TAMARIN PROVER の HEURISTIC ORACLE を利用した CPA MODELの安全性検証

2024.9.15

Authors

三重野 武彦

岡崎 裕之

布田 裕一

荒井 研一

Contents

1. 目的・目標・モチベーション　何をしたいのか

研究の目的：
ProVerif と Tamarin prover
それぞれの使い所を考察した上で、これらツールが、
暗号プロトコルの設計・開発に役立つことを示したい

目標: 設計・開発現場で容易にプロトコル検証のツールを利用したい

モチベーション
暗号の専門家でなくとも使用できるように、
検証ツールを含めたSDKの提供や、
生産性を上げる為の工夫を創っていく

2. 今回実施した内容と主張

Tamarin prover の検証対象(環境)として、
Heuristic Oracleがどのように扱われるか調査

- **black boxの証明内容を、証明目標の順位を並べ替える Heuristic Oracleを使用することで、ホワイトボックス化した。**

調査方法：証明が終わらないモデルをHeuristic Oracleを使って、手動で確認 **Heuristic Oracleは、Tamarin proverのInteractive Mode(Web I/F)を使用し、証明のステップを踏んで(手動で確認しつつ)構築した。**

主張

証明目標の順位を並べ替えるHeuristic Oracleを利用することで、
1. 目的の証明に繋がる選択がより自由に、かつ厳密に出来るようになる.
2. Tamarin proverが使用するデフォルトのヒューリスティック(標準的な攻撃発見手法)に比べて、優れた証明探索が実現できる場合がある。

What is Tamarin-prover

3.検証ツールの概要 1/2

A multiset rewriting rule is described as follows.

1 rule Name:

2 [Pre(x)] –[Action(x)]–> [Post(x)]

Lemma Annotations

lemma Name [Annotation1,Annotation2]:

e.g.)

lemma nonce_secret:

"All m #i. Secret(m) @i & K(m)@j ==> F"

Tamarin-prover query handling

Secret(session1, key)@i&Key(key)@j.

3.検証ツールの概要 2/2

Tamarin proverのHeuristic Oracle
ファクトの順序を好きなように直接変更するpython スクリプト

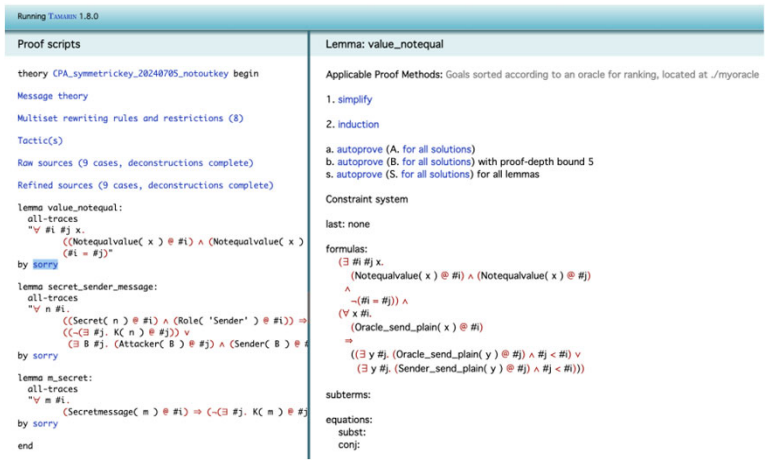
Heuristic Oracle の特徴

・Tamarin proverとは別に実行されるCodeの一部
・Proof Goalの番号付きListを入力として受け取り,
　どのProof Goalを最初に解決すべきか示す,
　番号の順序付きリストを生成する(図1)
（検証したいFactを直接プログラム内に埋め込む）

