チュートリアル：ProVerifによる結合可能安全性の形式検証

櫻田英樹
日本電信電話株式会社
NTT コミュニケーション科学基礎研究所

アウトライン

前半：ProVerifの紹介

後半：ProVerifを用いた結合可能安全性証明

☞[Dahl-Damgård, EuroCrypt2014, ePrint2013/296]の記号検証パート

ProVerif

フランス国立情報学自動制御研究所（INRIA）の
B. Blanchetを中心に開発されているプロトコル自動検証ツール

- π計算と呼ばれる言語（理論）に基づく
- 様々な暗号技術を等式を用いて定義・利用可能
- 様々な種類の安全性を定義・検証可能

  （認証、弱秘匿性、非干渉性、観測等価性）
- Unboundedな（プロトコルの並列実行数などを制限しない）検証が可能

ProVerifにおけるプロトコルの記述例(1)

$Q_R \doteq \text{new } r_b;$ 乱数生成

$\text{in}[in^R_{OT}, b];$ メッセージ受信

if $\mathbf{inType}_{bit}(b) = \mathbf{true}$ then 条件分岐

let $c'_b = \mathbf{enc}(b, r_b, ek_R);$

let $p_b = \mathbf{proof}_{bit}(c'_b, ek_R, crs_R);$ メッセージの操作（暗号化など）

let $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R);$

$\text{out}[send_{RS}, c_b];$ メッセージ送信

$\text{in}[receive_{SR}, c_x];$

if $\mathbf{verEvalPack}_{sel}(c_x) = \mathbf{true}$ then

if $\mathbf{encOf}_1(c_x) = c_b$ then

if $\mathbf{ekOf}(c_x) = ek_R$ then

if $\mathbf{crsOf}(c_b) = crs_R$ then

let $x_b = \mathbf{dec}(\mathbf{encOf}(c_x), dk_R);$

$\text{out}[out^R_{OT}, x_b]$

ProVerifにおけるプロトコルの記述例(2)

プロトコル全体

$$
\begin{aligned}
&\text{new } ck_A, ck_B;\\
&\text{new } dk_A, dk_B;\\
&\text{let } ek_A = \mathbf{ek}(dk_A);\\
&\text{let } ek_B = \mathbf{ek}(dk_B);\\
&\text{new } crs_A, crs_B;\\
&(Q_S \parallel Q_{\text{Auth}_{SR}} \parallel Q_{\text{Auth}_{RS}} \parallel Q_R)
\end{aligned}
$$

鍵などの生成（下のプロセスから参照できる）

前のスライドのプロセス

プロトコルで用いられる暗号等の性質（必要な分だけ定義）

$$
\begin{aligned}
\mathbf{dec}(\mathbf{enc}(v, x_r, \mathbf{ek}(x_{dk})), x_{dk}) &= v\\
\mathbf{first}(\mathbf{pair}(x_1, x_2)) &= x_1\\
\mathbf{second}(\mathbf{pair}(x_1, x_2)) &= x_2
\end{aligned}
$$

ProVerifによる安全性(観測等価性)検証

観測等価性(“攻撃者からは区別できない”)

$P \overset{s}{\sim} Q$ ⟺

任意のプロセスA(＝攻撃者)について、
$A \mid\mid P$ が通信路dからメッセージを出力
iff
$A \mid\mid Q$ が通信路dからメッセージを出力

※攻撃者Aも単なるプロセスなので、
メッセージの送受信と、決められた演算しか用いない
(☞ 記号モデルでの検証)

ProVerifでは、PとQが“ほぼ同じ形”をしている場合のみ扱える
(後述)

結合可能安全性の検証 （Dahl-Damgård 2014）

「実プロトコル（検証したいプロトコル）が理想プロトコル（理想機能＋シミュレータ）と同じように振る舞う（攻撃者から見て区別できない）」という等価性を示せば良い

$$\mathcal{RW}(Sys^{\mathcal{H}}_{\mathrm{real}}) \overset{c}{\sim} \mathcal{RW}(Sys^{\mathcal{H}}_{\mathrm{ideal}})$$

計算論的モデルでの
等価性（識別不能性）
☞自動検証が難しい

Theorem 5.6, 6.3
（プロトコルによらず成立）

$$\mathcal{S}(Sys^{\mathcal{H}}_{\mathrm{real}}) \overset{s}{\sim} \mathcal{S}(Sys^{\mathcal{H}}_{\mathrm{ideal}})$$

