修 士 論 文

SPINを用いたセキュリティプロトコルの安全性自動検証

平成23年度 修了
三重大学大学院　工学研究科
博士前期課程　情報工学専攻
計算機ソフトウェア研究室

松田 有司

概 要

セキュリティプロトコルは，悪意ある第三者（以降，攻撃者と呼ぶ）からユーザとサーバ間の通信路を，認証と暗号化により保護する技術である．そのため，セキュリティプロトコルには脆弱性が存在してはならず，十分に安全性を検証する必要がある．しかし，手作業でセキュリティプロトコルの全動作を模倣し，脆弱性の有無を確認する安全性検証は，膨大な時間や労力がかかり，見落としも生じる．一方，形式手法を用いた安全性検証では自動で正確な検証ができる．その中でもモデル検査法は，セキュリティプロトコルの安全性を専門家でなくても検証でき，脆弱性の原因を確認できる．本研究の目的は，モデル検査法で既存手法よりも多くのセキュリティプロトコルの安全性を，現実時間で自動検証できる手法の提案である．

モデル検査法を用いたセキュリティプロトコルの安全性自動検証には，Khan（2005）らの手法がある．Khan らは情報漏洩の有無や，セキュリティプロトコル上で扱える情報等を定義し，モデル検査器 SPIN を用いて特定のセキュリティプロトコルの脆弱性の発見に成功した．しかし，本研究により Khan らの手法の定義の 2 つの問題点が明らかになる．1 つ目は情報漏洩の定義が不適切であり，情報漏洩の誤検出や検出漏れが生じる問題である．2 つ目はセキュリティプロトコル上で扱える情報の定義が不十分であり，僅かなセキュリティプロトコルの安全性しか検証できない問題である．

そのため，本研究では情報漏洩の定義を見直し，セキュリティプロトコルの役割に応じて指定した機密情報が攻撃者に取得された時のみ情報漏洩であると再定義する．また，セキュリティプロトコル上で複数の構成要素からなる複雑な情報を扱うための解析・合成規則の定義を見直し，複雑な鍵情報や hash 関数などの特別な関数から作成された情報もセキュリティプロトコル上で扱えるように再定義する．しかし，これらの新たな定義により，解析・合成規則で攻撃者の所持情報の中から組み合わせて作成できる機密情報の候補数が膨大になる．

この候補数が膨大になる問題に対し，本研究では逆向き合成規則を用いた新手法を提案する．この手法は，攻撃者の所持情報から解析・合成規則で機密情報を作成できるか判定するのではなく，機密情報から逆向き合成規則で機密情報を作成できる情報の組み合わせ候補を作成し，いずれかの組み合わせを攻撃者が所持しているか判定する．攻撃者の所持情報の組み合わせの数より，機密情報を作成できる情報の組み合わせの数が圧倒的に少ないため，この手法により計算時間とメモリ消費量を大幅に抑制できる．

目 次

第1章　背景と目的

1.1　研究の背景

インターネットバンキング等の多くの電子商業アプリケーションでは，SSL や SET のようなセキュリティプロトコルを用いて認証や暗号化を行い，ユーザとサーバ間の通信路を攻撃者から保護する（図 1.1）.

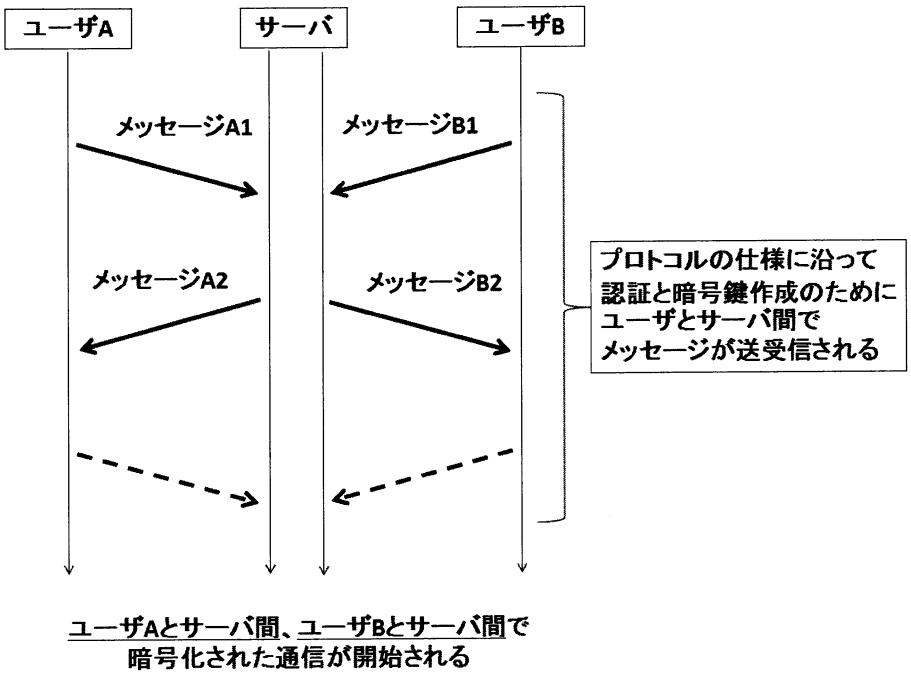

図 1.1: セキュリティプロトコルの概要

図 1.1 は，ユーザ A とユーザ B がサーバに対してメッセージを送受信している例である．セキュリティプロトコルはユーザ A とサーバ間，ユーザ B とサーバ間の通信路を認証と暗号化によって別々に保護するため，それぞれの通信路で様々なメッセージが送受信される．

しかし，セキュリティプロトコルに脆弱性が存在すると，攻撃者による機密情報の盗聴や悪用の危険性があるため，セキュリティプロトコルには十分な安全性検証が必要である．

一般に手作業でセキュリティプロトコルの全動作を模倣し，脆弱性の有無を確認する安全性検証は，膨大な時間や労力がかかり，見落としも生じる．一方，形式手法を用いた安全性検証では自動で正確に検証できる．そのため，セキュリティプロトコルの安全性検証に関する形式手法の様々な研究がなされてきた．

セキュリティプロトコルの安全性検証の形式手法は，大まかに次の 4 種類に分類できる [IPA04].

1. 状態探索法

2. 信念理論

3. 定理証明・帰納法

4. モデル検査法

状態探索法は状態空間を定義し，攻撃に成功するパスが存在するか検証器を用いて探索する手法であるが，認証が完了するかなど，プロセスが飢餓状態に陥らないこと（以降，活性と呼ぶ）を検証できず，未知の欠陥に対する攻撃についても検証できない欠点がある．

信念理論は認証プロトコルの検証用に開発されたBAN論理を用いて，送受信されるメッセージから推論される知識からプロトコルの欠陥を発見する手法であるが，モデルが明確でなく，完全性が成り立たない欠点がある．

定理証明・帰納法はプロトコルの実行ステップを形式仕様言語を用いて帰納的に定義し，証明支援系を用いて証明する手法であるが，活性がなく，証明に経験とテクニックが必要とされる欠点がある．

モデル検査法は有限状態システムとしてプロトコルをモデル化し，到達可能状態を探索して脆弱性の有無を検証する．また，モデル検査法のツールを用いると，脆弱性が存在する原因を図1.2のような状態遷移図で確認できる．図1.2は先の図1.1と対応しており，ユーザとサーバ間で送信したメッセージに合わせて状態の遷移する状態遷移図を表している．以降，脆弱性が有る状態を「拒否状態」と呼ぶ．

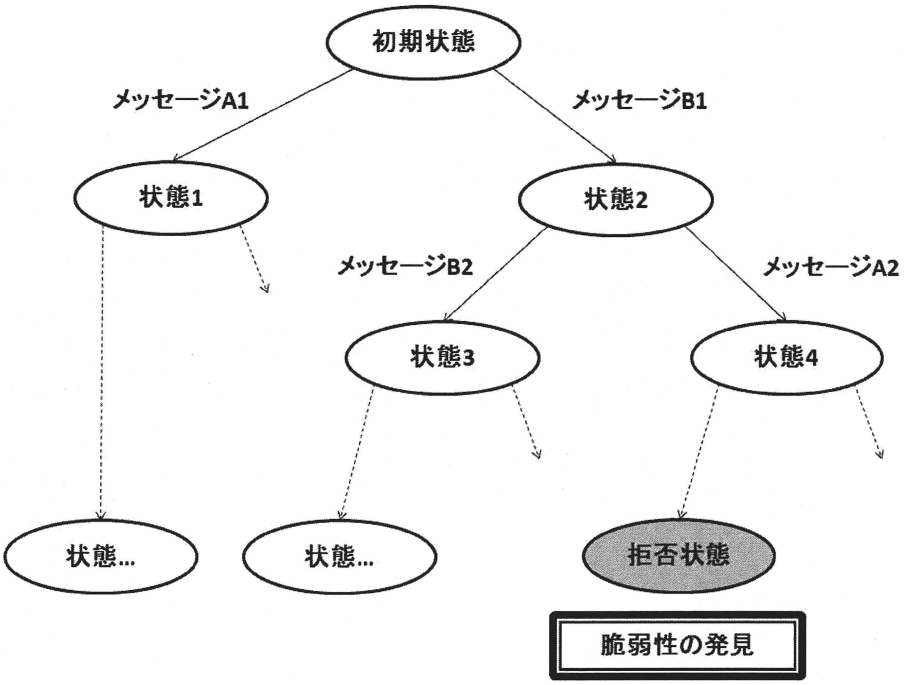

図 1.2: 状態遷移図

モデル検査法は自動化が可能であり，専門家でなくてもセキュリティプロトコルの安全性を検証できる利点がある．しかし，状態空間を制限しなければ，状態数が膨大になり現実時

間で探索できない問題（状態数爆発問題）が生じる．

本研究では，近年のハードウェア向上に伴い，これらの形式手法の中でもモデル検査法の問題が実用上では解決するのではないかと考え，モデル検査法を用いたセキュリティプロトコルの安全性検証の手法を提案する．

モデル検査法を用いたセキュリティプロトコルの安全性検証の既存手法には，Khan らの手法 [KMS05] がある．Khan らは情報漏洩の有無や攻撃者の動作を定義した上で，モデル検査器 SPIN を用い，特定のセキュリティプロトコルの安全性検証に成功した．しかし，安全性検証の際にセキュリティプロトコル上で扱える情報の種類は制限されており，大半のセキュリティプロトコルを検証できない問題がある．また，本研究により情報漏洩の有無の定義が不十分だと判明した．本研究では専門家が不要で自動化が可能なモデル検査法に着目し，そのツールである SPIN を用いて，セキュリティプロトコルの安全性を自動で検証できる手法を提案する．

1.2 研究の目的

本研究では，既存手法よりも扱える情報の種類が多い，大規模なセキュリティプロトコルの安全性検証を現実時間で可能な手法の提案を目的としている．そのため，既存手法ではセキュリティプロトコル上で扱えなかった複雑な情報を，本研究では情報操作規則の再定義により扱えるようにする．また，既存手法の問題点である情報漏洩の有無を再定義し，情報漏洩の誤検出や検出漏れを防ぐ．そして，提案手法を実装して特定のセキュリティプロトコルを検証し，評価と考察を行い，本研究の提案手法にて現実時間で検証可能なセキュリティプロトコルの種類や攻撃者の動作等を明らかにする．

第2章　モデル検査器SPINと既存研究

2.1　モデル検査器SPIN

モデル検査器SPINは，プロトコルの仕様をPromela言語で記述したプログラムを入力として，そのプログラムが取り得る全ての状態を生成し，拒否状態の有無を検証して結果を出力する．拒否状態が存在する場合，拒否状態への遷移過程を状態遷移図で確認できる（図2.1）．

