

CryptoVerif ハンズオン

FWS 運営委員会*

Contents

1 はじめに	1
2 準備	3
2.1 オンラインデモの使い方	3
2.2 予備知識	4
2.2.1 暗号の安全性	4
2.2.2 ゲーム列による安全性証明	4
3 CryptoVerif を用いた暗号システムの安全性検証	5
3.1 暗号システムの記述	6
3.1.1 暗号部品を書いてみよう	6
3.1.2 暗号システムを書いてみよう	7
3.2 安全性証明の記述	8
3.3 実行と実行結果	9
3.4 うまくいかない例	12
3.5 うまくいく例	14
4 応用編：Enc-then-MAC が IND-CCA2 を満たすこと	21
5 おわりに	23

1 はじめに

CryptoVerif [Blaa] は暗号プロトコルの形式検証ツールです。ゲーム変換を用いた安全性証明を支援するツールで、計算論的モデルに基づく安全性証明を（半）自動的に生成することができます（図 1）。ProVerif [Blab] や Tamarin Prover [BCDS] と比べてちょっと複雑ですが、その分示せることも強力です。

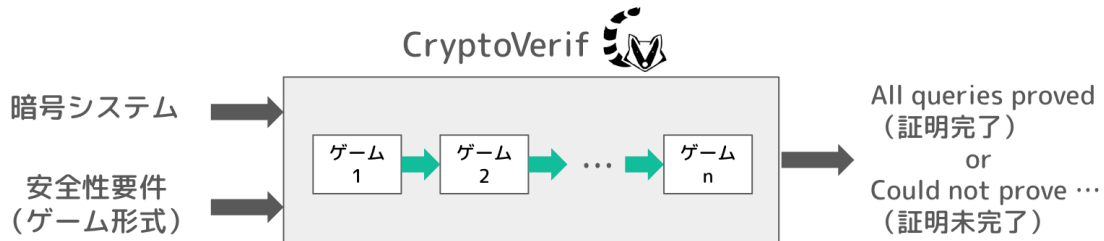


図 1: CryptoVerif のイメージ。アイコンは CryptoVerif 公式ページ [Blaa] より。

このハンズオンでは、CryptoVerif の簡単な使い方を習得することをゴールとします。具体的には、CryptoVerif における

*中林美郷（NTT），花谷嘉一（東芝），米山一樹（茨城大学）

- 暗号システムの記述方法
- 安全性モデルの記述方法
- 検証結果の読み方

の基礎を理解してもらうことをゴールとします。時間の都合上、細かい文法や記述方法については扱いません。興味を持っていただけた方は CryptoVerif の公式マニュアル [BCJ24] を参照してください。

なお、このハンズオンは、Marc Hafner 氏によるチュートリアル [Haf] を参考に作成しました。本テキストで扱うコードは FWS の Web ページ

- <https://www.iwsec.org/fws/2025/>

から入手可能です。

注意

本ハンズオンでは、**厳密さよりもわかりやすさと参考元チュートリアルへの準拠を優先しております**。何卒ご理解ください。暗号に関する厳密な定義や議論については、例えば [大陵龍 11] などをご参照ください。

2 準備

2.1 オンラインデモの使い方

このハンズオンでは CryptoVerif のオンラインデモ [BBC⁺] を使用します。

- <http://proverif24.paris.inria.fr/cryptoverif.php>

図 2: CryptoVerif オンラインデモの画面。

右側のフォームにコードを入力し、Verify ボタンを押すと検証が始まり、検証結果が出力されます。検証結果の===== Proof starts =====以下の部分にゲームの変換列が表示され、最終行に証明が完了したかどうかが表示されます(図 3)。証明が完了すると All queries proved と表示され、そうでない場合は Could not prove ... と表示されます。(注意: 証明が完了しなかった=安全性を満たさない、というわけではありません。あくまでも証明が完了しなかっただけです。)

図 3: CryptoVerif の出力の例。この例では証明が完了していません。

証明が完了しなかった場合であっても、CryptoVerif は ProVerif や Tamarin のように攻撃を導出することはできません。ですが、証明に失敗した際の出力が攻撃の発見に役立つこともあります。

2.2 予備知識

ここでは予備知識として暗号方式のゲームによる安全性定義とゲーム列の変換による安全性紹介について簡単に紹介します。

以下では次の2つの暗号部品が登場します。

- **共通鍵暗号**：暗号化と復号に同じ鍵を用いる暗号です。暗号化関数 enc と復号関数 dec から成ります。
 - enc (平文, 鍵) $\rightarrow$ 暗号文
 - dec (暗号文, 鍵) $\rightarrow$ 平文 or $\perp$ (復号失敗)
- **メッセージ認証コード (MAC)**：メッセージの改ざんの検出などに用いられる認証子です。MAC 生成関数 mac と MAC 検証関数 verify から成り、ハッシュ関数などを用いて実現されます。
 - mac (文, MAC 鍵) $\rightarrow$ MAC
 - verify (文, MAC 鍵, MAC) $\rightarrow$ $\top$ (認証成功) or $\perp$ (認証失敗)

2.2.1 暗号の安全性

暗号システムの安全性は挑戦者と攻撃者間のゲームで表現することができます。例えば、IND-CPA (INDistinguishability-Chosen Plaintext Attack) ゲームは次の図 8 のように表現されます。攻撃者は

IND-CPAゲーム

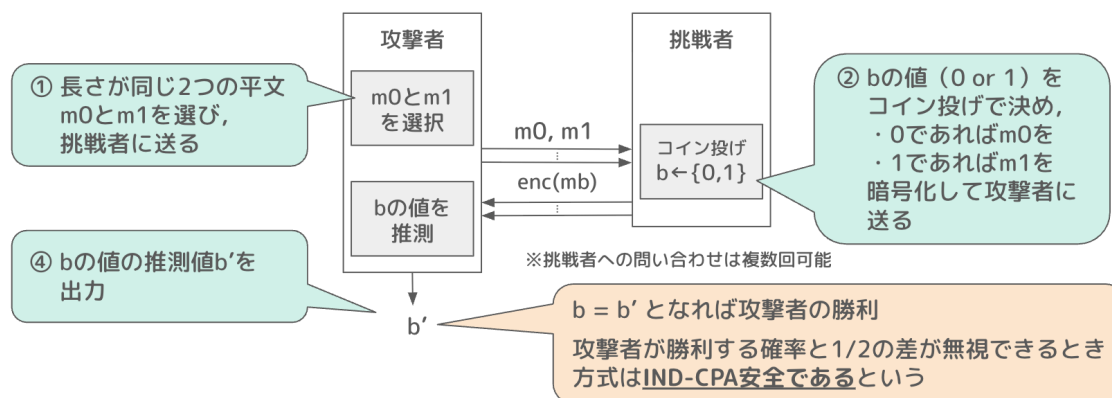


Figure 4: IND-CPA ゲーム。(注意：このハンズオンでは、わかりやすさと参考元チュートリアルへの準拠を優先してこのゲームを IND-CPA ゲームと呼びます。)

2つの平文 m_0 と m_1 を選び、挑戦者に送ります。それを受け取った挑戦者は変数 b の値 (0 or 1) をコイン投げをして決め、0 であれば m_0 を、1 であれば m_1 を暗号化して、その暗号文を攻撃者に送ります。このやり取りは複数回繰り返すことができます。そして、攻撃者は挑戦者から送られていた情報を元に b の値を推測し、その推測が当たれば攻撃者の勝利と成ります。攻撃者が勝利する確率と $1/2$ の差が無視できるとき (すなわち攻撃者の推測が成功する確率が当てずっぽうと変わらないとき)、方式は IND-CPA 安全であると言います。(注意：このハンズオンでは、わかりやすさと参考元チュートリアルへの準拠を優先してこのゲームを IND-CPA ゲームと呼びます。)