記号モデルでの
等価性（観測等価性）
☞自動検証が容易

個別のプロトコルについてはProVerifを使ってこれ検証すればよい

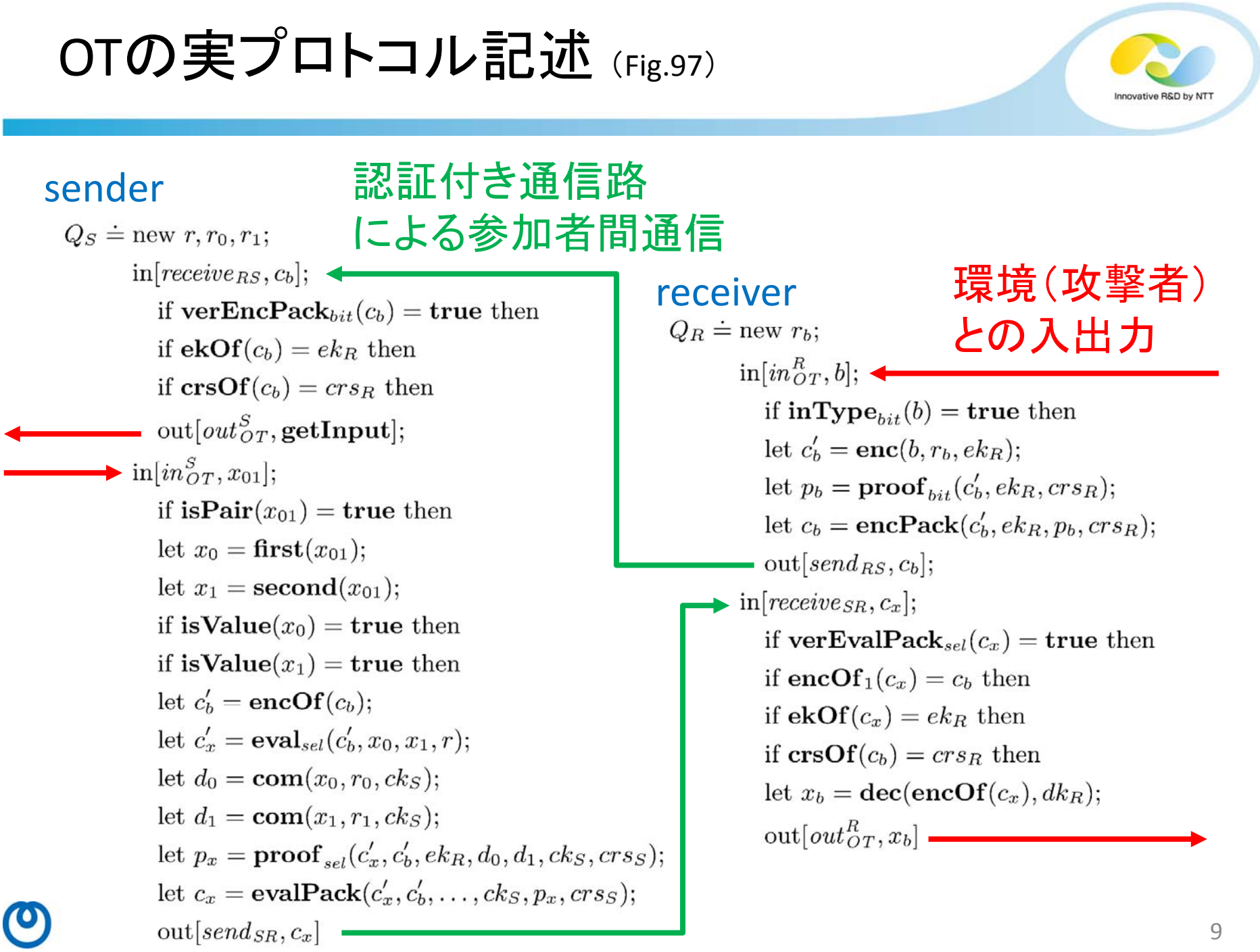
ProVerifを用いた検証の流れ

ProVerifはほぼ同じ形の2つのプロセスの等価性しか検証できないため、人手で以下を行う

1. 実プロトコルと理想プロトコルを記述
2. 両者がなるべく同じ形になるよう変形して、1つのプロセスBに”まとめる”
3. BをProVerifに入力して検証（ここは自動）