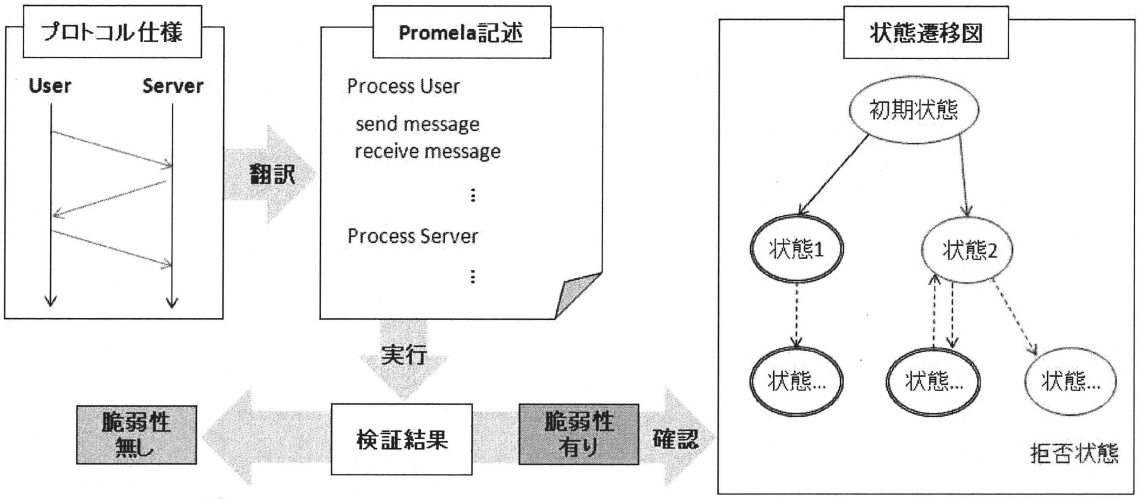

図 2.1: SPINの動作概要

Promela言語はC言語風の文法を採り入れており，C言語の履修者には親しみやすく，並行システムや分散システムのモデルを特有の文法にて記述できる．また，Promela言語のモデルはプロセスの集まりで構成されるので，ユーザやサーバ，攻撃者の各動作をプロセスとして定義できる．さらに，メッセージの送受信チャネルや，プロセス間の同期機構等を予め用意しており，実際のプロトコルの動作を容易に表現できる．そのため，SPINはプロトコルの安全性自動検証に適している．

2.2　既存研究

SPINを用いたセキュリティプロトコルの安全性検証の研究には，Khanらの研究がある．Khanらは，情報漏洩が起きるか否かを安全性の判断基準とし，セキュリティプロトコルの安全性自動検証法を提案した．

2.2.1 情報漏洩の定義と攻撃者モデル

Khan らは情報漏洩の定義を定義 2.2.1 の通りに定めた.

定義 2.2.1 (Khan らの情報漏洩の定義). 次の項目 1, 2 を同時に満たすとき, 情報漏洩と見なす.

1. ある状態で攻撃者が情報 A を所持している.
2. 項目 1 の状態以前に攻撃者は情報 A を所持しておらず, 攻撃者以外の動作主は情報 A を所持している.

また, 攻撃者は Dolev-Yao の攻撃者モデルの記述により 1 人で十分であるとされ [DY83], ユーザとサーバ間に入り込み, 全通信に対して次の盗聴・再送信・改ざん・なりすましの攻撃ができる.

- 盗聴は攻撃者がユーザとサーバ間で送信されたメッセージを取得すること.
- 再送信は攻撃者が取得したメッセージをそのままユーザやサーバに送信すること (図 2.2).

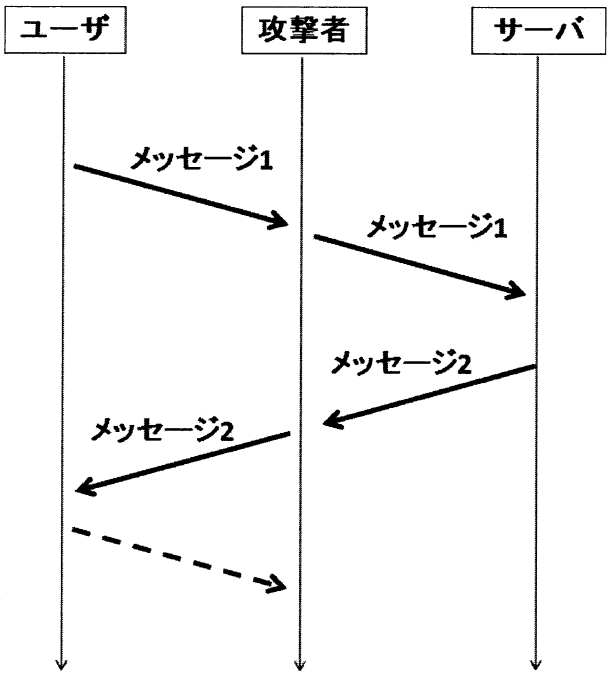

図 2.2: 攻撃者の再送信モデル

- 改ざんは攻撃者が取得したメッセージの一部の情報を他の情報と置き換えてユーザやサーバに送信すること.
- なりすましは攻撃者がユーザにはサーバのふりをし, サーバにはユーザのふりをしてメッセージの送受信を行うこと.

2.2.2 情報項の合成・解析規則

Khan らは情報項の合成・解析規則を定義し，メッセージに複合項を扱えるようにした．表 2.1 には解析・合成規則が示されており，各規則の上部分が規則適用前，下部分が規則適用後をそれぞれ表している．以下で各項目を説明する．

表 2.1: Khan らの合成・解析規則

	解析規則	合成規則
生成	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_a}$	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_s}$
対操作	$\dfrac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathrm{split_i}\ (i = 1, 2)$	$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathrm{pair}$
暗号操作	$\dfrac{T \vdash \{t\}_k \quad T \vdash \overline{k}}{T \vdash t}\ \mathrm{decrypt}$	$\dfrac{T \vdash t \quad T \vdash k}{T \vdash \{t\}_k}\ \mathrm{encrypt}$

T は情報項の集合を表しており，次のように定義される．

定義 2.2.2 (情報項の集合 T).

$$T ::= m \mid (t_1, t_2) \mid \{t\}_k$$

セキュリティプロトコル上の各メッセージにおいて，(t_1, t_2) は情報 t_1, t_2 の対を表し，$\{t\}_k$ は鍵 k で暗号化された情報 t を表す．また，k は鍵の集合 K の要素であり，次で定義される情報の基本項の集合 T_0 の要素が m である．

定義 2.2.3 (情報の基本項の集合 T_0).

$$T_0 = K \cup N \cup Ag$$

N は nonce[1] の集合を表し，Ag は動作主の集合を表す．動作主 Ag にはユーザやサーバ以外に攻撃者も含まれる．鍵 k について，鍵 k が暗号鍵であれば鍵 $\overline{k}$ を復号鍵とし，鍵 $\overline{k}$ が暗号鍵であれば鍵 k を復号鍵とする．また，鍵 k が共通鍵であれば鍵 $\overline{k}$ を鍵 k と同じ鍵とする．

上記を踏まえて，規則表 2.1 を用いた解析・合成処理の具体例を下記に示す．まず，ユーザが所持している情報を T_{user} で表し，下記の情報を所持しているとすると，次のように表せる．

$$T_{user} = \{\{a\}_{k_{server}}, \{(b, c)\}_{k_{user}}, \overline{k}_{user}\}$$

そして，このユーザから情報 c を取り出すには，図 2.3 のように規則を上から下へと適用させる．以降，本稿では生成の解析・合成規則の記載を省略する．

[1] number used once の略.

$$\dfrac{\dfrac{T \vdash \{(b,c)\}_{k_{user}} \qquad T \vdash \bar{k}_{user}}{T \vdash (b,c)}\ \text{decrypt}}{T \vdash c}\ \text{split}_2$$

図 2.3: Khan らの規則適用の具体例

解析・合成規則の適用により取り出した情報は，次々と所持情報に加えられる．また，情報 a のように，$\{a\}_{k_{server}}$ に対して適用できる解析規則が無い場合は情報を取り出して所持情報に加えられない．

Khan らは SPIN でこれらの定義を用い，Needham と Schroeder が考案したセキュリティプロトコル [NS78] の脆弱性の発見に成功した．

第3章　既存研究の問題点

3.1　情報漏洩の不適切な定義

Khan らのセキュリティプロトコルの検証では，情報漏洩の定義が不適切なために情報漏洩の誤検出や検出漏れが生じる．

3.1.1　情報漏洩の誤検出

図 3.1 は，Khan らの定義で情報漏洩の誤検出となる具体例である．

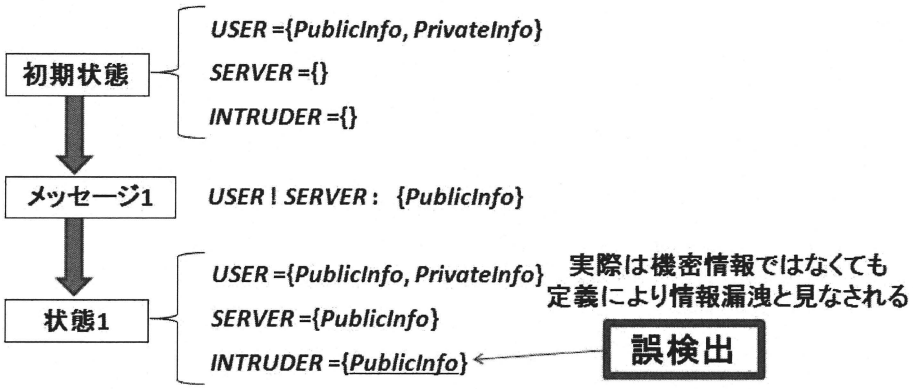

図 3.1: 情報漏洩の誤検出の具体例

図 3.1 の各項目を説明する．$PublicInfo$ は第三者に知られても良い情報（以下，公開情報と呼ぶ）を表しており，具体的な $PublicInfo$ の候補には公開鍵暗号方式の公開鍵等がある．$PrivateInfo$ は第三者に知られてはいけない機密情報を表しており，具体的な $PrivateInfo$ の候補には公開鍵暗号方式の秘密鍵等がある．$USER = \{\ \}$ は，その状態でユーザが所持している情報の内容を示しており，同様に $SERVER = \{\ \}$ はサーバが，$INTRUDER = \{\ \}$ は攻撃者がその状態で所持している情報を表している．$USER\ !\ SERVER : \{PublicInfo\}$ は，!の左の動作主から右の動作主に $\{\ \}$ 内の情報をメッセージとして送信する動作を表している．

上記を踏まえた上で図 3.1 を説明する．まず，初期状態でユーザは $PublicInfo$ と $PrivateInfo$ を所持しており，サーバと攻撃者は何も情報を所持していないと仮定する．次に，ユーザからサーバへ向けて $PublicInfo$ の情報をメッセージ 1 で送信する．そして，状態 1 でサーバはメッセージ 1 を受信して $PublicInfo$ を所持情報に加えるが，同時に攻撃者も盗聴により $PublicInfo$ を所持情報に加える．すると，状態遷移が僅か 1 回しかないが，図 3.1 中の状態 1 を定義 2.2.1 のある状態に，$PublicInfo$ を情報 A に置き換えると，

1. 状態 1 で攻撃者が $PublicInfo$ を所持している．

2. 項目 1 より前の状態で攻撃者は $PublicInfo$ を所持しておらず，攻撃者以外の動作主が $PublicInfo$ を所持している．

となり，定義を満たして情報漏洩と見なされてしまう．なお，メッセージ 1 で $PrivateInfo$ の情報を送信する場合は，実際に情報漏洩が起きているため，検証結果は正しいと言える．

