



Innovative R&D by NTT

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



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

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)



```
QR ≐ new rb; 乱数生成
in[inOTR, b]; メッセージ受信
  if inTypebit(b) = true then 条件分岐
  let c'b = enc(b, rb, ekR);
  let pb = proofbit(c'b, ekR, crsR); メッセージの操作(暗号化など)
  let cb = encPack(c'b, ekR, pb, crsR);
  out[sendRS, cb]; メッセージ送信
in[receiveSR, cx];
  if verEvalPacksel(cx) = true then
  if encOf1(cx) = cb then
  if ekOf(cx) = ekR then
  if crsOf(cb) = crsR then
  let xb = dec(encOf(cx), dkR);
  out[outOTR, xb]
```

ProVerifにおけるプロトコルの記述例(2)



プロトコル全体

```
new ckA, ckB;  
new dkA, dkB;  
let ekA = ek(dkA);  
let ekB = ek(dkB);  
new crsA, crsB;  
(QS || QAuthSR || QAuthRS || QR)
```

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)

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

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”)

$$P \stackrel{s}{\sim} Q \iff \begin{array}{l} \text{任意のプロセス } A (= \text{攻撃者}) \text{ について、} \\ A // P \text{ が通信路 } d \text{ からメッセージを出力} \\ \textit{iff} \\ A // Q \text{ が通信路 } d \text{ からメッセージを出力} \end{array}$$

※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

$$RW(Sys_{\text{real}}^{\mathcal{H}}) \stackrel{c}{\sim} RW(Sys_{\text{ideal}}^{\mathcal{H}})$$

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)

$$\mathcal{S}(Sys_{\text{real}}^{\mathcal{H}}) \stackrel{s}{\sim} \mathcal{S}(Sys_{\text{ideal}}^{\mathcal{H}})$$

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. 両者ができ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)



sender

認証付き通信路
による参加者間通信

```

 $Q_S \doteq \text{new } r, r_0, r_1;$ 
in[receiveRS, cb];
  if verEncPackbit(cb) = true then
    if ekOf(cb) = ekR then
      if crsOf(cb) = crsR then
        ← out[outSOT, getInput];
        → in[inSOT, x01];
        if isPair(x01) = true then
          let x0 = first(x01);
          let x1 = second(x01);
          if isValue(x0) = true then
            if isValue(x1) = true then
              let c'b = encOf(cb);
              let c'x = evalsel(c'b, x0, x1, r);
              let d0 = com(x0, r0, ckS);
              let d1 = com(x1, r1, ckS);
              let px = proofsel(c'x, c'b, ekR, d0, d1, ckS, crsS);
              let cx = evalPack(c'x, c'b, ..., ckS, px, crsS);
              out[sendSR, cx]
  
```

receiver

環境(攻撃者)
との入出力

```

 $Q_R \doteq \text{new } r_b;$ 
in[inROT, b];
  if inTypebit(b) = true then
    let c'b = enc(b, rb, ekR);
    let pb = proofbit(c'b, ekR, crsR);
    let cb = encPack(c'b, ekR, pb, crsR);
    out[sendRS, cb];
  in[receiveSR, cx];
  if verEvalPacksel(cx) = true then
    if encOf1(cx) = cb then
      if ekOf(cx) = ekR then
        if crsOf(cb) = crsR then
          let xb = dec(encOf(cx), dkR);
          out[outROT, xb]
  
```

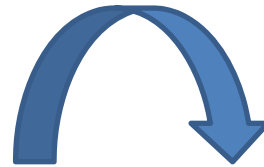


実プロトコルを変形 (Fig.99)



```
in[inOTR, b];  
  if inTypebit(b) = true then  
    new rb;  
    let c'b = enc(b, rb, ekR);  
    let pb = proofbit(c'b, ekR, crsR);  
    let cb = encPack(c'b, ekR, pb, crsR);  
    out[leakRS, cb];
```

不要なチェックを削除



```
in[inflRS, deliver];  
  if verEncPackbit(cb) = true then  
    if ekOf(cb) = ekR then  
      if crsOf(cb) = crsR then  
        out[outOTS, getInput];
```

```
in[inOTS, x01];  
  if isPair(x01) = true then  
    let x0 = first(x01);  
    let x1 = second(x01);  
    if isValue(x0) = true then  
      if isValue(x1) = true then
```

```
QleftSR ≐ in[inOTR, b];  
  if inTypebit(b) = true then  
    new rb;  
    let c'b = enc(b, rb, ekR);  
    let pb = proofbit(c'b, ekR, crsR);  
    let cb = encPack(c'b, ekR, pb, crsR);  
    out[leakRS, cb];
```

```
in[inOTS, x01];  
  if isPair(x01) = true then  
    let x0 = first(x01);  
    let x1 = second(x01);  
    if isValue(x0) = true then  
      if isValue(x1) = 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