(図1)

```
>>>>>>>>>>>>>>>>>>>>>>>> START INPUT
0:  (#i < #j)  ∨  (#j < #i)
1: St_ReceiveKeyOracleplain( m_, key_ ) ▶₀ #i
2: !KU( senc(<$A, m_>, key_) ) @ #vk
3: St_ReceiveKey( ~m, key ) ▶₂ #i
4: !KU( senc(<$B, ~m>, key) ) @ #vk.1
5: St_ReceiveKeyOracleplain( m_, key_ ) ▶₀ #j
6: !KU( senc(<$A.1, m_>, key_) ) @ #vk.2
7: St_ReceiveKey( ~m, key ) ▶₂ #j
8: !KU( senc(<$B.1, ~m>, key) ) @ #vk.3

>>>>>>>>>>>>>>>>>>>>>>>> START OUTPUT
1
3
5
7
4
0
2
6
8
    Raw sources (9 cases, deconstructions compl

    Refined sources (9 cases, deconstructions c

    lemma uniqueness:
      all-traces
      "∀ #i #j x.
            ((Unique( x ) @ #i) ∧ (Unique(
      simplify
      solve( St_ReceiveKeyOracleplain( m_, ke
>>>>>>>>>>>>>>>>>>>>>>>> END Oracle call
```

(図2)



注意事項)
・ Heuristic Oracleを使用したからといってModelによって,
　Tamarinは必ずしも証明が終了するわけではない.
・TamarinのInteractive Mode(対話Mode:図2)を使用し,手作業で
　Proof Goalを選ぶよりも,Oracleを書いて証明した方が便利なことが
　多々あるが、そうでない場合もある(先行研究もある)
・ Heuristic Oracleには元のモデルのFactの順序を直接変更することが
　できるため,Interactive Modeで必ず確認する必要がある.

4. 関連研究1

[タイトル]

Modelling and Analysis of Web Applications in Tamarin

[作者]

Sandra Dünki

A Oracle Template

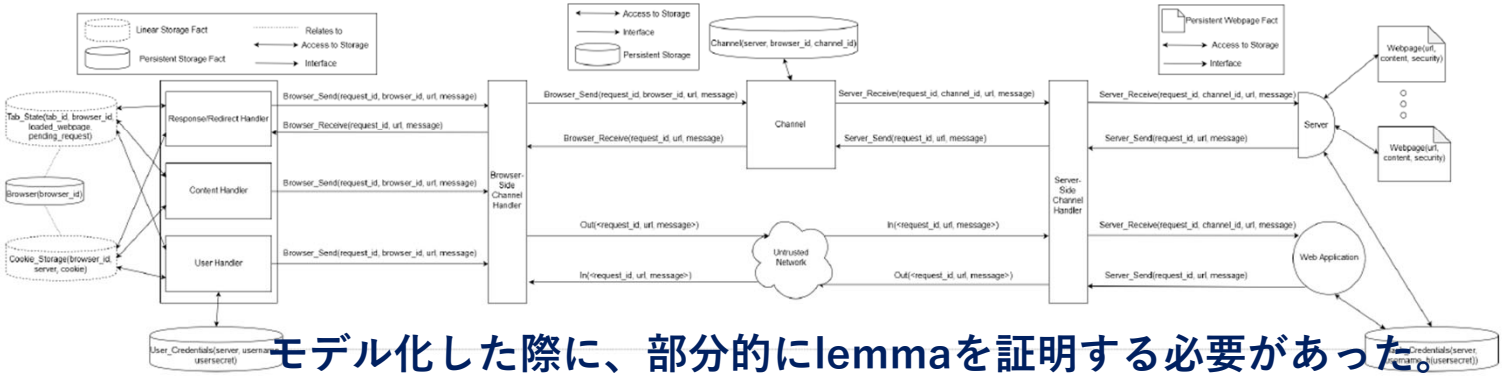
This oracle was needed in a previous version of the framework where HTTPS was modelled using encryption instead of confidential channels. We include the oracle here as a version of it might be needed when working with this earlier HTTPS model. Thanks to Ralf Sasse for providing me with a template I could use to write this oracle.

```
#!/usr/bin/python

import re
import os
import sys
debug = True

lines = sys.stdin.readlines()
lemma = sys.argv[1]

# INPUT:
# - lines contain a list of "%i:goal" where "%i" is the
    index of the goal
# - lemma contain the name of the lemma under scrutiny
# OUTPUT:
# - (on stdout) a list of ordered index separated by EOL


rank = []             # list of list of goals, main
    list is ordered by priority
maxPrio = 110
for i in range(0,maxPrio):
  rank.append([])

# SOURCES LEMMA
if lemma == "typing":
  for line in lines:
    num = line.split(':')[0]

    if re.match('.*!KU\( senc\(.*', line):
       rank[109].append(num)