※corruptionの各シナリオについて以上を行う

OTの実プロトコル記述（Fig.97）



実プロトコルを変形（Fig.99）

不要なチェックを削除

$\mathrm{in}[in^{R}_{OT}, b];$
  if $\mathbf{inType}_{bit}(b) = \mathbf{true}$ then
  new r_b;
  let $c'_b = \mathbf{enc}(b, r_b, ek_R)$;
  let $p_b = \mathbf{proof}_{bit}(c'_b, ek_R, crs_R)$;
  let $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R)$;
  $\mathrm{out}[leak_{RS}, c_b]$;
$\mathrm{in}[infl_{RS}, \mathbf{deliver}]$;
  if $\mathbf{verEncPack}_{bit}(c_b) = \mathbf{true}$ then
  if $\mathbf{ekOf}(c_b) = ek_R$ then
  if $\mathbf{crsOf}(c_b) = crs_R$ then
  $\mathrm{out}[out^{S}_{OT}, \mathbf{getInput}]$;
$\mathrm{in}[in^{S}_{OT}, x_{01}]$;
  if $\mathbf{isPair}(x_{01}) = \mathbf{true}$ then
  let $x_0 = \mathbf{first}(x_{01})$;
  let $x_1 = \mathbf{second}(x_{01})$;
  if $\mathbf{isValue}(x_0) = \mathbf{true}$ then
  if $\mathbf{isValue}(x_1) = \mathbf{true}$ then

$Q^{SR}_{left} \doteq \mathrm{in}[in^{R}_{OT}, b];$
  if $\mathbf{inType}_{bit}(b) = \mathbf{true}$ then
  new r_b;
  let $c'_b = \mathbf{enc}(b, r_b, ek_R)$;
  let $p_b = \mathbf{proof}_{bit}(c'_b, ek_R, crs_R)$;
  let $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R)$;
  $\mathrm{out}[leak_{RS}, c_b]$;
$\mathrm{in}[in^{S}_{OT}, x_{01}]$;
  if $\mathbf{isPair}(x_{01}) = \mathbf{true}$ then
  let $x_0 = \mathbf{first}(x_{01})$;
  let $x_1 = \mathbf{second}(x_{01})$;
  if $\mathbf{isValue}(x_0) = \mathbf{true}$ then
  if $\mathbf{isValue}(x_1) = \mathbf{true}$ then

- 複数のプロセスとして記述していたものをひとまとめに
- 不要なチェックを削除

正しく作った暗号文は
チェック不要なので削除

OTの理想プロトコル（Fig.98）

理想機能

$$
\begin{aligned}
Q_{\mathcal{F}_{OT}^{SR}} \doteq\ & \text{in}[in_{OT}^{R}, b];\\
& \quad \text{if } \mathbf{inType}_{bit}(b) = \mathbf{true} \text{ then}\\
& \quad \text{out}[leak_{OT}^{R}, \mathbf{bReceived}];\\
& \text{in}[infl_{OT}^{S}, \mathbf{getInput}];\\
& \quad \text{out}[out_{OT}^{S}, \mathbf{getInput}];\\
& \text{in}[in_{OT}^{S}, x_{01}];\\
& \quad \text{if } \mathbf{isPair}(x_{01}) = \mathbf{true} \text{ then}\\
& \quad \text{let } x_0 = \mathbf{first}(x_{01});\\
& \quad \text{let } x_1 - \mathbf{second}(x_{01});\\
& \quad \text{if } \mathbf{isValue}(x_0) = \mathbf{true} \text{ then}\\
& \quad \text{if } \mathbf{isValue}(x_1) = \mathbf{true} \text{ then}\\
& \quad \text{out}[leak_{OT}^{S}, \mathbf{xsReceived}];\\
& \text{in}[infl_{OT}^{R}, \mathbf{finish}];\\
& \quad \text{if } \mathbf{eqValue}(b, \mathbf{zero}) = \mathbf{true} \text{ then}\\
& \quad\quad \text{out}[out_{OT}^{R}, x_0]\\
& \quad \text{else}\\
& \quad\quad \text{out}[out_{OT}^{R}, x_1]
\end{aligned}
$$