3.1.2 情報漏洩の検出漏れ

図 3.2 は，Khan らの定義で情報漏洩の検出漏れとなる具体例である．

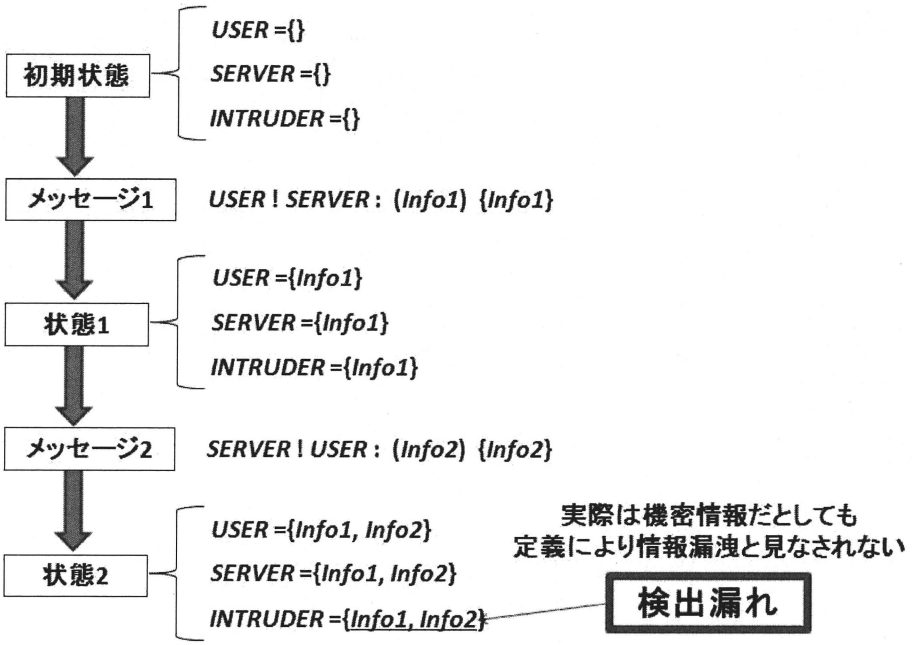

図 3.2: 情報漏洩の検出漏れの具体例

図 3.1 では現れなかった図 3.2 の各項目を説明する．$Info1$ は $Info2$ と対で第三者に知られてはいけない情報であり，$Info2$ も同様である．具体的な $Info1$ と $Info2$ の候補には，TLS プロトコルのマスター鍵の作成に必要なクライアントランダムとサーバランダム等がある．$USER$! $SERVER$: $(Info1)$ $\{Info1\}$ の ( ) は，メッセージの送信の際に生成する情報を表しており，$USER$! $SERVER$: $(Info1)$ $\{Info1\}$ は，ユーザが $Info1$ を作成し，メッセージとしてサーバへ送信する動作を表している．

上記を踏まえた上で図 3.2 を説明する．まず，初期状態では全動作主が情報を所持していないと仮定する．次に，ユーザが $Info1$ を作成し，メッセージ 1 としてサーバへ送信する．そして，状態 1 でサーバはメッセージ 1 を受信して $Info1$ を所持情報に加え，同時に攻撃者も盗聴により $Info1$ を所持情報に加える．同様に，メッセージ 2 もサーバが $Info2$ を作成してユーザへ送信し，状態 2 でユーザと攻撃者の所持情報に加えられると，結果として全ての動作主の所持情報に $Info1$ と $Info2$ の両方が存在してしまう．実際には情報漏洩であるが，定義 2.2.1 を満たさないため，Khan らのセキュリティプロトコルの検証結果では情報漏洩が起きないと判断される．

3.2 扱えない情報項の存在

Khan らのセキュリティプロトコルの検証では，3.1 節で扱った $(Info1,\ Info2)$ のような合成された情報を鍵として扱えない．これは Khan らの解析・合成規則の定義における decrypt と encrypt の規則において，暗号化と復号化の鍵の情報項が鍵 k だけであり，それ以外の情報項が扱えないからである．

また，hash 関数から出力された hash 値と呼ばれる値も情報項として扱えない．これは 2 つの情報項 $a,\ b$ を入力として hash 関数から出力された hash 値を情報項 h $(a,\ b)$ とした際，$a,\ b$ のどちらかと h $(a,\ b)$ の情報項を所持していたとしても，もう一方の情報項を生成できないという特別な値である．

hash 関数の具体例として，商の余りを出力する mod 関数を用いる．2 つの数値 25，10 を mod 関数への入力とすると，

$$\mathrm{mod}(25,\ 10)\ =\ 25\ \%\ 10\ =\ 5$$

となり，5 が出力される．ここで，mod 関数に入力した情報項の 1 つの 10 と，出力結果の 5 を所持しているとする．しかし，この 2 つの所持情報では，もう一方の入力した情報項の候補が 5, 15，25，35，45，…と無限にあり，もう一方の入力した情報項の 25 を推測できない．以降，このような関数を本稿では hash 関数と呼ぶ．

hash 関数は多くのセキュリティプロトコルで用いられており，セキュリティプロトコルの安全性を検証する上で欠かせない関数である．しかし，Khan らの定義した解析・合成規則では hash 関数を扱えないため，大半のセキュリティプロトコルの安全性を検証できない問題がある．

第4章　提案手法

4.1　情報漏洩の定義の変更

本研究では，第3章で述べたKhanらの不適切な情報漏洩の定義の問題点が，機密情報の指定方法にあると考えた．Khanらは攻撃者以外の動作主（以降，正直者と呼ぶ）が所持している全ての情報を機密情報と見なしている．そのため，機密情報は攻撃者よりも先に正直者に所持される必要があり，公開情報は全動作主に同時に所持される必要がある．しかし，正直者による情報の所持や不所持に関係無く，セキュリティプロトコル上の機密情報の多くは予め指定されており，Khanらの情報漏洩の定義を満たすとは限らない．そこで，本研究では情報漏洩を次のように再定義する．

定義 4.1.1 (本研究の情報漏洩の定義). 指定した機密情報を攻撃者が所持しているとき，情報漏洩が起きたと見なす．

セキュリティプロトコル上で指定される機密情報は，役割に応じて2種類ある．1つ目は暗号化用セキュリティプロトコル上で作られる正直者の共通鍵，または秘密鍵と，その作成に必要な情報の一部である．2つ目は相互認証用セキュリティプロトコル上で作られる正直者同士のnonceの組と，その作成に必要な情報の一部である．鍵やnonceの組のそれぞれの作成に必要な情報の一部とは，主に正直者が持つパスワードやパスフレーズなどの固有の機密情報を指す．これらの情報は，攻撃者に所持されると鍵やnonceの組を作成されてしまうため，セキュリティプロトコル上でも機密情報として扱われる．しかし，本研究では解析・合成規則により鍵やnonceの組が作成できるか判定できる．したがって，指定する機密情報をセキュリティプロトコルの用途に合わせて，次のように定義する．

定義 4.1.2 (本研究の機密情報の定義). 以下の情報を機密情報とする．

- 暗号化用セキュリティプロトコルでは，正直者の共通鍵と秘密鍵．
- 認証用セキュリティプロトコルでは，正直者同士のnonceの組．

定義4.1.1と定義4.1.2により，Khanらの情報漏洩の定義で問題となった誤検出や検出漏れを防げる．

4.2　複雑な情報の操作規則導入

本研究では合成処理を施された情報を鍵として扱えるように，情報項の集合Tを次のように再定義する．

定義 4.2.1 (情報項の集合T).

$$T ::= m \mid (t_1,\ t_2) \mid \{t_1\}_{t_2}$$

定義 4.2.1 は定義 2.2.2 と扱える暗号鍵の種類で異なる．また，表 4.1 に示す解析・合成規則の暗号操作において，複雑な鍵情報も解析・合成処理できるように規則を再定義する．これらの再定義により，図 4.1 に示す複雑な鍵情報を用いたセキュリティプロトコルも復号化で情報を取り出せるようになる．

表 4.1: 鍵情報を操作可能な合成・解析規則

	解析規則	合成規則
生成	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_a}$	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_s}$
対操作	$\dfrac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathrm{split_i}\ (i = 1, 2)$	$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathrm{pair}$
暗号操作	$\dfrac{T \vdash \{t_1\}_{t_2} \quad T \vdash \overline{t_2}}{T \vdash t_1}\ \mathrm{decrypt}$	$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash \{t_1\}_{t_2}}\ \mathrm{encrypt}$

$$\dfrac{\dfrac{\dfrac{T \vdash a \quad T \vdash c}{}\ \mathrm{encrypt}}{T \vdash \{(b,c)\}_{\{a\}_c} \quad T \vdash \{a\}_c}\ \mathrm{decrypt}}{T \vdash (b,c)}\ \mathrm{split_1}$$

図 4.1: 鍵情報を操作可能な合成・解析規則の具体例

図 4.1 の具体例では，情報 a と c を encrypt 規則で合成した後，情報 $\{a\}_c$ と情報 $\{(b,c)\}_{\{a\}_c}$ を decrypt 規則で解析して情報 (b,c) を所持情報に加え，結果として情報 b, c を所持情報に加える．この処理が検証において必要なセキュリティプロトコルは Khan らの定義で扱えず，情報 b は攻撃者に所持されないと見なされる．しかし，本研究の提案した規則表 4.1 により，複雑な鍵情報 $\{a\}_c$ も扱えるようになり，情報 b が機密情報の場合でも検証漏れを防止できる．

さらに，本研究では hash 値を情報項として扱えるよう，情報項の集合 T と解析・合成規則を再定義した（定義 4.2.2 と表 4.2）．定義 4.2.2 は定義 4.2.1 に情報項 $\mathrm{h}(\mathrm{t}_1,\ \mathrm{t}_2)$ が加わる．表 4.2 中の hash 操作の解析規則が空白なのは，hash 操作で合成された情報が解析できないためである．

定義 4.2.2 (情報項の集合 T).

$$T\ ::=\ m \mid (t_1,\ t_2) \mid \{t_1\}_{t_2} \mid \mathrm{h}(\mathrm{t}_1,\ \mathrm{t}_2)$$

図 4.2: 再定義した合成・解析規則

	解析規則	合成規則
生成	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_a}$	$\dfrac{}{T \cup \{t\} \vdash t}\ \mathrm{Ax_s}$
対操作	$\dfrac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathrm{split_i}\ (i = 1, 2)$	$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathrm{pair}$
暗号操作	$\dfrac{T \vdash \{t_1\}_{t_2} \quad T \vdash \bar{t}_2}{T \vdash t_1}\ \mathrm{decrypt}$	$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash \{t_1\}_{t_2}}\ \mathrm{encrypt}$
HASH操作		$\dfrac{T \vdash t_1 \quad T \vdash t_2}{T \vdash \mathrm{h}(t_1, t_2)}\ \mathrm{hash}$

4.3 組み合わせによる状態数爆発問題

本章では，これまでに Khan らの情報漏洩の定義による情報漏洩の誤検出や検出漏れを，情報漏洩の再定義により防いだ．また，複雑な鍵情報や hash 関数を含んだセキュリティプロトコルも解析・合成規則の再定義により扱えるようにした．しかし，これらの再定義により，検証の際に情報の組み合わせの数が飛躍的に増加し，状態数爆発問題が生じる．

