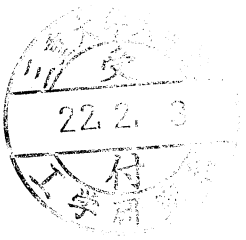
修 士 論 文

# 項の到達解析による
# プロトコル安全性検証



平成21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

牛之濱 智己

指導教員 山田俊行

# 目 次

# 第1章　背景

現在，Web 上の個人認証やメールなど，様々な用途で情報の通信が行われている．しかし，通信情報は第三者から閲覧可能なため，パスワードなどの機密情報を第三者に取得される危険性がある．そこで，安全な通信を提供するために，通信を暗号化している．通信を暗号化してもプロトコル (通信規約) の仕様次第では，第三者によるなりすましや機密情報漏洩といった危険性 [DY81] がある．そこで，暗号が安全であると仮定してプロトコルの安全性を検証する手法が広く研究されている．例えば，通信の開始者，受信者，第三者それぞれの動作をプロセス代数で定式化し，それらの並行動作を調べることで，攻撃が成立する通信手順がないか検証する手法 [Sch02] が挙げられる．この手法は，攻撃が行われる通信手順を発見するには有効な手法であると言われている．ただし，通信の定式化に用いた計算モデルには状態爆発という問題が存在する．これは，安全性検証のための探索空間が膨大になり，現実的な時間で検証できなくなるという問題である．この問題が起こる一因として，定式化する要素が膨大な場合が挙げられる．すなわち，通信者数や通信量が膨大な際に起こる攻撃は，現実的な時間で検証できないと考えられる．よって，通信の安全性を満たすためには，通信者や通信量などに制限が無い通信の並行動作を定式化して安全性を検証する必要がある．そこで，無限の通信の並行動作を検証する方法として，項書き換え系の到達解析を利用した手法 [GK00] が提案されている．

[GK00] の手法では，通信の状態や機密情報などを項，プロトコルの動作を書き換え規則，第三者の取得情報と取得情報から生成可能な情報を木オートマトンで表す．第三者の初期情報を表す木オートマトンの受理言語を項書き換え系で書き換えたとき，書き換えによって到達する項の全体集合は第三者の取得情報を表す．ただし，プロトコルの動作を表す書き換え規則を用いると無限に書き換えが可能なため，到達する項の全体集合，すなわち，第三者の取得情報は無限集合となる．よって，この集合を検証に用いるには不適切である．そこで，項の到達解析の手法を用いて，第三者の取得情報を包含する集合が受理言語となる近似オートマトンを作成する．近似オートマトンの受理言語に機密情報または通信の偽装を表す項が含まれていないか判定することで安全性を検証する．

本論文では，従来の定式化に基づいて検証すると，脆弱性があるのに安全であると判定することと検出漏れが起こることを明らかにする．これらの誤りには，次の 2 つの原因がある．一つは，攻撃が成立する通信を表せないことである．もう一つは，木オートマトンの受理言語に安全性を表す項が適切に含まれないことである．そこで本研究では，第三者が介入する通信を正しく表現し，検出漏れが起こらない定式化を提案する．[GK00] の手法で攻撃が成立する通信を表せないのは，第三者の取得情報を用いた情報作成を表現してないためである．そこで，従来の定式化の不備を改めるために，第三者の取得情報を用いた情報作成を表す書き換え規則を追加する．また，検出漏れ

を防ぐために，第三者が取得した情報を表す項は，全て近似オートマトンの受理状態に到達するように，書き換え規則を追加する．

以下に本論文の構成を示す．第二章では，プロトコルについての概要と定式化に用いる枠組みを解説する．第三章では，提案手法の元となる [GK00] の手法を解説する．第四章では，[GK00] の手法の問題点と，その解決法について記述する．さらに，従来手法では検出漏れが起こるプロトコルを提案手法に従って定式化した場合，攻撃が成立する通信を表現できることを確かめる．

# 第2章　準備

この章では，本論文の検証対象である暗号プロトコルと，暗号プロトコルの定式化に用いる，項書き換え系と木オートマトンについて解説する．

## 2.1　暗号プロトコル

プロトコルとは，通信を行う上で，通信者やサーバに送信するメッセージを，どのような順序で送信するか規定したものである．その中でも暗号技術を用いて，第三者に知られないように送信内容を暗号化しているプロトコルを暗号プロトコルと呼ぶ．暗号プロトコルの仕様は次の形式で記述される．

(番号)．　(送信者) → (受信者) : (送信内容)

この記法で定められた番号順にメッセージを送信し，認証や鍵配送などを行う．暗号プロトコルの例として，例 2.1.1 の暗号プロトコル [DY81] が有名である．

**例 2.1.1 (Needham Schroeder Public Key プロトコル)**

1. $A \rightarrow B \quad : \quad \{N_A, A\}K_B$
2. $B \rightarrow A \quad : \quad \{N_A, N_B\}K_A$
3. $A \rightarrow B \quad : \quad \{N_B\}K_B$

上記の A，B は通信者を表す．送信内容に含まれる A，B はそれぞれ A，B であることを証明する情報である．$N_A$, $N_B$ はノンスと呼ばれる一意な数である．$K_A$, $K_B$ は暗号化に使われた公開鍵を表す．$\{\cdots\}K_A$ は {} 内の内容を公開鍵 $K_A$ で暗号化した情報を表す．

Needham Schroeder Public Key プロトコル (以下 NSPK) は，相互認証を目的とした公開鍵暗号方式の暗号プロトコルである．相互認証とは，現在通信している相手が誰であるか確認することである．NSPK では，通信者が生成したノンスを互いに交換することで，認証する．NSPK で認証を行う場合，まず，A がノンス $N_A$ を生成し，認証をしたい相手 B にメッセージ $\{N_A, A\}K_B$ を送信する．メッセージ $\{N_A, A\}K_B$ を B が受理すると，メッセージの内容に A が含まれていることから，メッセージの送信者が A であると判断する．次に，ノンス $N_B$ を生成し，メッセージ $\{N_A, N_B\}K_A$ を A に返信する．メッセージ $\{N_A, N_B\}K_A$ を A が受理すると，メッセージの内容に自分が生成したノンス $N_A$ が含まれていれば，このメッセージは B からの返信であると判断する．ノンス $N_A$ は作成者 A か，1. のメッセージを受けとった B しか知らない情報であるため，返信に $N_A$ が含まれていれば，そのメッセージを作成できるのは A を除けば B のみである，と判断できるからである．ここまでの通信で，A は B と通信が確立されていると認識をする．最後に，A

がメッセージ $\{N_B\}K_B$ をBに返信することで，同じ理由により，Bは通信相手がAであると認識する．

### 2.1.1 安全性

本論文では，攻撃のうち機密情報の盗聴となりすましを問題視している．盗聴とは，自分以外の通信者に送信された情報を取得する動作である．この攻撃に関連した性質として，機密性と真正性を検証する．

機密性とは，通信者間でのみ知られている情報が第三者に漏れないことを示す性質である．先程の例では，ノンスが第三者に知られなければ，機密性が満たされると言える．真正性とは，第三者が他の通信者になりすましてプロトコルを終了できないことを示す性質である．

### 2.1.2 攻撃者

攻撃者は，機密情報の盗聴と通信者へのなりすましを目的とした第三者である．攻撃者は，他の通信者から通常の通信者と認識されている．さらに，通常の通信者と同じ行動が可能なことに加えて，次の特徴を持つ．

- 任意の通信者と通信を開始可能
- 他の通信者間で送信された任意のメッセージを取得可能
- 攻撃者が所持している情報から，任意にメッセージを作成可能
- 任意の通信に割り込んで，メッセージを送信可能

### 2.1.3 攻撃例

攻撃者が通信に加わることで，プロトコルによっては安全性が満たされない場合がある．
通常の通信者をA，B，攻撃者をCと記述し，NSPKを対象に攻撃例を述べる．

$$
\begin{array}{lllllll}
1. & A & \rightarrow & C & & : & \{N_A, A\}K_C \\
1'. & & & C & \rightarrow B & : & \{N_A, A\}K_B \\
2'. & & & C & \leftarrow B & : & \{N_A, N_B\}K_A \\
2. & A & \leftarrow & C & & : & \{N_A, N_B\}K_A \\
3. & A & \rightarrow & C & & : & \{N_B\}K_C \\
3'. & & & C & \rightarrow B & : & \{N_B\}K_B
\end{array}
$$

各番号の通信で行われている動作などを次に述べる.

1 : ユーザ A が攻撃者 C と通信を開始する.

1′ : 攻撃者 C がユーザ A になりすまして，ユーザ B と通信を開始する.
通信開始のメッセージに，先程取得したノンス $N_A$ を使用する.
ユーザ B はメッセージ中の情報 A から，通信相手は A であると認識する.

2′ : プロトコルの仕様に従い，1′ のメッセージに対して，ユーザ B が返信する.

2 : 2′ で取得したメッセージはユーザ A の公開鍵で暗号化されているため，
C には復号化不可である.
プロトコルの仕様から，1 の返信として利用可能なため，2′ で取得したメッセージを
ユーザ A に返信する.
メッセージ中にユーザ A が攻撃者 C に送信したノンス $N_A$ が含まれていることから，
ユーザ A は返信者が攻撃者 C であると認識する.

3 : 返信に含まれているノンス $N_B$ を攻撃者 C が取得する.
この時点で，A-B 間の通信の認証用に作成されたノンスが，攻撃者 C に取得されるので，
機密性が満たされない.

3′ : 3 で取得したノンス $N_B$ を用いてユーザ B へ返信する.
ユーザ B はメッセージにノンス $N_B$ が含まれていることから，返信相手がユーザ A だと認識する.
実際の通信相手は攻撃者 C なので，この時点でユーザ A へのなりすましが完了し，
真正性が満たされない.

## 2.2 項書き換え系

状態の推移を表すための計算モデルとして，項書き換え系を定義する．まず，項書き換え系で用いられる集合について述べる．関数記号の集合を $F$，変数の集合を $X$ と表す．次に，書き換えの対象となる項を定義する.

**定義 2.2.1 (項)**

項の全体集合 $\mathrm{T}(F, X)$ を次の条件を満たす集合と定める.

1. $X \subseteq \mathrm{T}(F, X)$
2. $f \in F, t_1, \ldots, t_n \in \mathrm{T}(F, X)$ ならば $f(t_1, \ldots, t_n) \in \mathrm{T}(F, X)$

項書き換え系を定義するために，項の位置集合，部分項，項に出現する変数集合をそれぞれ定義する.

**定義 2.2.2 (項 $s$ の位置集合)**

下記の定義では，$s, s_1, \ldots, s_n \in \mathrm{T}(F, X), f \in F$ である．$\mathbb{N}$ は自然数の集合，$\epsilon$ は空文字列を表す.
項 $s$ の位置集合 $\mathrm{Pos}(t)$ の定義を次に示す.

$\mathrm{Pos}(s) : \mathrm{T}(F, X) \to \mathbb{N}^*$

$$\mathrm{Pos}(s) = \begin{cases} \{\epsilon\} & (s \in X) \\ \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in \mathrm{Pos}(s_i)\} & (s = f(s_1, \ldots, s_n)) \end{cases}$$

集合 $\mathrm{Pos}(s)$ の要素は辞書式順序で順序付けられていると定義する.

