セキュリティプロトコルの略式記法からspi計算への変換

住井 英二郎[†] 立沢 秀晃[††] 米澤 明憲[††]

セキュリティプロトコルは往々にして非形式的な略式記法で定義されるが、この記法は明確な意味論がなく、プロトコルが正常に完了する場合のメッセージ列しか記述していない。そのためにプロトコルが誤って解釈されたり、形式的検証の妨げとなるおそれがある。

我々はこの問題に対処するべく、略式記法の構文を形式化し、Abadi と Gordon の spi 計算への半自動的な変換を定義する。spi 計算は、暗号プロトコルの形式的記述・検証においてもっとも成功している枠組みの一つである。変換の基本方針は、プロトコルの各時点における各参加者の**知識**(どのような鍵、ノンス、識別子を知っているか) にもとづき、各参加者の動作を明確化することである。

Translating Security Protocols from Informal Notation into Spi Calculus

Eijiro Sumii,[†] Hideaki Tatsuzawa[††] and Akinori Yonezawa[††]

Security protocols are often defined in informal notations which have no clear semantics and only describe the message sequence when the protocol is successfully completed. This can be a source of incorrect interpretations of protocols as well as an obstacle to their formal verification.

To address this problem, we formalize the syntax of the informal notations and present their semi-automatic translation into Abadi and Gordon's spi-calculus, one of the most successful frameworks for the formal specification and verification of cryptographic protocols. Our main idea is to make the actions of each principal explicit on the basis of its *knowledge*, i.e., what keys, nonces and names it knows at each point in a protocol.

1. 序 論

セキュリティプロトコルとは、暗号を使用したメッセージのやりとりにより、相手の認証や秘密の伝達といった目的を達成するためのアルゴリズムである。セキュリティプロトコルはしばしば以下のような外見の記法(**略式記法**ということにする)によって定義される。

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, B\}_{K_{AB}}$

これは ISO 対称鍵 2 パス一方向認証プロトコルという例である[9]。このプロトコルの目的は、**参加者** (principal) B が参加者 A の生存を確認することにある。そのために、B はまず一度だけ使用される適当な乱数である**ノンス** (nonce) N_B を新規に生成し、A に送信する。A はそれを受信し、B の名前と一緒に共有鍵 K_{AB} で暗号化して、B に返信する。B はそれを K_{AB} で復号化し、受信したノンスが以前に生成した N_B と一致していること、また、受信した名前が B であることを確認する。

しかし、以上のような動作を前述の略式記法だけから理解する方法は自明でない。というのは、略式記法では

- まず、B が A にノンス N_B を送る
- 次に、A は B に暗号文 $\{N_B, B\}_{K_{AB}}$ を送る

という動作しか明示されておらず、

- 1. において、B は N_B を新規に生成する
- 2. において、B は受信した N_B と B が本当に N_B と B であることを確認する

という動作を推測しなければならないためである。このような問題があるために、略式記法は一見すると簡潔でわかりやすいにもかかわらず、プロトコルを記述するための言語として実際の理論やシステムでは(非形式的な説明以外には)あまり使用されてこなかった。

けれども十分な経験のある人間ならば、自然言語に

† ペンシルバニア大学コンピュータ情報科学科
Department of Computer and Information Science, University of Pennsylvania
†† 東京大学情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo

よる記述がなくても略式記法だけからプロトコルの動作を推測することが可能である。これはまったくの直観だけによるわけではなく、ある程度の規則が存在すると想像される。たとえば、先の例の 2. において B が N_B と B を確認するのは、それらの値をすでに知っていて、特定のメッセージを受信することを期待しているためである。同様に、1. において B が N_B を新規に生成するのは、その値を知らない（まだ存在しない）のに送信することになっているからである。このように値の新規生成や確認といった、送信・受信に付随する暗黙の動作は、参加者がその値を知っているかどうかによって決まるように思われる。

以上のような考察にもとづき、本稿では参加者の**知識**(knowledge)にもとづいて略式記法の意味論を定義する方法を提案する。鍵となるアイディアは次の 4 つである。

- 知識にない値を送信するときは、新規に生成して、知識に追加する。
- 知識にある値を受信したときは、受信した値が知識にある値と同一であることを確認する。
- 知識にない値を受信したときは、知識に追加する。
- 鍵を知っている暗号文を受信したときは、その暗号文を復号化し、平文を受信したものとみなす。

ただし、各参加者の初期知識は外部からの入力として所与であると仮定する。

たとえば、先の例で A と B の初期知識をいずれも $\{A, B, K_{AB}\}$ とする。1. において、B は N_B を知らないので、新規に生成して送信する。A も N_B を知らないので、受信したら知識に追加する。さらに 2. では、A は N_B, B, K_{AB} をいずれも知っているので、$\{N_B, B\}_{K_{AB}}$ を送信する。B は K_{AB} を知っているので、$\{N_B, B\}_{K_{AB}}$ を復号化し、さらに N_B と B も知っているので、知識にある N_B や B と同一であることを確認する。これらの動作は自然言語による記述とも合致している。

本稿では、略式記法の意味を定義するための表示言語として spi 計算[6),21)] を使用する。略式記法から spi 計算への変換を定義することにより、略式記法で記述したプロトコルについて、spi 計算における双模倣[5),7),8)] や型検査[1),3),11),12)] といった様々な解析・検証の手法を適用することが可能となる。

以降の構成は次の通りである。2 節では略式記法の抽象構文を、3 節では spi 計算の抽象構文を定義する。4 節では略式記法の意味を spi 計算への変換により定義する。5 節では、複数のクライアントに応答するサーバのように、相手の特定できないプロトコルを変換するための拡張について説明し、6 節では関連研究と今後の課題について議論する。

プロトコル	π	::=	$\alpha_1; \ldots; \alpha_n$
行動	α	::=	$X \to Y : M_1, \ldots, M_n$
メッセージ	M, N	::=	v
		|	$op(M_1, \ldots, M_n)$
		|	$\{M_1, \ldots, M_n\}_M$
		|	M^+
		|	M^-
変数	v	::=	$X_{Y_1 \ldots Y_n}$
原子記号	X, Y	::=	$A, B, S, x, y, K, N, \ldots$