```

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

```

 $Q_{Sim_{OT}^{SR,S}} \doteq$  new  $r, r_0, r_1$ ;
    in[ $receive_{RS}, c_b$ ];
        if verEncPackbit( $c_b$ ) = true then
            if ekOf( $c_b$ ) =  $ek_R$  then
                if crsOf( $c_b$ ) =  $crs_R$  then
                    out[ $infl_{OT}^S, \mathbf{getInput}$ ];
            in[ $leak_{OT}^S, \mathbf{xsReceived}$ ];
                let  $c'_b = \mathbf{encOf}(c_b)$ ;
                let  $c'_x = \mathbf{eval}_e(c'_b, \mathbf{zero}, \mathbf{zero}, r)$ ;
                let  $d_0 = \mathbf{com}(\mathbf{zero}, r_0, ck_S)$ ;
                let  $d_1 = \mathbf{com}(\mathbf{zero}, r_1, ck_S)$ ;
                let  $p_x = \mathbf{proof}_{sel}(c'_x, c'_b, \dots, crs_S)$ ;
                let  $c'_x = \mathbf{evalPack}_{sel}(c'_x, c'_b, \dots, p_x, crs_S)$ ;
                out[ $send_{SR}, c'_x$ ]
    
```

シミュレータ
(senderをシミュレート)

```

 $Q_{Sim_{OT}^{SR,R}} \doteq$  new  $r_b$ ;
    in[ $leak_{OT}^R, \mathbf{bReceived}$ ];
        let  $c'_b = \mathbf{enc}(\mathbf{zero}, r_b, ek_R)$ ;
        let  $p_b = \mathbf{proof}_{bit}(c'_b, ek_R, crs_R)$ ;
        let  $c_b = \mathbf{encPack}(c'_b, ek_R, p_b, crs_R)$ ;
        out[ $send_{RS}, c_b$ ];
    in[ $receive_{SR}, c_x$ ];
        if verEvalPacksel( $c_x$ ) = true then
            if encOf1( $c_x$ ) =  $c'_b$  then
                if ekOf( $c_x$ ) =  $ek_R$  then
                    if ckOf( $c_x$ ) =  $ck_S$  then
                        if crsOf( $c_x$ ) =  $crs_S$  then
                            out[ $infl_{OT}^R, \mathbf{finish}$ ]
    
```

シミュレータ
(receiverをシミュレート)

理想プロトコルを変形 (Fig.100)



```

in[inOTR, b];
  if inTypebit(b) = true then
    new rb;
    let c'b = enc(zero, rb, ekR);
    let pb = proofbit(cb, ekR, crsR);
    let cb = encPack(c'b, ekR, pb, crsR);
    out[leakRS, cb];
  
```



不要なチェックを削除

```

in[inRSR, deliver]
  if verEncPackbit(cb) = true then
    if ekOf(cb) = ekR then
      if crsOf(cb) = crsR then
        out[outOTS, getInput];
  
```

```

QrightSR ≡ in[inOTR, b];
  if inTypebit(b) = true then
    new rb;
    let c'b = enc(zero, rb, ckR);
    let pb = proofbit(cb, ekR, crsR);
    let cb = encPack(c'b, ekR, pb, crsR);
    out[leakRS, cb];
  
```

```

in[inOTS, x01];
  if isPair(x01) = true then
    let x0 = first(x01);
    let x1 = second(x01);
    if isValue(x0) = true then
      if isValue(x1) = true then
        new r, r0, r1;
        let c'b = encOf(cb);
        let c'x = evalc(c'b, zero, zero, r);
        let d0 = com(zero, r0, ckS);
        let d1 = com(zero, r1, ckS);
        let px = proofsel(c'x, ..., crsS);
        let cx = evalPack(c'x, ..., crsS);
        out[leakSR, cx];
  
```

削除

```

in[inSRR, deliver];
  if verEvalPacksel(cx) = true then
    if encOf1(cx) = c'b then
      if ekOf(cx) = ekR then
        if ckOf(cx) = ckS then
          if crsOf(cx) = crsS then
            if eqValue(b, zero) = true then
              out[outOTR, x0]
            else
              out[outOTR, x1]
  
```

```

in[inOTS, x01];
  if isPair(x01) = true then
    let x0 = first(x01);
    let x1 = second(x01);
    if isValue(x0) = true then
      if isValue(x1) = true then
        new r, r0, r1;
        let c'x = evalc(c'b, zero, zero, r);
        let d0 = com(zero, r0, ckS);
        let d1 = com(zero, r1, ckS);
        let px = proofsel(c'x, ..., crsS);
        let cx = evalPack(c'x, ..., crsS);
        out[leakSR, cx];
  in[inSRR, deliver];
  if eqValue(b, zero) = true then
    out[outOTR, x0]
  else
    out[outOTR, x1]
  
```

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



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

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));
:
```



- 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
- ProVerif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)