項の位置情報を用いて，部分項を次に定義する．

**定義 2.2.3 (部分項)**
$n \in \mathbb{N}$, $f \in F$, $t, t_1, \ldots, t_n \in \mathrm{T}(F, X)$, $ip \in \mathrm{Pos}(f(t_1, \ldots, t_n))$, $p \in \mathrm{Pos}(t_i)$ の時，項 $t$ における位置 $p \in \mathrm{Pos}(t)$ の部分項 $t|_p$ の定義を次に示す．
$t|_\epsilon = t$
$f(t_1, \ldots, t_n)|_{ip} = t_i|_p$

**定義 2.2.4 (項に出現する変数集合)**
項 $t \in \mathrm{T}(F, X)$ に出現する変数の集合 $\mathrm{Var}(t)$ の定義を次に示す．
$\mathrm{Var}(t) = \{x \in X \mid p \in \mathrm{Pos}(t), t|_p = x\}$

**例 2.2.1**
項 $s$ の位置集合 $\mathrm{Pos}(s)$，部分項 $s|_p$ の例を述べる．
関数記号の集合 $F = \{\mathsf{g}, \mathsf{f}, \mathsf{a}\}$ と定義する．$\mathsf{g}$ は二つの引数を持つ関数記号，$\mathsf{f}$ は 1 つの引数を持つ関数記号，$\mathsf{a}$ は引数をもたない関数記号である．項 $s = \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{a})$ の場合，位置集合は $\mathrm{Pos}(s)$=$\{\epsilon, 1, 11, 2\}$ となる．$p \in \mathrm{Pos}(s)$ の場合，部分項 $s|_p$ はそれぞれ次の項を表す．$s|_\epsilon = \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{a})$, $s|_1 = \mathsf{f}(\mathsf{a})$, $s|_{11} = \mathsf{a}$, $s|_2 = \mathsf{a}$.

以上の定義を用いると，項書き換え系は次の定義となる．

**定義 2.2.5 (項書き換え系)**
項書き換え系とは，項から項への書き換え規則の集合である．項書き換え系 $R$ を条件を満たす集合として定める．
$R \subseteq \{l \to r \mid l, r \in \mathrm{T}(F, X), l \notin X, \mathrm{Var}(r) \subseteq \mathrm{Var}(l)\}$

項書き換え系が左線形であるとは，項書き換え系に左線形な書き換え規則が存在することである．書き換え規則が左線形の定義を次に示す．

**定義 2.2.6 (左線形)**
書き換え規則 $l \to r$ が左線形であるとは，任意の変数 $x \in \mathrm{Var}(l)$ が項 $l$ の部分項に一度しか現れない書き換え規則である．

また，書き換えの系列を述べるために，書き換えを表す二項関係を定義する．

**定義 2.2.7 (二項関係 $\to_R$)**
項書き換え系 $R$ を用いて，項 $s$ から項 $t$ へ書き換えたことを，$s \to_R t$ と記述する．二項関係 $\to_R$ は次の条件を満たす．
$\forall s, t \in \mathrm{T}(F, X)(s \to_R t \Leftrightarrow \exists p \in \mathrm{Pos}(s), \exists l \to r \in R\ (\ s|_p = l \wedge t|_p = r\ ))$

上記の二項演算 $\to_R$ を用いて，書き換えによって到達可能な項の集合と正規形の集合を定義する．

**定義 2.2.8 (到達可能な項の集合)**
項の集合 $L$ に項書き換え系 $R$ を適用した場合の，到達可能な項の集合 $R^*(L)$ を次の集合と定める．
$R^*(L) = \{t \in \mathrm{T}(F, X) \mid \exists s \in L, s \to_R^* t\}$

**定義 2.2.9 (正規形)**
項書き換え系 $R$ の任意の書き換え規則で書き換え不能な項を正規形と呼ぶ．項書き換え系 $R$ における正規形の集合を $\mathrm{NF}(R)$ と記述すると，次の集合となる．
$\mathrm{NF}(R) = \{t \in \mathrm{T}(F, X) \mid \forall s \in \mathrm{T}(F, X), \neg t \to s\}$

**例 2.2.2**
項書き換え系 $R$ と到達可能な項の集合 $R^*(L)$ の例を示す．関数記号の集合 $F$ は例 2.2.1 と同じ集合を用いる．項の集合を $L = \{\mathsf{f(a)}, \mathsf{g(a, a)}\}$，書き換え規則の集合を $R = \{\mathsf{f(a)} \to \mathsf{f(f(a))}, \mathsf{g(a, a)} \to \mathsf{g(f(a), f(a))}\}$ と定義する．この時，項の集合 $L$ の要素について成立する書き換えは，$\mathsf{f(a)} \to_R \mathsf{f(f(a))} \to_R \mathsf{f(f(f(a)))} \to_R \cdots$, $\mathsf{g(a, a)} \to_R \mathsf{g(f(a), f(a))} \to_R \mathsf{g(f(f(a)), f(a))} \to_R \cdots$, $\mathsf{g(a, a)} \to_R \mathsf{g(f(a), f(a))} \to_R \mathsf{g(f(a), f(f(a)))} \to_R \cdots$，である．すなわち，到達可能な項の集合は $R^*(L) = \{\mathsf{g}(\mathsf{f}^n(\mathsf{a}), \mathsf{f}^m(\mathsf{a})) \mid n, m \in \mathbb{N}\} \cup \{\mathsf{f}^n(\mathsf{a}) \mid n \in \mathbb{N}\}$ となる．

## 2.3 木オートマトン

項の無限集合に有限の記述を与えるために木オートマトンを定義する．木オートマトンは $A = \langle F, Q, Q_f, \Delta \rangle$ と記述する．木オートマトン $A$ は $F$, $Q$, $Q_f$, $\Delta$ の 4 つの集合で定義される項書き換え系である．$F$ は木オートマトンで用いられる関数記号の集合，$Q$ は状態の有限集合，$Q_f$ は終了状態の集合である．ただし，$Q_f \subseteq Q$ と定義する．$\Delta$ は遷移規則である．遷移規則の定義は後述する．通常の項書き換え系における規則が項から項への書き換え規則であるのに対し，木オートマトンにおける規則，すなわち遷移規則は，部分項に状態をもつ項から状態への書き換え規則である．まず，木オートマトンで使われる項の集合を定義する．

**定義 2.3.1 (木オートマトンで使われる項集合)**
遷移規則に使われる項の集合 $\mathrm{T}(F \cup Q)$ の定義を次に示す．
$\mathrm{T}(F \cup Q) = Q \cup \{f(q_1, \ldots, q_k) \mid k \in \mathbb{N}, f \in F, q_1, \ldots, q_k \in Q, k \geq 1\}$
また，関数記号のみで構成される項の集合 $\mathrm{T}(F)$ の定義を次に示す．
$\mathrm{T}(F) = \{f(f_1, \ldots, f_k) \mid k \in \mathbb{N}, f, f_1, \ldots, f_k \in F, k \geq 1\}$

次に，木オートマトンの遷移規則 $\Delta$ の定義を行う．遷移規則とは，正規化遷移規則の集合である．正規化遷移規則とは，項から状態への書き換え規則である．正規化遷移規則の定義を次に示す．

**定義 2.3.2 (正規化遷移規則)**
$t \in \mathrm{T}(F \cup Q)$, $q \in Q$ の時，正規化遷移規則とは，書き換え規則 $t \to q$ である．

また，木オートマトン $A$ の遷移規則を用いて書き換えたことを表す二項関係 $s \to_A t$ の定義を次に示す．

**定義 2.3.3 (二項関係 $\to_A$)**
$\forall s, t \in \mathrm{T}(F, Q)(s \to_A t \Leftrightarrow \exists p \in \mathrm{Pos}(s), \exists t' \to q \in \Delta \ (\ s|_p = t' \wedge t|_p = q\ ))$

二項関係 $\to_A$ を利用して，状態 $q$ に到達する項の集合を定義する．

**定義 2.3.4 (状態 $q$ に到達する項の集合)**
木オートマトン $A$ の遷移規則を適用して，状態 $q$ に到達する項の集合 $\mathrm{L}(A,\ q)$ は次の集合である．
$\mathrm{L}(A,\ q) = \{t \in \mathrm{T}(F) \mid \exists q \in \mathrm{Q}, t \to_A^* q\}$

$\mathrm{L}(A,\ q)$ の中でも，終了状態に到達する項の集合を受理言語と呼ぶ．受理言語の定義は次の通りである．

**定義 2.3.5 (木オートマトンの受理言語)**
木オートマトン $A$ の受理言語 $\mathrm{L}(A)$ は次の集合である．
$\mathrm{L}(A) = \{t \in \mathrm{T}(F) \mid \exists q \in Q_f,\ t \to_A^* q\}$

また，木オートマトン $A$ に遷移規則を追加する場合，追加したい規則が正規化遷移規則でない時には，正規化を行う．正規化とは，全ての部分項に状態を割り当てる処理を指す．正規化は写像 $\alpha$:$\{s|_p \mid \exists s \in \mathrm{T}(F \cup Q)\backslash Q,\ p \in \mathrm{Pos}(s)\} \to Q$ と正規化関数 $\mathrm{Norm}_\alpha$ で行われる．正規化関数 $\mathrm{Norm}_\alpha$ は次の定義である．

**定義 2.3.6 (正規化関数 $\mathrm{Norm}_\alpha$)**
$s,\ t_1, \ldots,\ t_n \in \mathrm{T}(F \cup Q),\ q \in Q$, 遷移規則の全体集合を $\mathrm{Rule}(F,\ Q)$，正規化遷移規則の全体集合を $\mathrm{NRule}(F,\ Q)$ と記述する．
$\mathrm{Norm}_\alpha : \mathrm{Rule}(F,\ Q) \to \mathrm{NRule}(F,\ Q)$

$$\mathrm{Norm}_\alpha(s \to q) = \begin{cases} \emptyset & (s = q) \\ \{s \to q\} & (s \neq q,\ s \in Q) \\ \{f(\alpha(t_1), \ldots, \alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} \mathrm{Norm}_\alpha(t_i \to \alpha(t_i)) & (s = f(t_1, \ldots,\ t_n)) \end{cases}$$

**例 2.3.1**
木オートマトンと正規化の例を次に示す．関数記号の集合 $F$ は例 2.2.1 と同じ集合を用いる．
木オートマトン $A = \langle F,\ Q,\ Q_f, \Delta_0 \rangle$ の各要素を次に定める．

$$\begin{array}{lll} F & = \{ & \mathsf{g},\ \mathsf{f},\ \mathsf{a}\ \} \\ Q & = \{ & \mathsf{q_a},\ \mathsf{q_f},\ \mathsf{q_{end}}\ \} \\ Q_f & = \{ & \mathsf{q_{end}}\ \} \\ \Delta_0 & = \{ & \mathsf{a} \to \mathsf{q_a} \qquad \mathsf{f}(\mathsf{q_a}) \to \mathsf{q_f} \quad \mathsf{f}(\mathsf{q_f}) \to \mathsf{q_f} \\ & & \mathsf{g}(\mathsf{q_f},\ \mathsf{q_f}) \to \mathsf{q_{end}} \qquad\qquad\qquad\qquad\qquad\quad \} \end{array}$$

この木オートマトンの受理言語は $\mathrm{L}(A) = \{\mathsf{g}(\mathsf{f}^n(\mathsf{a}),\ \mathsf{f}^m(\mathsf{a})) \mid n,\ m \in \mathbb{N},\ n \geq 1,\ m \geq 1\}$ である．木オートマトン $A$ の受理言語を $\mathrm{L}(A) = \{\mathsf{g}(\mathsf{f}^n(\mathsf{a}),\ \mathsf{f}^m(\mathsf{a})) \mid n,\ m \in \mathbb{N}\}$ にするために，遷移規則 $\mathsf{g}(\mathsf{a}, \mathsf{a}) \to \mathsf{q_{end}}$ を $\Delta_0$ に追加する．ただし，規則 $\mathsf{g}(\mathsf{a},\ \mathsf{a}) \to \mathsf{q_{end}}$ は正規化遷移規則ではないため，正規化を行う．$\alpha(\mathsf{a}) = \mathsf{q_a}$ の時，正規化関数 $\mathrm{Norm}_\alpha$ から次の規則を得る．
$\mathrm{Norm}_\alpha(\mathsf{g}(\mathsf{a},\ \mathsf{a}) \to \mathsf{q_{end}}) = \{\mathsf{g}(\mathsf{q_a},\ \mathsf{q_a}) \to \mathsf{q_{end}},\ \mathsf{a} \to \mathsf{q_a}\}$
これらの規則を $\Delta_0$ に追加することで，木オートマトン $A$ の受理言語が $\mathrm{L}(A) = \{\mathsf{g}(\mathsf{f}^n(\mathsf{a}),\ \mathsf{f}^m(\mathsf{a})) \mid n,\ m \in \mathbb{N}\}$ となる．すなわち，規則追加後の木オートマトン $A$ は次の通りである．

