(NET)
  if (crypt(pubk(B), (NI, X)) in getmsg(NET)) .

-- Message2 send
  ceq getpri(nspk2(NET, B, nspkR1(X, NI), NR, nspkS2(X, NI, NR),
                                              crypt(pubk(X), (NI, NR, B)))) =
        (B : (states = add(nspkS2(X, NI, NR), del(nspkR1(X, NI),
           getstates(B, getpri(NET))))), (notes = NR U getnotes(B, getpri(NET)))) U
        (getpri(NET) - (B : (states = getstates(B, getpri(NET))),
                                        (notes = getnotes(B, getpri(NET)))))
  if nonce?(NR, getmsg(NET)) and (nspkR1(X, NI) in getstates(B, getpri(NET))) .

-- Message2 receive
  ceq getpri(nspk3(NET, A, nspkS1(X, NI), crypt(pubk(A), (NI, NR, B)),
                                                    nspkR2(X, NI, NR))) =
        (A : (states = add(nspkR2(X, NI, NR), del(nspkS1(X, NI),
           getstates(A, getpri(NET))))), (notes = NR U getnotes(A, getpri(NET)))) U
        (getpri(NET) - (A : (states = getstates(A, getpri(NET))),
                                          (notes = getnotes(A, getpri(NET)))))
  if (nspkS1(X, NI) in getstates(A, getpri(NET))) and
                     (crypt(pubk(A), (NI, NR)) in getmsg(NET)) and (B == X) .

-- Message3 send
  ceq getpri(nspk4(NET, A, nspkR2(X, NI, NR), nspkS3(X, NI, NR), crypt(pubk(X), NR))) =
        (A : (states = add(nspkS3(X, NI, NR), del(nspkR2(X, NI, NR),
           getstates(A, getpri(NET))))), (notes = getnotes(A, getpri(NET)))) U
        (getpri(NET) - (A : (states = getstates(A, getpri(NET))),
                                          (notes = getnotes(A, getpri(NET)))))
  if (nspkR2(X, NI, NR) in getstates(A, getpri(NET))) .

-- Message3 receive
  ceq getpri(nspk5(NET, B, nspkS2(X, NI, NR), crypt(pubk(B), NR), nspkR3(X, NI, NR))) =
        (B : (states = add(nspkR3(X, NI, NR), del(nspkS2(X, NI, NR),
           getstates(B, getpri(NET))))), (notes = getnotes(B, getpri(NET)))) U
        (getpri(NET) - (B : (states = getstates(B, getpri(NET))),
```

```
                                        (notes = getnotes(B, getpri(NET)))))
  if (nspkS2(X, NI, NR) in getstates(B, getpri(NET))) and
                                         (crypt(pubk(B), NR) in getmsg(NET)) .
}
```

e 実行例

```
** ************************************************************ **
** FILE : simulate.mod                                          **
** CONTENTS: Simulation of Needham-Schroeder Public Key Protocol    **
** AUTHOR: Naoki Kinjo                                          **
** ************************************************************ **

in nspk.mod
in nspk-rv.mod

mod SIMULATE1 {
  protecting(NSPK)

  ops alice bob chris : -> Agent

  op  RESULT : Agent Agent Agent -> NW
  ops STEP0 STEP1 : NW Agent Agent Agent Nonce -> NW
  ops STEP2 STEP3 STEP4 STEP5 : NW Agent Agent Agent Nonce Nonce -> NW

  var  NET : NW
  vars A B C : Agent
  vars NI NR : Nonce

  trans STEP0(NET, A, B, C, NI) =>
nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, C))) .

  trans STEP1(NET, A, B, C, NI) =>
nspk1(STEP0(NET, A, B, C, NI), B,
crypt(pubk(B), (NI, C)), nspkR1(C, NI)) .

  trans STEP2(NET, A, B, C, NI, NR) =>
nspk2(STEP1(NET, A, B, C, NI), B, nspkR1(C, NI), NR,
nspkS2(C, NI, NR), crypt(pubk(C), (NI, NR))) .

  trans STEP3(NET, A, B, C, NI, NR) =>
nspk3(STEP2(NET, A, B, C, NI, NR), A, nspkS1(B, NI),
crypt(pubk(A), (NI, NR)), nspkR2(B, NI, NR)) .

  trans STEP4(NET, A, B, C, NI, NR) =>
nspk4(STEP3(NET, A, B, C, NI, NR), A, nspkR2(B, NI, NR),
nspkS3(B, NI, NR), crypt(pubk(B), NR)) .