図 4.3 は攻撃者が情報 t_1 と t_2 を所持している場合の合成規則の適用例である．再定義で合成規則には 3 つの操作があるため，合成規則を 1 回適用すると所持情報の候補は 3 通りできる（図 4.3 中の所持情報 1〜3）．合成規則をさらに適用すると，規則の種類の選択で所持情報はさらに 3 通りずつ増え，候補が 9 通りできる（図 4.3 では所持情報 1-1，所持情報 1-2，所持情報 1-3 等）．しかし，規則の適用は情報の組み合わせ方も重要であるため，1 回目の適用では情報の組み合わせ方で 2 通りずつあり，候補が全 6 通りでき，2 回目の適用では情報の組み合わせ方で 6 通りずつあるため，結果として候補が全 108 通りできる．

このように，僅かな所持情報でも合成規則の適用で所持情報の候補は増加する．また，所持情報によっては解析規則の適用により候補がさらに増加する．そのため，増加した所持情報の候補と機密情報との比較作業が処理時間を大幅に増やし，現実時間で安全性を検証できない．実際にどの程度増加するかは，第 5 章を参照されたい．

4.4 逆向き合成の導入

4.3 節の問題に対して，本研究では解析・合成処理の機密情報の構成要素が含まれていない情報を，情報漏洩に無関係な情報として除き，組み合わせの数を減らして解決を図る．しかし，その情報の中には機密情報の構成要素の作成に関わる情報も存在し，全ての情報の依存関係を把握してから解析・合成処理を行わなければならない問題が生じる．

図 4.4 は所持情報 $T = \{\{a\}_b, \{b\}_c, c\}$ から規則適用により機密情報 a を所持情報に加える

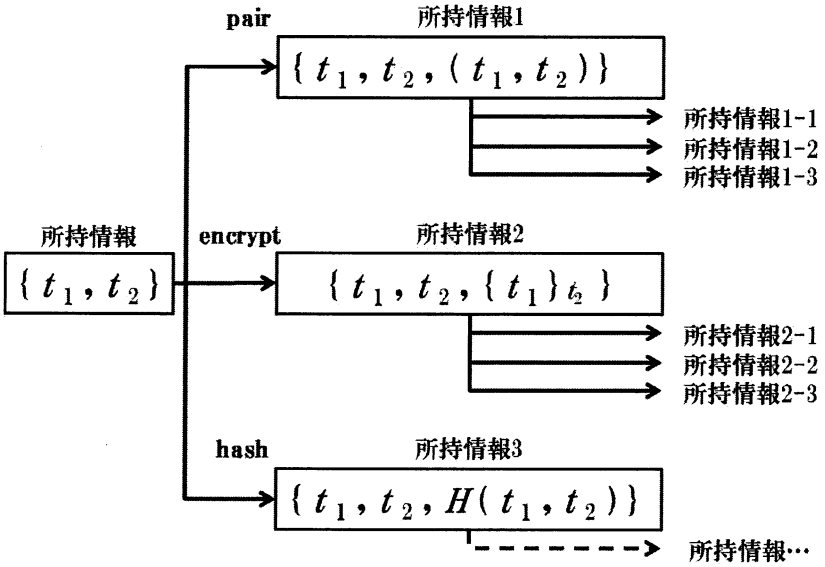

図 4.3: 規則適用による状態数爆発問題

$$\dfrac{\dfrac{T \vdash \{b\}_c \quad T \vdash c}{\cdots}\ \text{decrypt} \quad \cdots}{\dfrac{T \vdash \{a\}_b \quad T \vdash b}{T \vdash a}\ \text{decrypt}}$$

図 4.4: 規則適用の具体例

具体例である．ここで，図 4.4 の所持情報から組み合わせの数を減らすため，機密情報の構成要素が含まれていない情報 $\{b\}_c, c$ を除く．すると，所持情報は $T = \{\{a\}_b\}$ のみとなり，組み合わせの数は減ったが，機密情報を所持できないと見なされる問題が生じる．この問題を解決するためには，情報間の依存関係を把握する必要があるが，依存関係の把握も処理時間を増加させる要因となる．

そのため，本研究では情報漏洩に無関係な情報を除く手法ではなく，逆向き合成処理の手法を新たに提案する．本手法は，機密情報の構成要素である情報の全ての組み合わせと，繰り返し解析した攻撃者の所持情報を比較する手法である．構成要素である情報の全ての組み合わせは，機密情報に対して合成規則を逆向き[1]に適用させて生成する．合成規則を逆向きに適用させて生成した情報が合成情報であれば，再び逆向きに合成規則を適用させ，構成要素である情報の組み合わせ候補を追加していく．また，比較に用いる攻撃者の所持情報も，解析規則を適用できなくなるまで繰り返し解析する．

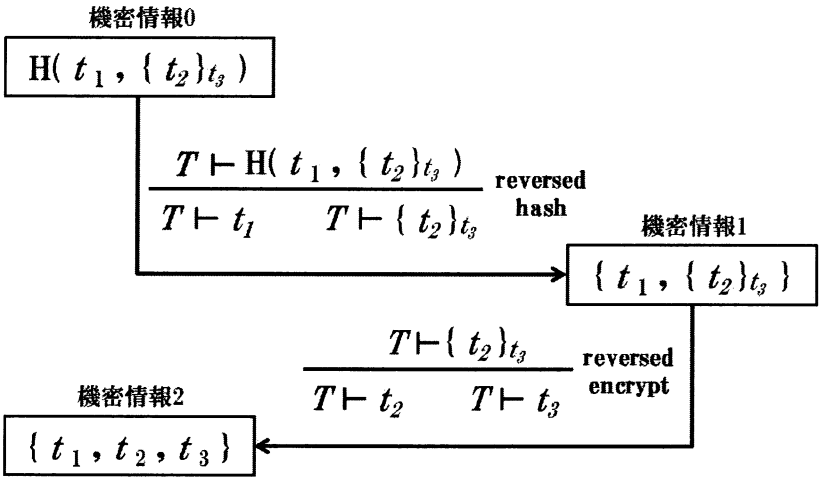

図 4.5: 逆向き合成規則適用の具体例

図 4.5 は逆向き合成規則適用の具体例である．機密情報を図 4.5 中の機密情報 0 である h $(t_1, \{t_2\}_{t_3})$ とする．すると，機密情報 0 は合成規則の hash 規則の逆向き（図 4.5 中の reversed hash 規則）が適用できるため，t_1 と $\{t_2\}_{t_3}$ の組み合わせである機密情報 1 が生成される．そして，同様に encrypt 規則の逆向き（図 4.5 中の reversed encrypt 規則）の適用により機密情報 1 から t_1 と t_2，t_3 の組み合わせである機密情報 2 が生成される．逆向き合成の規則は機密情報に対して適用できなくなるまで繰り返される．適用できなくなると規則の適用により生成された機密情報の候補（図 4.5 中の機密情報 0〜2）と，解析処理を繰り返し適用させた攻撃者の所持情報で比較する．

この手法では規則を 2 回適用させても機密情報の候補数は 3 通りしかなく，攻撃者の所持情報の候補は 1 通りしかないため，僅か 3 通りの比較で済む．そのため，通常の解析・合成規則を用いた手法よりも大規模なセキュリティプロトコルを現実時間で安全性を検証できる．

[1]規則の下部分を適用前，上部分を適用後とすること．

第5章　実装と考察

5.1　提案手法の実装

本研究の提案手法の実装に関して，図 5.1 のように情報項のデータ構造は 1 次元配列（図 5.1 中の Elem）を用いて実装しており，合成処理が施された複雑な情報は適用された操作をタグ（図中の CRYPT や HASH に該当する）で識別している．また，攻撃者は情報項を複数所持できるため，攻撃者の所持情報には 2 次元配列（図 5.1 中の GET）を用いて実装している．そのため，情報項の合成回数や攻撃者の所持情報には上限がある．

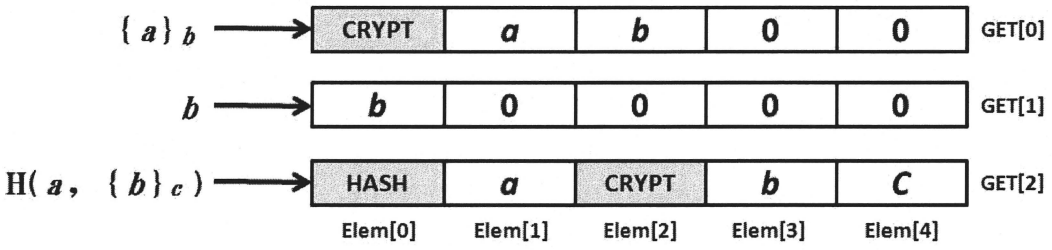

図 5.1: 情報項のデータ構造

5.2　検証結果の考察

本研究の逆向き合成処理により，メモリ使用量や処理時間をどの程度抑えられたか把握するため，逆向き合成処理の手法を用いない通常の判定処理の場合と逆向き合成の判定処理の場合において，所持情報から機密情報を作成できるか判定するプログラムを作成し，SPIN で実行してメモリ使用量や処理時間を測定した（付録 A 参照）．

図 5.2 は通常の判定処理の場合（normal）と逆向き合成の判定処理の場合（reversed）で，メモリ使用量（eq memory）と処理時間（elapsed time）を比較したグラフである．判定結果が偽の場合のメモリ使用量や処理時間のデータは，検証で生成される状態空間中の拒否状態の位置次第で大きく変わる上に，状態空間を全探索する判定結果が真のデータよりも値が小さくなるため，図 5.2 は判定結果が真のデータを比較している．各グラフのタイトルにある E03 や E04 は，5.1 節で述べた情報項のデータ構造における Elem の上限数に該当する．つまり，E03 であれば情報項は 1 回まで合成規則を適用でき，E05 であれば 2 回まで適用できる．また，同様に各グラフの横軸にある G06 や G12 は，5.1 節で述べた攻撃者の所持情報の上限数に該当する．

図 5.2 の各グラフを見ると，通常の判定処理は G08 で処理時間に指数関数的な増加が見られる．一方，逆向き合成の判定処理は G06 から G15 までほとんど増加が見られない．付録