シミュレータ
（senderをシミュレート）

$$
\begin{aligned}
Q_{Sim_{OT}^{SR,S}} \doteq\ & \text{new } r, r_0, r_1;\\
& \text{in}[receive_{RS}, c_b];\\
& \quad \text{if } \mathbf{verEncPack}_{bit}(c_b) = \mathbf{true} \text{ then}\\
& \quad \text{if } \mathbf{ekOf}(c_b) = ek_R \text{ then}\\
& \quad \text{if } \mathbf{crsOf}(c_b) = crs_R \text{ then}\\
& \quad \text{out}[infl_{OT}^{S}, \mathbf{getInput}];\\
& \text{in}[leak_{OT}^{S}, \mathbf{xsReceived}];\\
& \quad \text{let } c_b' = \mathbf{encOf}(c_b);\\
& \quad \text{let } c_x' = \mathbf{eval}_e(c_b', \mathbf{zero}, \mathbf{zero}, r);\\
& \quad \text{let } d_0 = \mathbf{com}(\mathbf{zero}, r_0, ck_S);\\
& \quad \text{let } d_1 = \mathbf{com}(\mathbf{zero}, r_1, ck_S);\\
& \quad \text{let } p_x = \mathbf{proof}_{sel}(c_x', c_b', \ldots, crs_S);\\
& \quad \text{let } c_x' = \mathbf{evalPack}_{sel}(c_x', c_b', \ldots, p_x, crs_S);\\
& \quad \text{out}[send_{SR}, c_x']
\end{aligned}
$$

シミュレータ
（receiverをシミュレート）

$$
\begin{aligned}
Q_{Sim_{OT}^{SR,R}} \doteq\ & \text{new } r_b;\\
& \text{in}[leak_{OT}^{R}, \mathbf{bReceived}];\\
& \quad \text{let } c_b' = \mathbf{enc}(\mathbf{zero}, r_b, ek_R);\\
& \quad \text{let } p_b = \mathbf{proof}_{bit}(c_b', ek_R, crs_R);\\
& \quad \text{let } c_b = \mathbf{encPack}(c_b', ek_R, p_b, crs_R);\\
& \quad \text{out}[send_{RS}, c_b];\\
& \text{in}[receive_{SR}, c_x];\\
& \quad \text{if } \mathbf{verEvalPack}_{sel}(c_x) = \mathbf{true} \text{ then}\\
& \quad \text{if } \mathbf{encOf}_1(c_x) = c_b' \text{ then}\\
& \quad \text{if } \mathbf{ekOf}(c_x) = ek_R \text{ then}\\
& \quad \text{if } \mathbf{ckOf}(c_x) = ck_S \text{ then}\\
& \quad \text{if } \mathbf{crsOf}(c_x) = crs_S \text{ then}\\
& \quad \text{out}[infl_{OT}^{R}, \mathbf{finish}]
\end{aligned}
$$

理想プロトコルを変形（Fig.100）

不要なチェックを削除

削除

$\text{in}[in^{R}_{OT}, b];$
  if $\mathbf{inType}_{bit}(b) = \mathbf{true}$ then
  new r_b;
  let $c'_b = \mathbf{enc}(\mathbf{zero}, r_b, ek_R)$;
  let $p_b = \mathbf{proof}_{bit}(c_b, ek_R, crs_R)$;
  let $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R)$;
  $\text{out}[leak_{RS}, c_b]$;
$\text{in}[infl_{RS}, \mathbf{deliver}]$
  if $\mathbf{verEncPack}_{bit}(c_b) = \mathbf{true}$ then
  if $\mathbf{ekOf}(c_b) = ek_R$ then
  if $\mathbf{crsOf}(c_b) = crs_R$ then
  $\text{out}[out^{S}_{OT}, \mathbf{getInput}]$;
$\text{in}[in^{S}_{OT}, x_{01}]$;
  if $\mathbf{isPair}(x_{01}) = \mathbf{true}$ then
  let $x_0 = \mathbf{first}(x_{01})$;
  let $x_1 = \mathbf{second}(x_{01})$;
  if $\mathbf{isValue}(x_0) = \mathbf{true}$ then
  if $\mathbf{isValue}(x_1) = \mathbf{true}$ then
  new r, r_0, r_1;
  let $c'_b = \mathbf{encOf}(c_b)$;
  let $c'_x = \mathbf{eval}_c(c'_b, \mathbf{zero}, \mathbf{zero}, r)$;
  let $d_0 = \mathbf{com}(\mathbf{zero}, r_0, ck_S)$;
  let $d_1 = \mathbf{com}(\mathbf{zero}, r_1, ck_S)$;
  let $p_x = \mathbf{proof}_{sel}(c'_x, \ldots, crs_S)$;
  let $c_x = \mathbf{evalPack}(c'_x, \ldots, crs_S)$;
  $\text{out}[leak_{SR}, c_x]$;
$\text{in}[infl_{SR}, \mathbf{deliver}]$;
  if $\mathbf{verEvalPack}_{sel}(c_x) = \mathbf{true}$ then
  if $\mathbf{encOf}_1(c_x) = c'_b$ then
  if $\mathbf{ekOf}(c_x) = ek_R$ then
  if $\mathbf{ckOf}(c_x) = ck_S$ then
  if $\mathbf{crsOf}(c_x) = crs_S$ then
  if $\mathbf{eqValue}(b, \mathbf{zero}) = \mathbf{true}$ then
    $\text{out}[out^{R}_{OT}, x_0]$
  else
    $\text{out}[out^{R}_{OT}, x_1]$