2.2.2 ゲーム列による安全性証明

暗号システムが安全性を満たしていることは、ゲームを変換することで証明することができます。図 5 のように、示したい安全性のゲーム表現である初期ゲームから始めて、攻撃者が勝利する確率がほぼ変わらない範囲でゲームを変換していき、少しずつ理想的なゲーム (攻撃者が勝利する確率が明らかに十分小さいゲーム) に近づけていきます。最終的に理想的なゲームに変換ができれば、初期ゲームでも攻撃者が勝利する確率が十分に小さいことを示すことができます。

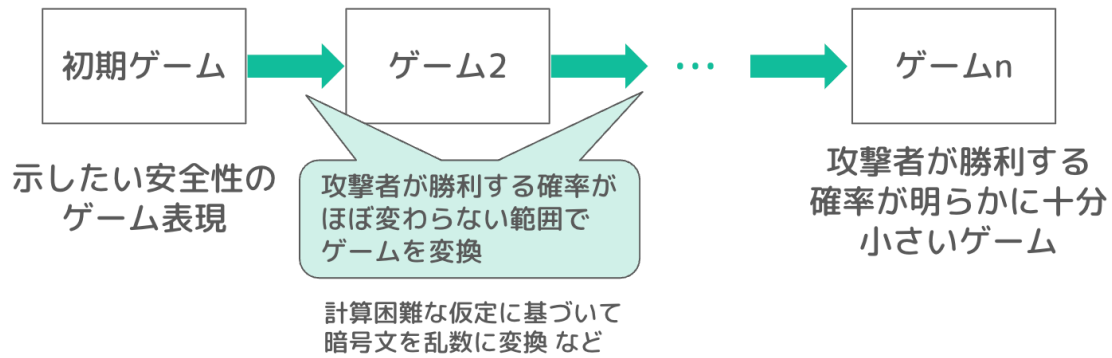


Figure 5: ゲーム列による安全性証明のイメージ.

3 CryptoVerifを用いた暗号システムの安全性検証

まず, CryptoVerif のコードの全体像を見てみましょう. 次のコード 1 をご覧ください.

Code 1: Enc-and-MAC の IND-CPA 安全性の一部 (1_Enc-and-MAC_IND-CPA_part.cv)

```

1 (* Encrypt-and-MAC is IND-CPA *)
2 param qEnc.
3
4 type mkey [fixed].
5 type key [fixed].
6 type macs [fixed].
7
8 (* Shared-key encryption (CPA Stream cipher) *)
9 proba Penc.
10 expand IND_CPA_sym_enc(key, bitstring, bitstring, enc, dec, injbot, Z, Penc).
11
12 (* Mac *)
13 proba Pmac.
14 expand SUF_CMA_det_mac(mkey, bitstring, macs, mac, verify, Pmac).
15
16 fun concat(bitstring, macs): bitstring [data].
17
18 letfun full_enc(m: bitstring, k: key, mk: mkey) =
19   enc(m, k).
20
21 (* Queries *)
22 query secret b.
23
24 let QencLR(b0: bool, k: key, mk: mkey) =
25   foreach i <= qEnc do
26     Oenc (m0: bitstring, m1: bitstring) :=
27       if Z(m0) = Z(m1) then (* m0 and m1 have the same length *)
28         mb <- if b0 then m0 else m1;
29         return(full_enc(mb, k, mk)).
30
31 process
32   Ostart() :=
33     b <-R bool;
34     k <-R key;
35     mk <-R mkey;
36     return;
37     run QencLR(b, k, mk)

```

CryptoVerif の入力コードの拡張子は.cv です. このコードは, この後説明する Enc-and-MAC の IND-CPA 安全性に関するものです.

ざっくり分けると, CryptoVerif のコードは

- 宣言部 (1 行-6 行)

- 暗号システム部（8 行-19 行）
- 安全性証明部（21 行-37 行）

から構成されます。宣言部ではパラメータの宣言や型の宣言を行い、暗号システム部では暗号部品と部品が満たす安全性、証明したい暗号システムを記述します。安全性証明部では安全性要件と安全性ゲームを記述します。

特に、宣言部の 2 行目ではパラメータ `qEnc` を宣言しています。 `param` はセキュリティパラメータに依存する定数を宣言するコマンドで、後ほど攻撃者が呼び出すオラクルの最大回数として用いられます。では改めまして、はじめに暗号システムの記述について見ていきましょう。

3.1 暗号システムの記述

例として、図 6 の Enc-and-MAC を考えてみます。

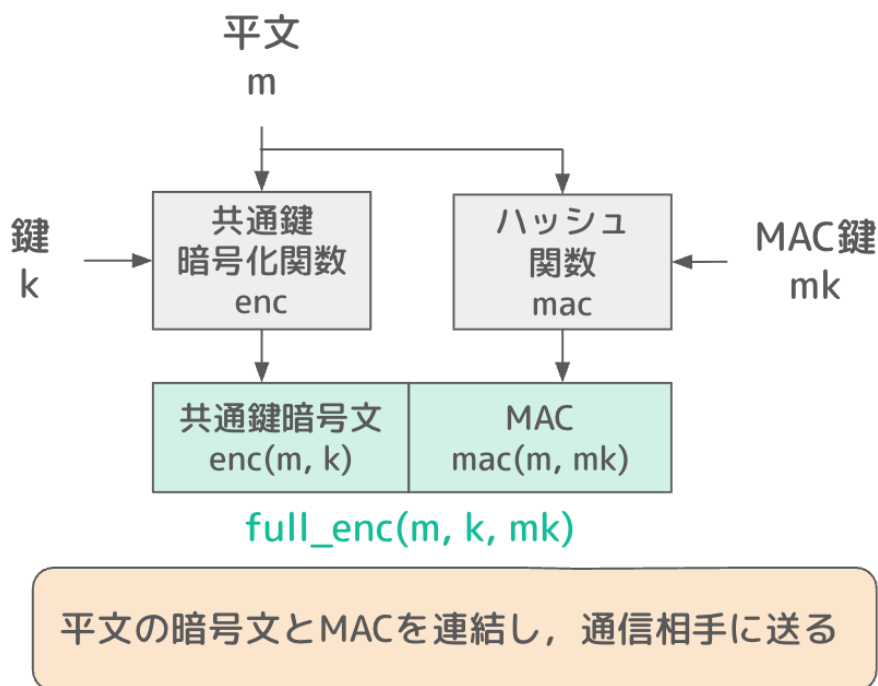


Figure 6: Enc-and-MAC. 平文の共通鍵暗号文と MAC を連結したものを通信相手に送ります。

この方式を記述するために、まず暗号部品として共通鍵暗号と MAC を呼び出します。

3.1.1 暗号部品を書いてみよう

CryptoVerif では、標準でいくつかの安全性仮定を満たす暗号部品が用意されています（マニュアル [BCJ24] の Section 6 参照）。ここでは、その中から IND-CPA を満たす共通鍵暗号と SUF-CMA を満たす MAC を呼び出します。

```

8 (* IND-CPA Symmetric Encryption *)
9 proba Penc.
10 expand IND_CPA_sym_enc(key, bitstring, bitstring, enc, dec, injbot, Z, Penc).
11
12 (* SUF-CMA MAC *)
13 proba Pmac.
14 expand SUF_CMA_det_mac(mkey, bitstring, macs, mac, verify, Pmac).
```

では、1 行ずつ見ていきましょう。9 行目の `proba ...` は、確率的なイベント（probability event）を宣言する構文です。「攻撃者が IND-CPA を破る確率」として使うために `Penc` という名前の確率的イベントを宣言しています。3 行目では標準ライブラリにある IND-CPA 安全な共通鍵暗号の定義を展開しています。引数は、左から

- `key`: 鍵の型
- `bitstring`: 平文の型
- `bitstring`: 暗号文の型
- `enc`: 暗号化関数
- `dec`: 復号関数
- `injb0t`: $\perp$ (復号エラー) を返す関数
- `Z`: 平文に対して, 長さが等しい 0 のみの文字列を返す関数
- `Penc`: 攻撃者が IND-CPA を破る確率のイベント

です. この行では IND-CPA 安全な共通鍵暗号の関数ペア `enc:bitstring  $\times$  key  $\rightarrow$  bitstring`, `dec:bitstring  $\times$  key  $\rightarrow$  bitstring or  $\perp$` を宣言しています. 関数 `Z` は平文の長さを比較するために用いられます.