図 1 略式記法の抽象構文
Fig. 1 Abstract Syntax of Informal Notation

2. 略式記法

我々の略式記法の抽象構文は図 1 の通りである。プロトコル π は 0 個以上の行動 α の列である。行動 $X \to Y : M_1, \ldots, M_n$ は、参加者 X から参加者 Y へメッセージ列 $M_1, \ldots, M_n$ を送信することを表現している。ただし、n は 0 以上の整数であり、以後も同様とする。メッセージ M は、変数 v、算術演算 $op(M_1, \ldots, M_n)$、鍵 M によるメッセージ列 $M_1, \ldots, M_n$ の暗号化 $\{M_1, \ldots, M_n\}_M$、メッセージ M にもとづく非対称鍵 M^+, M^- のいずれかである。直観としては、M^+ で暗号化されたメッセージは M^- によってのみ復号化でき、また、M^- で署名されたメッセージは M^+ によってのみ確認できる。変数 v は $X_{Y_1 \ldots Y_n}$ $(n \geq 0)$ という形をしており、たとえば A, K_{AB}, N_B などが含まれる。算術演算 $op(M_1, \ldots, M_n)$ としては、ハッシュ関数 $hash(M_1, \ldots, M_n)$ や整数定数 i、ノンスに 1 を加える $succ(M)$ などが考えられる。

3. spi 計算

本稿で使用する spi 計算の抽象構文を図 2 に示す。プロセス P は、

- 何もしないプロセス 0
- 値を新規に生成し、v に束縛してから P を実行するプロセス $\nu v.\, P$
- 通信チャンネル v に $M_1, \ldots, M_n$ を送信してから P を実行するプロセス $\overline{v}\langle M_1, \ldots, M_n\rangle.\, P$
- 通信チャンネル v からメッセージ列を受信し、$v_1, \ldots, v_n$ に束縛してから P を実行するプロセス $v(v_1, \ldots, v_n).\, P$
- P と Q を並列に実行するプロセス $P \mid Q$

プロセス P, Q ::=
- 0
- $\nu v.\, P$
- $\overline{v}\langle M_1, \ldots, M_n \rangle.\, P$
- $v(v_1, \ldots, v_n).\, P$
- $P \mid Q$
- $!P$
- $\texttt{if}\ M = N\ \texttt{then}\ P$
- $\texttt{case}\ M\ \texttt{of}\ \{v_1, \ldots, v_n\}_N\ \texttt{in}\ P$

図 2 spi 計算の抽象構文
Fig. 2 Abstract Syntax of spi-Calculus

- $P \mid P \mid P \mid \ldots$ に相当する無限の並列な複製のプロセス $!P$
- M と N が同一の場合だけ P を実行するプロセス $\texttt{if}\ M = N\ \texttt{then}\ P$
- 暗号文 M を鍵 N で復号化し、成功したら平文を $v_1, \ldots, v_n$ に束縛してから P を実行するプロセス $\texttt{case}\ M\ \texttt{of}\ \{v_1, \ldots, v_n\}_N\ \texttt{in}\ P$

のいずれかである。簡便のために、$P_1 \mid \ldots \mid P_n$ のような並列実行を $\prod_{1 \le i \le n} P_i$ のように表記する。また、$\widehat{M^+} = M^-$, $\widehat{M^-} = M^+$, $\widehat{M} = M$（それ以外の場合）と定義する。

プロセスの**文脈**(context) C とは、任意のプロセスにおいて一個の部分プロセスを**穴**(hole) [] でおきかえたものである。文脈 C の穴にプロセス P を代入したプロセスを $C[P]$ と表記する。たとえば $C = \nu x.\, \nu y.\, []$ は文脈であり、$C[\overline{z}\langle x, y \rangle.\, 0] = \nu x.\, \nu y.\, \overline{z}\langle x, y \rangle.\, 0$ である。

spi 計算の形式的な意味論についてはオリジナルの文献[6] を参照されたい。

4. 変　　換

略式記法のプロトコル π から spi 計算のプロセス P への変換 $P = \mathcal{T}(\pi)$ を図 3 のように定義する。ただし、I は各参加者から初期知識（メッセージの集合）への写像であり、外から与えられているものとする。また、条件が重複する定義は「上から下へ」適用し、下の定義はそれより上の定義が適用できなかった場合のみ適用できるものとする。

まず、プロトコルの全体 π は、各参加者 X を表現するプロセス $\mathcal{T}_X(\pi)$ の並列実行に変換される。$\mathcal{T}_X(\alpha_1; \ldots; \alpha_n)$ は、それぞれの行動 α_i を変換した文脈 C_i の合成に変換される。変換にあたっては、行動 α_i によって X の知識が ρ_{i-1} から ρ_i へ増加していくことに注意する。（一般に知識 ρ とは、略式記法におけるメッセージから、それに対応する spi 計算におけるメッセージへの部分写像である。）$\texttt{After}^{\sigma}_X$ は、プロトコルが終了した以降の X の動作を表現するプロセスであり、我々の変換では特定しない。

X の行動 α の変換 $\mathcal{T}_X(\rho, \alpha)$ は、X がメッセージを送信するか、受信するか、あるいはどちらもしないかによって場合わけされる。X がメッセージ $M_1, \ldots, M_n$ を送信する場合は、それぞれのメッセージ M_i について、知識にない値を新規に生成する文脈 C_i と、実際に送信するメッセージ N_i を、$\mathcal{S}_X(\rho_{i-1}, M_i)$ により計算していく（1 行目）。X がメッセージ $M_1, \ldots, M_n$ を受信する場合は、知識にある値と同一かどうか確認する文脈 C_i を、$\mathcal{R}_X(\rho_{i-1}, M_i)$ により計算していく（2 行目）。X が送信も受信もしない場合は、何もしない空の文脈 [] に変換される（3 行目）。

$\mathcal{S}_X(\rho, M)$ は、メッセージ M を知識 ρ から作成する。M 自体が ρ で定義されている場合は、それを使用する（1 行目）。そうでない場合は、まず M の部分式を再帰的に作成し、そこから M を作成する（2〜5 行目）。ただし、1 節で説明したように、知識にない変数は新規に生成する（6 行目）。

$\mathcal{R}_X(\rho, M)$ は、メッセージ M を知識 ρ から作成できる場合は、受信した x が M と同一であることを確認する（1 行目）。また、メッセージが暗号文 $\{M_1, \ldots M_n\}_M$ であり、M を ρ から作成できる場合は、復号化して再帰的に平文 $M_1, \ldots M_n$ を処理する（2 行目）。どちらでもない場合は、メッセージを知識に追加する（3 行目）。

$map_X(\rho, M)$ は、メッセージ M を知識 ρ から作成する。知識にない変数を新規に生成しない（したがって知識も増加しない）こと以外は、$\mathcal{S}_X(\rho, M_i)$ と同様である。

なお、$\mathcal{S}_X(\rho, M)$, $\mathcal{R}_X(\rho, M)$, $map_X(\rho, M)$ の X は図 3 の定義で使用されていないが、次節の拡張で必要となる。