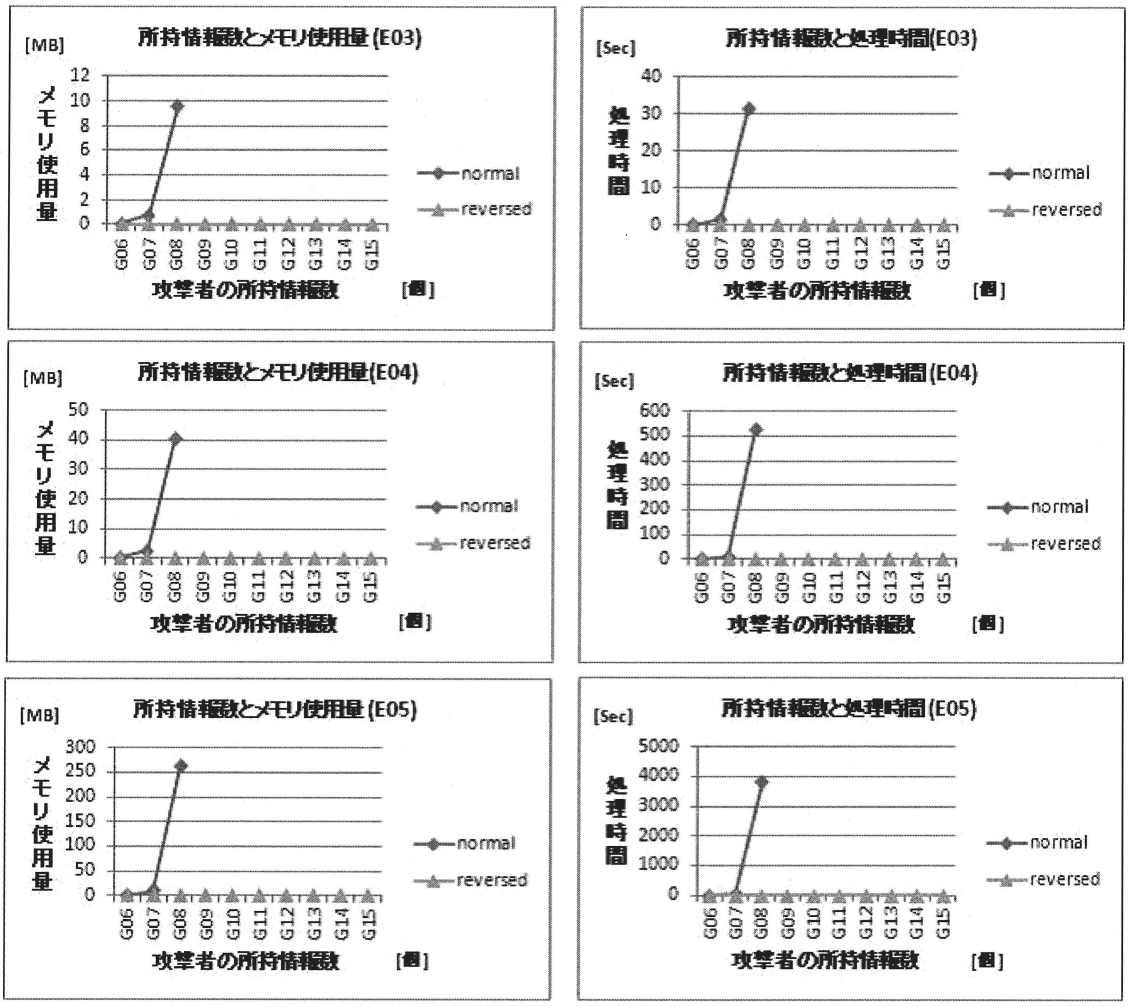

図 5.2: 通常と逆向き合成の判定処理の比較結果

A の通常の判定処理（検証結果：真）を見ると，E05 の項目において，攻撃者の所持情報が 1 つずつ増加すると，処理時間は 50 倍近く増加している．本稿の記載で G09 以降が無いのは，1 回の判定処理に 3810 秒の 50 倍程の時間がかかるためである．一方，逆向き合成の判定処理（検証結果：真）を見ると、処理時間の増加は僅かである．

このように，本研究の提案した逆向き合成処理の適用手法は，機密情報を攻撃者の所持情報から作れるか判定する際に有効であり，通常の手法よりも大規模なセキュリティプロトコルを安全性検証できる．

尚，今回判定に用いたプログラムでは，メモリ使用量や処理時間の指数関数的な増加により通常の判定処理で検証が終わらない状況を避けるため，解析・合成規則の暗号操作と hash 操作を省いている（付録 B 参照）．

第6章　結論と今後の課題

6.1　結論

本研究ではセキュリティプロトコルの安全性自動検証において，専門家が不要で自動化が可能なモデル検査法に着目した．そして，既存研究の Khan らの手法における情報漏洩の誤検出や検出漏れを指摘し，機密情報を定義した上で情報漏洩の有無を再定義した．また，Khan らの手法では扱えない複雑な情報の存在を指摘して，複雑な鍵情報や hash 関数を操作できる解析・合成規則を再定義し，Khan らの手法よりも多くの種類のセキュリティプロトコルを検証可能にした．しかし，情報漏洩の定義や複雑な情報操作の再定義により，情報を合成する際に状態数爆発問題が生じ，計算時間量・メモリ使用量が増加した．そのため，本研究では新たに逆向き合成処理の適用手法を提案して実装を行い，検証により状態数の増加の抑制ができることを示した．

6.2　今後の課題

本手法では，既存手法に対して複雑な情報の操作規則を導入したため，複雑な情報を扱うセキュリティプロトコルも検証可能となった．しかし，既存手法で扱える規模よりも小さなセキュリティプロトコルでなければ，状態数爆発問題が生じて現実時間で自動検証できないという課題がある．さらに，これまで本研究ではセキュリティプロトコルの安全性自動検証について，Khan らの手法と検証時間や検証可能プロトコルの点で比較したが，モデル検査法を用いた他の検査器 [RS00] との比較や他の形式手法との比較を今後行う必要がある．

謝辞

日頃から熱心なご指導頂いた山田俊行講師，講義で大変お世話になった大山口通夫教授に感謝の意を表します．研究室の備品利用等でお世話になった落合美子事務員，研究や発表の場で多くの刺激と示唆を与えてくださった松永朋樹君と木下将典君，計算機ソフトウェア研究室の皆さんに感謝致します．

参考文献

[IPA04] 独立行政法人 情報処理推進機構, 「セキュアプロトコルに対する攻撃法等に関する技術調査」, 第 5 章, 2004 年 3 月.

[KMS05] A.S.Khan, M.Mukund and S.P.Suresh. "Generic Verification of Security Protocols". Proceedings of the 12th International SPIN Workshop, 2005.

[DY83] Danny Dolev and Andrew Yao. "On the Security of public-key protocols". IEEE Transactions on Information Theory, 29:198-208, 1983.

[NS78] R.M.Needham and M.D.Schroeder. "Using Encryption for Authentication in Large Networks of Computers". Communications of the ACM, 21(12):993-999, 1978.

[RS00] Peter Ryan and Steve Schneider. "Modelling and Analysis of Security Protocols". Addison-Wesley Professional, 2000.

付録A　逆向き合成処理の比較評価

通常の判定処理 （検証結果:真）

Name	eq memory	elapsed time
G06E03	0.089	0.093
G07E03	0.784	1.62
G08E03	9.564	31.6
G09E03		
G10E03		
G11E03		
G12E03		
G13E03		
G14E03		
G15E03		
G06E04	0.166	0.234
G07E04	2.466	10.5
G08E04	40.219	524
G09E04		
G10E04		
G11E04		
G12E04		
G13E04		
G14E04		
G15E04		
G06E05	0.504	0.795
G07E05	10.464	49.5
G08E05	262.452	3810
G09E05		
G10E05		
G11E05		
G12E05		
G13E05		
G14E05		
G15E05		
	[Sec]	[MB]

通常の判定処理 （検証結果:偽）

Name	eq memory	elapsed time
G06E03	0	0.015
G07E03	0	0.015
G08E03	0	0.015
G09E03	0	0.015
G10E03	0	0.015
G11E03	0	0.015
G12E03	0	0.015
G13E03	0	0.015
G14E03	0	0.015
G15E03	0	0.015
G06E04	0	0.015
G07E04	0	0.015
G08E04	0	0.015
G09E04	0	0.015
G10E04	0	0.015
G11E04	0	0.015
G12E04	0	0.015
G13E04	0	0.015
G14E04	0	0.015
G15E04	0	0.015
G06E05	0	0.015
G07E05	0	0.015
G08E05	0	0.015
G09E05	0	0.015
G10E05	0	0.015
G11E05	0	0.015
G12E05	0	0.015
G13E05	0	0.015
G14E05	0	0.015
G15E05	0	0.015
	[Sec]	[MB]

図 6.1: 通常の合成処理の評価データ

逆向き合成の判定処理 （検証結果：真）

Name	eq memory	elapsed time
G06E03	0.002	0.015
G07E03	0.002	0.015
G08E03	0.003	0.015
G09E03	0.003	0.015
G10E03	0.003	0.015
G11E03	0.003	0.015
G12E03	0.003	0.015
G13E03	0.004	0.015
G14E03	0.004	0.015
G15E03	0.004	0.015
G06E04	0.003	0.015
G07E04	0.003	0.015
G08E04	0.003	0.015
G09E04	0.003	0.015
G10E04	0.004	0.015
G11E04	0.004	0.015
G12E04	0.004	0.015
G13E04	0.004	0.015
G14E04	0.005	0.015
G15E04	0.005	0.015
G06E05	0.003	0.015
G07E05	0.003	0.015
G08E05	0.004	0.015
G09E05	0.004	0.015
G10E05	0.004	0.015
G11E05	0.005	0.015
G12E05	0.005	0.015
G13E05	0.005	0.015
G14E05	0.006	0.015
G15E05	0.006	0.015
	[Sec]	[MB]

逆向き合成の判定処理 （検証結果：偽）

Name	eq memory	elapsed time
G06E03	0	0.015
G07E03	0	0.015
G08E03	0	0.015
G09E03	0	0.015
G10E03	0	0.015
G11E03	0	0.015
G12E03	0	0.015
G13E03	0	0.015
G14E03	0	0.015
G15E03	0	0.015
G06E04	0	0.015
G07E04	0	0.015
G08E04	0	0.015
G09E04	0	0.015
G10E04	0	0.015
G11E04	0	0.015
G12E04	0	0.015
G13E04	0	0.015
G14E04	0	0.015
G15E04	0	0.015
G06E05	0	0.015
G07E05	0	0.015
G08E05	0	0.015
G09E05	0	0.015
G10E05	0	0.015
G11E05	0	0.015
G12E05	0	0.015
G13E05	0	0.015
G14E05	0	0.015
G15E05	0	0.015
	[Sec]	[MB]