```

OpenID プロトコルの安全性を Tamarin proverで証明

Tamarin prover のデフォルト探索では上手く証明できないが、
Heuristic Oracleを利用して証明可

B Model



モデル化した際に、部分的にlemmaを証明する必要があった。

Figure 10: Complete Diagram of Model

4. 関連研究2

Tamarin proverは終了せず、利用可能なメモリをすべて消費した。[2/3]

Tamarin proverのHeuristic Oracleを利用

[タイトル]
Formal Analysis and Applications of Direct Anonymous Attestation

[作者]
Jorden D.Whitefield

Tamarin proverを用いたECC-DAA 仕様のシンボリック解析により、信頼されたプラットフォーム・モジュールを悪用して、他の妥協されていないモジュールを弱体化させる攻撃を特定した論文

ISO/IEC 20008-2:2013規格に基づくECC-DAA(Direct Anonymous Attestation)

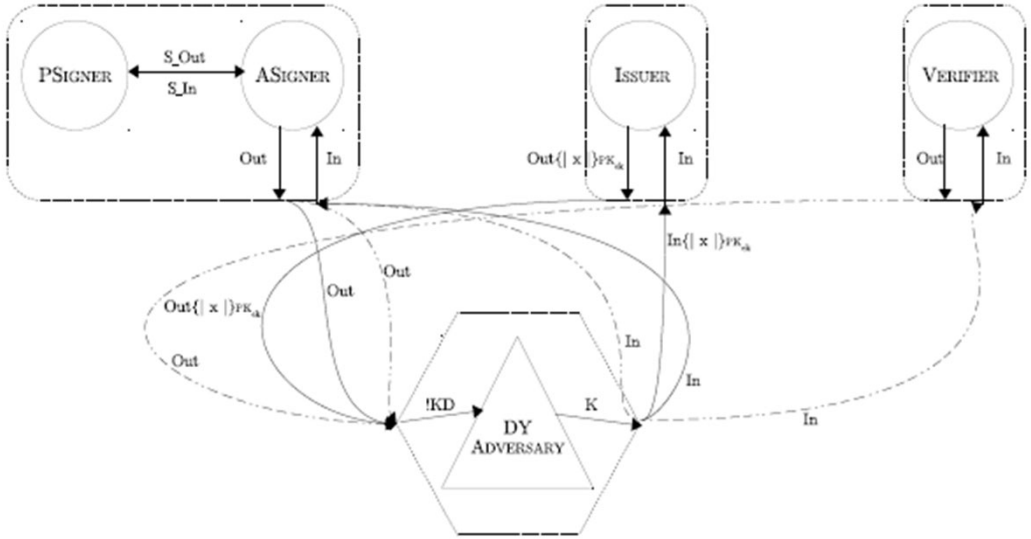


Figure 6.3 Network with all communication routed through the DY adversary

Goal	Lemma	Model A	Model B
G1	functional_correctness_group_verification	✓	✓
G2	functional_correctness	✓	✓
G3	functional_correctness_dishonest_send	✓	✓
G4	aliveness	✓	✓
G5	weak_agreement_any_reveal	✓	✓
G6	weak_agreement	×	×
G7	ni_agreement_any_reveal	✓	✓
G8	ni_agreement	×	×
G9	i_agreement	×	×
G10	secrecy_cre	×	×
G11	can_be_deanonymised	✓	✓
G12	user_controlled_independent_link_tokens	✓	n/a
G13	user_controlled_linkability	n/a	✓
Goal	**Observational Equivalence**	**Model C**	
G14	unlinkability	✓	