例 1 1 節で例に挙げた ISO 対称鍵 2 パス一方向認証プロトコル

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, B\}_{K_{AB}}$

を初期知識 $I(A) = I(B) = \{A, B, K_{AB}\}$ で変換すると以下のようになる。

$$
\begin{array}{lcl}
\mathcal{T}(\pi) & = & \prod_{X \in dom(I)} \mathcal{T}_X(\pi) \\
\\
\mathcal{T}_X(\alpha_1;\ldots;\alpha_n) & = & C_1[\ldots[C_n[\mathtt{After}_X^{\sigma}]]] \\
 & & \text{if } \mathcal{T}_X(\rho_0,\alpha_1) = (\rho_1,C_1),\ldots,\mathcal{T}_X(\rho_{n-1},\alpha_n) = (\rho_n,C_n) \\
 & & \text{for } \rho_0 = \{M \mapsto M \mid M \in I(X)\} \\
 & & \text{and } \sigma = \rho_n \setminus \{M \mapsto M \mid \text{any } M\} \\
\\
\mathcal{T}_X(\rho, X \to Y : M_1,\ldots,M_n) & = & (\rho_n, C_1[\ldots[C_n[\overline{\mathtt{chan}_Y}\langle N_1,\ldots,N_n\rangle.\,[\,]]]]) \\
 & & \text{if } Y \neq X \\
 & & \text{and } \mathcal{S}_X(\rho,M_1) = (\rho_1,C_1,N_1),\ldots,\mathcal{S}_X(\rho_{n-1},M_n) = (\rho_n,C_n,N_n) \\
\mathcal{T}_X(\rho, Y \to X : M_1,\ldots,M_n) & = & (\rho_n, \mathtt{chan}_X(x_1,\ldots,x_n).\,C_1[\ldots[C_n[\,]]]) \\
 & & \text{if } Y \neq X,\ x_1,\ldots,x_n \text{ fresh}, \\
 & & \text{and } \mathcal{R}_X(\rho,x_1,M_1) = (\rho_1,C_1),\ldots,\mathcal{R}_X(\rho_{n-1},x_n,M_n) = (\rho_n,C_n) \\
\mathcal{T}_X(\rho, Y \to Z : M_1,\ldots,M_n) & = & (\rho,[\,]) \\
 & & \text{if } Y \neq X \text{ and } Z \neq X \\
\\
\mathcal{S}_X(\rho,M) & = & (\rho,[\,],\rho(M)) \\
 & & \text{if } M \in dom(\rho) \\
\mathcal{S}_X(\rho, op(M_1,\ldots,M_n)) & = & (\rho_n, C_1[\ldots[C_n[\,]]], op(N_1,\ldots,N_n)) \\
 & & \text{if } \mathcal{S}_X(\rho,M_1) = (\rho_1,C_1,N_1),\ldots,\mathcal{S}_X(\rho_{n-1},M_n) = (\rho_n,C_n,N_n) \\
\mathcal{S}_X(\rho, M^+) & = & (\rho',C[\,],N^+) \\
 & & \text{if } \mathcal{S}_X(\rho,M) = (\rho',C,N) \\
\mathcal{S}_X(\rho, M^-) & = & (\rho',C[\,],N^-) \\
 & & \text{if } \mathcal{S}_X(\rho,M) = (\rho',C,N) \\
\mathcal{S}_X(\rho, \{M_1,\ldots,M_n\}_M) & = & (\rho_n, C[C_1[\ldots[C_n[\,]]]], \{N_1,\ldots,N_n\}_N) \\
 & & \text{if } \mathcal{S}_X(\rho,M) = (\rho_0,C,N) \\
 & & \text{and } \mathcal{S}_X(\rho_0,M_1) = (\rho_1,C_1,N_1),\ldots,\mathcal{S}_X(\rho_{n-1},M_n) = (\rho_n,C_n,N_n) \\
\mathcal{S}_X(\rho, v) & = & ((\rho, v \mapsto v), \nu v.\,[\,], v) \\
\\
\mathcal{R}_X(\rho,x,M) & = & (\rho, C[\mathtt{if}\ x = N\ \mathtt{then}\ [\,]]) \\
 & & \text{if } map_X(\rho,M) = (C,N) \\
\mathcal{R}_X(\rho,x,\{M_1,\ldots,M_n\}_M) & = & (\rho_n, C[\mathtt{case}\ x\ \mathtt{of}\ \{y_1,\ldots,y_n\}_N\ \mathtt{in}\ C_1[\ldots[C_n[\,]]]]) \\
 & & \text{if } map_X(\rho,\widehat{M}) = (C,N) \\
 & & \text{and } \mathcal{R}_X(\rho,y_1,M_1) = (\rho_1,C_1),\ldots,\mathcal{R}_X(\rho_{n-1},y_n,M_n) = (\rho_n,C_n) \\
\mathcal{R}_X(\rho,x,M) & = & ((\rho_n, M \mapsto x),[\,]) \\
\\
map_X(\rho,M) & = & ([\,],\rho(M)) \\
 & & \text{if } M \in dom(\rho) \\
map_X(\rho, op(M_1,\ldots,M_n)) & = & (C_1[\ldots[C_n[\,]]], op(N_1,\ldots,N_n)) \\
 & & \text{if } map_X(\rho,M_1) = (C_1,N_1),\ldots,map_X(\rho,M_n) = (C_n,N_n) \\
map_X(\rho, M^+) & = & (C[\,],N^+) \\
 & & \text{if } map_X(\rho,M) = (C,N) \\
map_X(\rho, M^-) & = & (C[\,],N^-) \\
 & & \text{if } map_X(\rho,M) = (C,N) \\
map_X(\rho, \{M_1,\ldots,M_n\}_M) & = & (C_1[\ldots[C_n[C[\,]]]], \{N_1,\ldots,N_n\}_N) \\
 & & \text{if } map_X(\rho,M) = (C,N) \\
 & & \text{and } map_X(\rho,M_1) = (C_1,N_1),\ldots,map_X(\rho,M_n) = (C_n,N_n)
\end{array}
$$