図 6.2: 逆向き合成処理の評価データ

付録B　逆向き合成規則適用の判定プログラム

```
#define GET_MAX 6
#define ELEMENT_MAX 3
#define COMB_CRYPT 1
#define COMB_HASH 2
#define COMB_PAIR 3
#define ID_A 4
#define ID_B 5
#define ID_I 6
#define PUB_A 7
#define PUB_B 8
#define PUB_I 9
#define PRI_A 10
#define PRI_B 11
#define PRI_I 12
#define NONCE_X 13
#define NONCE_Y 14

byte sizeGet, sizeGetBefore, sizeSec, sizeSecBefore; /* ユーザの所持情報の数 */
typedef VECTOR{
byte element[ELEMENT_MAX] = 0;
}
VECTOR getInfo[GET_MAX]; /* 所持情報 */
VECTOR secInfo[GET_MAX]; /* 機密情報 */
VECTOR tempAnalz1, tempAnalz2; /* 解析により得られた情報 */
bool flawA = false; /* 機密情報漏洩の判定用フラグ */
bool flawAnalz1, flawAnalz2; /* 所持情報との比較結果 */
bool tempFlaw;
byte numAnalz; /* 解析する所持情報の要素番号 */
byte countInfo, countElement, countResult;
byte cgi_countInfo, cgi_countElement;

/* ********************************** */
/* 指定範囲内でランダムに番号を選択   */
/* 入力：選択した番号                 */
/* 入力：選択する番号の上限           */
/* ********************************** */
inline chooseNumFromInfo(cnfi_num, cnfi_size){
cnfi_num = 0;
do
:: (cnfi_num < cnfi_size-1) -> cnfi_num++;
:: (1) -> break;
od;
}

/* ********************************** */
/* 特定の情報が所持情報内に無いか検査 */
/* 入力１：検査したい特定の情報       */
/* 入力２：検査結果を格納する bool 変数 */
/* ********************************** */
inline checkGetInfo(cgi_checkInfo, cgi_flaw){
cgi_flaw = false;
cgi_countInfo = 0;
do
:: ((cgi_countInfo < sizeGet) && !cgi_flaw) ->
cgi_countElement = 0;
do
:: (cgi_countElement < ELEMENT_MAX) ->
if
:: (cgi_checkInfo.element[cgi_countElement]
 == getInfo[cgi_countInfo].element[cgi_countElement]) ->
```

```
61 cgi_countElement++;
62 :: else -> break;
63 fi;
64 :: else ->
65 cgi_flaw = true;
66 break;
67 od;
68 cgi_countInfo++;
69 :: else -> break;
70 od;
71 }
72
73 /* ********************************** */
74 /* 合成情報を別々の情報に分解して格納 */
75 /* 入力１：分解する情報               */
76 /* ********************************** */
77 inline separateGetPAIR(rsi_numAnalz){
78 countElement = 0;
79 do
80 :: (countElement < ELEMENT_MAX) ->
81 tempAnalz1.element[countElement] = 0;
82 tempAnalz2.element[countElement] = 0;
83 countElement++;
84 :: else -> break;
85 od;
86
87 countElement = 1;
88 countInfo = 1;
89 countResult = 0;
90 do
91 :: (countElement < ELEMENT_MAX && 1 <= countInfo) ->
92 if
93 :: ((getInfo[rsi_numAnalz].element[countElement] == COMB_CRYPT)
94 || (getInfo[rsi_numAnalz].element[countElement] == COMB_HASH)
95 || (getInfo[rsi_numAnalz].element[countElement] == COMB_PAIR)) ->
96 countInfo++;
97 :: else ->
98 countInfo--;
99 fi;
100 tempAnalz1.element[countResult] = getInfo[rsi_numAnalz].element[countElement];
101 countElement++;
102 countResult++;
103 :: else -> break;
104 od;
105 checkGetInfo(tempAnalz1, flawAnalz1);
106
107 countResult = 0;
108 do
109 :: (_countElement < ELEMENT_MAX) ->
110 tempAnalz2.element[countResult] = getInfo[rsi_numAnalz].element[countElement];
111 countElement++;
112 countResult++;
113 :: else -> break;
114 od;
115 checkGetInfo(tempAnalz2, flawAnalz2);
116
117 countElement = 0;
118 if
119 :: (flawAnalz1 && flawAnalz2) ->
120 do
121 :: (countElement < ELEMENT_MAX) ->
122 getInfo[rsi_numAnalz].element[countElement] = getInfo[sizeGet-1].element[countElement];
123 countElement++;
124 :: else ->
125 sizeGet--;
126 break;
127 od;
128 :: (!flawAnalz1 && flawAnalz2) ->
129 do
130 :: (countElement < ELEMENT_MAX) ->
131 getInfo[rsi_numAnalz].element[countElement] = tempAnalz1.element[countElement];
```

```
132 countElement++;
133 :: else -> break;
134 od;
135 :: (flawAnalz1 && !flawAnalz2) ->
136 do
137 :: (countElement < ELEMENT_MAX) ->
138 getInfo[rsi_numAnalz].element[countElement] = tempAnalz2.element[countElement];
139 countElement++;
140 :: else -> break;
141 od;
142 :: else ->
143 do
144 :: (countElement < ELEMENT_MAX) ->
145 getInfo[rsi_numAnalz].element[countElement] = tempAnalz1.element[countElement];
146 getInfo[sizeGet].element[countElement] = tempAnalz2.element[countElement];
147 countElement++;
148 :: else ->
149 sizeGet++;
150 break;
151 od;
152 fi;
153 }
154
155 /* ********************************** */
156 /* 機密情報を別々の情報に分解して格納 */
157 /* 入力1 : 分解する情報               */
158 /* ********************************** */
159 inline separateSec(rsi_numAnalz){
160 countElement = 1;
161 countInfo = 1;
162 countResult = 0;
163 do
164 :: (countElement < ELEMENT_MAX && 1 <= countInfo) ->
165 if
166 :: ((secInfo[rsi_numAnalz].element[countElement] == COMB_CRYPT)
167 || (secInfo[rsi_numAnalz].element[countElement] == COMB_HASH)
168 || (secInfo[rsi_numAnalz].element[countElement] == COMB_PAIR)) ->
169 countInfo++;
170 :: else ->
171 countInfo--;
172 fi;
173 tempAnalz1.element[countResult] = secInfo[rsi_numAnalz].element[countElement];
174 countElement++;
175 countResult++;
176 :: else -> break;
177 od;
178 checkGetInfo(tempAnalz1, flawAnalz1);
179
180 countResult = 0;
181 do
182 :: (countElement < ELEMENT_MAX) ->
183 tempAnalz2.element[countResult] = secInfo[rsi_numAnalz].element[countElement];
184 countElement++;
185 countResult++;
186 :: else -> break;
187 od;
188 checkGetInfo(tempAnalz2, flawAnalz2);
189
190 countElement = 0;
191 if
192 :: (flawAnalz1 && flawAnalz2) ->
193 do
194 :: (countElement < ELEMENT_MAX) ->
195 secInfo[rsi_numAnalz].element[countElement] = secInfo[sizeSec-1].element[countElement];
196 countElement++;
197 :: else ->
198 sizeSec--;
199 break;
200 od;
201 :: (!flawAnalz1 && flawAnalz2) ->
202 do
```

```
203 :: (countElement < ELEMENT_MAX) ->
204 secInfo[rsi_numAnalz].element[countElement] = tempAnalz1.element[countElement];
205 countElement++;
206 :: else -> break;
207 od;
208 :: (flawAnalz1 && !flawAnalz2) ->
209 do
210 :: (countElement < ELEMENT_MAX) ->
211 secInfo[rsi_numAnalz].element[countElement] = tempAnalz2.element[countElement];
212 countElement++;
213 :: else -> break;
214 od;
215 :: else ->
216 do
217 :: (countElement < ELEMENT_MAX) ->
218 secInfo[rsi_numAnalz].element[countElement] = tempAnalz1.element[countElement];
219 secInfo[sizeSec].element[countElement] = tempAnalz2.element[countElement];
220 countElement++;
221 :: else ->
222 sizeSec++;
223 break;
224 od;
225 fi;
226 }
227
228 /* ********************************** */
229 /*            解析処理                */
230 /* ********************************** */
231 inline procAnalz(){
232 do
233 :: (sizeGet < GET_MAX-1) ->
234 sizeGetBefore = sizeGet;
235 chooseNumFromInfo(_numAnalz, sizeGet);
236 if
237 :: (getInfo[numAnalz].element[0] == COMB_PAIR) -> separateGetPAIR(numAnalz);
238 :: else -> break;
239 fi;
240 if
241 :: (sizeGetBefore > sizeGet) -> break;
242 :: else -> skip;
243 fi;
244 :: (1) -> break;
245 od;
246
247 do
248 :: (sizeSec < GET_MAX-1) ->
249 sizeSecBefore = sizeSec;
250 chooseNumFromInfo(numAnalz, sizeSec);
251 if
252 :: ((secInfo[numAnalz].element[0] == COMB_CRYPT)
253 || (secInfo[numAnalz].element[0] == COMB_HASH)
254 || (secInfo[numAnalz].element[0] == COMB_PAIR)) -> separateSec(numAnalz);
255 :: else -> break;
256 fi;
257 if
258 :: (sizeSecBefore > sizeSec) -> break;
259 :: else -> skip;
260 fi;
261 :: (1) -> break;
262 od;
263 }
264
265 /* ********************************** */
266 /*            比較処理                */
267 /* ********************************** */
268 inline procCheck(){
269 flawA = true;
270 countInfo = 0;
271 do
272 :: (countInfo < sizeSec) ->
273 checkGetInfo(secInfo[countInfo], tempFlaw);
```

```
274 flawA = (flawA && tempFlaw);
275 countInfo++;
276 :: else -> break;
277 od;
278 }
279
280 /* ********************************** */
281 /*            開始処理                */
282 /* ********************************** */
283 init{
284 /* 初期情報を入力 */
285 atomic{
286 /* 初期所持情報の入力 */
287 getInfo[0].element[0] = NONCE_X;
288 getInfo[1].element[0] = NONCE_Y;
289 getInfo[2].element[0] = COMB_PAIR;
290 getInfo[2].element[1] = PUB_A;
291 getInfo[2].element[2] = ID_A;
292 sizeGet = 3;
293
294 /* 機密情報の入力 */
295 secInfo[0].element[0] = COMB_PAIR;
296 secInfo[0].element[1] = PUB_A;
297 secInfo[0].element[2] = ID_A;
298 sizeSec = 1;
299
300 procAnalz();
301 procCheck();
302 assert(!flawA);
303 }
304 }
```