次は MAC です. 13 行目では 9 行目と同様に, 「攻撃者が SUF-CMA を破る」確率的イベント `Pmac` を宣言しています. 14 行目では, 標準ライブラリにある SUF-CMA 安全な MAC の定義を展開しています. 引数は左から

- `mkey`: 鍵の型
- `bitstring`: 入力値の型
- `macs`: MAC (出力値) の型
- `mac`: MAC 関数
- `verify`: MAC が正しければ $\top$, 正しくなければ $\perp$ を返す関数
- `Pmac`: 攻撃者が SUF-CMA を破る確率のイベント

です. SUF-CMA 安全な MAC の関数ペア `mac: bitstring  $\times$  mkey  $\rightarrow$  macs`, `verify: bitstring  $\times$  mkey  $\times$  macs  $\rightarrow$   $\top$  or  $\perp$` を宣言しています.

演習 1 オンラインデモを使ってみよう

1.Enc-and-MAC.IND-CPA.part.cv をオンラインデモに入力し, 結果を見てみましょう.

3.1.2 暗号システムを書いてみよう

これで必要な暗号部品が揃いました. 次に Enc-and-MAC のプロトコルを構成していきます. `CryptoVerif` では, OCaml 風の関数型言語でプロトコルを記述します.

まず, Enc-and-MAC の構成のために文字列の連結のための関数を定義します.

```
1 fun concat(bitstring, macs): bitstring [data].
```

関数 `concat` は `bitstring` 型と `macs` 型の入力を受け取り, `bitstring` 型の出力を返します. ここで, 最後の `[data]` は, この関数が単射であり, 逆関数が効率的に計算できることを意味するオプションです. このオプションを付けることによって, 出力から入力が計算できるという文字列の連結関数の性質を付与しています.

この関数 `concat` を用いて, Enc-and-MAC を構成します.

演習 2 *Enc-and-MAC* を構成しよう

2.Enc-and-MAC_IND-CPA_fill.cv の ??? 部分を埋めて、Enc-and-MAC を完成させましょう。???部分は平文 m 、鍵 k 、MAC 鍵 mk を受け取り、暗号文と MAC の連結を返す関数 `full_enc` です。要素の連結は 16 行目で定義している `concat` 関数を使ってください。

Code 2: 2.Enc-and-MAC_IND-CPA_fill.cv

```
1 (* Encrypt-and-MAC is IND-CPA *)
2 param qEnc.
3
4 type mkey [fixed].
5 type key [fixed].
6 type macs [fixed].
7
8 (* Shared-key encryption (CPA Stream cipher) *)
9 proba Penc.
10 expand IND_CPA_sym_enc(key, bitstring, bitstring, enc, dec, injbot, Z, Penc).
11
12 (* Mac *)
13 proba Pmac.
14 expand SUF_CMA_det_mac(mkey, bitstring, macs, mac, verify, Pmac).
15
16 fun concat(bitstring, macs): bitstring [data].
17
18 letfun full_enc(m: bitstring, k: key, mk: mkey) =
19   ??? .
20
21 (* Queries *)
22 query secret b.
23
24 let QencLR(b0: bool, k: key, mk: mkey) =
25   foreach i <= qEnc do
26     Oenc (m0: bitstring, m1: bitstring) :=
27       if Z(m0) = Z(m1) then (* m0 and m1 have the same length *)
28         mb <- if b0 then m0 else m1;
29         return(full_enc(mb, k, mk)).
30
31 process
32   Ostart() :=
33     b <-R bool;
34     k <-R key;
35     mk <-R mkey;
36     return;
37   run QencLR(b, k, mk)
```

Enc-and-MAC は次のように表現できます。

```
18 letfun full_enc(m: bitstring, k: key, mk: mkey) =
19   concat(enc(m, k), mac(m, mk)).
```

この宣言で、Enc-then-MAC の暗号文を作る関数 `full_enc` を定義しています。平文 m と共通鍵 k 、MAC 用鍵 mk を受け取り、`concat(enc(m, k), mac(m, mk))` を返します。宣言文はピリオド. で終わります。ここでは Enc-then-MAC を関数として定義しましたが、安全性ゲームの中でプロトコルの構成を定義することもできます。

3.2 安全性証明の記述

次に安全性ゲームを見ていきましょう。Code 1 と 2 では既に IND-CPA 安全性ゲームが記述されていました。各行について説明していきます。

まず、安全性ゲームのゴールとして

```
21 (* Queries *)
22 query secret b.
```

を記述します。これは、攻撃者は b （安全性ゲームのコイン投げ部分）に関する情報を持たない、という意味を持ちます。「攻撃者が b の値を当てる確率は $1/2$ 」と同値です。

次に、オラクルを記述します。

```
24 let QencLR(b0: bool, k: key, mk: mkey) =
25   foreach i <= qEnc do
26     Oenc (m0: bitstring, m1: bitstring) :=
27       if Z(m0) = Z(m1) then (* m0 and m1 have the same length *)
28         mb <- if b0 then m0 else m1;
29       return(full_enc(mb, k, mk)).
```

これはゲームの中で攻撃者がメッセージのペア (m_0, m_1) を出したときに、 $b = 0$ ならば m_0 の暗号文を、 $b = 1$ ならば m_1 の暗号文を返すようなオラクルです。引数としてコイン投げの値 b と暗号化鍵 k 、MAC 鍵 mk を受け取り、 b の値に応じて暗号文 `full_enc(m0, k, mk)` または `full_enc(m1, k, mk)` を返します。25 行目は攻撃者がオラクルを呼び出せる回数の上限を指定しています。また、IND-CPA では m_0 と m_1 のメッセージの長さが同じことが前提となっているため、27 行目でそれらの長さが等しいことを要請しています。

そして、安全性証明のための初期ゲーム (initial game) を記述します。

```
31 process
32   Ostart() :=
33     b <-R bool;
34     k <-R key;
35     mk <-R mkey;
36     return;
37   run QencLR(b, k, mk)
```

CryptoVerif では、`process` はゲームのはじまりを意味します。2.2.1 節の IND-CPA ゲームの通り、 b と鍵 k 、 mk を一様ランダムに決め、それらを用いてひとつ前で定義したオラクルを呼び出します。

3.3 実行と実行結果

演習 3 安全性ゲームを入力してみよう

演習 2 で作ったコードをオンラインデモに入力し、結果を見てみましょう。（参考コードは `3.Enc-and-MAC_IND-CPA.cv` にあります。）

出力結果を見てみると、結構長いようです。まずは何も考えず一番下までスクロールしていただいて、一番最後の行を見てみましょう。

```
1 RESULT Could not prove secrecy of b.
```