Table 6.1 Summary of Results

4. 関連研究3

[タイトル]

The Design and Analysis of Real-World Cryptographic Protocols

[作者]

Samuel Scott

TLS1.3仕様のセキュリティ特性を証明するために、Tamarin proverを用いて、セキュリティ解析への記号的アプローチを実施。

Heuristic Oracleを利用して
上手く証明できないlemmaを証明

```
lemma secret_session_keys:
 "All tid actor peer wkey rkey peer_auth_status #i.
    SessionKey(tid, actor, peer, <peer_auth_status, 'auth'>, <wkey, rkey>)@i &
    not (Ex #r. RevLtk(peer)@r & #r < #i) &
    not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
    not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(actor, resumption_ms)@r) &
    not (Ex resumption_ms #r. RevealPSK(peer, resumption_ms)@r)
  ==> not Ex #j. K(read_key)@j"
```

Figure 5.9: The secret_session_keys lemma.

```
lemma entity_authentication [use_induction, reuse]:
 "All tid actor peer nonces cauth_status #i.
    CommitNonces(tid, actor, 'client', nonces)@i &
    CommitIdentity(tid, actor, 'client', peer, <cauth_status, 'auth'>)@i &
    not (Ex #r. RevLtk(peer)@r & #r < #i) &
    not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
    not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(actor, resumption_ms)@r & #r < #i) &
    not (Ex resumption_ms #r. RevealPSK(peer, resumption_ms)@r & #r < #i)
        ==> (Ex tid2 #j. RunningNonces(tid2, peer, 'server', nonces)@j & #j < #i)"
```

Figure 5.10: The entity_authentication lemma.

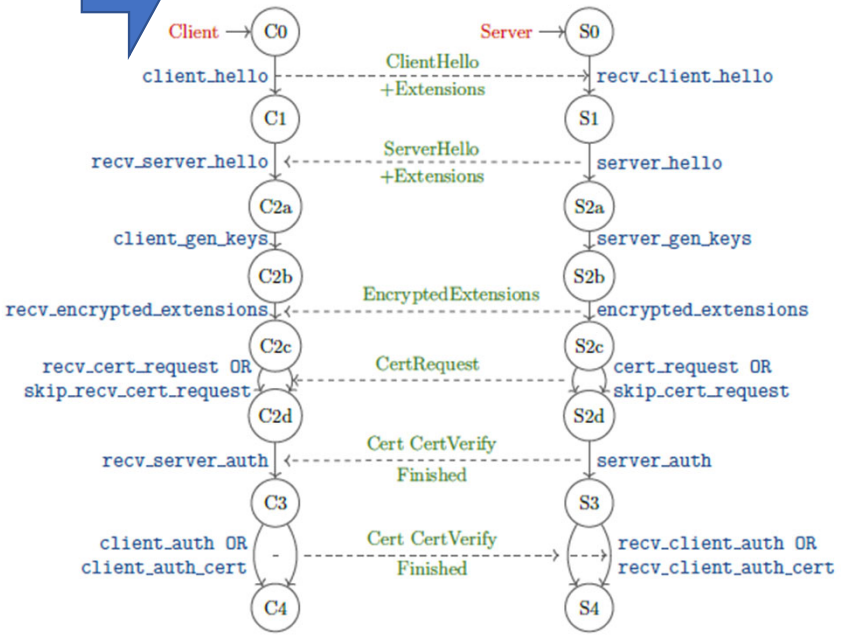


Figure 5.7: Partial state diagram for full TLS 1.3 handshake. TAMARIN rules are written in snake case on edges. The messages exchanged between entities are displayed in the middle along the dashed edges. Our full model contains many more transitions, and can be found at the end of the chapter.