図 3 略式記法から spi 計算への変換
Fig. 3 Translation from Informal Notation to spi-Calculus

$$
\begin{array}{ll}
 & \mathtt{chan}_A(x).\,\overline{\mathtt{chan}_B}\langle\{x,B\}_{K_{AB}}\rangle.\,\mathtt{After}_A^{N_B \mapsto x} \\
\mid & \nu N_B.\,\overline{\mathtt{chan}_A}\langle N_B\rangle. \\
 & \mathtt{chan}_B(y).\,\mathtt{case}\ y\ \mathtt{of}\ \{z_1,z_2\}_{K_{AB}}\ \mathtt{in} \\
 & \mathtt{if}\ z_1 = N_B\ \mathtt{then}\ \mathtt{if}\ z_2 = B\ \mathtt{then}\ \mathtt{After}_B^{\emptyset}
\end{array}
$$

このプロセスを簡約すると

$$\nu N_B.\,(\mathtt{After}_A^{N_B \mapsto N_B} \mid \mathtt{After}_B^{\emptyset})$$

になり、プロトコルが正常に実行されることがわかる。なお、通信チャンネル $\mathtt{chan}_A$ と $\mathtt{chan}_B$ が ν によって束縛されずオープンであることは、プロトコルの実行されるネットワークが公開されており盗聴や改竄が可能であるという通常の仮定[10] を反映している。

例 2 2 人の参加者 A, B が相互に認証することを目的とする Needham-Schroeder 公開鍵プロトコル[18]

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

を、初期知識 $I(A) = \{A, B, K_A^+, K_A^-, K_B^+\}$, $I(B) = \{A, B, K_A^+, K_B^+, K_B^-\}$ で変換すると以下のようになる。この変換結果は、人間の直観による通常の表現[2] と等価である☆。

$$
\begin{array}{l}
\nu N_A.\, \overline{\mathtt{chan}_B}\langle \{N_A, A\}_{K_B^+} \rangle. \\
\mathtt{chan}_A(y).\, \mathtt{case}\ y\ \mathtt{of}\ \{y_1, y_2\}_{K_A^-}\ \mathtt{in} \\
\mathtt{if}\ y_1 = N_A\ \mathtt{then} \\
\overline{\mathtt{chan}_B}\langle \{y_2\}_{K_B^+} \rangle.\, \mathtt{After}_A^{N_B \mapsto y_2} \\
\mid\ \mathtt{chan}_B(x).\, \mathtt{case}\ x\ \mathtt{of}\ \{x_1, x_2\}_{K_B^-}\ \mathtt{in} \\
\mathtt{if}\ x_2 = A\ \mathtt{then} \\
\nu N_B.\, \overline{\mathtt{chan}_A}\langle \{x_1, N_B\}_{K_A^+} \rangle. \\
\mathtt{chan}_B(z).\, \mathtt{case}\ z\ \mathtt{of}\ \{z_1\}_{K_B^-}\ \mathtt{in} \\
\mathtt{if}\ z_1 = N_B\ \mathtt{then}\ \mathtt{After}_B^{N_A \mapsto x_1}
\end{array}
$$

公開鍵による暗号化 $\{N_A, A\}_{K_B^+}$, $\{N_A, N_B\}_{K_A^+}$, $\{N_B\}_{K_B^+}$ とその復号化が変換される方法に注意されたい。このプロセスは簡約すると

$$\nu N_A.\, \nu N_B.\, (\mathtt{After}_A^{N_B \mapsto N_B} \mid \mathtt{After}_B^{N_A \mapsto N_A})$$

となる。なお、このプロトコルは使用の方法によっては欠陥があることで有名である[15]。これについては次節で議論する。

例 3 2 人の参加者 A, B が相互に認証し、さらに鍵 K'_{AB} とノンス N'_B を共有することを目的とする Andrew Secure RPC プロトコル[9]

1. $A \rightarrow B : A, \{N_A\}_{K_{AB}}$
2. $B \rightarrow A : \{succ(N_A), N_B\}_{K_{AB}}$
3. $A \rightarrow B : \{succ(N_B)\}_{K_{AB}}$
4. $B \rightarrow A : \{K'_{AB}, N'_B\}_{K_{AB}}$

を、初期知識 $I(A) = I(B) = \{K_{AB}, A, B\}$ で変換すると以下のようになる。

$$
\begin{array}{l}
\nu N_A.\, \overline{\mathtt{chan}_B}\langle A, \{N_A\}_{K_{AB}} \rangle. \\
\mathtt{chan}_A(y).\, \mathtt{case}\ y\ \mathtt{of}\ \{y_1, y_2\}_{K_{AB}}\ \mathtt{in} \\
\mathtt{if}\ y_1 = succ(N_A)\ \mathtt{then} \\
\overline{\mathtt{chan}_B}\langle \{succ(y_2)\}_{K_{AB}} \rangle. \\
C_A(w).\, \mathtt{case}\ w\ \mathtt{of}\ \{w_1, w_2\}_{K_{AB}}\ \mathtt{in} \\
\mathtt{After}_A^{N_B \mapsto y_2, K'_{AB} \mapsto w_1, N'_B \mapsto w_2} \\
\mid\ \mathtt{chan}_B(x_1, x_2).\, \mathtt{if}\ x_1 = A\ \mathtt{then} \\
\mathtt{case}\ x_2\ \mathtt{of}\ \{x_3\}_{K_{AB}}\ \mathtt{in} \\
\nu N_B.\, \overline{\mathtt{chan}_A}\langle \{succ(x_3), N_B\}_{K_{AB}} \rangle. \\
\mathtt{chan}_B(z).\, \mathtt{case}\ z\ \mathtt{of}\ \{z_1\}_{K_{AB}}\ \mathtt{in} \\
\mathtt{if}\ z_1 = succ(N_B)\ \mathtt{then} \\
\nu K'_{AB}.\, \nu N'_B.\, \overline{\mathtt{chan}_A}\langle \{K'_{AB}, N'_B\}_{K_{AB}} \rangle. \\
\mathtt{After}_B^{N_A \mapsto z_1}
\end{array}
$$

この例題では算術演算 $succ(N_A)$, $succ(N_B)$ が変換される方法に注意されたい。上述のプロセスを簡約すると

$$
\begin{array}{l}
\nu N_A.\, \nu N_B.\, \nu K'_{AB}.\, \nu N'_B. \\
\quad \mathtt{After}_A^{N_B \mapsto N_B, K'_{AB} \mapsto K'_{AB}, N'_B \mapsto N'_B} \mid \\
\quad \mathtt{After}_B^{N_A \mapsto N_A}
\end{array}
$$

となる。なお、このプロトコルはメッセージ 4. に A の確認できるノンスが存在しないので、リプレイ攻撃が可能である。実際に、上述の変換結果 P について

$$P \mid P \mid \mathtt{chan}_A(x_4).\, \overline{\mathtt{chan}_A}(x_4).\, \overline{\mathtt{chan}_A}(x_4).\, 0$$