とあります。どうやら証明できなかったようです。CryptoVerif は ProVerif のように攻撃を導出することはできませんが、証明に失敗した際の出力が具体的な攻撃を見つけるのに役立つ場合もあります。では、出力を見てみましょう。

```
1 ===== Proof starts =====
2 Initial state
3 Game 1 is
4   Ostart() :=
5     b <-R bool;
6     k_3 <-R key;
7     mk_2 <-R mkey;
8     return();
9     foreach i <= qEnc do
10      Oenc(m0: bitstring, m1: bitstring) :=
11        if Z(m0) = Z(m1) then
12          mb: bitstring <- (if b then m0 else m1);
13          return((m: bitstring <- mb; concat((m_1: bitstring <- m; k_2: key <- k_3; r <-R
              enc_seed; enc_r(m_1, k_2, r)), mac(m, mk_2))))
14
15
16 Applying expand
17   - Expand if/find/let
18 yields
```

```

19
20 Game 2 is
21   Ostart() :=
22     b <-R bool;
23     k_3 <-R key;
24     mk_2 <-R mkey;
25     return();
26     foreach i <= qEnc do
27       Oenc(m0: bitstring, m1: bitstring) :=
28         if Z(m0) = Z(m1) then
29           if b then
30             mb: bitstring <- m0;
31             m: bitstring <- mb;
32             m_1: bitstring <- m;
33             k_2: key <- k_3;
34             r <-R enc_seed;
35             return(concat(enc_r(m_1, k_2, r), mac(m, mk_2)))
36           else
37             mb: bitstring <- m1;
38             m: bitstring <- mb;
39             m_1: bitstring <- m;
40             k_2: key <- k_3;
41             r <-R enc_seed;
42             return(concat(enc_r(m_1, k_2, r), mac(m, mk_2)))
43
44
45 Applying remove assignments of findcond
46 - Remove assignments on mb (definition removed, all usages removed)
47 - Remove assignments on m (definition removed, all usages removed)
48 - Remove assignments on m_1 (definition removed, all usages removed)
49 - Remove assignments on k_2 (definition removed, all usages removed)
50 - Remove assignments on mb (definition removed, all usages removed)
51 - Remove assignments on m (definition removed, all usages removed)
52 - Remove assignments on m_1 (definition removed, all usages removed)
53 - Remove assignments on k_2 (definition removed, all usages removed)
54 yields
55
56 Game 3 is
57   Ostart() :=
58     b <-R bool;
59     k_3 <-R key;
60     mk_2 <-R mkey;
61     return();
62     foreach i <= qEnc do
63       Oenc(m0: bitstring, m1: bitstring) :=
64         if Z(m0) = Z(m1) then
65           if b then
66             r <-R enc_seed;
67             return(concat(enc_r(m0, k_3, r), mac(m0, mk_2)))
68           else
69             r <-R enc_seed;
70             return(concat(enc_r(m1, k_3, r), mac(m1, mk_2)))
71
72
73 Applying SA rename new without array accesses and remove assignments of findcond
74 - Rename variable r into r_2, r_1
75 yields
76
77 Game 4 is
78   Ostart() :=
79     b <-R bool;
80     k_3 <-R key;
81     mk_2 <-R mkey;
82     return();
83     foreach i <= qEnc do
84       Oenc(m0: bitstring, m1: bitstring) :=
85         if Z(m0) = Z(m1) then
86           if b then
87             r_2 <-R enc_seed;

```

```

88     return(concat(enc_r(m0, k_3, r_2), mac(m0, mk_2)))
89   else
90     r_1 <-R enc_seed;
91     return(concat(enc_r(m1, k_3, r_1), mac(m1, mk_2)))
92
93
94 Applying equivalence ind_cpa(enc) [probability Penc(time_1, qEnc, max(maxlength(game 4:
    m0), maxlength(game 4: m1)))]
95 - Equivalence ind_cpa(enc) with variables: r_2 -> r_1, k_3 -> k_2, r_1 -> r_1
96 yields
97
98 Game 5 is
99   Ostart() :=
100   b <-R bool;
101   k_4 <-R key;
102   mk_2 <-R mkey;
103   return();
104   foreach i <= qEnc do
105   Oenc(m0: bitstring, m1: bitstring) :=
106   if Z(m0) = Z(m1) then
107   if b then
108     r_4 <-R enc_seed;
109     return(concat((x_1: bitstring <- m0; enc_r'(Z(x_1), k_4, r_4)), mac(m0, mk_2)))
110   else
111     r_3 <-R enc_seed;
112     return(concat((x: bitstring <- m1; enc_r'(Z(x), k_4, r_3)), mac(m1, mk_2)))
113
114
115 Applying expand
116 - Expand if/find/let
117 yields
118
119 Game 6 is
120   Ostart() :=
121   b <-R bool;
122   k_4 <-R key;
123   mk_2 <-R mkey;
124   return();
125   foreach i <= qEnc do
126   Oenc(m0: bitstring, m1: bitstring) :=
127   if Z(m0) = Z(m1) then
128   if b then
129     r_4 <-R enc_seed;
130     x_1: bitstring <- m0;
131     return(concat(enc_r'(Z(x_1), k_4, r_4), mac(m0, mk_2)))
132   else
133     r_3 <-R enc_seed;
134     x: bitstring <- m1;
135     return(concat(enc_r'(Z(x), k_4, r_3), mac(m1, mk_2)))
136
137
138 Applying remove assignments of findcond
139 - Remove assignments on x (definition removed, all usages removed)
140 - Remove assignments on x_1 (definition removed, all usages removed)
141 yields
142
143 Game 7 is
144   Ostart() :=
145   b <-R bool;
146   k_4 <-R key;
147   mk_2 <-R mkey;
148   return();
149   foreach i <= qEnc do
150   Oenc(m0: bitstring, m1: bitstring) :=
151   if Z(m0) = Z(m1) then
152   if b then
153     r_4 <-R enc_seed;
154     return(concat(enc_r'(Z(m0), k_4, r_4), mac(m0, mk_2)))
155   else

```

```

156     r_3 <-R enc_seed;
157     return(concat(enc_r'(Z(m1), k_4, r_3), mac(m1, mk_2)))
158
159 RESULT time_1 = qEnc * time(= bitstring, length(Z, maxlength(game 4: m0)), length(Z,
    maxlength(game 4: m1))) + qEnc * time(Z, maxlength(game 4: m1)) + qEnc * time(Z,
    maxlength(game 4: m0)) + qEnc * time(concat, length(enc_r, maxlength(game 4: m0)))
    + qEnc * time(mac, maxlength(game 4: m0)) + qEnc * time(concat, length(enc_r,
    maxlength(game 4: m1))) + qEnc * time(mac, maxlength(game 4: m1)) + time
160 RESULT Could not prove secrecy of b.

```

CryptoVerifは自動でゲーム変換を行ってくれます。この入力に対しては、初期ゲーム（Game 1）から始まって Game 7 まで変換したようです。では、それぞれ詳しく見ていきましょう。

3.4 うまくいかない例

証明に失敗した際の出力を見ていきます。それぞれのゲーム変換と、なぜうまくいかなかったのかを見ていきましょう。

ゲーム 1 からゲーム 2 へ

ゲーム 1 は初期ゲームです。

```

1 Game 1 is
2   Ostart() :=
3   b <-R bool;
4   k_3 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  mb: bitstring <- (if b then m0 else m1);
11  return((m: bitstring <- mb; concat((m_1: bitstring <- m; k_2: key <- k_3; r <-R
    enc_seed; enc_r(m_1, k_2, r)), mac(m, mk_2))))

```

10 行目の b の値による分岐を展開し、次のゲーム 2 に変換されます。もちろん、この変換で攻撃者の勝利する確率は変わりません。

```

1 Game 2 is
2   Ostart() :=
3   b <-R bool;
4   k_3 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  if b then
11    mb: bitstring <- m0;
12    m: bitstring <- mb;
13    m_1: bitstring <- m;
14    k_2: key <- k_3;
15    r <-R enc_seed;
16    return(concat(enc_r(m_1, k_2, r), mac(m, mk_2)))
17  else
18    mb: bitstring <- m1;
19    m: bitstring <- mb;
20    m_1: bitstring <- m;
21    k_2: key <- k_3;
22    r <-R enc_seed;
23    return(concat(enc_r(m_1, k_2, r), mac(m, mk_2)))