5. 検証

定義 1 Let Σ be an ecnryption scheme, Σ has security against CPA security. If $\mathcal{L}^{\Sigma}_{cpa_x} \equiv \mathcal{L}^{\Sigma}_{cpa_y}$, where: $\mathcal{L}^{\Sigma}_{cpa_x}$: $key \leftarrow \Sigma$ KeyGen, $C_x = \Sigma$ Enc(key, m_x), return C_x, $\mathcal{L}^{\Sigma}_{cpa_y}$: $key \leftarrow \Sigma$ KeyGen, $C_y = \Sigma$ Enc(key, m_x), return C_y. Arbitary choose distinct plaintexts $m_x, m_y \in \mathcal{M}$.

Verification

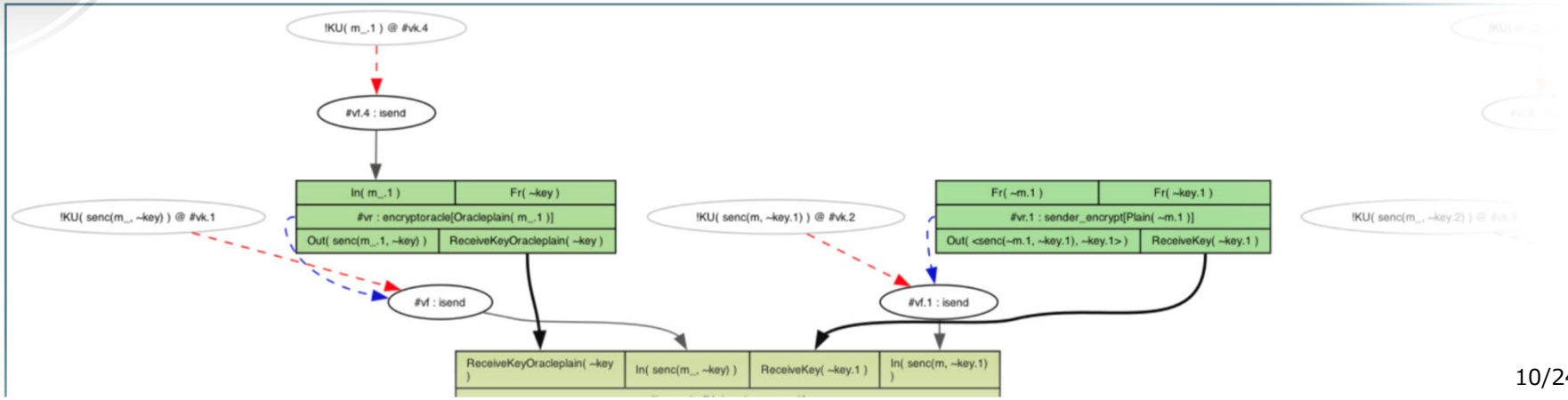
Chosen-plaintext attack (CPA) Model

選択平文攻撃

```
uniqueness:
all-traces
"∀ #i #j x.
  ((Unique( x ) @ #i) ∧ (Unique( x )
simplify
solve( ReceiveKeyOracleplain( key_ ) ▶₀ #i
case encryptoracle
solve( ReceiveKey( key.1 ) ▶₂ #i )
case sender_encrypt
solve( ReceiveKeyOracleplain( key_ ) ▶
case encryptoracle
solve( ReceiveKey( key.3 ) ▶₂ #j )
case sender_encrypt
solve( !KU( senc(m, ~key.3) ) @ #vk
case c_senc
by sorry
next
case sender_encrypt
by sorry
qed
```