$A = \langle F,\ Q,\ Q_f, \Delta \rangle$

$F, Q, Q_{\mathrm{f}}$ には変更がないため，略す．

$$\begin{array}{lll} \Delta = \{ & \mathsf{a} \to \mathrm{q_a} & \mathsf{f}(\mathrm{q_a}) \to \mathrm{q_f} \qquad \mathsf{f}(\mathrm{q_f}) \to \mathrm{q_f} \\ & \mathsf{g}(\mathrm{q_f},\ \mathrm{q_f}) \to \mathrm{q_{end}} & \mathsf{g}(\mathrm{q_a},\ \mathrm{q_a}) \to \mathrm{q_{end}} \qquad \} \end{array}$$

2 つの木オートマトンの受理言語の共通部分を計算するために，積オートマトンの定義を次に示す．

**定義 2.3.7 (木オートマトンの積)**

木オートマトン $A_1 = \langle F,\ Q_1,\ Q_{f1}, \Delta_1 \rangle$ と $A_2 = \langle F,\ Q_2,\ Q_{f2}, \Delta_2 \rangle$ の積は次の木オートマトン $A$ である．

$A = \langle F,\ Q_1 \times Q_2,\ Q_{f1} \times Q_{f2}, \Delta_1 \times \Delta_2 \rangle$

ただし，$\Delta_1 \times \Delta_2 = \{ f((q_1,\ q_1'), \ldots, (q_n,\ q_n')) \to (q,\ q') \mid q_1, \ldots, q_n \in \mathrm{Q}_1,\ q_1', \ldots, q_n' \in \mathrm{Q}_2,\ f(q_1, \ldots, q_n) \to q \in \Delta_1,\ f(q_1', \ldots, q_n') \to q' \in \Delta_2 \}$ である．

積オートマトンの例を次に示す．

**例 2.3.2**

まず，木オートマトン $A_1 = \langle F,\ Q_1,\ Q_{f1}, \Delta_1 \rangle$ の各要素を次に定める．

$$\begin{array}{lll} F & = \{ & \mathsf{g},\ \mathsf{f},\ \mathsf{a} \ \} \\ Q_1 & = \{ & \mathrm{q_a},\ \mathrm{q_f},\ \mathrm{q_{end}} \ \} \\ Q_{f1} & = \{ & \mathrm{q_{end}} \ \} \\ \Delta_1 & = \{ & \mathsf{a} \to \mathrm{q_a} \qquad \mathsf{f}(\mathrm{q_a}) \to \mathrm{q_f} \qquad \mathsf{f}(\mathrm{q_f}) \to \mathrm{q_f} \\ & & \mathsf{g}(\mathrm{q_f},\ \mathrm{q_f}) \to \mathrm{q_{end}} \quad \mathsf{g}(\mathrm{q_a},\ \mathrm{q_a}) \to \mathrm{q_{end}} \qquad \} \end{array}$$

木オートマトン $A_1$ の受理言語は $\mathrm{L}(A_1) = \{ \mathsf{g}(\mathsf{f}^n(\mathsf{a}), \mathsf{f}^m(\mathsf{a})) \mid n, m \in \mathbb{N} \}$ である．次に，$A_2 = \langle F,\ Q_2,\ Q_{f2}, \Delta_2 \rangle$ の各要素を次に定める．

$$\begin{array}{lll} F & = \{ & \mathsf{g},\ \mathsf{f},\ \mathsf{a} \ \} \\ Q_2 & = \{ & \mathrm{q_a},\ \mathrm{q_{end}} \ \} \\ Q_{f2} & = \{ & \mathrm{q_{end}} \ \} \\ \Delta_2 & = \{ & \mathsf{a} \to \mathrm{q_a} \qquad \mathsf{a} \to \mathrm{q_{end}} \quad \mathsf{g}(\mathrm{q_a},\ \mathrm{q_a}) \to \mathrm{q_{end}} \} \end{array}$$

$A_2$ の受理言語は $\mathrm{L}(A_2) = \{ \mathsf{g}(\mathsf{a},\ \mathsf{a}),\ \mathsf{a} \}$ である．この時，$A_1$ と $A_2$ の積 $A = \langle F',\ Q',\ Q_f',\Delta' \rangle$ の各要素は次の通りである．

$$\begin{array}{lll} F' & = F & \\ Q' & = \{ & (\mathrm{q_a},\ \mathrm{q_a}),\ (\mathrm{q_a},\ \mathrm{q_{end}}),\ (\mathrm{q_f},\ \mathrm{q_a}),\ (\mathrm{q_f},\ \mathrm{q_{end}}),\ (\mathrm{q_{end}},\ \mathrm{q_a}),\ (\mathrm{q_{end}},\ \mathrm{q_{end}}) \ \} \\ Q_f' & = \{ & (\mathrm{q_{end}},\ \mathrm{q_{end}}) \ \} \\ \Delta' & = \{ & \mathsf{a} \to (\mathrm{q_a},\ \mathrm{q_a}) \qquad \mathsf{a} \to (\mathrm{q_a},\ \mathrm{q_{end}}) \\ & & \mathsf{g}((\mathrm{q_f},\ \mathrm{q_a}),\ (\mathrm{q_f},\ \mathrm{q_a})) \to (\mathrm{q_{end}}\cdot\ \mathrm{q_{end}}) \quad \mathsf{g}((\mathrm{q_a},\ \mathrm{q_a}),\ (\mathrm{q_a},\ \mathrm{q_a})) \to (\mathrm{q_{end}},\ \mathrm{q_{end}}) \qquad \} \end{array}$$

この時，$A$ の受理言語は $\mathrm{L}(A) = \{ \mathsf{g}(\mathsf{a},\ \mathsf{a}) \}$ となる．すなわち，木オートマトンの積オートマトンを計算することで，受理言語の共通部分を計算できる．

# 第3章　従来手法

この章では，本手法の基盤となる [GK00] の手法について解説する．[GK00] は，項書き換え系と木オートマトンを用いて NSPK プロトコルと攻撃者を定式化し，機密性と真正性の検証法を提案した．

## 3.1　現実の通信の定式化

暗号プロトコルの安全性を検証するためには，現実の通信を定式化する必要がある．すなわち，次の四つを定式化しなければならない．

- 公開鍵など通信に用いる情報
- プロトコルと攻撃者それぞれの動作
- 攻撃者が取得可能な情報
- 安全性をみたす条件

[GK00] の手法では，通信に用いる情報を次に定義される項集合の要素で表す．

**定義 3.1.1 (通信に用いる情報)**

$$
\begin{aligned}
\mathrm{ENEMY} &= \{\ 0\ \} \cup \{\mathsf{s}(t) \mid t \in \mathrm{ENEMY}\} \\
\mathrm{AGT} &= \{\mathsf{agt}(a) \mid a \in \{\mathsf{A},\ \mathsf{B}\} \cup \mathrm{ENEMY}\} \\
\mathrm{NONCE} &= \{\mathsf{N}(a,\ b) \mid a,\ b \in \mathrm{AGT}\} \\
\mathrm{KEY} &= \{\mathsf{pubkey}(a) \mid a \in \mathrm{AGT}\} \\
\mathrm{ENC} &= \{\mathsf{encr}(k,\ a,\ m) \mid k \in \mathrm{KEY},\ a \in \mathrm{AGT},\ m \in \mathrm{CONS}\} \\
\mathrm{MESG} &= \{\mathsf{mesg}(a,\ b,\ m) \mid a,\ b \in \mathrm{AGT},\ m \in \mathrm{ENC} \cup \mathrm{CONS} \cup \{\mathsf{null}\}\} \\
\mathrm{GOAL} &= \{\mathsf{goal}(a,\ b) \mid a,\ b \in \mathrm{AGT}\ \} \\
\mathrm{INIT} &= \{\mathsf{c\_init}(a,\ b,\ c) \mid a,\ b,\ c \in \mathrm{AGT}\} \\
\mathrm{RESP} &= \{\mathsf{c\_resp}(b,\ a,\ c) \mid a,\ b,\ c \in \mathrm{AGT}\}
\end{aligned}
$$

CONS，$\mathrm{T_G}$ は以下を満たす最小の集合とする．

$$
\begin{aligned}
\mathrm{CONS} &= \{\mathsf{cons}(m,\ m') \mid m \in \mathrm{AGT} \cup \mathrm{NONCE} \cup \mathrm{KEY} \cup \mathrm{ENC},\ m' \in \mathrm{CONS} \cup \{\mathsf{null}\}\} \\
\mathrm{T_G} &= \mathrm{ENEMY} \cup \mathrm{AGT} \cup \mathrm{NONCE} \cup \mathrm{KEY} \cup \mathrm{ENC} \cup \mathrm{MESG} \cup \mathrm{CONS} \cup \mathrm{INIT} \cup \mathrm{RESP} \\
&\quad \cup \mathrm{GOAL} \cup \{x \sqcup y \mid x,\ y \in \mathrm{T_G}\} \cup \{\mathsf{A},\ \mathsf{B}\}
\end{aligned}
$$

それぞれの項は次の情報，または動作を表す．

| | | |
|---|---|---|
| A, B | : | 正常な通信者 |
| $\mathsf{s}^n(0)$ | : | 攻撃者 $(n \geq 0)$ |
| $\mathsf{agt}(a)$ | : | 通信者 $a$ であると証明する情報 |
| $\mathsf{N}(a, b)$ | : | $b$ と認証するために $a$ が作成したノンス |
| $\mathsf{pubkey}(a)$ | : | 通信者 $a$ の公開鍵 |
| $\mathsf{encr}(k, a, m)$ | : | 通信者 $a$ が鍵 $k$ で情報 $m$ を暗号化した情報 |
| $\mathsf{mesg}(a, b, m)$ | : | 通信者 $a$ から 受信者 $b$ へメッセージ $m$ を送信 |
| $\mathsf{cons}(m, m')$ | : | リスト構造 $[m, m']$ |
| $\mathsf{goal}(a, b)$ | : | $a$ から $b$ への通信要求 |
| $x \sqcup y$ | : | 集合 $\{x\} \cup \{y\}$ |
| $\mathsf{c\_init}(a, b, c)$ | : | 通信の開始者 $a$ が受信者 $b$ と通信を確立したと判断したが，<br>実際は $c$ と通信中である．例えば，c_init(agt(A), agt(B), agt(0)) は，<br>攻撃者が受信者 B に偽装して通信の開始者 A と通信を確立したことを表す． |
| $\mathsf{c\_resp}(b, a, c)$ | : | 受信者 $b$ が通信の開始者 $a$ と通信を確立したと判断したが，<br>実際は $c$ と通信中である．例えば，c_resp(agt(B), agt(A), agt(0)) は，<br>攻撃者が通信の開始者 A に偽装して受信者 B と通信を確立したことを表す． |

### 3.1.1 プロトコルの定式化

[GK00] では，$l.\ r \in \mathrm{T_{Gv}}$ を満たす書き換え規則 $l \to r$ を用いてプロトコルを表す．まず，変数の集合を VAR と定義して，引数に変数を持つ項の集合を定義した．

**定義 3.1.2 (変数を持つ項の集合)**

$$
\begin{aligned}
\mathrm{AGT'} &= \{\mathsf{agt}(x) \mid x \in \mathrm{VAR}\} \\
\mathrm{NONCE'} &= \{\mathsf{N}(a, b) \mid a, b \in \mathrm{VAR} \cup \mathrm{AGT'}\} \\
\mathrm{KEY'} &= \{\mathsf{pubkey}(a) \mid a \in \mathrm{VAR} \cup \mathrm{AGT'}\} \\
\mathrm{ENC'} &= \{\mathsf{encr}(k, a, m) \mid k \in \mathrm{VAR} \cup \mathrm{KEY'},\ a \in \mathrm{VAR} \cup \mathrm{AGT'},\ m \in \mathrm{VAR} \cup \mathrm{CONS'}\} \\
\mathrm{MESG'} &= \{\mathsf{mesg}(a, b, m) \mid a, b \in \mathrm{VAR} \cup \mathrm{AGT'},\ m \in \mathrm{VAR'} \cup \mathrm{ENC'} \cup \mathrm{CONS'} \cup \{\mathsf{null}\}\} \\
\mathrm{GOAL'} &= \{\mathsf{goal}(a, b) \mid a, b \in \mathrm{VAR} \cup \mathrm{AGT'}\ \} \\
\mathrm{INIT'} &= \{\mathsf{c\_init}(a, b, c) \mid a, b, c \in \mathrm{VAR} \cup \mathrm{AGT'}\} \\
\mathrm{RESP'} &= \{\mathsf{c\_resp}(b, a, c) \mid a, b, c \in \mathrm{VAR} \cup \mathrm{AGT'}\}
\end{aligned}
$$