付録C　本研究の提案手法の検証プログラム

```
#define ID_MAX 4 /* ID_MAX = the number of agents */
#define ELEMENT_MAX 11 /* ELEMENT_MAX = the times of synth * 2 + 1 */
#define ARRAY_MAX 20
#define STORE_MAX 10 /* STORE_MAX = the number of intruder's information */
#define COMB_PAIR 252
#define COMB_CRYPT 253
#define COMB_HASH 254
#define NULL 255

mtype{msg1, msg2, msg3};
typedef Contents{
byte element[ELEMENT_MAX] = NULL
};
Contents getInfo[ARRAY_MAX]; /* intruder's information */
Contents secInfo[ARRAY_MAX]; /* secret information & reversed composition pattern */
Contents tempAnalz1, tempAnalz2;
typedef channel{
chan C = [0] of { mtype,/* message type */
byte,/* partner's id */
Contents /* message contents */
};
};
channel proc_chan[ID_MAX];
typedef stored_data{
mtype msg_type;
Contents data;
}
stored_data stored_mesg[STORE_MAX];
byte public_key[ID_MAX] = NULL;
byte private_key[ID_MAX] = NULL;
byte getInfoSize = 0, secInfoSize = 1;
byte s_ptr = 0, agent_num = 3;
byte countArray1, countArray2, countElement, countInfo, countResult;
byte tempSize1, tempSize2;
bool tempFlaw1, tempFlaw2, flaw; /* security property */

/* ********************************** */
/*            配列要素の初期化          */
/* ********************************** */
inline setInitArray(sia_array){
countElement = 0;
do
:: (countElement >= ELEMENT_MAX) -> break;
:: else ->
sia_array.element[countElement] = NULL;
countElement++;
od;
}

/* ********************************** */
/*          配列先頭要素の初期化        */
/* ********************************** */
inline setInitSecretInfoHead(){
countInfo = 0;
do
:: (countInfo >= ARRAY_MAX) -> break;
:: else ->
secInfo[countInfo].element[0] = NULL;
countInfo++;
od;
```

```
61 }
62
63 /* ********************************** */
64 /*          攻撃者の初期所持情報          */
65 /* ********************************** */
66 inline addInitIntrudersInfo(){
67 getInfo[0].element[0] = private_key[0];
68
69 countElement = 0;
70 do
71 :: (countElement >= agent_num) -> break;
72 :: else ->
73 getInfo[getInfoSize+countElement].element[0] = public_key[countElement];
74 countElement++;
75 od;
76 getInfoSize = agent_num + 1;
77 }
78
79 /* ********************************** */
80 /*          合成情報の要素数を調査          */
81 /* ********************************** */
82 inline countInfoSize(cis_info, cis_size){
83 cis_size = 0;
84 countInfo = 1;
85 do
86 :: (countInfo <= 0) -> break;
87 :: else ->
88 if
89 :: (cis_info.element[cis_size] == COMB_PAIR
90 || cis_info.element[cis_size] == COMB_CRYPT
91 || cis_info.element[cis_size] == COMB_HASH) ->
92 countInfo++;
93 :: else ->
94 countInfo--;
95 fi;
96 cis_size++;
97 od;
98 }
99
100 /* ********************************** */
101 /*       非決定的に実行プロセスを選択       */
102 /* ********************************** */
103 inline getAValidProcess(gvp_num){
104 gvp_num = 1;
105 do /* choose the process to send the message */
106 :: break;
107 :: (gvp_num < agent_num - 1) -> gvp_num++;
108 od;
109 }
110
111 /* ********************************** */
112 /*    非決定的に所持情報配列番号を選択      */
113 /* ********************************** */
114 inline chooseGetNum(cgn_num){
115 cgn_num = 0;
116 do
117 :: break;
118 :: (cgn_num < getInfoSize - 1) -> cgn_num++;
119 od;
120 }
121
122 /* ********************************** */
123 /*    非決定的に機密情報配列番号を選択      */
124 /* ********************************** */
125 inline chooseSecNum(csn_num){
126 csn_num = 0;
127 do
128 :: break;
129 :: (csn_num < secInfoSize - 1) -> csn_num++;
130 od;
131 }
```

```
132
133 /* ********************************** */
134 /*    所持情報に含まれている情報か確認    */
135 /* ********************************** */
136 inline checkGetInfo(cgi_info, cgi_flaw){
137 cgi_flaw = false;
138 countArray2 = 0;
139 do
140 :: ((countArray2 < getInfoSize) && !cgi_flaw) ->
141 countInfo = 1;
142 countResult = 0;
143 do
144 :: (countInfo == 0) ->
145 cgi_flaw = true;
146 break;
147 :: else ->
148 if
149 :: ((cgi_info.element[countResult] == COMB_PAIR)
150 || (cgi_info.element[countResult] == COMB_CRYPT)
151 || (cgi_info.element[countResult] == COMB_HASH)) -> countInfo++;
152 :: else -> countInfo--;
153 fi;
154 if
155 :: (cgi_info.element[countResult] == getInfo[countArray2].element[countResult]) ->
156 countResult++;
157 :: else -> break;
158 fi;
159 od;
160 countArray2++;
161 :: else -> break;
162 od;
163 }
164
165 /* ********************************** */
166 /*     所持情報内の同構成合成情報と交換    */
167 /* ********************************** */
168 inline changeSameStructure(css_info, css_size){
169 countArray2 = 0;
170 do
171 :: (countArray2 >= getInfoSize) -> break;
172 :: else ->
173 if
174 :: skip;
175 :: (1) ->
176 countElement = 0;
177 do
178 :: (countElement >= css_size) ->
179 countElement = 0;
180 do
181 :: (countElement >= css_size) -> break;
182 :: else ->
183 css_info.element[countElement] = getInfo[countArray2].element[countElement];
184 countElement++;
185 od;
186 countArray2 = getInfoSize;
187 break;
188 :: else ->
189 if
190 :: ((css_info.element[countElement] == COMB_PAIR
191 || css_info.element[countElement] == COMB_CRYPT
192 || css_info.element[countElement] == COMB_HASH)
193 && css_info.element[countElement] != getInfo[countArray2].element[countElement]) ->
194 break;
195 :: ((getInfo[countArray2].element[countElement] == COMB_PAIR
196 || getInfo[countArray2].element[countElement] == COMB_CRYPT
197 || getInfo[countArray2].element[countElement] == COMB_HASH)
198 && css_info.element[countElement] != getInfo[countArray2].element[countElement]) ->
199 break;
200 :: else -> countElement++;
201 fi;
202 od;
```

```
203 fi;
204 countArray2++;
205 od;
206 }
207
208 /* ********************************** */
209 /*     受信メッセージを所持情報に加える      */
210 /* ********************************** */
211 inline copyContents(){
212 addInitIntrudersInfo();
213
214 countArray1 = 0;
215 do
216 :: (countArray1 < s_ptr && (getInfoSize+countArray1) < ARRAY_MAX) ->
217 countElement = 0;
218 do
219 :: (countElement >= ELEMENT_MAX) -> break;
220 :: else ->
221 getInfo[getInfoSize+countArray1].element[countElement]
222  = stored_mesg[countArray1].data.element[countElement];
223 countElement++;
224 od;
225 countArray1++;
226 :: else -> break;
227 od;
228 getInfoSize = getInfoSize + countArray1;
229 }
230
231 /* ********************************** */
232 /*    機密情報を別々の情報に分解して格納     */
233 /* ********************************** */
234 inline separateData(sd_num){
235 countElement = 1;
236 countResult = 0;
237 countInfo = 1;
238
239 do
240 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
241 if
242 :: ((secInfo[sd_num].element[countElement] == COMB_PAIR)
243 || (secInfo[sd_num].element[countElement] == COMB_CRYPT)
244 || (secInfo[sd_num].element[countElement] == COMB_HASH)) ->
245 countInfo++;
246 :: else ->
247 countInfo--;
248 fi;
249 secInfo[sd_num+1].element[countResult] = secInfo[sd_num].element[countElement];
250 countElement++;
251 countResult++;
252 :: else -> break;
253 od;
254
255 countArray2 = countResult;
256 countResult = 0;
257 countInfo = 1;
258 do
259 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
260 if
261 :: ((secInfo[sd_num].element[countElement] == COMB_PAIR)
262 || (secInfo[sd_num].element[countElement] == COMB_CRYPT)
263 || (secInfo[sd_num].element[countElement] == COMB_HASH)) ->
264 countInfo++;
265 :: else ->
266 countInfo--;
267 fi;
268 secInfo[sd_num+1+countArray2].element[countResult] = secInfo[sd_num].element[countElement];
269 countElement++;
270 countResult++;
271 :: else -> break;
272 od;
273 }
```

```
274
275 /* ********************************** */
276 /*           メッセージ生成処理           */
277 /* ********************************** */
278 inline makeMessage(result){
279 if
280 :: (1) ->/* replay mssage */
281 countElement = 0;
282 do
283 :: (countElement >= ELEMENT_MAX) -> break;
284 :: else ->
285 result.element[countElement] = secInfo[0].element[countElement];
286 countElement++;
287 od;
288 break;
289
290 :: (1) ->/* make fasilitation massage */
291 countArray1 = 0;
292 do
293 :: break;
294 :: (countArray1 < ARRAY_MAX) ->
295 if
296 :: skip;
297 :: ((secInfo[countArray1].element[0] == COMB_PAIR)
298 || (secInfo[countArray1].element[0] == COMB_CRYPT)
299 || (secInfo[countArray1].element[0] == COMB_HASH)) ->
300 result.element[countArray1] = secInfo[countArray1].element[0];
301 secInfo[countArray1].element[0] = NULL;
302 separateData(countArray1);
303 countArray1++;
304 fi;
305 od;
306
307 countArray1 = 0;
308 do
309 :: (countArray1 >= ARRAY_MAX) -> break;
310 :: else ->
311 if
312 :: (secInfo[countArray1].element[0] == NULL) -> skip;
313 :: else ->
314 countInfoSize(secInfo[countArray1], countResult);
315 changeSameStructure(secInfo[countArray1], countResult);
316 countElement = 0;
317 do
318 :: (countElement >= countResult) -> break;
319 :: else ->
320 result.element[countArray1+countElement]
321  = secInfo[countArray1].element[countElement];
322 countElement++;
323 od;
324 fi;
325 countArray1++;
326 od;
327
328 fi;
329 }
330
331 /* ********************************** */
332 /*   機密情報を別々の情報に分解して格納    */
333 /* ********************************** */
334 inline separateSec(ss_num){
335 setInitArray(tempAnalz1);
336 setInitArray(tempAnalz2);
337 countElement = 1;
338 countResult = 0;
339 countInfo = 1;
340
341 do
342 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
343 if
344 :: ((secInfo[ss_num].element[countElement] == COMB_PAIR)
```

```
345 || (secInfo[ss_num].element[countElement] == COMB_CRYPT)
346 || (secInfo[ss_num].element[countElement] == COMB_HASH)) ->
347 countInfo++;
348 :: else ->
349 countInfo--;
350 fi;
351 tempAnalz1.element[countResult] = secInfo[ss_num].element[countElement];
352 countElement++;
353 countResult++;
354 :: else -> break;
355 od;
356 checkGetInfo(tempAnalz1, tempFlaw1);
357
358 countResult = 0;
359 countInfo = 1;
360 do
361 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
362 if
363 :: ((secInfo[ss_num].element[countElement] == COMB_PAIR)
364 || (secInfo[ss_num].element[countElement] == COMB_CRYPT)
365 || (secInfo[ss_num].element[countElement] == COMB_HASH)) ->
366 countInfo++;
367 :: else ->
368 countInfo--;
369 fi;
370 tempAnalz2.element[countResult] = secInfo[ss_num].element[countElement];
371 countElement++;
372 countResult++;
373 :: else -> break;
374 od;
375 checkGetInfo(tempAnalz2, tempFlaw2);
376
377 countElement = 0;
378 if
379 :: (tempFlaw1 && tempFlaw2) ->
380 do
381 :: (countElement < ELEMENT_MAX) ->
382 secInfo[ss_num].element[countElement] = secInfo[secInfoSize-1].element[countElement];
383 countElement++;
384 :: else ->
385 secInfoSize--;
386 break;
387 od;
388 :: (!tempFlaw1 && tempFlaw2) ->
389 do
390 :: (countElement < ELEMENT_MAX) ->
391 secInfo[ss_num].element[countElement] = tempAnalz1.element[countElement];
392 countElement++;
393 :: else -> break;
394 od;
395 :: (tempFlaw1 && !tempFlaw2) ->
396 do
397 :: (countElement < ELEMENT_MAX) ->
398 secInfo[ss_num].element[countElement] = tempAnalz2.element[countElement];
399 countElement++;
400 :: else -> break;
401 od;
402 :: else ->
403 do
404 :: (countElement >= ELEMENT_MAX) ->
405 secInfoSize++;
406 break;