を簡約すると、単一の K'_{AB}, N'_B について $\mathtt{After}_A$ が二回、重複して実行されうる。また、Needham-Schroeder 公開鍵プロトコルと同様の問題も存在する。

例 4 参加者 B がサーバ S を経由して参加者 A を認証することを目的とする、Woo と Lam の認証プロトコル Π[9]

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

を、初期知識 $I(A) = \{A, B, S, K_{AS}\}$, $I(B) = \{A, B, S, K_{BS}\}$, $I(S) = \{A, B, S, K_{AS}, K_{BS}\}$ で変換すると以下のようになる。

☆ ただし、文献[2] では解析の都合により、$\overline{v}\langle \ldots \rangle.\, P$ という形のプロセスを許さず、かわりに $\overline{v}\langle \ldots \rangle \mid P$ という形のプロセスを用いている点が異なる。

$$
\begin{array}{l}
\quad \overline{\mathtt{chan}_B}\langle A\rangle. \\
\quad \mathtt{chan}_A(y).\,\overline{\mathtt{chan}_B}\langle\{y\}_{K_{AS}}\rangle.\,\mathtt{After}_A^{N_B\mapsto y} \\
\mid\ \mathtt{chan}_B(x).\,\mathtt{if}\ x = A\ \mathtt{then}\ \nu N_B.\,\overline{\mathtt{chan}_A}\langle N_B\rangle. \\
\quad \mathtt{chan}_B(z).\,\overline{\mathtt{chan}_S}\langle\{A,z\}_{K_{BS}}\rangle. \\
\quad \mathtt{chan}_B(u).\,\mathtt{case}\ u\ \mathtt{of}\ \{u_1\}_{K_{BS}}\ \mathtt{in} \\
\quad \mathtt{if}\ u_1 = N_B\ \mathtt{then}\ \mathtt{After}_B^{\{N_B\}_{K_{AS}}\mapsto z} \\
\mid\ \mathtt{chan}_S(w).\,\mathtt{case}\ w\ \mathtt{of}\ \{w_1,w_2\}_{K_{BS}}\ \mathtt{in} \\
\quad \mathtt{if}\ w_1 = A\ \mathtt{then}\ \mathtt{case}\ w_2\ \mathtt{of}\ \{w_3\}_{K_{AS}}\ \mathtt{in} \\
\quad \overline{\mathtt{chan}_B}\langle\{w_3\}_{K_{BS}}\rangle.\,\mathtt{After}_S^{N_B\mapsto w_3}
\end{array}
$$

B は K_{AS} を知らないので、3. で受信したメッセージを復号化せず、暗号文のままで 4. のメッセージの一部として送信することに注意されたい。このプロセスを簡約すると

$$
\begin{array}{l}
\nu N_B. \\
\quad \mathtt{After}_A^{N_B\mapsto N_B}\ \mid \\
\quad \mathtt{After}_B^{\{N_B\}_{K_{AS}}\mapsto\{N_B\}_{K_{AS}}}\ \mid \\
\quad \mathtt{After}_S^{N_B\mapsto N_B}
\end{array}
$$

となる。

5. 変換の拡張

Needham-Schroeder 公開鍵プロトコルの次のような使い方を考える。

$$
\begin{array}{llcll}
1. & B & \to & A & : & B \\
2. & A & \to & B & : & \{N_A, A\}_{K_B^+} \\
3. & B & \to & A & : & \{N_A, N_B\}_{K_A^+} \\
4. & A & \to & B & : & \{N_B\}_{K_B^+}
\end{array}
$$

これは、サーバ A がクライアント B からのリクエストにより相互認証を開始する、というシステムである。

もし A の初期知識に特定の名前 B があるとすると、A を変換したプロセスは

$$\mathtt{chan}_A(x).\,\mathtt{if}\ x = B\ \mathtt{then}\ \ldots$$

となるから、A はその B からのリクエストしか処理できない。では、もし A の初期知識に B がなければ、A は複数の B からのリクエストを処理できるのだろうか? $I(A) = \{A, K_A^+, K_A^-, K_B^+\}$ として、図 3 の定義にしたがい実際に変換してみると、以下の結果がえられる。

$$
\begin{array}{l}
\quad \mathtt{chan}_A(w). \\
\quad \nu N_A.\,\overline{\mathtt{chan}_B}\langle\{N_A, A\}_{K_B^+}\rangle. \\
\quad \mathtt{chan}_A(y).\,\mathtt{case}\ y\ \mathtt{of}\ \{y_1,y_2\}_{K_A^-}\ \mathtt{in} \\
\quad \mathtt{if}\ y_1 = N_A\ \mathtt{then} \\
\quad \overline{\mathtt{chan}_B}\langle\{y_2\}_{K_B^+}\rangle.\,\mathtt{After}_A^{N_B\mapsto y_2} \\
\mid\ \overline{\mathtt{chan}_A}\langle B\rangle. \\
\quad \mathtt{chan}_B(x).\,\mathtt{case}\ x\ \mathtt{of}\ \{x_1,x_2\}_{K_B^-}\ \mathtt{in} \\
\quad \mathtt{if}\ x_2 = A\ \mathtt{then} \\
\quad \nu N_B.\,\overline{\mathtt{chan}_A}\langle\{x_1, N_B\}_{K_A^+}\rangle. \\
\quad \mathtt{chan}_B(z).\,\mathtt{case}\ z\ \mathtt{of}\ \{z_1\}_{K_B^-}\ \mathtt{in} \\
\quad \mathtt{if}\ z_1 = N_B\ \mathtt{then}\ \mathtt{After}_B^{N_A\mapsto x_1}
\end{array}
$$

しかし、この結果には問題がある。A は任意の B すなわち w からのリクエストを処理したいにもかかわらず、特定の B の通信チャンネル $\mathtt{chan}_B$ および公開鍵 K_B^+ を使用するからである。直観としては $\mathtt{chan}_w$ ないし K_w^+ のように表記したいところだが、このようなパラメタのある変数は、spi 計算の操作的意味論の範疇から逸脱している。