```

ゲーム 2 からゲーム 3 へ

ゲーム 2 から、不要な一時変数 mb, m, m_1, k_2 を削除し、次のゲーム 3 が得られます。この変換もゲームの中身は変わっていないので、攻撃者の勝利する確率は変わりません。

```

1 Game 3 is
2   Ostart() :=
3     b <-R bool;
4     k_3 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10           if b then
11             r <-R enc_seed;
12             return(concat(enc_r(m0, k_3, r), mac(m0, mk_2)))
13           else
14             r <-R enc_seed;
15             return(concat(enc_r(m1, k_3, r), mac(m1, mk_2)))

```

ゲーム 3 からゲーム 4 へ

ゲーム 3 の乱数 r の名前を変更し、次のゲーム 4 が得られます。こちらも同様にゲームの中身は変わっていないので、攻撃者の勝利する確率は変わりません。

```

1 Game 4 is
2   Ostart() :=
3     b <-R bool;
4     k_3 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10           if b then
11             r_2 <-R enc_seed;
12             return(concat(enc_r(m0, k_3, r_2), mac(m0, mk_2)))
13           else
14             r_1 <-R enc_seed;
15             return(concat(enc_r(m1, k_3, r_1), mac(m1, mk_2)))

```

ゲーム 4 からゲーム 5 へ

ゲーム 4 の暗号文の部分を実数暗号文に差し替えて、次のゲーム 5 が得られます。共通鍵暗号が IND-CPA であることからこの変換を行うことができます。最終的な安全性評価に、攻撃者が共通鍵暗号を破る確率である Penc が計上されます。

```

1 Game 5 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10           if b then
11             r_4 <-R enc_seed;
12             return(concat((x_1: bitstring <- m0; enc_r'(Z(x_1), k_4, r_4)), mac(m0, mk_2)))
13           else
14             r_3 <-R enc_seed;
15             return(concat((x: bitstring <- m1; enc_r'(Z(x), k_4, r_3)), mac(m1, mk_2)))

```

ここで、12 行目と 15 行目の enc_r' は平文の長さが等しい実数暗号文を出力する関数です。

ゲーム 5 からゲーム 6 へ

ゲーム 5 の代入の順序を入れ替えて、ゲーム 6 が得られます。こちらもゲームの中身は変わらないため、攻撃者の勝利する確率は変わりません。

```

1 Game 6 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10          if b then
11            r_4 <-R enc_seed;
12            x_1: bitstring <- m0;
13            return(concat(enc_r'(Z(x_1), k_4, r_4), mac(m0, mk_2)))
14          else
15            r_3 <-R enc_seed;
16            x: bitstring <- m1;
17            return(concat(enc_r'(Z(x), k_4, r_3), mac(m1, mk_2)))

```

ゲーム 6 からゲーム 7 へ

さらに不要な一時変数を削除して、次のゲーム 7（最終ゲーム）が得られます。

```

1 Game 7 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10          if b then
11            r_4 <-R enc_seed;
12            return(concat(enc_r'(Z(m0), k_4, r_4), mac(m0, mk_2)))
13          else
14            r_3 <-R enc_seed;
15            return(concat(enc_r'(Z(m1), k_4, r_3), mac(m1, mk_2)))

```

戻り値にメッセージ（ m_0 または m_1 ）の情報が含まれているため、攻撃者は暗号文オラクルを用いて b の値が 0 なのか 1 なのかを容易に推測することができます。そのため証明が失敗し、

```

1 RESULT Could not prove secrecy of b.

```

という結果が返ってきたようです。

3.5 うまくいく例

では、CryptoVerif の出力を基にプロトコルを IND-CPA 安全に修正していきましょう。出力を見ると、出力にメッセージの MAC がそのまま入っていることが証明の失敗の主な原因のようです。

演習 4 システムを IND-CPA 安全にしよう

先ほどのコードのシステム部分を修正して、IND-CPA 安全を満たすようにしましょう。出力の最後が All queries proved. となれば成功です。

参考コード（穴埋め前）は 4_Enc-and-MAC_IND-CPA_secure.cv にあります。

出力にメッセージの MAC がそのまま含まれていることが攻撃者による b の推測につながっていたので、メッセージを暗号文に置き換えるとよさそうです。次の図 7 のように、暗号化してから MAC に入力する方式（Enc-then-MAC）を考えてみましょう。（修正の方法はもちろんこの限りではありません。）

Enc-then-MAC は次のように記述できます。

```

18 letfun full_enc(m: bitstring, k: key, mk: mkey) =
19   concat(enc(m, k), mac(enc(m, k), mk)).

```

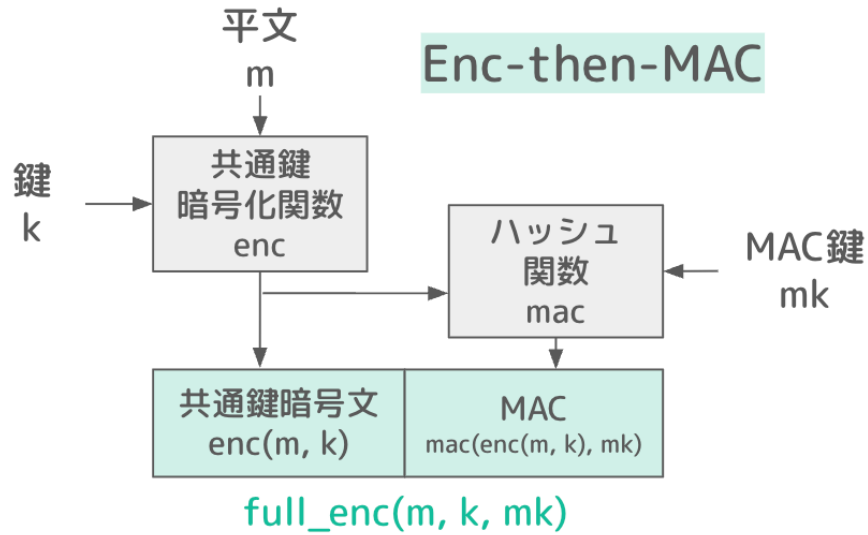


Figure 7: Enc-then-MAC. 平文の共通鍵暗号文とその暗号文の MAC を連結したものを通信相手に送ります。

Enc-and-MAC では `concat(enc(m, k), mac(m, mk))` と書いていましたが, `mac` の第一引数がメッセージから暗号文になりました. これを `CryptoVerif` に入力してみましょう.
出力は次のようになるはずです.

```

1 ===== Proof starts =====
2 Initial state
3 Game 1 is
4   Ostart() :=
5   b <-R bool;
6   k_3 <-R key;
7   mk_2 <-R mkey;
8   return();
9   foreach i <= qEnc do
10  Oenc(m0: bitstring, m1: bitstring) :=
11  if Z(m0) = Z(m1) then
12  mb: bitstring <- (if b then m0 else m1);
13  return((m: bitstring <- mb; concat((r_1 <-R enc_seed; enc_r(m, k_3, r_1)), mac((
    r_2 <-R enc_seed; enc_r(m, k_3, r_2)), mk_2))))
14
15
16 Applying expand
17   - Expand if/find/let
18 yields
19
20 Game 2 is
21   Ostart() :=
22   b <-R bool;
23   k_3 <-R key;
24   mk_2 <-R mkey;
25   return();
26   foreach i <= qEnc do
27   Oenc(m0: bitstring, m1: bitstring) :=
28   if Z(m0) = Z(m1) then
29   if b then
30     mb: bitstring <- m0;
31     m: bitstring <- mb;
32     r_1 <-R enc_seed;
33     r_2 <-R enc_seed;
34     return(concat(enc_r(m, k_3, r_1), mac(enc_r(m, k_3, r_2), mk_2)))
35   else
36     mb: bitstring <- m1;
37     m: bitstring <- mb;