  trans STEP5(NET, A, B, C, NI, NR) =>
nspk5(STEP4(NET, A, B, C, NI, NR), B, nspkS2(C, NI, NR),
crypt(pubk(B), NR), nspkR3(C, NI, NR)) .

  trans RESULT(A, B, C) =>
STEP5(net, A, B, C, n(A, 0), n(B, 0)) .
}

exec in SIMULATE1 : RESULT(alice, bob, alice) .
```

```
mod SIMULATE2 {
  protecting(NSPK-RV)

  ops alice bob chris david : -> Agent

  op  RESULT : Agent Agent Agent Agent -> NW
  ops STEP0 STEP1 : NW Agent Agent Agent Nonce -> NW
  ops STEP2 STEP3 STEP4 STEP5 : NW Agent Agent Agent Agent Nonce Nonce -> NW

  var  NET : NW
  vars A B C D : Agent
  vars NI NR : Nonce

  trans STEP0(NET, A, B, C, NI) =>
nspk0(NET, A, NI, nspkS1(B, NI), crypt(pubk(B), (NI, C))) .

  trans STEP1(NET, A, B, C, NI) =>
nspk1(STEP0(NET, A, B, C, NI), B,
crypt(pubk(B), (NI, C)), nspkR1(C, NI)) .

  trans STEP2(NET, A, B, C, D, NI, NR) =>
nspk2(STEP1(NET, A, B, C, NI), B, nspkR1(C, NI), NR,
nspkS2(C, NI, NR), crypt(pubk(C), (NI, NR, D))) .

  trans STEP3(NET, A, B, C, D, NI, NR) =>
nspk3(STEP2(NET, A, B, C, D, NI, NR), A, nspkS1(B, NI),
crypt(pubk(A), (NI, NR, D)), nspkR2(B, NI, NR)) .

  trans STEP4(NET, A, B, C, D, NI, NR) =>
nspk4(STEP3(NET, A, B, C, D, NI, NR), A, nspkR2(B, NI, NR),
nspkS3(B, NI, NR), crypt(pubk(B), NR)) .

  trans STEP5(NET, A, B, C, D, NI, NR) =>
nspk5(STEP4(NET, A, B, C, D, NI, NR), B, nspkS2(C, NI, NR),
crypt(pubk(B), NR), nspkR3(C, NI, NR)) .

  trans RESULT(A, B, C, D) =>
STEP5(< empty-msg, peopleless >, A, B, C, D, n(A, 0), n(B, 0)) .
}

exec in  SIMULATE2 : RESULT(alice, bob, alice, bob) .

open SIMULATE1

  ops  a b c : -> Agent .

  exec closure?(a, n(a, 0), STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0)))
 and closure?(a, a, STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(b, n(a, 0), STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(b, a, STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(c, n(a, 0), STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(c, a, STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0))) .


  exec closure?(a, n(a, 0), STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(a, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(b, n(a, 0), STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(b, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(c, n(a, 0), STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(c, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
```

```
  exec closure?(a, n(a, 0), STEP0(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(a, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(b, n(a, 0), STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(b, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .
  exec closure?(c, n(a, 0), STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0)))
and closure?(c, a, STEP1(< empty-msg, peopleless >, a, b, a, n(a, 0))) .

close
```

参考文献

[1] Răzvan Diaconescu and Kokichi Futatsugi, CafeOBJ Report, World Scientific, 1998.

[2] Roger Needham and Michael Schroeder, Using Encryption for Authentication in Large Networks of Computers. Communication sf the ACM, 21(12):993-999, 1978.

[3] Gavin Lowe, An Attack on the Needham-Schroeder Public-Key Authentication Protocol Information Processing Letters, volume 56, number 3, pages 131-133.

[4] Gavin Lowe, Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In Proceedings of TACAS, volume 1055 of Lecture Notes in Computer Science, pages 147-166. Springer-Verlag, 1996.

[5] G.Denker and J.Meseguer, C.Talcott, Formal Specification and Analysis of Active Network and Communication Protocols: The Maude Experience Internal Report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.

[6] Masami Hagiya, Yozo Toda and Yoshiki Fukuba, Implementation and Verification of Authentication Protocols Using Proof Procedures in HOL 2nd IMC Enterprise Security Workshop, Information Media Center, Science University of Tokyo, November 1999.

[7] Lawrence C. Paulson The Inductive Approach to Verifying Cryptographic Protocols Journal of Computer Security, 6(1):85-128, 1998.

[8] 日経 BP 社編電子商取引のセキュリティ技術 -決済プロトコル実用システム構築に向けて 日経デジタルマネーシステム 別冊 電子商取引のセキュリティ技術, 日経 BP 社, 1998.