CONS′，$\mathrm{T_{Gv}}$ は以下を満たす最小の集合と定義する．

$$
\begin{aligned}
\mathrm{CONS'} &= \{\mathsf{cons}(m, m') \mid m \in \mathrm{VAR'} \cup \mathrm{AGT'} \cup \mathrm{NONCE'} \cup \mathrm{KEY'} \cup \mathrm{ENC'}, \\
&\qquad\qquad m' \in \mathrm{CONS'} \cup \{\mathsf{null}\}\} \\
\mathrm{T_{Gv}} &= \mathrm{ENEMY'} \cup \mathrm{AGT'} \cup \mathrm{NONCE'} \cup \mathrm{KEY'} \cup \mathrm{ENC'} \cup \mathrm{MESG'} \cup \mathrm{CONS'} \cup \mathrm{INIT'} \\
&\quad \cup \mathrm{RESP'} \cup \mathrm{GOAL'} \cup \{x \sqcup y \mid x, y \in \mathrm{T_{Gv}}\}
\end{aligned}
$$

プロトコルを表す項書き換え系を $R$ と定義すると，任意の書き換え規則 $l \to r \in R$ は，$l, r \in \mathrm{T_{Gv}}$ である．また，書き換え規則の左辺は送信メッセージを表す項である．右辺は左辺が表すメッセージに対する返信と付加情報を表す項である．付加情報とは，定義 3.1.2 の集合 INIT′，RESP′ の要素である．すなわち，実際にどの通信者と通信が確立されているか示す情報である．

例として，NSPK(例 2.1.1 参照) を項書き換え系で表すと次の集合となる．

**例 3.1.1 (NSPK を表す項書き換え系 $R_N$)**

LHS は書き換え規則 $l \to r$ の左辺 $l$ を表す.

$R_{\mathrm{N}} = \{$ goal(agt($a$), agt($b$))
$\to$ LHS
$\sqcup$ mesg(agt($a$), agt($b$),
encr(pubkey(agt($b$)), agt($a$), cons(N(agt($a$), agt($b$)), cons(agt($a$), null))),

mesg(agt($a$), agt($b$), encr(pubkey(agt($b$)), agt($c$), cons($n$, cons(agt($a$), null)))
$\to$ LHS
$\sqcup$ mesg(agt($b$), agt($a$),
encr(pubkey(agt($a$)), agt($b$), cons($n$, cons(N(agt($b$), agt($a$)), null))),

mesg(agt($b$), agt($a$), encr(pubkey(agt($a$)), agt($c$), cons(N(agt($a$), agt($b$)), cons($n$, null)))
$\to$ LHS
$\sqcup$ mesg(agt($a$), agt($b$), encr(pubkey(agt($b$)), agt($a$), cons(N(agt($b$), agt($a$)), null))
$\sqcup$ c_init($a$, $b$, $c$),

mesg(agt($a$), agt($b$), encr(pubkey(agt($b$)), agt($c$), cons(N(agt($b$), agt($a$)), null))
$\to$ LHS $\sqcup$ c_resp($b$, $a$, $c$)
$\}$

### 3.1.2 攻撃者の定式化

従来手法では，攻撃者の動作を次の書き換え規則で表す.

**定義 3.1.3 (攻撃者の動作)**

cons($x$, $y$) $\sqcup$ $z \to$ LHS $\sqcup x$

cons($x$, $y$) $\sqcup$ $z \to$ LHS $\sqcup y$

mesg($x$, $y$, $z$) $\sqcup$ $u \to$ LHS $\sqcup z$

encr(pubkey(agt(0)), $y$, $z$) $\sqcup$ $u \to$ LHS $\sqcup z$

encr(pubkey(agt(s($x$))), $y$, $z$) $\sqcup$ $u \to$ LHS $\sqcup z$

1 行目から 3 行目の規則は，メッセージや情報の集合を表す項からその内容を取り出す動作を表す．また，4 行目，5 行目の規則は，メッセージを復号化し，その内容を攻撃者の取得情報に加える動作を表す.

次に，攻撃者の初期情報および取得情報と，取得情報から作成可能な情報を木オートマトン $A_0$ で表す.

**定義 3.1.4 (攻撃者の取得情報)**

$\mathrm{A}_0 = \langle F, Q, Q_f, \Delta \rangle$

$F = \{0, \mathrm{A}, \mathrm{B}, \mathrm{mesg}, \mathrm{encr}, \mathrm{cons}, \mathrm{agt}, \mathrm{pubkey}, \mathrm{N}, \mathrm{goal}, \mathrm{c\_init}, \mathrm{c\_resp}, \mathrm{null}, \sqcup\}$

$Q = \{\mathrm{q_{int}}, \mathrm{q_A}, \mathrm{q_B}, \mathrm{q_{agtA}}, \mathrm{q_{agtB}}, \mathrm{q_{agtI}}, \mathrm{q_{net}}\}$

$Q_f = \{\mathrm{q_{net}}\}$

$$
\begin{array}{lllll}
\Delta_0 = \{ & 0 \to q_{\mathrm{int}} & \mathsf{pubkey}(q_{\mathrm{agtI}}) \to q_{\mathrm{net}} & q_{\mathrm{net}} \sqcup q_{\mathrm{net}} \to q_{\mathrm{net}} \\
& \mathsf{A} \to q_{\mathrm{A}} & \mathsf{pubkey}(q_{\mathrm{agtA}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& \mathsf{s}(q_{\mathrm{int}}) \to q_{\mathrm{int}} & \mathsf{pubkey}(q_{\mathrm{agtB}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{B} \to q_{\mathrm{B}} & \mathsf{N}(q_{\mathrm{agtI}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{int}}) \to q_{\mathrm{agtI}} & \mathsf{N}(q_{\mathrm{agtI}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{A}}) \to q_{\mathrm{agtA}} & \mathsf{N}(q_{\mathrm{agtI}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtA}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{B}}) \to q_{\mathrm{agtB}} & \mathsf{cons}(q_{\mathrm{net}}, q_{\mathrm{net}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtI}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{int}}) \to q_{\mathrm{net}} & \mathsf{null} \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtB}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{A}}) \to q_{\mathrm{net}} & \mathsf{encr}(q_{\mathrm{net}}, q_{\mathrm{agtI}}, q_{\mathrm{net}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtI}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{B}}) \to q_{\mathrm{net}} & \mathsf{mesg}(q_{\mathrm{net}}, q_{\mathrm{net}}, q_{\mathrm{net}}) \to q_{\mathrm{net}} & \mathsf{goal}(q_{\mathrm{agtI}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \}
\end{array}
$$

## 3.2 検証方法

定義 3.1.4 の木オートマトン $\mathrm{A}_0$ と，定義 3.1.3 とプロトコルを表す項書き換え系 $R$ を用いて，プロトコルの検証を行う．

[GK00] では，攻撃者が取得可能な全ての情報を，初期情報に書き換え規則を適用した時に到達可能な項の集合，すなわち，$R^*(\mathrm{L}(\mathrm{A}_0))$ で表せるように，プロトコルや攻撃者を定式化した．しかし，$R^*(\mathrm{L}(\mathrm{A}_0))$ は無限集合であるため，この集合に安全性を表す項が含まれるか探索すると，現実的な時間で処理が終わらない場合がある．そこで，現実的な時間で検証するために，$R^*(\mathrm{L}(\mathrm{A}_0)) \subseteq \mathrm{L}(\mathrm{T}_R\uparrow(\mathrm{A}_0))$ を満たす近似オートマトン $\mathrm{T}_R\uparrow(\mathrm{A}_0)$ を生成する．近似オートマトン $\mathrm{T}_R\uparrow(\mathrm{A}_0)$ の受理言語に，機密情報や，通信の偽装を表す項が存在するか判定することで，安全性の検証を行う．

### 3.2.1 近似オートマトン生成

[GK00] の近似オートマトン生成手法は，入力に近似関数，項書き換え系，木オートマトンを受け取り，近似オートマトン $\mathrm{T}_R\uparrow(\mathrm{A}_0)$ を計算する．近似関数や後述する危険対に必要となる Q-代入の定義を次に示す．

**定義 3.2.1 (Q-代入)**
Q-代入とは項が持つ変数に状態を割り当てることである．Q-代入 $\sigma$ は写像 $\sigma : X \to Q$ である．Q-代入 $\sigma$ が状態を割り当てる変数の集合を $\sigma$ のドメインと呼ぶ．$\sigma$ のドメインを次に定める．
$\mathrm{Dom}(\sigma) = \{x \in X \mid \sigma(x) \neq x\}$
$\mathrm{Dom}(\sigma) = \{x_1, \ldots, x_k\}$ のとき，Q-代入 $\sigma$ は $\sigma = \{x_1 \mapsto \sigma(x_1), \ldots, x_k \mapsto \sigma(x_k)\}$ と記述する．項 $l \in \mathrm{T}(F, X)$ に対して，Q-代入 $\sigma$ に従って変数に状態を代入した項を $l\sigma$ と表す．また，Q-代入の全体集合を $\Sigma(Q, X)$ と記述する．

Q-代入の例を次に示す．

**例 3.2.1**
項 $l = \mathsf{goal}(a, b)$，Q-代入 $\sigma = \{a \mapsto q_{\mathrm{A}}, b \mapsto q_{\mathrm{int}}\}$ の場合，$l\sigma = \mathsf{goal}(q_{\mathrm{A}}, q_{\mathrm{int}})$ である．

入力となる近似関数 $\gamma$ は次の定義である.

**定義 3.2.2 (近似関数 $\gamma$)**
状態の集合を $Q = \{\mathrm{q}_i \mid i \in \mathbb{N}\}$ と定義する.
近似関数 $\gamma$ は写像 $\gamma : R \times Q \times \Sigma(Q, X) \to Q^*$ である.
近似関数 $\gamma(l \to r, q, \sigma)$ は, $r\sigma$ の値は部分項に対応する状態列である. この値を用いて正規化関数 $\mathrm{Norm}_\alpha$ で用いる写像 $\alpha$ の値を定める. すなわち, $\mathrm{Pos}(r\sigma) = \{p_1, \ldots, p_k\}$ (ただし, $p_1, \ldots, p_k$ は辞書式順序にしたがって整列しているとする) かつ $\gamma(l \to r, q, \sigma) = \mathrm{q}_1 \ldots \mathrm{q}_k$ の場合, $\alpha(r\sigma|_{p_i}) = \mathrm{q}_i$ (ただし, $1 \leq i \leq k$) という出力になる.

近似関数 $\gamma$ の例を次に述べる.

**例 3.2.2**
f, g は例 2.2.1 と同じ関数記号である. また, $Q = \{\mathrm{q}, \mathrm{q_a}, \mathrm{q_f}, \mathrm{q_m}, \mathrm{q_n}\}$ と定義する.
$\gamma(\ \mathrm{g}(x, y) \to \mathrm{g}(\mathrm{f}(x), \mathrm{f}(y)), \mathrm{q}, \{x \mapsto \mathrm{q_a}, y \mapsto \mathrm{q_f}\}) = \mathrm{q_m q_n}$ のとき, 正規化関数 $\mathrm{Norm}_\alpha(\ \mathrm{g}(\mathrm{f}(\mathrm{q_a}), \mathrm{f}(\mathrm{q_f})) \to \mathrm{q}\ )$ で使われる写像 $\alpha$ の値は次のように規定される.
$\alpha(\mathrm{f}(\mathrm{q_a})) = \mathrm{q_m}, \alpha(\mathrm{f}(\mathrm{q_f})) = \mathrm{q_n}$

近似オートマトン生成の際に, 遷移規則を追加する基準となる危険対の定義を次に示す.