$Q^{SR}_{right} \doteq \text{in}[in^{R}_{OT}, b];$
  if $\mathbf{inType}_{bit}(b) = \mathbf{true}$ then
  new r_b;
  let $c'_b = \mathbf{enc}(\mathbf{zero}, r_b, ek_R)$;
  let $p_b = \mathbf{proof}_{bit}(c_b, ek_R, crs_R)$;
  let $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R)$;
  $\text{out}[leak_{RS}, c_b]$;
$\text{in}[in^{S}_{OT}, x_{01}]$;
  if $\mathbf{isPair}(x_{01}) = \mathbf{true}$ then
  let $x_0 = \mathbf{first}(x_{01})$;
  let $x_1 = \mathbf{second}(x_{01})$;
  if $\mathbf{isValue}(x_0) = \mathbf{true}$ then
  if $\mathbf{isValue}(x_1) = \mathbf{true}$ then
  new r, r_0, r_1;
  let $c'_x = \mathbf{eval}_c(c'_b, \mathbf{zero}, \mathbf{zero}, r)$;
  let $d_0 = \mathbf{com}(\mathbf{zero}, r_0, ck_S)$;
  let $d_1 = \mathbf{com}(\mathbf{zero}, r_1, ck_S)$;
  let $p_x = \mathbf{proof}_{sel}(c'_x, \ldots, crs_S)$;
  let $c_x = \mathbf{evalPack}(c'_x, \ldots, crs_S)$;
  $\text{out}[leak_{SR}, c_x]$;
$\text{in}[infl_{SR}, \mathbf{deliver}]$;
  if $\mathbf{eqValue}(b, \mathbf{zero}) = \mathbf{true}$ then
    $\text{out}[out^{R}_{OT}, x_0]$
  else
    $\text{out}[out^{R}_{OT}, x_1]$