407 :: else ->
408 secInfo[ss_num].element[countElement] = tempAnalz1.element[countElement];
409 secInfo[secInfoSize].element[countElement] = tempAnalz2.element[countElement];
410 countElement++;
411 od;
412 fi;
413 }
414
415 /* ********************************** */
```

```
416 /*      合成情報 (PAIR) を分解して格納      */
417 /* ********************************** */
418 inline separateGetPAIR(sgp_num){
419 countElement = 1;
420 countResult = 0;
421 countInfo = 1;
422 do
423 :: (countInfo <= 0) -> break;
424 :: else ->
425 if
426 :: ((getInfo[sgp_num].element[countElement] == COMB_PAIR)
427 || (getInfo[sgp_num].element[countElement] == COMB_CRYPT)
428 || (getInfo[sgp_num].element[countElement] == COMB_HASH)) ->
429 countInfo++;
430 :: else ->
431 countInfo--;
432 fi;
433 tempAnalz1.element[countResult] = getInfo[sgp_num].element[countElement];
434 countElement++;
435 countResult++;
436 od;
437 checkGetInfo(tempAnalz1, tempFlaw1);
438
439 countResult = 0;
440 countInfo = 1;
441 do
442 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
443 if
444 :: ((getInfo[sgp_num].element[countElement] == COMB_PAIR)
445 || (getInfo[sgp_num].element[countElement] == COMB_CRYPT)
446 || (getInfo[sgp_num].element[countElement] == COMB_HASH)) ->
447 countInfo++;
448 :: else ->
449 countInfo--;
450 fi;
451 tempAnalz2.element[countResult] = getInfo[sgp_num].element[countElement];
452 countElement++;
453 countResult++;
454 :: else -> break;
455 od;
456 checkGetInfo(tempAnalz2, tempFlaw2);
457
458 countElement = 0;
459 if
460 :: (tempFlaw1 && tempFlaw2) ->
461 do
462 :: (countElement < ELEMENT_MAX) ->
463 getInfo[sgp_num].element[countElement] = getInfo[getInfoSize-1].element[countElement];
464 countElement++;
465 :: else ->
466 getInfoSize--;
467 break;
468 od;
469 :: (!tempFlaw1 && tempFlaw2) ->
470 do
471 :: (countElement < ELEMENT_MAX) ->
472 getInfo[sgp_num].element[countElement] = tempAnalz1.element[countElement];
473 countElement++;
474 :: else -> break;
475 od;
476 :: (tempFlaw1 && !tempFlaw2) ->
477 do
478 :: (countElement < ELEMENT_MAX) ->
479 getInfo[sgp_num].element[countElement] = tempAnalz2.element[countElement];
480 countElement++;
481 :: else -> break;
482 od;
483 :: else ->
484 do
485 :: (countElement >= ELEMENT_MAX) ->
486 getInfoSize++;
```

```
487 break;
488 :: else ->
489 getInfo[sgp_num].element[countElement] = tempAnalz1.element[countElement];
490 getInfo[getInfoSize].element[countElement] = tempAnalz2.element[countElement];
491 countElement++;
492 od;
493 fi;
494 }
495
496 /* ********************************** */
497 /*      合成情報 (CRYPT) を分解して格納     */
498 /* ********************************** */
499 inline separateGetCRYPT(sgc_num){
500 countElement = 1;
501 countResult = 0;
502 countInfo = 1;
503
504 do
505 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
506 if
507 :: ((getInfo[sgc_num].element[countElement] == COMB_PAIR)
508 || (getInfo[sgc_num].element[countElement] == COMB_CRYPT)
509 || (getInfo[sgc_num].element[countElement] == COMB_HASH)) ->
510 countInfo++;
511 :: else ->
512 countInfo--;
513 fi;
514 tempAnalz1.element[countResult] = getInfo[sgc_num].element[countElement];
515 countElement++;
516 countResult++;
517 :: else -> break;
518 od;
519
520 countResult = 0;
521 do
522 :: (countResult >= agent_num) -> checkGetInfo(tempAnalz1, tempFlaw1);
523 :: else ->
524 if
525 :: (tempAnalz1.element[0] == private_key[countResult]) ->
526 tempFlaw1 = true;
527 break;
528 :: (tempAnalz1.element[0] == public_key[countResult]) ->
529 tempAnalz1.element[0] = private_key[countResult];
530 checkGetInfo(tempAnalz1, tempFlaw1);
531 break;
532 :: else -> countResult++;
533 fi;
534 od;
535
536 countResult = 0;
537 countInfo = 1;
538 do
539 :: (countElement < ELEMENT_MAX && 0 < countInfo) ->
540 if
541 :: ((getInfo[sgc_num].element[countElement] == COMB_PAIR)
542 || (getInfo[sgc_num].element[countElement] == COMB_CRYPT)
543 || (getInfo[sgc_num].element[countElement] == COMB_HASH)) ->
544 countInfo++;
545 :: else ->
546 countInfo--;
547 fi;
548 tempAnalz2.element[countResult] = getInfo[sgc_num].element[countElement];
549 countElement++;
550 countResult++;
551 :: else -> break;
552 od;
553
554 countResult = 0;
555 do
556 :: (countResult >= agent_num) -> checkGetInfo(tempAnalz2, tempFlaw2);
557 :: else ->
```

```
558 if
559 :: (tempAnalz2.element[0] == private_key[countResult]) ->
560 tempFlaw2 = true;
561 break;
562 :: (tempAnalz2.element[0] == public_key[countResult]) ->
563 tempAnalz2.element[0] = private_key[countResult];
564 checkGetInfo(tempAnalz2, tempFlaw2);
565 break;
566 :: else -> countResult++;
567 fi;
568 od;
569
570 countElement = 0;
571 if
572 :: (tempFlaw1 && tempFlaw2) ->
573 do
574 :: (countElement < ELEMENT_MAX) ->
575 getInfo[sgc_num].element[countElement]
576  = getInfo[getInfoSize-1].element[countElement];
577 countElement++;
578 :: else ->
579 getInfoSize--;
580 break;
581 od;
582 :: (!tempFlaw1 && tempFlaw2) ->
583 do
584 :: (countElement < ELEMENT_MAX) ->
585 getInfo[sgc_num].element[countElement] = tempAnalz1.element[countElement];
586 countElement++;
587 :: else -> break;
588 od;
589 :: (tempFlaw1 && !tempFlaw2) ->
590 do
591 :: (countElement < ELEMENT_MAX) ->
592 getInfo[sgc_num].element[countElement] = tempAnalz2.element[countElement];
593 countElement++;
594 :: else -> break;
595 od;
596 :: else -> skip;
597 fi;
598 }
599
600 /* ********************************** */
601 /*            逆向き合成処理            */
602 /* ********************************** */
603 inline ruleReversedComposite(){
604 do
605 :: break;
606 :: (secInfoSize < ARRAY_MAX-1) ->
607 chooseSecNum(countArray1);
608 if
609 :: ((secInfo[countArray1].element[0] == COMB_PAIR)
610 || (secInfo[countArray1].element[0] == COMB_CRYPT)
611 || (secInfo[countArray1].element[0] == COMB_HASH)) -> separateSec(countArray1);
612 :: else -> break;
613 fi;
614 od;
615 }
616
617 /* ********************************** */
618 /*               解析処理               */
619 /* ********************************** */
620 inline ruleDecomposite(){
621 do
622 :: break;
623 :: (getInfoSize < ARRAY_MAX - 1) ->
624 chooseGetNum(countArray1);
625 if
626 :: (getInfo[countArray1].element[0] == COMB_PAIR) -> separateGetPAIR(countArray1);
627 :: (getInfo[countArray1].element[0] == COMB_CRYPT) -> separateGetCRYPT(countArray1);
628 :: else -> break;
```

```
fi;
od;
}

/* ********************************** */
/*              比較処理                */
/* ********************************** */
inline ruleCompare(){
flaw = true;
countArray1 = 0;
do
:: (countArray1 >= secInfoSize) -> break;
:: else ->
checkGetInfo(secInfo[countArray1], tempFlaw1);
flaw = (flaw && tempFlaw1);
countArray1++;
od;
assert(!flaw);
}

/* ********************************** */
/*              プロセス A              */
/* ********************************** */
proctype procA(byte MY_ID){
byte MY_NONCE, P_ID, P_NONCE;
chan my_chan, p_chan;
Contents data;
atomic{
private_key[MY_ID] = ID_MAX + MY_ID;
public_key[MY_ID] = (ID_MAX * 2) + MY_ID;
MY_NONCE = (ID_MAX * 3) + MY_ID;
my_chan = proc_chan[MY_ID].C;
getAValidProcess(P_ID); /* choose the partner */
p_chan = proc_chan[P_ID].C;

data.element[0] = COMB_CRYPT;
data.element[1] = COMB_PAIR;
data.element[2] = MY_NONCE;
data.element[3] = MY_ID;
data.element[4] = public_key[P_ID];
endA1: p_chan!msg1(MY_ID, data);
}

endA2: my_chan?msg2(_, data);

endA3: do /* if unexpected message go to infinite loop */
:: ((data.element[3] == MY_NONCE)
&& (data.element[5] == P_ID)
&& (data.element[6] == public_key[MY_ID])) -> break;
od;

atomic{
P_NONCE = data.element[4];

data.element[0] = COMB_CRYPT;
data.element[1] = P_NONCE;
data.element[2] = public_key[P_ID];
endA4: p_chan!msg3(MY_ID, data);
}
}

/* ********************************** */
/*              プロセスB               */
/* ********************************** */
proctype procB(byte MY_ID){
byte MY_NONCE, P_ID, P_NONCE;
chan my_chan, p_chan;
Contents data;

atomic{
private_key[MY_NONCE] = ID_MAX + MY_ID;
```

```
700 public_key[MY_NONCE] = (ID_MAX * 2) + MY_ID;
701 MY_NONCE = (ID_MAX * 3) + MY_ID;
702
703 my_chan = proc_chan[MY_ID].C;
704 }
705
706 endB1: my_chan?msg1(_, data);
707 endB2: do /* if unexpected message go to infinite loop */
708 :: (data.element[4] == public_key[MY_ID]) -> break;
709 od;
710
711 atomic{
712 P_ID = data.element[3];
713 p_chan = proc_chan[P_ID].C;
714 P_NONCE = data.element[2];
715
716 data.element[0] = COMB_CRYPT;
717 data.element[1] = COMB_PAIR;
718 data.element[2] = COMB_PAIR;
719 data.element[3] = P_NONCE;
720 data.element[4] = MY_NONCE;
721 data.element[5] = MY_ID;
722 data.element[6] = public_key[P_ID];
723 endB3: p_chan!msg2(MY_ID, data);
724 }
725
726 endB4: my_chan?msg3(_, data);
727 endB5: do /* if unexpected message go to infinite loop */
728 :: ((data.element[1] == MY_NONCE)
729 && (data.element[2] == public_key[MY_ID])) -> break;
730 od;
731 }
732
733 /* ********************************** */
734 /*              プロセス I                */
735 /* ********************************** */
736 proctype procI(byte MY_ID){
737 mtype msg;
738 byte P_ID;
739 Contents data;
740
741 atomic{
742 private_key[MY_ID] = ID_MAX;
743 public_key[MY_ID] = ID_MAX * 2;
744 }
745
746 do
747 :: (1) ->/* intercept a message */
748 endI1: if /* choose a channel to intercept (動作主の数分必要) */
749 :: proc_chan[1].C?msg(_, data);
750 :: proc_chan[2].C?msg(_, data);
751 :: proc_chan[3].C?msg(_, data);
752 fi;
753
754 atomic{
755 if
756 :: (s_ptr >= STORE_MAX) -> skip;
757 :: else ->
758 stored_mesg[s_ptr].msg_type = msg;
759 countElement = 0;
760 do
761 :: (countElement >= ELEMENT_MAX) -> break;
762 :: else ->
763 stored_mesg[s_ptr].data.element[countElement] = data.element[countElement];
764 countElement++;
765 od;
766 s_ptr++;
767 fi;
768 }
769
770 :: (1) ->/* message replay or send fasilitation message */
```

```
771 atomic{
772 countArray1 = 0;
773 do
774 :: (countArray1 >= s_ptr) -> break;
775 :: else ->
776 if
777 :: countArray1++;
778 :: (1) ->
779 setInitSecretInfoHead();
780 msg = stored_mesg[countArray1].msg_type;
781 countElement = 0;
782 do
783 :: (countElement >= ELEMENT_MAX) -> break;
784 :: else ->
785 secInfo[0].element[countElement]
786  = stored_mesg[countArray1].data.element[countElement];
787 countElement++;
788 od;
789 copyContents();
790 ruleDecomposite();
791
792 makeMessage(data);
793
794 getAValidProcess(P_ID);
795 endI2: proc_chan[P_ID].C!msg(MY_ID, data);
796 break;
797 fi;
798 od;
799 }
800 od;
801 }
802
803 /* ********************************** */
804 /*             初動プロセス             */
805 /* ********************************** */
806 init{
807 atomic{
808 run procI(0);
809 run procA(1);
810 run procB(2);
811 do /* create processes nondeterministically */
812 :: break;
813 :: (agent_num < ID_MAX) ->
814 if
815 :: run procA(agent_num);
816 :: run procB(agent_num);
817 fi;
818 agent_num++;
819 od;
820 }
821
822 atomic{
823 /* 試験的に 250 */
824 secInfo[0].element[0] = 250;
825 secInfoSize = 1;
826
827 copyContents();
828 ruleDecomposite();
829 ruleReversedComposite();
830 ruleCompare();
831 }
832 }
```