**定義 3.2.3 (危険対)**
$R$ を書き換え規則の集合, $Q$ を木オートマトンの状態の集合, $X$ を変数の集合, $A$ を木オートマトンとする. 危険対とは, ある項 $\mathrm{t}_1, \mathrm{t}_2 \in \mathrm{T}(F, Q) \cup \mathrm{T}(F)$ について, $t_1$ から $t_2$ へ $R$ の規則で書き換えられることに加えて, $t_1$ と $t_2$ に木オートマトン $A$ の規則を 0 回以上適用した時, 別の状態に到達しうる対である (図 3.1 参照). すなわち, 危険対とは, 次の条件を満たす項の対 $(l\sigma, r\sigma)$ である.
$\exists l \to r \in R, \exists \sigma \in \Sigma(Q, X), \exists q \in Q\ (l\sigma \to_A^* q \wedge r\sigma \not\to_A^* q)$

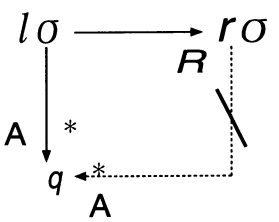



図 3.1: 危険対

近似オートマトン作成の手順を次に示す.

0. 入力を木オートマトン $A_i = \langle F, Q, Q_f, \Delta_i \rangle$, 近似関数 $\gamma$, 項書き換え系 $R$ とする.
1. 危険対を探索する. すなわち, $l\sigma \to_{A_i}^* q$ と $r\sigma \not\to_{A_i}^* q$ を満たすような $l \to r \in R$, $\sigma \in \Sigma(Q, Var(l))$, $q \in Q_f$ の組み合わせを探索する.

2-1. 危険対が見つからなければ終了する.

2-2. 危険対が見つかった場合，次の処理を実行する．

$\Delta_{i+1} = \Delta_i \cup \mathrm{Norm}_\alpha(r\sigma \to q)$

3. 入力の木オートマトンを $A_{i+1} = \langle F, Q, Q_f, \Delta_{i+1} \rangle$ に変更して，1 に戻る

上記の手順では，危険対 $(l\sigma, r\sigma)$ を発見すると，$r\sigma$ が受理状態に到達するように木オートマトンの遷移規則を追加する (図 3.2 参照)．作成した近似オートマトンの受理言語は $R^*(\mathrm{L}(\mathrm{A}_0))$ の受理言語を包含することが [GK00] に示されている．その定理と定理に必要な定義を次に示す．

**定義 3.2.4 (線形化)**
項 $t \in \mathrm{T}(F, X)$ を線形化した項を $t_{lin}$ と記述する．線形化とは，項 $t$ 中の非左線形な変数を全て異なる変数に置き換える処理である．

**例 3.2.3**
項 $\mathrm{g}(x, \mathrm{g}(x, y))$ を線形化すると，項 $\mathrm{g}(x, \mathrm{g}(x', y))$ になる．

**定義 3.2.5 (状態一致)**
$A$ を木オートマトン，$R$ を項書き換え系，$Q$ を近似オートマンの状態集合，$t \in \mathrm{T}(F, X)$ を非左線形な項，$\{p_1, \ldots, p_n\} \subseteq \mathrm{Pos}(t)$ を非左線形な変数の位置集合とする．
状態 $q, q_1, \ldots, q_n \in Q$ が変数 $x$ に一致するとは，$t_{lin}\sigma \to_A^* q$ かつ $t_{lin}\sigma|_{p_1} = q_1, \ldots, t_{lin}\sigma|_{p_n} = q_n$ を満たす Q-代入 $\sigma \in \Sigma(Q, X)$ が存在することである．

**定理 3.2.1 (近似オートマトン $\mathrm{T}_R\!\uparrow\!(\mathrm{A})$ の定理)**
$A$ を木オートマトン，$R$ を項書き換え系，$Q$ を近似オートマンの状態集合とする．任意の非左線形な書き換え規則 $l \to r \in R$, 項 $l$ 中の任意の非左線形な変数 $x$，変数 $x$ に一致する任意の状態 $q_1, \ldots, q_n \in Q$ について，$q_1 = \cdots = q_n$ もしくは $L(\mathrm{T}_R\!\uparrow\!(A), q_1) \cap \ldots \cap \mathrm{L}(\mathrm{T}_R\!\uparrow\!(A), q_n) = \emptyset$ ならば次の論理式が成り立つ．

$$\mathrm{L}(\mathrm{T}_R\!\uparrow\!(A)) \subseteq R^*(\mathrm{L}(A))$$

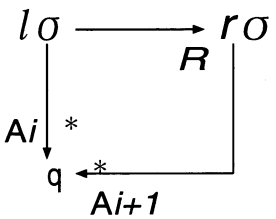



図 3.2: 遷移規則追加後の関係

### 3.2.2 安全性検証

プロトコルの安全性を検証する場合，通信者数や通信量が膨大だと [Sch02] の手法では，状態爆発が起こりうると述べた．これに対し，[GK00] では，通信者数などを項で定式化し，木オートマ

トンの受理言語を探索空間とすることで，通信者数などが膨大でも現実時間で計算可能な安全性検証法を提案している．

具体的には，前述の近似オートマトン $\mathrm{T}_R\uparrow(\mathrm{A}_0)$，機密情報を表す項を受理する木オートマトン $\mathrm{A}_{\mathrm{conf}}$，通信の偽装を表す項を受理する木オートマトン $\mathrm{A}_{\mathrm{aut}}$ を用いる．これらの木オートマトンの関数記号の集合 $F$，状態集合 $Q$，終了状態の集合 $Q_f$ は定義 3.1.4 で用いた集合と同様である．$\mathrm{A}_{\mathrm{conf}}$，$\mathrm{A}_{\mathrm{aut}}$ の木オートマトンの遷移規則 $\Delta_{\mathrm{conf}}$,$\Delta_{\mathrm{aut}}$ を次に定義する．

**定義 3.2.6** ($\mathrm{A}_{\mathrm{conf}}$の遷移規則)

$$
\begin{aligned}
\Delta_{\mathrm{conf}} = \{\quad & \mathsf{A} \to q_{\mathrm{A}} && \mathsf{agt}(q_{\mathrm{B}}) \to q_{\mathrm{agtB}} && \mathsf{N}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{B} \to q_{\mathrm{B}} && \mathsf{N}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} && \mathsf{N}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{A}}) \to q_{\mathrm{agtA}} && \mathsf{N}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} && q_{\mathrm{net}} \sqcup q_{\mathrm{net}} \to q_{\mathrm{net}}\}
\end{aligned}
$$

**定義 3.2.7** ($\mathrm{A}_{\mathrm{aut}}$の遷移規則)

$$
\begin{aligned}
\Delta_{\mathrm{aut}} = \{\quad & 0 \to q_{\mathrm{int}} && \mathsf{c\_init}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{s}(q_{\mathrm{int}}) \to q_{\mathrm{int}} && \mathsf{c\_resp}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{A} \to q_{\mathrm{A}} && \mathsf{c\_init}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{B} \to q_{\mathrm{B}} && \mathsf{c\_resp}(q_{\mathrm{agtA}}, q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{int}}) \to q_{\mathrm{agtI}} && \mathsf{c\_init}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{A}}) \to q_{\mathrm{agtA}} && \mathsf{c\_resp}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{agt}(q_{\mathrm{B}}) \to q_{\mathrm{agtB}} && \mathsf{c\_init}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& q_{\mathrm{net}} \sqcup q_{\mathrm{net}} \to q_{\mathrm{net}} && \mathsf{c\_resp}(q_{\mathrm{agtB}}, q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} \\
& \mathsf{c\_init}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} && \mathsf{c\_init}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{c\_resp}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} && \mathsf{c\_resp}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}, q_{\mathrm{agtI}}) \to q_{\mathrm{net}} \\
& \mathsf{c\_init}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} && \mathsf{c\_init}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \\
& \mathsf{c\_resp}(q_{\mathrm{agtA}}, q_{\mathrm{agtA}}, q_{\mathrm{agtB}}) \to q_{\mathrm{net}} && \mathsf{c\_resp}(q_{\mathrm{agtB}}, q_{\mathrm{agtB}}, q_{\mathrm{agtA}}) \to q_{\mathrm{net}} \quad \}
\end{aligned}
$$

$\mathrm{T}_R\uparrow(\mathrm{A}_0)$ と $\mathrm{A}_{\mathrm{conf}}$, $\mathrm{T}_R\uparrow(\mathrm{A}_0)$ と $\mathrm{A}_{\mathrm{aut}}$，それぞれの積オートマトン $\mathrm{A}'_{\mathrm{conf}}$, $\mathrm{A}'_{\mathrm{aut}}$ を生成し，それらに受理言語が存在しないとき，$\mathrm{T}_R\uparrow(\mathrm{A}_0)$ には機密情報または通信の偽装を表す項が存在しないと言える．$R^*(L(A_0)) \subseteq \mathrm{T}_R\uparrow(\mathrm{A}_0)$ であることから，それらの項は，攻撃者の取得情報を表す項集合 $R^*(\mathrm{L}(\mathrm{A}_0))$ にも存在しない．すなわち，$\mathrm{T}_R\uparrow(\mathrm{A}_0)$ で安全性が満たされていれば，$R^*(\mathrm{L}(\mathrm{A}_0))$ でも安全性が満たされている．

# 第4章　提案手法

この章では，[GK00] の手法の問題点を指摘し，その改善法を提案する．

## 4.1　従来法の問題点

[GK00] の定式化には 2 つの問題点が存在する．一つは，定式化に不備があるため，現実の通信が表せていないことである．もう一つは，近似関数 $\gamma$ の出力によっては，検証を行った際に，検出漏れが起こりうることである．

4.1.1 節，4.1.2 節ではそれぞれの問題点が起こる場合について解説する．

### 4.1.1　定式化の不備

従来手法では，$R^*(\mathrm{L}(\mathrm{A}_0))$ が現実の通信を表すと主張している．しかし，$R^*(\mathrm{L}(\mathrm{A}_0))$ は攻撃者が介入していない通信は表せているが，攻撃者を含めた通信は表せていない．そこで，本節では [GK00] の項書き換え系で表せない通信の例を述べる．ここでは，2.1.3 節に記述されている NSPK の中間者攻撃を例として，安全性が満たされない通信を [GK00] の項書き換え系で表現できるか確認する．

A，B を通信者，C を攻撃者とすると，2.1.3 節の通信は A-C 間通信と C(A に偽装)-B 間通信が並行に動作している．[GK00] の主張が正しいならば，項 goal(agt(A), agt(0)) ⊔ goal(agt(0), agt(B)) を起点として，項書き換え系 $R$ の書き換え規則を繰り返し適用することで，安全性を満たさない通信を表せる．この通信を表現できるか確かめるために，書き換えの系列を示す．下記の書き換え系列のうち，下線部が書き換えによって得られた項である．

goal(agt(A), agt(0)) ⊔ goal(agt(0), agt(B))
$\rightarrow_R^*$ goal(agt(A), agt(0)) ⊔ goal(agt(0), agt(B)) ⊔
mesg{agt(A), agt(0), encr[pubkey(agt(0)), agt(A), cons(N(agt(A), agt(0)), cons(agt(A), null))]}
$\rightarrow_R^*$ goal(agt(A), agt(0)) ⊔ goal(agt(C), agt(B)) ⊔
mesg{agt(A), agt(0), encr[pubkey(agt(0)), agt(A), cons(N(agt(A), agt(0)), cons(agt(A), null))]}
⊔ N(agt(A), agt(0)) ⊔ agt(A)
$\rightarrow_R^*$ goal(agt(A), agt(0)) ⊔ goal(agt(0), agt(B)) ⊔
mesg{agt(A), agt(0), encr[pubkey(agt(0)), agt(A), cons(N(agt(A), agt(0)), cons(agt(A), null))]}
⊔ N(agt(A), agt(0)) ⊔ agt(A) ⊔
mesg{agt(0), agt(B), encr[pubkey(agt(B)), agt(0), cons(N(agt(0), agt(B)), cons(agt(0), null))]}

上記の書き換えの 1〜6 行目は，2.1.3 節の攻撃例に記述されている通信 1 を表す．

7行目以降は通信 $1'$. を表す．ここで作成された項が意味するのは，“A-B 間のノンスと A を表す情報を，B の公開鍵で暗号化して送信”である．しかし，$1'$. は，“A-C 間のノンスと A を表す情報を，B の公開鍵で暗号化して送信”である．以上より，項書き換え系に従うと，実際の通信と違うメッセージが表現される場合がある．