そこで本稿では変換を拡張し、w のような変数から $\mathtt{chan}_w$ ないし K_w^+ のような通信チャンネルや公開鍵を動的に検索するための、いわば「データベース」に相当するプロセスを生成・使用する。たとえば、上の例は以下のように変換される。

$$
\begin{array}{l}
\nu\mathtt{getc}.\,\nu\mathtt{getk}. \\
\quad !\mathtt{getc}(r,x).\ \prod_{Z\in dom(I)\wedge Z\notin I(A)} \\
\qquad \mathtt{if}\ x = Z\ \mathtt{then}\ \overline{r}\langle\mathtt{chan}_Z\rangle\ \mid \\
\quad !\mathtt{getk}(r,x).\ \prod_{K_Z^+\in I(A)\wedge Z\notin I(A)} \\
\qquad \mathtt{if}\ x = Z\ \mathtt{then}\ \overline{r}\langle K_Z^+\rangle\ \mid \\
\quad \mathtt{chan}_A(w). \\
\quad \nu N_A.\,\nu r_1.\,\overline{getc}\langle r_1, w\rangle.\,r_1(c). \\
\quad \nu N_B.\,\nu r_2.\,\overline{getk}\langle r_2, w\rangle.\,r_2(k). \\
\quad \overline{c}\langle\{N_A, A\}_k\rangle. \\
\quad \mathtt{chan}_A(y).\,\mathtt{case}\ y\ \mathtt{of}\ \{y_1,y_2\}_{K_A^-}\ \mathtt{in} \\
\quad \mathtt{if}\ y_1 = N_A\ \mathtt{then} \\
\quad \overline{c}\langle\{y_2\}_k\rangle.\,\mathtt{After}_A^{N_B\mapsto y_2} \\
\mid\ \ldots
\end{array}
$$

プロセス $!\mathtt{getc}(r,x).\ldots$ が名前 x から通信チャンネル $\mathtt{chan}_x$ を、プロセス $!\mathtt{getk}(r,x).\ldots$ が名前 x から鍵 K_x^+ を動的に検索するためのデータベースに相当する。ただし、r は返信のための通信チャンネルである。すなわち、$\mathtt{getc}$ ないし $\mathtt{getk}$ に、通信チャンネル r と参加者の名前 Z を送信すると、r に $\mathtt{chan}_Z$ ないし K_Z^+ が返信される。ここでは $\nu r_1.\,\overline{getc}\langle r_1, w\rangle.\,r_1(c)$ と $\nu r_2.\,\overline{getk}\langle r_2, w\rangle.\,r_2(k)$ が、そのような動的な検索

$$
\begin{array}{lcl}
\mathcal{T}_X(\alpha_1;\ldots;\alpha_n) & = & \nu \mathtt{get}_{XY_1}.\ldots.\nu \mathtt{get}_{XY_m}.\\
& & \nu \mathtt{get{+}}_{XY_1}.\ldots.\nu \mathtt{get{+}}_{XY_m}.\\
& & \quad \prod_{i,n} !\mathtt{get}_{XY_i}(r,x_1,\ldots,x_n).\\
& & \qquad \prod_{Y_i Z_1\ldots Z_n \in I(X)\cup\{\mathtt{chan}_Z \mid Z\in dom(I)\}\wedge\{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset}\\
& & \qquad\quad \mathtt{if}\ x_1 = Z_1\ \mathtt{then}\ \ldots\ \mathtt{if}\ x_n = Z_n\ \mathtt{then}\ r\langle Y_{Z_1\ldots Z_n}\rangle \mid\\
& & \quad \prod_{i,n} !\mathtt{get{+}}_{XY_i}(r,x_1,\ldots,x_n).\\
& & \qquad \prod_{Y_i{}^{+}_{Z_1\ldots Z_n} \in I(X)\wedge\{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset}\\
& & \qquad\quad \mathtt{if}\ x_1 = Z_1\ \mathtt{then}\ \ldots\ \mathtt{if}\ x_n = Z_n\ \mathtt{then}\ r\langle Y^{+}_{Z_1\ldots Z_n}\rangle \mid\\
& & \quad C_1[\ldots[C_n[\mathtt{After}^{\sigma}_{X}]]])\\
& & \text{if } \{Y_1,\ldots,Y_m\} = \{Y \mid Y \text{ appears in } I\}\cup\{\mathtt{chan}\},\\
& & \mathcal{T}_X(\rho_0,\alpha_1) = (\rho_1,C_1),\ldots,\mathcal{T}_X(\rho_{n-1},\alpha_n) = (\rho_n,C_n)\\
& & \text{for } \rho_0 = \{M\mapsto M \mid M\in I(X)\}\\
& & \text{and } \sigma = \rho_n \setminus \{M\mapsto M \mid \text{any } M\}\\
\\
\mathcal{T}_X(\rho, X\to Y : M_1,\ldots,M_n) & = & (\rho_n, \nu r.\overline{\mathtt{get}_{X\,\mathtt{chan}}}\langle r,\rho(Y)\rangle.\, r(c).\, C_1[\ldots[C_n[\overline{c}\langle N_1,\ldots,N_n\rangle.\,[]]]])\\
& & \text{if } r,c \text{ fresh, } Y\neq X,\ Y\notin I(X),\\
& & \text{and } \mathcal{S}_X(\rho,M_1) = (\rho_1,C_1,N_1),\ldots,\mathcal{S}_X(\rho_{n-1},M_n) = (\rho_n,C_n,N_n)\\
\\
\mathcal{S}_X(\rho, Y_{Z_1\ldots Z_n}) & = & (\rho, \nu r.\overline{\mathtt{get}_{XY}}\langle r,\rho(Z_1),\ldots,\rho(Z_n)\rangle.\, r(k).\,[],k)\\
& & \text{if } r,k \text{ fresh, } Y_{Z_1\ldots Z_n}\in I(X) \text{ and } \{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset\\
\mathcal{S}_X(\rho, Y^{+}_{Z_1\ldots Z_n}) & = & (\rho, \nu r.\overline{\mathtt{get{+}}_{XY}}\langle r,\rho(Z_1),\ldots,\rho(Z_n)\rangle.\, r(k).\,[],k)\\
& & \text{if } r,k \text{ fresh, } Y^{+}_{Z_1\ldots Z_n}\in I(X) \text{ and } \{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset\\
\\
map_X(\rho, Y_{Z_1\ldots Z_n}) & = & (\nu r.\overline{\mathtt{get}_{XY}}\langle r,\rho(Z_1),\ldots,\rho(Z_n)\rangle.\, r(k).\,[],k)\\
& & \text{if } r,k \text{ fresh, } Y_{Z_1\ldots Z_n}\in I(X) \text{ and } \{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset\\
map_X(\rho, Y^{+}_{Z_1\ldots Z_n}) & = & (\nu r.\overline{\mathtt{get{+}}_{XY}}\langle r,\rho(Z_1),\ldots,\rho(Z_n)\rangle.\, r(k).\,[],k)\\
& & \text{if } r,k \text{ fresh, } Y^{+}_{Z_1\ldots Z_n}\in I(X) \text{ and } \{Z_1,\ldots,Z_n\}\setminus I(X)\neq\emptyset
\end{array}
$$