実プロトコルと同様に、

- 複数のプロセス（理想機能、シミュレータ）をひとまとめに
- 不要なチェックを削除

実プロトコルと理想プロトコルをまとめる

$$
\begin{aligned}
B^{SR}_{OT} \doteq\ & \mathrm{in}[in^{R}_{OT}, b];\\
& \quad \text{if } \mathbf{inType}_{bit}(b) = \mathbf{true} \text{ then}\\
& \quad \text{new } r_b;\\
& \quad \text{let } c'_b = \mathbf{enc}(\mathrm{choice}[b \cdot \mathbf{zero}], r_b, ek_R);\\
& \quad \text{let } p_b = \mathbf{proof}_{bit}(c'_b, ek_R, crs_R);\\
& \quad \text{let } c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R);\\
& \quad \mathrm{out}[leak_{RS}, c_b];\\
& \mathrm{in}[in^{S}_{OT}, x_{01}];\\
& \quad \text{if } \mathbf{isPair}(x_{01}) = \mathbf{true} \text{ then}\\
& \quad \text{let } x_0 = \mathbf{first}(x_{01});\\
& \quad \text{let } x_1 = \mathbf{second}(x_{01});\\
& \quad \text{if } \mathbf{isValue}(x_0) = \mathbf{true} \text{ then}\\
& \quad \text{if } \mathbf{isValue}(x_1) = \mathbf{true} \text{ then}\\
& \quad \text{new } r, r_0, r_1;\\
& \quad \text{let } c'_x = \mathbf{eval}_{sel}(c'_b, \mathrm{choice}[x_0 \cdot \mathbf{zero}], \mathrm{choice}[x_1 \cdot \mathbf{zero}], r);\\
& \quad \text{let } d_0 = \mathbf{com}(\mathrm{choice}[x_0 \cdot \mathbf{zero}], r_0, ck_S);\\
& \quad \text{let } d_1 = \mathbf{com}(\mathrm{choice}[x_1 \cdot \mathbf{zero}], r_1, ck_S);\\
& \quad \text{let } p_x = \mathbf{proof}_{sel}(c'_x, \ldots, crs_S);\\
& \quad \text{let } c_x = \mathbf{evalPack}(c'_x, \ldots, crs_S);\\
& \quad \mathrm{out}[leak_{SR}, c_x];\\
& \mathrm{in}[infl_{SR}, \mathbf{deliver}];\\
& \quad \text{let } x_b = \mathrm{choice}[\mathbf{dec}(c'_x, dk_R) \cdot \mathbf{zero}];\\
& \quad \text{if } \mathbf{eqValue}(b, \mathbf{zero}) = \mathbf{true} \text{ then}\\
& \quad\quad \mathrm{out}\left[out^{R}_{OT}, \mathrm{choice}[x_b \cdot x_0]\right]\\
& \quad \text{else}\\
& \quad\quad \mathrm{out}\left[out^{R}_{OT}, \mathrm{choice}[x_b \cdot x_0]\right]
\end{aligned}
$$

両者の違いをchoiceで吸収：

choice[M0・M1]の形の箇所を
- すべてM0 で置き換えると実プロトコル
- すべてM1で置き換えると理想プロトコル

が得られるようにする

（Fig.101）

ProVerifに入力して検証

- その前に、暗号（コミットメント、NIZK）の性質なども記述する必要がある（実はたくさんあります）
- 停止しない場合は工夫が必要（例では準同型暗号の暗号文の書き換え回数を制限）
- 安全でない場合は（内部モデルの）攻撃例を出力

```
$ proverif -in pi honest-tagged.PI
Process:
{1}new ckS_483;
                              ⋮   （ノートPCで90秒程度）
                              ⋮
4800 rules inserted. The rule base contains 4708 rules. 661 rules in the queue.
5000 rules inserted. The rule base contains 4908 rules. 565 rules in the queue.
5200 rules inserted. The rule base contains 5108 rules. 417 rules in the queue.
5400 rules inserted. The rule base contains 5308 rules. 217 rules in the queue.
5600 rules inserted. The rule base contains 5508 rules. 17 rules in the queue.
RESULT Observational equivalence is true (bad not derivable).
```

停止しない場合の工夫(“タグ付け”)

準同型暗号の暗号文の書換規則(eval_sel関数)

書き換えられた暗号文は更に書換可能だが、
そのせいでProVerifが停止しなくなる(無限ループ)

```
private reduc
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, zero, xr) = enc(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, one, xr) = enc(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, two, xr) = enc(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), one, zero, xr) = enc(one, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), one, one, xr) = enc(one, xr, ek(x_dk));
                                        :
```

書き換えられた暗号文に違う記号を割り当てて
再度の書換を禁止、ProVerifが停止するように

```
private reduc
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, zero, xr) = encN(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, one, xr) = encN(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), zero, two, xr) = encN(zero, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), one, zero, xr) = encN(one, xr, ek(x_dk));
    eval_sel(enc(zero, x_rb, ek(x_dk)), one, one, xr) = encN(one, xr, ek(x_dk));
                                        :
```

まとめ

- ProVerifの紹介
  - 等式により暗号の性質を記述可能
  - 観測等価性で表される安全性を検証可能

    ☞結合可能安全性の証明に使える
- ProVeirfを用いた結合可能安全性証明
  1. 計算論的モデルでの安全性を記号モデルでの安全性に帰着(いわゆる計算論的健全性)
  2. 記号モデルでの安全性をProVerif自動検証

     (とは言うものの、結構手作業があります)