### 4.1.2 検出漏れ

[GK00] の検証法では，書き換え規則 $l \to r$ の右辺 $r$ に対応する項は左辺 $l$ に対応する項と同じ状態に到達するように木オートマトンの規則を追加すると定義している．しかし，右辺 $r$ の部分項については何の状態を割り当てるか正確な定義がない．右辺 $r$ の部分項に不適切な状態を割り当てると，木オートマトンの規則を追加しても検証に必要な項が受理状態まで到達しない場合がある．そこで，攻撃者が取得した情報が受理状態に含まれない例を述べる．

以下のプロトコルを対象として検証を行う．

1. $\mathrm{A} \to \mathrm{B} \quad : \quad \mathrm{N_A}$

このプロトコルは，2 者間の通信で暗号化せずにノンスを送信する，という動作を記述する．1. の通信を行うと攻撃者にノンス $\mathrm{N_A}$ を盗聴されるため，機密性を満たさない．次に，このプロトコルを検証するための入力を述べる．まず，プロトコルの動作を表す書き換え規則は次の定義である．

$$\mathsf{goal}(\mathsf{agt}(a), \mathsf{agt}(b)) \to \mathsf{goal}(\mathsf{agt}(a), \mathsf{agt}(b)) \sqcup \mathsf{mesg}(\mathsf{agt}(a), \mathsf{agt}(b), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(a), \mathsf{agt}(b)), \mathsf{null}))$$

攻撃者の動作を表した書き換え規則，攻撃者の取得情報を表す木オートマトン $\mathrm{A_0}$ は定義 3.1.4 と同じ木オートマトンを用いる．最後に，近似関数 $\gamma$ のうち，今回の例に必要な部分のみ，次に述べる．

$$\gamma(\mathsf{goal}(a, b) \to \mathsf{goal}(a, b) \sqcup \mathsf{mesg}(a, b, \mathsf{cons}(\mathsf{N}(a, b), \mathsf{null})), \mathrm{q_{net}}, \{a \mapsto \mathrm{q_{agtA}}, b \mapsto \mathrm{q_{agtB}}\}) = \mathrm{q_{net}q_m q_c q_N}$$

$$\gamma(\mathsf{cons}(x, y) \sqcup z \to (\mathsf{cons}(x, y) \sqcup z) \sqcup a, \mathrm{q_{net}}, \{a \mapsto \mathrm{q_N}, y \to \mathrm{q_{net}}, z \mapsto \mathrm{q_{net}}\}) = \mathrm{q_{net}q_N}$$

以上を用いて，検証を行う．NSPK で起こる脆弱性を検出するためには，書き換え規則を用いた後に，$\mathrm{N}(\mathsf{agt}(a), \mathsf{agt}(b))$ が受理状態に到達するように木オートマトンの規則を追加する必要がある．

実際に，この入力について，木オートマトン $\mathrm{A_0}$ に規則を追加する．次の書き換えでは，A-B 間通信でノンス $\mathrm{N_A}$ が攻撃者に取得された場合を表す．

1. $\mathsf{goal}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B})) \to_R \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B}), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B})), \mathsf{null}))$

書き換えの左辺が $\mathrm{A_0}$ で受理されることから，項 $\mathsf{goal}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B})) \sqcup \mathsf{mesg}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B}), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B})), \mathsf{null}))$ が受理状態に到達するように規則を追加する．この項に対応する近似関数 $\gamma$ から，以下の規則を木オートマトン $\mathrm{A_0}$ に追加する．

$$\mathsf{N}(\mathrm{q_{agtA}}, \mathrm{q_{agtB}}) \to \mathrm{q_N}, \mathsf{cons}(\mathrm{q_N}, \mathrm{q_{net}}) \to \mathrm{q_c}, \mathsf{mesg}(\mathrm{q_{agtA}}, \mathrm{q_{agtB}}, \mathrm{q_c}) \to \mathrm{q_m}, \mathrm{q_{net}} \sqcup \mathrm{q_m} \to \mathrm{q_{net}}$$

以上の規則を木オートマトン $\mathrm{A_0}$ の規則に追加した木オートマトンを $\mathrm{A_1}$ と定義する．

2. goal(agt(A), agt(B)) ⊔ mesg(agt(A), agt(B), cons(N(agt(A), agt(B)), null))
$\rightarrow_R$ LHS ⊔ cons(N(agt(A), agt(B)), null)

1. の書き換えから，この書き換えの左辺も受理状態に到達する．
よって，以下の規則を木オートマトン $A_1$ に追加する．

$q_{net} \sqcup q_c \rightarrow q_{net}$

以上の規則を木オートマトン $A_1$ の規則に追加した木オートマトンを $A_2$ と定義する．

3. goal(agt(A), agt(B))⊔mesg(agt(A), agt(B), cons(N(agt(A), agt(B)), null))⊔cons(N(agt(A), agt(B)), null)
$\rightarrow_R$ LHS ⊔ N(agt(A), agt(B))

2. の書き換えから，この書き換えの左辺も受理状態に到達する．よって，以下の規則を木オートマトン $A_2$ に追加する．

$q_{net} \sqcup q_N \rightarrow q_{net}$

以上の規則を木オートマトン $A_2$ の規則に追加した木オートマトンを $A_3$ と定義する．3. の書き換えを行った時点で，⊔ に関する書き換え規則以外は適用できなくなるため，これ以上木オートマトンの規則は追加されない．この時点で，攻撃者が A-B 間のノンス N(agt(A), agt(B)) を取得したことを表しているので，検証を正しく行うためには，$N(agt(A), agt(B)) \rightarrow^*_{A_3} q_{net}$ である必要がある．しかし，木オートマトン $A_3$ の受理言語には，項 N(agt(A), agt(B)) は含まれないため，$N(agt(A), agt(B)) \not\rightarrow^*_{A_3} q_{net}$ である．

[GK00] の手法では近似関数 $\gamma$ の値に依存して，木オートマトンに規則を追加していく．[GK00] の手法で攻撃者の取得情報を計算すると，以上のように，近似関数 $\gamma$ の値によっては，攻撃者の取得情報を表す項が木オートマトンの受理状態に到達しない．すなわち，[GK00] の手法で検証を行うと，攻撃者の取得情報が正しく計算できない場合があるため，検出漏れが起こりうる．

## 4.2 問題点の改善

本論文では，4.1 節で示した問題点を解消するために，新たに書き換え規則を追加した．それぞれの問題点に対応した書き換え規則と，解説をこの節で述べる．

### 4.2.1 定式化の改変

4.1.1 節で [GK00] の項書き換え系だけでは，現実の通信を表現できないことを明らかにした．[GK00] の項書き換え系は，プロトコルの動作，攻撃者の動作を表す．攻撃者の動作で表しているのは，復号化と取得メッセージの分解だけで，偽装メッセージの作成を表現していないため，現実の通信を表せない．

そこで，改善案として [GK00] の項書き換え系に，攻撃者の取得情報を利用したメッセージ生成を表す書き換え規則を追加する．攻撃者の取得情報には，エンティティ，ノンス，といった基本的

な情報に加え，それらを暗号化した情報も含まれる．これらの取得情報からメッセージを作成する場合，あるメッセージ作成に必要な情報は，複数のパターンが存在する．例えば，メッセージ $\{\mathrm{N_A}, \{\mathrm{N_B}\}\mathrm{K_B}\}\mathrm{K_B}$ を，攻撃者が送信する場合を考える．この時，メッセージ $\{\mathrm{N_A}, \{\mathrm{N_B}\}\mathrm{K_B}\}\mathrm{K_B}$ の作成に必要な情報は，次の 3 パターンである．

1. ノンス $\mathrm{N_A}$, $\mathrm{N_B}$，公開鍵 $\mathrm{K_B}$
   攻撃者がメッセージ作成に必要な情報を全て所持している．
2. ノンス $\mathrm{N_A}$，暗号化情報 $\{\mathrm{N_B}\}\mathrm{K_B}$
   攻撃者がメッセージ作成に必要な情報を一部所持し，残りを盗聴で得た暗号化情報で補う．
3. 暗号化情報 $\{\mathrm{N_A}, \{\mathrm{N_B}\}\mathrm{K_B}\}\mathrm{K_B}$
   まったく同じ形のメッセージを他の通信から盗聴して送信する．

このように，攻撃者は暗号化情報を利用して偽装メッセージを作成できる．これらのパターンを網羅できるように，書き換え規則を定義する．以降の定義では，変数記号の集合 $\mathrm{VAR}' = \{v_i | i \in \mathbb{N}\} \cup \{v_z\}$, $\mathrm{VAR} \cap \mathrm{VAR}' = \emptyset$ である．まず，書き換え規則の定義のために，準備を行う．[GK00] で定義されている項書き換え系を $R$ として，プロトコルに従ったメッセージを表す項の集合を次に定義する．

**定義 4.2.1 (プロトコルの送信メッセージ)**
プロトコルに従ったメッセージを表す項の集合を $\mathrm{T_{pm}}$ と記述する．$\mathrm{T_{pm}}$ は次の集合である．

$$\mathrm{T_{pm}} \quad = \quad \{l \mid l \to r \in R, l \notin \mathrm{GOAL}'\}$$

また，項 $t$ に関数記号 encr が出現する回数を $\mathrm{enc\_num}(t)$ と定義する．これを利用して，攻撃者が項 $t$ を作成する際に，可能な暗号化回数の集合 $\mathrm{Enc\_num}(t)$ を $\mathrm{Enc\_num}(t) = \{n \mid n \in \mathbb{N}, n \leq \mathrm{enc\_num}(t)\}$ と定義する．
次に，項集合の演算 times を定義する．この演算は，書き換え規則の左辺の集合を計算する際に用いる．

**定義 4.2.2 (項集合の演算)**
$\mathrm{times} : \mathcal{P}(\mathrm{T_{Gv}}) \times \mathcal{P}(\mathrm{T_{Gv}}) \to \mathcal{P}(\mathrm{T_{Gv}})$

$$\mathrm{times}(\mathrm{S_1}, \mathrm{S_2}) \quad = \quad \{x \sqcup y \mid x \in \mathrm{S_1}, y \in \mathrm{S_2}\}$$

次に，書き換え規則の生成に用いる関数を定義する．

**定義 4.2.3 (関数 elem)**

下記の定義において，$\mathrm{E}_x(i) = \{n \mid n \in \mathrm{Enc\_num}(x),\ n \leq i\}$，$\mathrm{E}_y(i,\ i') = \{n \mid n \in \mathrm{Enc\_num}(y)$ , $n \leq i - i'\}$ と定義する．

$\mathrm{elem} : \mathrm{T_{Gv}} \times \mathbb{N} \times \mathbb{N} \to \mathcal{P}(\mathrm{T_{Gv}})$

$$
\mathrm{elem}(m,\ i,\ j) = \begin{cases}
\mathrm{times}(\{a \sqcup b\},\ \mathrm{elem}(m',\ i,\ j)) & (m = \mathsf{mesg}(a,\ b,\ m')) \\
\mathrm{times}(\{k\},\ \mathrm{elem}(m',\ i-1,\ j)) & (m = \mathsf{encr}(k,\ a,\ m') \text{ かつ } i > 0) \\
\{\mathsf{encr}(k,\ v_j,\ m'') \mid m'' \in \mathrm{elem}(m',\ i,\ j+1)\} & (m = \mathsf{encr}(k,\ a,\ m') \text{ かつ } i = 0) \\
\bigcup\limits_{i' \in \mathrm{E_x}(i)} \mathrm{times}\Big(\mathrm{elem}(x,\ i',\ j),\ \bigcup\limits_{j \in \mathrm{E_y}(i,\ i')} \mathrm{elem}(y,\ j',\ j + \mathrm{enc\_num}(x))\Big) & (m = \mathsf{cons}(x,\ y) \text{ または } m = x \sqcup y) \\
\{m\} & (\text{上記以外})
\end{cases}
$$