```

```

38     r_1 <-R enc_seed;
39     r_2 <-R enc_seed;
40     return(concat(enc_r(m, k_3, r_1), mac(enc_r(m, k_3, r_2), mk_2)))
41
42
43 Applying remove assignments of findcond
44 - Remove assignments on mb (definition removed, all usages removed)
45 - Remove assignments on m (definition removed, all usages removed)
46 - Remove assignments on mb (definition removed, all usages removed)
47 - Remove assignments on m (definition removed, all usages removed)
48 yields
49
50 Game 3 is
51   Ostart() :=
52     b <-R bool;
53     k_3 <-R key;
54     mk_2 <-R mkey;
55     return();
56     foreach i <= qEnc do
57       Oenc(m0: bitstring, m1: bitstring) :=
58         if Z(m0) = Z(m1) then
59           if b then
60             r_1 <-R enc_seed;
61             r_2 <-R enc_seed;
62             return(concat(enc_r(m0, k_3, r_1), mac(enc_r(m0, k_3, r_2), mk_2)))
63           else
64             r_1 <-R enc_seed;
65             r_2 <-R enc_seed;
66             return(concat(enc_r(m1, k_3, r_1), mac(enc_r(m1, k_3, r_2), mk_2)))
67
68
69 Applying SA rename new without array accesses and remove assignments of findcond
70 - Rename variable r_1 into r_5, r_3
71 - Rename variable r_2 into r_6, r_4
72 yields
73
74 Game 4 is
75   Ostart() :=
76     b <-R bool;
77     k_3 <-R key;
78     mk_2 <-R mkey;
79     return();
80     foreach i <= qEnc do
81       Oenc(m0: bitstring, m1: bitstring) :=
82         if Z(m0) = Z(m1) then
83           if b then
84             r_5 <-R enc_seed;
85             r_6 <-R enc_seed;
86             return(concat(enc_r(m0, k_3, r_5), mac(enc_r(m0, k_3, r_6), mk_2)))
87           else
88             r_3 <-R enc_seed;
89             r_4 <-R enc_seed;
90             return(concat(enc_r(m1, k_3, r_3), mac(enc_r(m1, k_3, r_4), mk_2)))
91
92
93 Applying equivalence ind_cpa(enc) [probability Penc(time_1, 2 * qEnc, max(maxlength(
    game 4: m1), maxlength(game 4: m0)))]
94 - Equivalence ind_cpa(enc) with variables: r_3 -> r_1, r_5 -> r_1, r_6 -> r_1, k_3
    -> k_2, r_4 -> r_1
95 yields
96
97 Game 5 is
98   Ostart() :=
99     b <-R bool;
100    k_4 <-R key;
101    mk_2 <-R mkey;
102    return();
103    foreach i <= qEnc do
104      Oenc(m0: bitstring, m1: bitstring) :=

```

```

105   if Z(m0) = Z(m1) then
106   if b then
107     r_9 <-R enc_seed;
108     r_8 <-R enc_seed;
109     return(concat((x_2: bitstring <- m0; enc_r'(Z(x_2), k_4, r_9)), mac((x_1:
      bitstring <- m0; enc_r'(Z(x_1), k_4, r_8)), mk_2)))
110   else
111     r_10 <-R enc_seed;
112     r_7 <-R enc_seed;
113     return(concat((x_3: bitstring <- m1; enc_r'(Z(x_3), k_4, r_10)), mac((x:
      bitstring <- m1; enc_r'(Z(x), k_4, r_7)), mk_2)))
114
115
116 Applying expand
117   - Expand if/find/let
118 yields
119
120 Game 6 is
121   Ostart() :=
122   b <-R bool;
123   k_4 <-R key;
124   mk_2 <-R mkey;
125   return();
126   foreach i <= qEnc do
127   Oenc(m0: bitstring, m1: bitstring) :=
128   if Z(m0) = Z(m1) then
129   if b then
130     r_9 <-R enc_seed;
131     r_8 <-R enc_seed;
132     x_2: bitstring <- m0;
133     x_1: bitstring <- m0;
134     return(concat(enc_r'(Z(x_2), k_4, r_9), mac(enc_r'(Z(x_1), k_4, r_8), mk_2)))
135   else
136     r_10 <-R enc_seed;
137     r_7 <-R enc_seed;
138     x_3: bitstring <- m1;
139     x: bitstring <- m1;
140     return(concat(enc_r'(Z(x_3), k_4, r_10), mac(enc_r'(Z(x), k_4, r_7), mk_2)))
141
142
143 Applying remove assignments of findcond
144   - Remove assignments on x_3 (definition removed, all usages removed)
145   - Remove assignments on x (definition removed, all usages removed)
146   - Remove assignments on x_2 (definition removed, all usages removed)
147   - Remove assignments on x_1 (definition removed, all usages removed)
148 yields
149
150 Game 7 is
151   Ostart() :=
152   b <-R bool;
153   k_4 <-R key;
154   mk_2 <-R mkey;
155   return();
156   foreach i <= qEnc do
157   Oenc(m0: bitstring, m1: bitstring) :=
158   if Z(m0) = Z(m1) then
159   {16}if b then
160     r_9 <-R enc_seed;
161     r_8 <-R enc_seed;
162     return(concat(enc_r'(Z(m0), k_4, r_9), mac(enc_r'(Z(m0), k_4, r_8), mk_2)))
163   else
164     r_10 <-R enc_seed;
165     r_7 <-R enc_seed;
166     return(concat(enc_r'(Z(m1), k_4, r_10), mac(enc_r'(Z(m1), k_4, r_7), mk_2)))
167
168
169 Applying merge branches
170   - Merge branches of test at 16
171 yields

```

```

172
173 Game 8 is
174   Ostart() :=
175     b <-R bool;
176     k_4 <-R key;
177     mk_2 <-R mkey;
178     return();
179   foreach i <= qEnc do
180     Oenc(m0: bitstring, m1: bitstring) :=
181       if Z(m0) = Z(m1) then
182         r_10 <-R enc_seed;
183         r_7 <-R enc_seed;
184         return(concat(enc_r'(Z(m1), k_4, r_10), mac(enc_r'(Z(m1), k_4, r_7), mk_2)))
185
186
187 Proved secrecy of b in game 8
188 Adv[Game 1: secrecy of b] <= 2 * Penc(time_1, 2 * qEnc, max(maxlength(game 4: m1),
189   maxlength(game 4: m0))) + Adv[Game 8: secrecy of b]
189 Adv[Game 8: secrecy of b] <= 0
190 RESULT Proved secrecy of b up to probability 2 * Penc(time_1, 2 * qEnc, max(maxlength(
191   game 4: m1), maxlength(game 4: m0)))
191 RESULT time_1 = qEnc * time(= bitstring, length(Z, maxlength(game 4: m0)), length(Z,
192   maxlength(game 4: m1))) + qEnc * time(Z, maxlength(game 4: m1)) + qEnc * time(Z,
193   maxlength(game 4: m0)) + qEnc * time(concat, length(enc_r, maxlength(game 4: m0)))
194   + qEnc * time(mac, length(enc_r, maxlength(game 4: m0))) + qEnc * time(concat,
195   length(enc_r, maxlength(game 4: m1))) + qEnc * time(mac, length(enc_r, maxlength(
196   game 4: m1))) + time
197 All queries proved.