図 4 鍵データベースによる変換の拡張
Fig. 4 Extension of the Translation with Key Database

に相当するプロセスである。

鍵ないし通信チャンネルの動的な検索がどのような場合に必要かは、以下のように判定する。先の例で、知識 $I(A)$ によく注意すると、参加者 B を知らないにもかかわらず、その公開鍵 K_B^+ を知っていることになっている。これは、B が静的にはわからず、K_B^+ を動的に検索する必要があることを暗示している。一般に、変数 $X_{Y_1\ldots Y_n}$ が初期知識にあるにもかかわらず、$Y_1, \ldots, Y_n$ のいずれかが初期知識にない場合は、$X_{Y_1\ldots Y_n}$ を動的に検索しなければならない。

以上の考察にもとづいて、図 3 の定義を図 4 のように拡張する。ただし、条件の重複する定義については図 4 を優先するものとする。先の例は、この定義の特殊な場合となっている。

図 4 の前半では、参加者 X を変換するときに、X が動的に検索する（かもしれない）すべての鍵ないし通信チャンネル $Y_{ix_1\ldots x_n}$ および $Y^{+}_{ix_1\ldots x_n}$ について、それらのデータベースに相当するプロセスを並列に生成している。プロセス $!\mathtt{get}_{XY_i}(r,x_1,\ldots,x_n).\ldots$ は、参加者 X が共有鍵ないし通信チャンネル $Y_{ix_1\ldots x_n}$ を動的に検索するためのデータベースに相当する。先の例でいえば、A が X に、$\mathtt{chan}$ が Y_i に、$\mathtt{getc}$ が $\mathtt{get}_{XY_i}$ に、x が $x_1,\ldots,x_n$ に当たる。同様に、プロセス $!\mathtt{get{+}}_{XY_i}(r,x_1,\ldots,x_n).\ldots$ は、参加者 X が公開鍵 $Y^{+}_{ix_1\ldots x_n}$ を動的に検索するためのデータベースに相当する。先の例でいえば、A が X に、K が Y_i に、$\mathtt{getk}$ が $\mathtt{get{+}}_{XY_i}$ に、x が $x_1,\ldots,x_n$ に当たる。

図 4 の後半では、参加者 X の各行動を変換するときに、（必要ならば）データベースの動的な検索に相当するプロセスを生成している。動的な検索が必要かどうかは、前述のように「使用したい変数 $Y_{Z_1\ldots Z_n}$ ないし $Y^{+}_{Z_1\ldots Z_n}$ が初期知識にあるにもかかわらず、$Z_1, \ldots, Z_n$ のいずれかが初期知識にない」ことにより判定する。

たとえば、A は攻撃者 E の公開鍵 K_E^+ を知っているものとして、前述の変換結果を P とする。

$\overline{\mathtt{chan}_A}\langle E\rangle.$
$\mathtt{chan}_E(x).\,\mathtt{case}\ x\ \mathtt{of}\ \{x_1, x_2\}_{K_E^-}\ \mathtt{in}$
$\mathtt{chan}_A(y).\,\mathtt{if}\ y = B\ \mathtt{then}$
$\overline{\mathtt{chan}_B}\langle\{x_1, x_2\}_{K_B^+}\rangle.$
$\mathtt{chan}_E(z).\,\mathtt{case}\ z\ \mathtt{of}\ \{z_1\}_{K_E^-}\ \mathtt{in}$
$\overline{\mathtt{chan}_B}\langle\{z_1\}_{K_B^+}\rangle.$
$\mathtt{After}_E^{N_A \mapsto x_1, N_B \mapsto z_1}$

なる攻撃プロセス Q について $P \mid Q$ を実行すると、

$\nu N_A.\,\nu N_B.$
$\mathtt{After}_A^{N_B \mapsto N_B} \mid \mathtt{After}_B^{N_A \mapsto N_A} \mid$
$\mathtt{After}_E^{N_A \mapsto N_A, N_B \mapsto N_B}$

のように、E が A, B のノンス N_A, N_B を共有するプロセスに簡約されてしまう。このように、A が B だけでなく E からのリクエストも処理する場合は、いわゆる man-in-the-middle 攻撃が可能である[15] ことが、拡張された変換によって正確に反映されている。

なお、既存の文献[2] では同様の攻撃を表現するにあたり、本稿のように鍵や通信チャンネルを動的に検索するのではなく、B と E のそれぞれについて別々の A が存在することを仮定している。この仮定は場合によっては非現実的だが、非依存型システムによりプロトコルを解析するために要求されている。依存型により解析を拡張すれば、本稿と同様の表現も使用できると予想される。

6. 関連研究と今後の課題

本稿ではセキュリティプロトコルの略式表現から spi 計算への変換を定義し、略式表現の意味を明確にした。我々は本稿のアイディアにもとづいて、略式表現のプレーンテキストから spi 計算のプロセス（および双方を整形した TEX ソース）を生成する Objective Caml[14] のプログラムを実装した。そもそも略式表現の意味が変換以外に定義されていないので、変換の妥当さを厳密に証明したり自動で検査することはできないが、我々は文献[9] にある 24 個のプロトコルについて、変換の結果が直観と合致していることを人手で確認した[19]。24 個の中で主な 4 つを 4 節に挙げたが、他の 20 個のプロトコルも 4 つのいずれかと同じような形をしており、(5 節の拡張も含めて) 同じように変換できる。

関連研究としては、すでに言及した spi 計算や spi 計算における検証以外に、Casper[16] と CAPSL[17] がある。Casper は略式記法に類似したスクリプトを、プロセス計算の一種である CSP[13] に変換するコンパイラである。CAPSL はセキュリティプロトコルを記述・検証するための形式言語である。しかしどちらの言語も、参加者の予備知識だけでなく、参加者の役割（initiator か responder か）や変数の種類（メッセージかノンスか公開鍵か）の宣言が必要であり、プロトコルの**動作**を記述する目的では、我々の方法のほうが簡潔である。一方で、本稿の略式記法ではプロトコルの**目的**は記述できない。これを可能にするには、秘密にしたいメッセージに annotation をつけたり、認証を表現する対応表明[11),20)] を挿入するといった拡張が必要だろう。