関数 elem は書き換え規則の左辺の集合を計算する関数である．第 1 引数は作成するメッセージを表す項，第 2 引数はメッセージ作成のために攻撃者が暗号化する回数を表す．第 3 引数は，入力の部分項に項 encr が複数存在した場合に，項 encr の第 2 引数の変数記号を重複させないために利用する．

4.2.1 節で述べた通り，暗号化情報を含むメッセージ作成のために必要な情報のパターンは複数存在する．この関数では，攻撃者がメッセージ $m$ を $i$ 回の暗号化で作成する時に，必要な情報の全てのパターンを表す．暗号化回数 $i$ が 0 回になった時に，項 encr を発見した場合は，項 encr の第 2 引数を変数集合 $\mathrm{VAR}'$ に含まれる変数に置き換える．これは，盗聴で得た暗号化情報を利用したメッセージ作成を表すためである．

**(関数 elem の使用例)**

入力がユーザ A からユーザ B へのメッセージ $\{\mathrm{N_A}\}\mathrm{K_B}$ を表す項の場合，値は送信者 A，受信者 B，公開鍵 $\mathrm{K_B}$，ノンス $\mathrm{N_A}$ の集合を表す項である．実際の値を次に示す．

$\mathrm{elem}($ $\mathsf{mesg}\{\mathsf{agt}(a),\ \mathsf{agt}(b),\ \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(b)),\ \mathsf{agt}(a),\ \mathsf{cons}(\mathsf{N}(\mathsf{agt}(a),\ \mathsf{agt}(b)),\ \mathsf{null})]\}$ , 1, 0)

$= \{\mathsf{agt}(a) \sqcup \mathsf{agt}(b) \sqcup \mathsf{pubkey}(\mathsf{agt}(b)) \sqcup \mathsf{N}(\mathsf{agt}(a),\ \mathsf{agt}(b)) \sqcup \mathsf{null}\}$

$\mathrm{elem}($ $\mathsf{mesg}\{\mathsf{agt}(a),\ \mathsf{agt}(b),\ \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(b)),\ \mathsf{agt}(a),\ \mathsf{cons}(\mathsf{N}(\mathsf{agt}(a),\ \mathsf{agt}(b)),\ \mathsf{null})]\}$ , 0, 0)

$= \{\mathsf{agt}(a) \sqcup \mathsf{agt}(b) \sqcup \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(b)),\ v_0,\ \mathsf{cons}(\mathsf{N}(\mathsf{agt}(a),\ \mathsf{agt}(b)),\ \mathsf{null})]\}$

**定義 4.2.4 (関数 same)**

真偽値の集合を $\mathrm{Bool} = \{\mathrm{true},\ \mathrm{false}\}$ と定める．

また，引数をもたない項と変数の集合を BASE = ENEMY ∪ {A, B} ∪ VAR と定める．

same : $T_{Gv} \times T_{Gv} \rightarrow$ Bool

$$\text{same}(t, t') = \begin{cases} \bigwedge_{i=1}^{n} \text{same}(t, t') & (t = f(t_1, \ldots, t_n) \text{ かつ } t' = f(t'_1, \ldots, t'_n)) \\ \text{true} & (t, t' \in \text{BASE}, t = t', \text{ または} \\ & \ t \in \text{VAR} \cup \text{VAR}', t' \in \text{VAR}', \text{ または} \\ & \ t \in \text{VAR}', t' \in \text{VAR} \cup \text{VAR}') \\ \text{false} & (\text{それ以外}) \end{cases}$$

関数 same は変数を持つ項の等価性を判定する関数である．項 encr の第 2 引数が変数集合 VAR の要素である場合，その暗号化情報は任意の通信者が作成した情報を表す．そのため，比較する 2 つの項が項 encr であり，さらに項 encr の第 2 引数が変数である場合，任意の $v_i \in \text{VAR}'$ は任意の $x \in \text{VAR}$ と同じである，と判定する．

**定義 4.2.5 (関数 trans)**

trans : $T_{Gv} \times T_{Gv} \rightarrow T_{Gv}$

$$\text{trans}(l, m) = \begin{cases} \text{mesg}(a, b, \text{trans}(l, m')) & (m = \text{mesg}(a, b, m')) \\ \text{encr}(k, v_z, \text{trans}(l, m')) & (m = \text{encr}(k, a, m') \text{ かつ} \\ & \ \exists p \in \text{Pos}(l), l|_p = \text{encr}(k, v_z, m'') \text{ かつ} \\ & \ \text{same}(m', m'') = \text{true}) \\ \text{encr}(k, 0, \text{trans}(l, m')) & (m = \text{encr}(k, a, m') \text{ かつ} \\ & \ \exists p \in \text{Pos}(l), l|_p = \text{encr}(k, a, m'') \text{ かつ} \\ & \ \text{same}(m', m'') = \text{false}) \\ \text{cons}(\text{trans}(l, t), \text{trans}(l, t')) & (m = \text{cons}(t, t')) \\ \text{trans}(l, t) \sqcup \text{trans}(l, t') & (m = t \sqcup t') \\ m & (\text{上記以外}) \end{cases}$$

関数 trans は書き換え規則の左辺 $l$ に対応した右辺を計算する関数である．第 1 引数は書き換え規則の左辺，第 2 引数は左辺の元となったメッセージが入力される．第 2 引数の部分項に，偽装したいユーザが作成した暗号化情報が存在し，かつ $l$ にその暗号化情報が存在しなかったら，それを攻撃者が暗号化したことに変更する．この処理を行うことで，攻撃者がユーザ $x$ に偽装して，メッセージを作成したことを表す．

また，第 2 引数の部分項の暗号化情報とまったく同じ項が，$l$ の部分項に存在した場合，攻撃者が暗号化情報そのものを用いてメッセージを作成したことを表す．そのため，その部分項を任意の通信者が暗号化した情報であることに変更する．

**(関数 trans の使用例)**

書き換え規則 $l \rightarrow r$ について，左辺 $l$ = agt($a$) ⊔ agt($b$) ⊔ pubkey(agt($b$)) ⊔ N(agt($a$), agt($b$)) ⊔ null とする．関数 trans を用いて左辺 $l$ に対応した右辺 $r$ を計算する．

trans( agt($a$) ⊔ agt($b$) ⊔ pubkey(agt($b$)) ⊔ N(agt($a$), agt($b$)) ⊔ null,
mesg{agt($a$), agt($b$), encr[pubkey(agt($b$)),agt($a$), cons(N(agt($a$), agt($b$)), null)]} )
= mesg{agt($a$), agt($b$), encr[pubkey(agt($b$)),agt(0), cons(N(agt($a$), agt($b$)), null)]}

**定義 4.2.6 (関数 any)**

$\mathrm{any} : \mathrm{T_{Gv}} \times \mathrm{VAR} \times \mathrm{VAR} \rightarrow \mathrm{T_{Gv}}$

$$\mathrm{any}(t,\ a,\ b) = \begin{cases} \mathsf{mesg}(a',\ b',\ \mathrm{any}(t',\ a,\ b)) & (t = \mathsf{mesg}(a',\ b',\ t')) \\ \mathsf{encr}(k,\ a',\ \mathrm{any}(t',\ a,\ b)) & (t = \mathsf{encr}(k,\ a',\ t')) \\ \mathsf{cons}(\mathrm{any}(t_1,\ a,\ b),\ \mathrm{any}(t_2,\ a,\ b)) & (t = \mathsf{cons}(t_1,\ t_2)) \\ \mathrm{any}(t_1,\ a,\ b) \sqcup \mathrm{any}(t_2,\ a,\ b) & (t = t_1 \sqcup t_2) \\ \mathsf{N}(v_x,\ v_y) & (t = \mathsf{N}(a,\ b)) \\ t & (\text{上記以外}) \end{cases}$$

関数 any は前述の関数 elem，関数 trans で計算された，書き換え規則の左辺または右辺を第 1 引数に入力し，ノンスを表す項 N の引数を変換する関数である．

この関数の第 1 引数は項，第 2 引数はノンスの作成者，第 3 引数はノンスを受け取るユーザを入力する．攻撃者が偽装メッセージを作成する場合，メッセージの送信者が作成するノンスを，別のノンスで代用する場合がある．そのため，攻撃者の規則では，メッセージの送信者が作成するノンスは，任意のノンスを表現できる項でなければならない．このような任意のノンスを表すためにこの関数を用いる．

**(関数 any の使用例)**

A から B へのメッセージ $\mathrm{N_A}$ を攻撃者が A に偽装して作成する場合，攻撃者はノンス $\mathrm{N_A}$ を，任意のノンスで代用可能である．これを書き換え規則で表すために，次のように関数 any を用いる．書き換え規則 $\mathsf{agt}(a) \sqcup \mathsf{agt}(b) \sqcup \mathsf{N}(\mathsf{agt}(a), \mathsf{agt}(b)) \sqcup \mathsf{null} \rightarrow \mathsf{mesg}\{\mathsf{agt(a)}, \mathsf{agt(b)}, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(a), \mathsf{agt}(b)), \mathsf{null})\}$ の左辺を $l$，右辺を $r$ とする．$l$, $r$ それぞれに関数 any を用いたとき，下線部が変換される．

$\mathrm{any}\ (l,\ a,\ b) = \mathsf{agt}(a) \sqcup \mathsf{agt}(b) \sqcup \underline{\mathsf{N}(v_x,\ v_y)} \sqcup \mathsf{null}$

$\mathrm{any}\ (r,\ a,\ b) = \mathsf{mesg}\{\mathsf{agt(a)}, \mathsf{agt(b)}, \mathsf{cons}(\underline{\mathsf{N}(v_x,\ v_y)}, \mathsf{null})\}$

以上の定義を用いて，攻撃者の取得情報を利用したメッセージ生成を表す項書き換え系 $R'$ を次に定義する．

**定義 4.2.7 (メッセージ作成の項書き換え系)**

$$R' = \{l \rightarrow l \sqcup r \mid \exists a,\ b \in \mathrm{AGT'},\ \exists m \in \mathrm{MESG'},\ \exists \mathsf{mesg}(a,\ b,\ m) \in \mathrm{T_{pm}},$$
$$\exists i \in \mathrm{Enc\_Num}(\mathsf{mesg}(a,\ b,\ m)),$$
$$(l,\ r) \in \bigcup_{l' \in \mathrm{elem}(\mathsf{mesg}(a,\ b,\ m),\ i,\ 0)} (\mathrm{any}(l',\ a,\ b)\ ,\ \mathrm{any}(\mathrm{trans}(a,\ l',\ \mathsf{mesg}(a,\ b,\ m)),\ a,\ b))\}$$

プロトコルに従ったメッセージを表す項を関数 elem に与えることで，メッセージ作成に必要な情報の全パターンを計算する．その結果を関数 any に与えることで，特定のノンスを任意のノンスに置き換えて書き換え規則の左辺 $l$ を計算する．右辺 $r$ は，関数 elem の結果を関数 trans に与えることで，どのメッセージを攻撃者が作成したのかを変換し，さらに関数 any に関数 trans の値を与えることで計算される．この処理で作成される書き換え規則の例は 4.3 節に述べる．

### 4.2.2 検出漏れの対処

4.1.2 節で述べた検出漏れが起きる原因は，近似関数 $\gamma$ の状態割り当て次第で，攻撃者の取得情報に受理状態が割り当てられない場合があるためである．すなわち，[GK00] の定式化では，入力に応じて検出漏れが起こりうる．[GK00] の定式化において，検出漏れが起こるのは攻撃者の取得情報を表す項が，木オートマトンの受理言語に加えられない場合があるためである．よって，攻撃者の取得情報を表す項から書き換えられる全ての項が，危険対の関係になるように書き換え規則を追加すればよい．そこで，下記の書き換え規則の追加が必要であると考えた．

$x \sqcup y \to x$

この書き換え規則を追加すると，$\sqcup$ で繋がった項，すなわち，攻撃者の取得情報を表す項は全て受理状態に到達できるように，木オートマトンの規則が追加されるので，検出漏れを防ぐことができる．