```

最後の行を見ると、証明が完了していることがわかります。この例ではゲーム8まで変換したようです。そして、最終ゲームのゲーム8でbの secrecy を示すことができます。これにより、Enc-then-MAC の IND-CPA 安全性を示すことができました。

演習 5 各ゲーム変換を理解しよう

5_Enc-then-MAC_IND-CPA.cv を入力し、Enc-then-MAC の安全性ゲーム変換列を手に入れましょう。

各安全性ゲームの変換はそれぞれ何を意味しているのでしょうか？

- if 文を展開
- if 文をマージ
- 乱数の名前を変更
- 不要な一時変数の削除
- 順序を変更
- 暗号文をダミーに差し替え

では、出力されたゲーム変換を一つずつ見ていきましょう。

ゲーム 1 からゲーム 2 へ (if 文を展開)

ゲーム 1 は初期ゲームです。

```

1 Game 1 is
2   Ostart() :=
3     b <-R bool;
4     k_3 <-R key;
5     mk_2 <-R mkey;
6     return();
7   foreach i <= qEnc do
8     Oenc(m0: bitstring, m1: bitstring) :=
9       if Z(m0) = Z(m1) then
10         mb: bitstring <- (if b then m0 else m1);

```

```

11     return((m: bitstring <- mb; concat((r_1 <-R enc_seed; enc_r(m, k_3, r_1)), mac((
        r_2 <-R enc_seed; enc_r(m, k_3, r_2)), mk_2))))

```

Enc-and-MAC の場合と同様、b の値による分岐を展開し、次のゲーム 2 が得られます。

```

1 Game 2 is
2   Ostart() :=
3   b <-R bool;
4   k_3 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  if b then
11    mb: bitstring <- m0;
12    m: bitstring <- mb;
13    r_1 <-R enc_seed;
14    r_2 <-R enc_seed;
15    return(concat(enc_r(m, k_3, r_1), mac(enc_r(m, k_3, r_2), mk_2)))
16  else
17    mb: bitstring <- m1;
18    m: bitstring <- mb;
19    r_1 <-R enc_seed;
20    r_2 <-R enc_seed;
21    return(concat(enc_r(m, k_3, r_1), mac(enc_r(m, k_3, r_2), mk_2)))

```

ゲーム 2 からゲーム 3 へ (不要な一時変数の削除)

こちらも同様に、不要な一時変数 mb, m を削除し、ゲーム 3 が得られます。

```

1 Game 3 is
2   Ostart() :=
3   b <-R bool;
4   k_3 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  if b then
11    r_1 <-R enc_seed;
12    r_2 <-R enc_seed;
13    return(concat(enc_r(m0, k_3, r_1), mac(enc_r(m0, k_3, r_2), mk_2)))
14  else
15    r_1 <-R enc_seed;
16    r_2 <-R enc_seed;
17    return(concat(enc_r(m1, k_3, r_1), mac(enc_r(m1, k_3, r_2), mk_2)))

```

ゲーム 3 からゲーム 4 へ (乱数の名前を変更)

またまた同様に、乱数の名前を変更し、次のゲーム 4 が得られます。

```

1 Game 4 is
2   Ostart() :=
3   b <-R bool;
4   k_3 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  if b then
11    r_5 <-R enc_seed;
12    r_6 <-R enc_seed;
13    return(concat(enc_r(m0, k_3, r_5), mac(enc_r(m0, k_3, r_6), mk_2)))
14  else

```

```

15     r_3 <-R enc_seed;
16     r_4 <-R enc_seed;
17     return(concat(enc_r(m1, k_3, r_3), mac(enc_r(m1, k_3, r_4), mk_2)))

```

ゲーム 4 からゲーム 5 へ (暗号文をダミーに差し替え)

ゲーム 4 の 2 箇所の暗号文部分をダミー暗号文に差し替えて, ゲーム 5 が得られます. これまた Enc-and-MAC の場合と同様に, 共通鍵暗号が IND-CPA であることから変換することができます.

```

1 Game 5 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10          if b then
11            r_9 <-R enc_seed;
12            r_8 <-R enc_seed;
13            return(concat((x_2: bitstring <- m0; enc_r'(Z(x_2), k_4, r_9)), mac((x_1:
14              bitstring <- m0; enc_r'(Z(x_1), k_4, r_8)), mk_2)))
15          else
16            r_10 <-R enc_seed;
17            r_7 <-R enc_seed;
18            return(concat((x_3: bitstring <- m1; enc_r'(Z(x_3), k_4, r_10)), mac((x:
19              bitstring <- m1; enc_r'(Z(x), k_4, r_7)), mk_2)))

```

ゲーム 5 からゲーム 6 へ (順序を変更)

代入の順序を 2 箇所変更し, ゲーム 6 が得られます.

```

1 Game 6 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();
7     foreach i <= qEnc do
8       Oenc(m0: bitstring, m1: bitstring) :=
9         if Z(m0) = Z(m1) then
10          if b then
11            r_9 <-R enc_seed;
12            r_8 <-R enc_seed;
13            x_2: bitstring <- m0;
14            x_1: bitstring <- m0;
15            return(concat(enc_r'(Z(x_2), k_4, r_9), mac(enc_r'(Z(x_1), k_4, r_8), mk_2)))
16          else
17            r_10 <-R enc_seed;
18            r_7 <-R enc_seed;
19            x_3: bitstring <- m1;
20            x: bitstring <- m1;
21            return(concat(enc_r'(Z(x_3), k_4, r_10), mac(enc_r'(Z(x), k_4, r_7), mk_2)))

```

ゲーム 6 からゲーム 7 へ (不要な一時変数の削除)

不要な一時変数 x_2 , x_1 , x_3 , x を削除し, ゲーム 7 が得られます.

```

1 Game 7 is
2   Ostart() :=
3     b <-R bool;
4     k_4 <-R key;
5     mk_2 <-R mkey;
6     return();

```

```

7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10  {16}if b then
11      r_9 <-R enc_seed;
12      r_8 <-R enc_seed;
13      return(concat(enc_r'(Z(m0), k_4, r_9), mac(enc_r'(Z(m0), k_4, r_8), mk_2)))
14  else
15      r_10 <-R enc_seed;
16      r_7 <-R enc_seed;
17      return(concat(enc_r'(Z(m1), k_4, r_10), mac(enc_r'(Z(m1), k_4, r_7), mk_2)))

```

ゲーム7からゲーム8へ (if文をマージ)

最後に, if 文による分岐を統合して, 最終ゲームのゲーム8に変換されます. Enc-and-MAC の場合とは異なり, if 文の分岐先の内容が区別できないため統合することができます. このゲーム8では, 出力は b に全く依存なくなっています. そのため, b の secrecy を示すことができます.

```

1 Game 8 is
2 Ostart() :=
3   b <-R bool;
4   k_4 <-R key;
5   mk_2 <-R mkey;
6   return();
7   foreach i <= qEnc do
8   Oenc(m0: bitstring, m1: bitstring) :=
9   if Z(m0) = Z(m1) then
10      r_10 <-R enc_seed;
11      r_7 <-R enc_seed;
12      return(concat(enc_r'(Z(m1), k_4, r_10), mac(enc_r'(Z(m1), k_4, r_7), mk_2)))

```

4 応用編：Enc-then-MACがIND-CCA2を満たすこと

物足りない方向けに, 応用編です. 応用編では, ひとつ前で考えた Enc-then-MAC が次の IND-CCA2 を満たすことを示していきましょう.