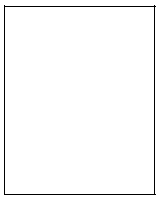
また、本稿の変換では算術演算の性質やタイムスタンプの概念に依存したセキュリティプロトコルを正確に定義することができない。図 3 の定義は、たとえば加算の結合法則を反映していないので、もし $x+y$ と z の値を知っていても、$x+(y+z)$ の値は知らないことになってしまう。この問題を解決するためには、たとえば $x+(y+z)$ を $(x+y)+z$ と解釈させるような annotation を導入するか、あるいは応用 π 計算[4] と同様の等式理論で変換を拡張し、$M = N$ かつ $map_X(\rho, N)$ が定義されている N が存在したら M の値を知っているとみなさなければならない。タイムスタンプについては、fresh なノンスとして解釈するだけならば本稿の変換で可能だが、時間の概念を導入した解釈はそもそも spi 計算やほとんどの formal method でも手法が確立されておらず、今後の課題である。

参 考 文 献

1) Abadi, M.: Secrecy by Typing in Security Protocols, *Journal of the ACM*, Vol. 46, No. 5, pp. 749–786 (1999). Preliminary version appeared in *Theoretical Aspects of Computer Software*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1281, pp. 611–638, 1997.
2) Abadi, M. and Blanchet, B.: Secrecy Types for Asymmetric Communication, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, Vol. 2030, Springer-Verlag, pp. 25–41 (2001).
3) Abadi, M. and Blanchet, B.: Secrecy Types for Asymmetric Communication, *Theoretical Computer Science*, Vol. 298, No. 3, pp. 387–415 (2003). Preliminary version appeared in *Foundations of Software Science and Computation Structures*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 2030, pp. 25–41, 2001.
4) Abadi, M. and Fournet, C.: Mobile Values, New Names, and Secure Communication, *Pro-*

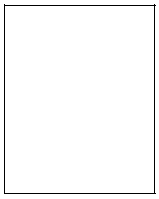
ceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 104–115 (2001).
5) Abadi, M. and Gordon, A. D.: A Bisimulation Method for Cryptographic Protocols, *Nordic Journal of Computing*, Vol. 5, pp. 267–303 (1998). Preliminary version appeared in *Programming Languages and Systems – ESOP'98, 7th European Symposium on Programming*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1381, pages 12–26, 1998.
6) Abadi, M. and Gordon, A. D.: A Calculus for Cryptographic Protocols: The spi Calculus, *Information and Computation*, Vol. 148, No. 1, pp. 1–70 (1999). Preliminary version appeared in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 36–47, 1997.
7) Boreale, M., De Nicola, R. and Pugliese, R.: Proof Techniques for Cryptographic Processes, *SIAM Journal on Computing*, Vol.31, No.3, pp. 947–986 (2002). Preliminary version appeared in *14th Annual IEEE Symposium on Logic in Computer Science*, pp. 157–166, 1999.
8) Borgström, J. and Nestmann, U.: On Bisimulations for the Spi Calculus, *9th International Conference on Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science, Vol.2422, Springer-Verlag, pp.287–303 (2002).
9) Clark, J. and Jacob, J.: A Survey of Authentication Protocol Literature: Version 1.0, `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz` (1997).
10) Dolev, D. and Yao, A. C.: On the security of public key protocols, *IEEE Transactions on Information Theory*, Vol. 29, No. 2, pp. 198–208 (1983).
11) Gordon, A. D. and Jeffrey, A.: Authenticity by Typing for Security Protocols, *Journal of Computer Security* (2003). To appear. Extended abstract appeared in *14th IEEE Computer Security Foundations Workshop*, pp. 145–159, 2001.
12) Gordon, A. D. and Jeffrey, A.: Types and Effects for Asymmetric Cryptographic Protocols, *Journal of Computer Security* (2004). To appear. Extended abstract appeared in *15th IEEE Computer Security Foundations Workshop*, pp. 77–91, 2002.
13) Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice Hall (1985). `http://www.usingcsp.com/`.
14) Leroy, X.: Objective Caml, `http://caml.inria.fr/`.
15) Lowe, G.: An Attack on the Needham-Schroeder Public-Key Authentication Protocol, *Information Processing Letters*, Vol. 56, No. 3, pp. 131–133 (1995).
16) Lowe, G.: Casper: A Compiler for the Analysis of Security Protocols, *10th IEEE Computer Security Foundations Workshop*, pp. 18–30 (1997).
17) Millen, J. K.: Common Authentication Protocol Specification Language, `http://www.csl.sri.com/users/millen/capsl/`.
18) Needham, R. and Schroeder, M.: Using encryption for authentication in large networks of computers, *Communications of the ACM*, Vol. 21, No. 12, pp. 993–999 (1978).
19) Tatsuzawa, H.: Translating Security Protocols from Informal Notation into Spi Calculus, Bachelor's Thesis, Department of Information Science, Faculty of Science, University of Tokyo (2003).
20) Woo, T. Y. C. and Lam, S. S.: A Semantic Model for Authentication Protocols, *IEEE Symposium on Security and Privacy*, pp. 178–194 (1993).
21) 住井英二郎: チュートリアル: 暗号化通信の spi 計算による形式的検証, コンピュータソフトウェア (2003). 掲載予定.

(平成 16 年 2 月 23 日受付)
(平成 16 年 5 月 10 日採録)



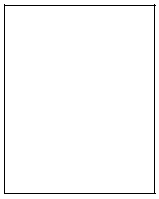
住井英二郎

1975 年生。1998 年 3 月東京大学理学部情報科学科卒業。2000 年 3 月同大学院理学系研究科情報科学専攻修士課程修了。2000 年 4 月より 2001 年 3 月まで同博士課程学生、日本学術振興会特別研究員、およびペンシルバニア大学コンピュータ情報科学科 Visiting Scholar。2001 年 4 月より 2003 年 3 月まで東京大学大学院情報理工学系研究科コンピュータ科学専攻助手。2003 年 4 月よりペンシルバニア大学コンピュータ情報科学科 Research Associate。情報セキュリティ、プロセス計算、部分評価等の領域における、プログラミング言語および型システムの理論と応用について研究。ACM 会員。



立沢　秀晃

昭和 55 年生。平成 15 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程入学。現在修士 2 年。



米澤　明憲（正会員）

1947 年 6 月に生まれ、MIT 計算機科学科博士課程修了 (Ph.D. in Computer Science), MIT 計算機科学研究所および人工知能研究所において並列・分散計算モデルの研究に従事し、「並列オブジェクト」概念のパイオニアの一人として知られている。帰国後、東工大を経て、1988 年に東京大学理学部情報科学科教授に着任した。米国計算機学会終身フェロー (ACM Fellow) であり、ACM TOPLAS 副編集長、日本ソフトウエア科学会理事長、ドイツ国立情報科学技術研究所 (GMD) 科学顧問、政府情報科学技術委員会委員などを歴任。