ここで，この書き換え規則の追加により，誤検出の可能性も疑われる．すなわち，この規則の追加によって，復号化できない情報まで受理状態に到達するかもしれない．この規則を追加したことで誤検出が起こりえるのは，暗号化メッセージを表す項 $\mathsf{encr}(k, a, m)$ の部分項 $m$ が，書き換え規則 $x \sqcup y \to x$ によって，書き換え可能になる場合である．その場合，部分項 $m$ の部分項が受理状態に到達する．これは，復号化されていない情報まで，攻撃者の取得情報に加えられていることを表す．つまり，項 $\mathsf{encr}(k, a, m)$ の部分項 $m$ が書き換えれないならば，規則追加による誤検出は起こらないと考えられる．定義から部分項 $m \in \mathrm{CONS}$ なので，今回追加した規則の集合 $R'$ を $R' = \{x \sqcup y \to x\}$ と定義した時，命題 $\mathrm{CONS} \subseteq \mathrm{NF}(R')$ が真であればよい．

**補題 4.2.1** ($\mathrm{CONS} \subseteq \mathrm{NF}(R')$)

**証明.** 任意の項 $m \in \mathrm{CONS}$ が $R'$ の規則では書き換えられないことを証明する．

CONS の定義から，$m$ は再帰的に定義されているので，$m$ 中の関数記号 cons の出現回数 $n$ に関する帰納法で証明する．

(1)$n = 0$ の時
$m$ 中に関数記号 cons が出現しない場合を考える．$m \in \mathrm{CONS}$ であることから，集合 CONS の定義を見ると，集合 CONS の任意の項は 1 個以上の関数記号 cons を含むことが分かる．すなわち，$m \notin \mathrm{CONS}$ となり，前提と矛盾する．よって $\mathrm{CONS} \subseteq \mathrm{NF}(R')$ が成り立つ．

(2)$n = k + 1$(但し, k $\geq 0$) の時
$m$ 中の cons の出現回数が $k$ の時，$\mathrm{CONS} \subseteq \mathrm{NF}(R')$ であると仮定して，出現回数 $k + 1$ の場合を考える．すなわち，任意の $\mathsf{cons}(m', m) \in \mathrm{CONS}$ について $\mathsf{cons}(m', m) \in \mathrm{NF}(R')$ が成り立つことを証明する．帰納法の仮定より，$\mathsf{cons}(m', m)$ の部分項 $m$ は $R'$ の規則で書き換えられない．また，集合 CONS の定義から，(1) の証明と同様に，部分項 $m'$ も書き換えられない．以上から，項 $\mathsf{cons}(m', m)$ は $R'$ の規則で書き換えられない．よって $n = k + 1$(但し, $k \geq 1$) の時，$\mathrm{CONS} \subseteq \mathrm{NF}(R')$ が成立する．以上 (1), (2) から $\mathrm{CONS} \subseteq \mathrm{NF}(R')$ が示された．

上記の証明より，書き換え規則 $x \sqcup y \to x$ の追加による誤検出は起こらないと言える．

## 4.3 提案手法の適用例

4.2 節で述べた書き換え規則を追加することで，問題点が解決したことを，2.1.1 節に挙げた通信で示す．この通信は 4.1.1 節で [GK00] の書き換え規則では，表現できない通信である．

例 3.1.1 の項書き換え系 $R_{\mathrm{N}}$ を用いると，提案手法では次の項書き換え系 $R_{\mathrm{add}}$ が計算される．

$R_{\mathrm{add}}$={ $\mathsf{agt}(x) \sqcup \mathsf{agt}(y) \sqcup \mathsf{pubkey}(\mathsf{agt}(y)) \sqcup n \sqcup \mathsf{agt}(x) \sqcup \mathsf{null}$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(x), \mathsf{agt}(y), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), 0, \mathsf{cons}(n, \mathsf{cons}(\mathsf{agt}(x), \mathsf{null})))$,
$\mathsf{agt}(y) \sqcup \mathsf{agt}(x) \sqcup \mathsf{pubkey}(\mathsf{agt}(x)) \sqcup \mathsf{N}(\mathsf{agt}(x), \mathsf{agt}(y)) \sqcup n \sqcup \mathsf{null}$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(y), \mathsf{agt}(x), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(x)), 0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(x), \mathsf{agt}(y)), \mathsf{cons}(n, \mathsf{null})))$,
$\mathsf{agt}(x) \sqcup \mathsf{agt}(y) \sqcup \mathsf{pubkey}(\mathsf{agt}(y)) \sqcup \mathsf{N}(\mathsf{agt}(y), \mathsf{agt}(x))) \sqcup \mathsf{null}$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(x), \mathsf{agt}(y), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), 0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(y), \mathsf{agt}(x))), \mathsf{null}))$
$\mathsf{agt}(x) \sqcup \mathsf{agt}(y) \sqcup \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), v_0, \mathsf{cons}(n, \mathsf{cons}(\mathsf{agt}(x), \mathsf{null}))$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(x), \mathsf{agt}(y), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), v_0, \mathsf{cons}(n, \mathsf{cons}(\mathsf{agt}(x), \mathsf{null})))$
$\mathsf{agt}(y) \sqcup \mathsf{agt}(x) \sqcup \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(x)), v_0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(x), \mathsf{agt}(y)), \mathsf{cons}(n, \mathsf{null}))$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(y), \mathsf{agt}(x), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(x)), v_0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(x), \mathsf{agt}(y)), \mathsf{cons}(n, \mathsf{null})))$
$\mathsf{agt}(x) \sqcup \mathsf{agt}(y) \sqcup \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), v_0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(y), \mathsf{agt}(x)), \mathsf{null}))$
$\rightarrow \mathrm{LHS} \sqcup \mathsf{mesg}(\mathsf{agt}(x), \mathsf{agt}(y), \mathsf{encr}(\mathsf{pubkey}(\mathsf{agt}(y)), v_0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(y), \mathsf{agt}(x)), \mathsf{null})))$}

項 $\mathsf{goal}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)) \sqcup \mathsf{agt}(\mathsf{A}) \sqcup \mathsf{agt}(\mathsf{B}) \sqcup \mathsf{pubkey}(\mathsf{agt}(\mathsf{B})) \sqcup \mathsf{null} \in \mathrm{L}(\mathrm{A}_0)$ を起点として，項書き換え系 $R$, $R_{\mathrm{add}}$, $\{x \sqcup y \rightarrow x\}$ の書き換え規則を繰り返し適用することで，2.1.1 の通信を表す．簡略化のため，書き換え系列の $i$ 番目に記述している項を $\mathrm{LHS}_i$ と表記する．

**(書き換え系列)**

1. $\mathsf{goal}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)) \sqcup \mathsf{agt}(\mathsf{A}) \sqcup \mathsf{agt}(\mathsf{B}) \sqcup \mathsf{pubkey}(\mathsf{agt}(\mathsf{B})) \sqcup \mathsf{null}$
2. $\rightarrow_R^*$ $\mathrm{LHS}_1$
   $\sqcup \mathsf{mesg}\{\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0), \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(0)), \mathsf{agt}(\mathsf{A}), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)), \mathsf{cons}(\mathsf{agt}(\mathsf{A}), \mathsf{null}))]$
3. $\rightarrow_R^*$ $\mathrm{LHS}_2 \sqcup \mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)) \sqcup \mathsf{agt}(\mathsf{A})$
4. $\rightarrow_{R_{\mathrm{add}}}^*$ $\mathrm{LHS}_3$
   $\sqcup \mathsf{mesg}\{\mathsf{agt}(\mathsf{A}), \mathsf{agt}(\mathsf{B}), \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(\mathsf{B})), 0, \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)), \mathsf{cons}(\mathsf{agt}(\mathsf{A}), \mathsf{null}))]$
5. $\rightarrow_R^*$ $\mathrm{LHS}_4$
   $\sqcup \mathsf{mesg}\{\mathsf{agt}(\mathsf{B}), \mathsf{agt}(\mathsf{A}), \mathsf{encr}[\mathsf{pubkey}(\mathsf{agt}(\mathsf{A})), \mathsf{agt}(\mathsf{B}), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{A}), \mathsf{agt}(0)), \mathsf{cons}(\mathsf{N}(\mathsf{agt}(\mathsf{B}), \mathsf{agt}(\mathsf{A})), \mathsf{null}))]$

6. $\rightarrow^*_{R_{\mathrm{add}}}$ $\mathrm{LHS}_5$
   $\sqcup$ mesg{agt(0), agt(A),
   encr[pubkey(agt(A)), agt(B), cons(N(agt(A), agt(0)),
   cons(N(agt(B), agt(A)), null))]
7. $\rightarrow^*_R$ $\mathrm{LHS}_6$
   $\sqcup$ mesg{agt(A), agt(0),
   encr[pubkey(agt(0)), agt(A), cons(N(agt(B), agt(A)), null))]
   $\sqcup$ c_init(agt(A), agt(0), agt(B))
8. $\rightarrow^*_R$ $\mathrm{LHS}_7$ $\sqcup$ c_resp(agt(0), agt(A), agt(A)) $\sqcup$ N(agt(B), agt(A))
9. $\rightarrow^*_R$ $\mathrm{LHS}_8$
   $\sqcup$ mesg{agt(A), agt(B),
   encr[pubkey(agt(B)), agt(0), cons(N(agt(B), agt(A)), null))]
10. $\rightarrow^*_R$ $\mathrm{LHS}_9$ $\sqcup$ c_resp(agt(B), agt(A), agt(0))

1. ~3. までは 4.1.1 節の例と同様である．4.1.1 節では，この時点で適用できる書き換え規則がなくなった．しかし，今回提案した関数から計算された書き換え規則を適用すると，実際の通信を表す書き換え系列が計算できる．以上から，項書き換え系 $R_{\mathrm{add}}$ の書き換え規則を従来の項書き換え系に追加すると，上記の系列中の 4 や 7 のように，取得情報を用いた偽装メッセージ作成が書き換えにより表現できる．

# 第5章　結論

といった既存のプロトコル検証法では，状態爆発問題が起こりうることから，通信者数や通信量が膨大な場合の検証が困難であると考えられる．そこで，無限の通信者数や通信量に対応した [GK00] の検証法が提案されたが，本研究より，この手法には問題点があることが判明した．そこで本論文では，[GK00] の問題点を修正したプロトコル検証法を提案した．

[GK00] の手法には，問題点が 2 つ存在する．まず，攻撃者が偽装メッセージを送信する動作が表現されていなかったため，攻撃者を交えた通信が表現できなかった．そこで，入力で与えられる，プロトコルの動作を表す項書き換え系を利用して，偽装メッセージ作成を表す書き換え規則を計算する関数を定義した．次に，入力で与える近似関数 $\gamma$ の値によっては，検出漏れが起こる可能性があることである．これについて，攻撃者が取得した情報を表す項は，近似関数 $\gamma$ の値に関わらず，全て近似オートマトン $\mathrm{T}_R\uparrow(\mathrm{A}_0)$ の受理状態に到達するように，書き換え規則を追加した．

# 謝辞

本論文を作成するにあたり，研究の議論，発表指導，他にも数多くの御指導を熱心にしてくださった山田俊行講師，講義等様々な場で御指導いただいた大山口通夫教授，重要な連絡，各種手続きをしていただいた落合美子事務員に心より感謝申し上げます．

また，研究や資料作成について様々な指摘をしてくださった座礼晃一氏に感謝の意を申し上げます．そして，研究に関して相談にのっていただき，さらに数々の御指摘をしてくださった研究室の諸氏に深く感謝いたします．

# 参考文献


[BN99] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge Univercity Press, 1999.

[CDG+05] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. 2005. `http://www.grappa.univ-lille3.fr/tata`.

[DEK82] D. Dolev, S. Even, and R.M. Karp. Security of Ping-Pong Protocols. *Information and Control*, Vol. 55, No. 1, pp. 57–68, 1982.

[DY81] D. Dolevt and A.C. Yao. On the Security of Public Key Protocols·. In *the Proceedings of Foundations of Computer Science 1981 (FOCS1981)*, pp. 350–357, 1981.

[GK00] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *the Proceedings of Conference on Automated Deduction 2000 (CADE2000)*, Lecture Notes in Artificial Intelligence 1831(LNAI 1831), pp. 271–290. Springer, 2000.

[Sch02] S. Schneider. Verifying Authentication Protocol Implementations. In *the Proceedings of Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2002)*, Vol. 209, pp. 5–24, 2002.