IND-CCA2ゲーム

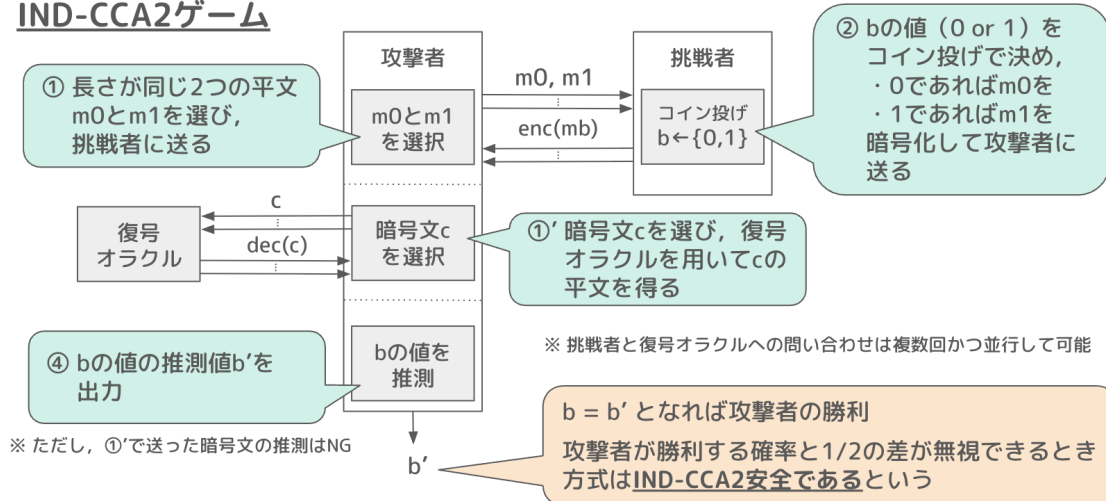


Figure 8: IND-CCA2 ゲーム. (注意: このハンズオンでは, わかりやすさと参考元チュートリアルへの準拠を優先してこのゲームを IND-CCA2 ゲームと呼びます.)

IND-CPA は攻撃者はメッセージに対して暗号文を自由に手に入れることができる (暗号化オラクルを使える) 設定でした. IND-CCA2 では, 攻撃者は暗号文の復号も行うことができます. そのため暗号化オラクルに加えて復号化オラクルも定義する必要があります.

応用演習 IND-CCA2ゲームを記述しよう

6_Enc-then-MAC_IND-CCA2_fill.cv の ??? 部分を埋めて復号オラクルを完成させましょう。
そしてコードを実行し、どのような実行結果とゲーム変換が出力されるのか確認してみましょう。

Code 3: 6_Enc-then-MAC_IND-CCA2_fill.cv

```
1 (* Encrypt-then-MAC is IND-CCA2 *)
2 param qEnc.
3 param qDec.
4
5 type mkey [fixed].
6 type key [fixed].
7 type macs [fixed].
8
9 (* Shared-key encryption (CPA Stream cipher) *)
10 proba Penc.
11 expand IND_CPA_sym_enc(key, bitstring, bitstring, enc, dec, injbot, Z, Penc).
12
13 (* Mac *)
14 proba Pmac.
15 expand SUF_CMA_det_mac(mkey, bitstring, macs, mac, verify, Pmac).
16
17 fun concat(bitstring, macs): bitstring [data].
18
19 letfun full_enc(m: bitstring, k: key, mk: mkey) =
20   c <- enc(m, k);
21   concat(c, mac(c, mk)).
22
23 letfun full_dec(c: bitstring, k: key, mk: mkey) =
24   let concat(c1, mac1) = c in
25   (
26     if verify(c1, mk, mac1) then
27       dec(c1, k)
28     else
29       bottom
30   )
31   else
32     bottom.
33
34 table ciphertexts(bitstring).
35
36 (* Queries *)
37 query secret b.
38
39 let QencLR(b0: bool, k: key, mk: mkey) =
40   foreach ienc <= qEnc do
41     Qenc (m0: bitstring, m1: bitstring) :=
42       if Z(m0) = Z(m1) then (* m0 and m1 have the same length *)
43         mb <- if b0 then m0 else m1;
44         cb <- full_enc(mb, k, mk);
45         insert ciphertexts(cb);
46   return(cb).
47
48 let Qdec(k: key, mk: mkey) =
49   foreach idec <= qDec do
50     Qdec (c: bitstring) :=
51       get ciphertexts(=c) in return(bottom) else
52       return(???).
53
54 process
55   Ostart() :=
56     b <-R bool;
57     k <-R key;
58     mk <-R mkey;
59     return;
60     (run QencLR(b, k, mk) | run Qdec(k, mk))
```

復号オラクルは次のように記述できます。

```
48 let Qdec(k: key, mk: mkey) =  
49   foreach ideo <= qDec do  
50     Odec (c: bitstring) :=  
51     get ciphertxts(=c) in return(bottom) else  
52     return(full_dec(c, k, mk)).
```

6.Enc-then-MAC.IND-CCA2.fill.cv を簡単に説明します。まず、復号オラクルを定義するために Enc-then-MAC の復号関数 full_dec を定義しています (23 行目から 32 行目)。そして、IND-CCA2 ゲームで注意しなければならないのは、攻撃者が復号オラクルを用いて自明にゲームに勝利してしまうことを防ぐ必要があるということです。復号オラクルを用いると、チャレンジ暗号文を復号することで攻撃者は容易に b の値を推測できてしまいます。そのため、チャレンジ暗号文に対しては復号オラクルを使えないように定義する必要があります。この機能は CryptoVerif のテーブル機能を用いて記述することができます。34 行目でテーブルを定義し、暗号化オラクルの中で生成した暗号文をテーブルに記録していきます (45 行目)。そして、復号オラクルの中で、オラクルに問い合わせた暗号文がそのテーブルに記録されているかをチェックし、記録されていたら平文ではなく $\perp$ を返すようにします (51 行目)。このようにすることで、攻撃者が復号オラクルを用いて明らかにゲームに勝利する場合を除くことができます。また、暗号化オラクルと復号オラクルを並列して動かすようにゲームを定義しています (60 行目)。

余談ですが、IND-CPA のコードと IND-CCA2 のコードで Enc-then-MAC の記述方法が異なっています。(気になる方は見比べてみてください。) IND-CCA2 の証明では、IND-CPA のコードのように Enc-then-MAC 部分を記述すると証明が自動で完了しませんでした。(繰り返しになりますが、証明が完了しない=安全性を満たさない、ではありません。) このように、1 行書き方を変えるだけで自動で証明できる/できないが変わることもあります。証明を完了させる匠の技もあるのかもしれませんが。

5 おわりに

本ハンズオンでは、Enc-and-MAC と Enc-then-MAC を例に CryptoVerif による安全性証明について紹介しました。時間の都合上、細かい文法や記述方法については扱えなかったため、興味を持っていただけた方はぜひ公式マニュアル [BCJ24] 等を見ていただければと思います。また、応用例は Web ページ [Blaa] に掲載されておりますので、ぜひご覧ください。

ハンズオンにご参加いただきありがとうございました！

References

- [BBC⁺] Bruno Blanchet, Pierre Boutry, David Cadé, Christian Doczkal, Aymeric Fromherz, Charlie Jacomme, Benjamin Lipp, and Pierre-Yves Strub. Online demo for cryptoverif. <http://proverif24.paris.inria.fr/cryptoverif.php>. Accessed: 2025-10-28.
- [BCDS] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. The tamarin prover. <https://tamarin-prover.com/>. Accessed: 2025-10-28.
- [BCJ24] Bruno Blanchet, David Cadé, and Charlie Jacomme. Cryptoverif computationally sound cryptographic protocol verifier user manual, 2024. <https://bblanche.gitlabpages.inria.fr/CryptoVerif/manual.pdf>. Accessed: 2025-10-28.
- [Blaa] Bruno Blanchet. Cryptoverif: Cryptographic protocol verifier in the computational model. <https://bblanche.gitlabpages.inria.fr/CryptoVerif/>. Accessed: 2025-10-28.
- [Blab] Bruno Blanchet. Proverif: Cryptographic protocol verifier in the formal model. <https://prosecco.gforge.inria.fr/personal/bblanche/proverif/>. Accessed: 2025-10-28.
- [Haf] Marc Hafner. Cryptoverif tutorial. https://rub-nds.github.io/AKE-Cryptoverif-Tutorial/Tutorial_md. Accessed: 2025-10-28.
- [大陵龍 11] 森山 大輔, 西巻 陵, and 岡本 龍明. 公開鍵暗号の数理. シリーズ応用数理. 共立出版, 2011.