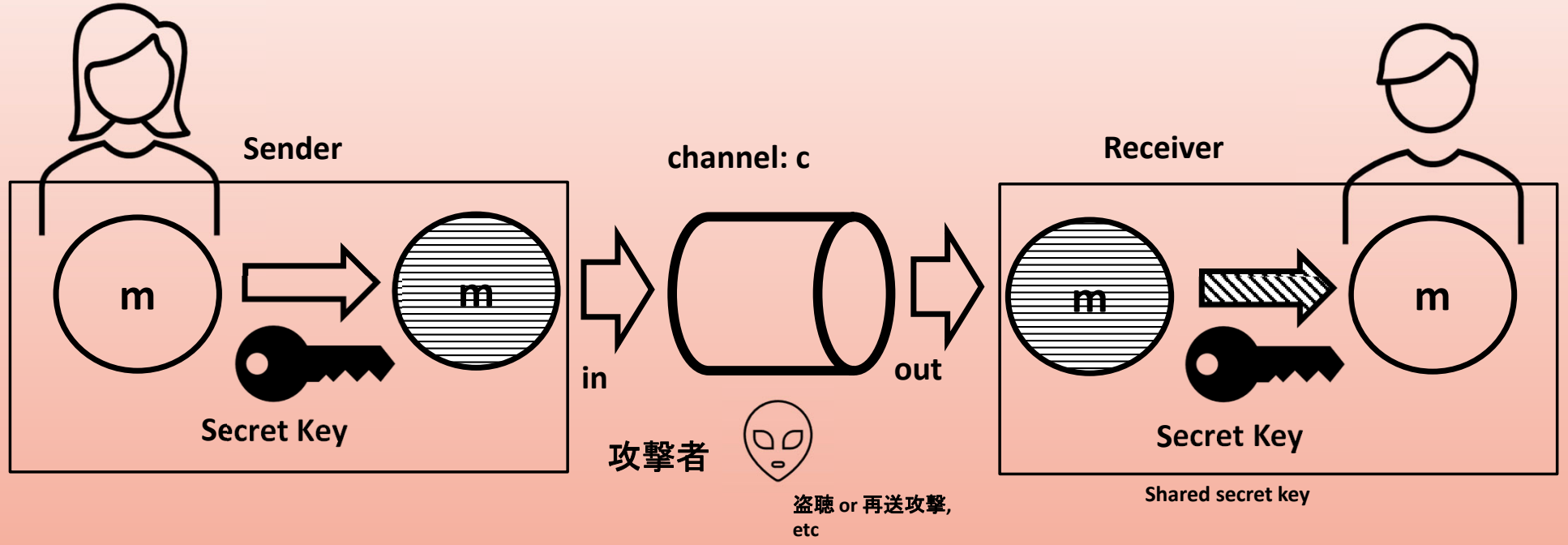
a. autoprove (A. for all solutions)
b. autoprove (B. for all solutions) with proof-depth bound 5
s. autoprove (S. for all solutions) for all lemmas

Constraint system



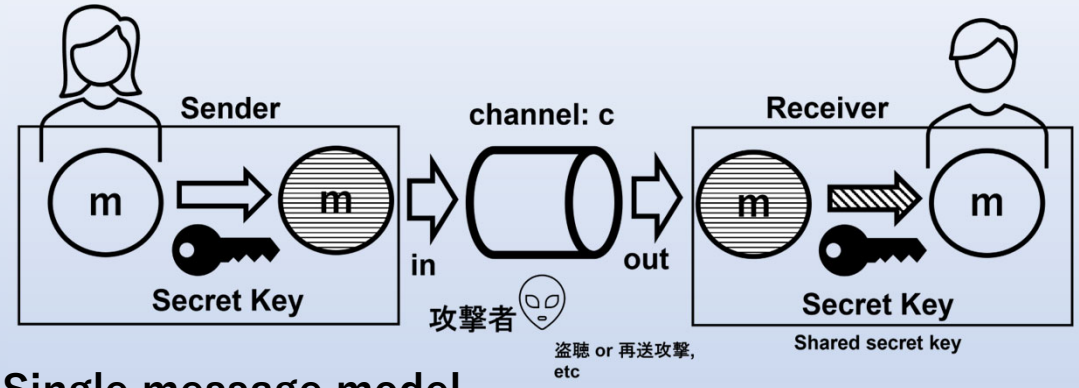
5.1 検証 CPA model

共通鍵暗号



攻撃者が、選んだ平文とそれに対する暗号文を手に入れることができる状況で攻撃するモデル。

5.1 検証モデル CPA model



Single message model

モデルの説明

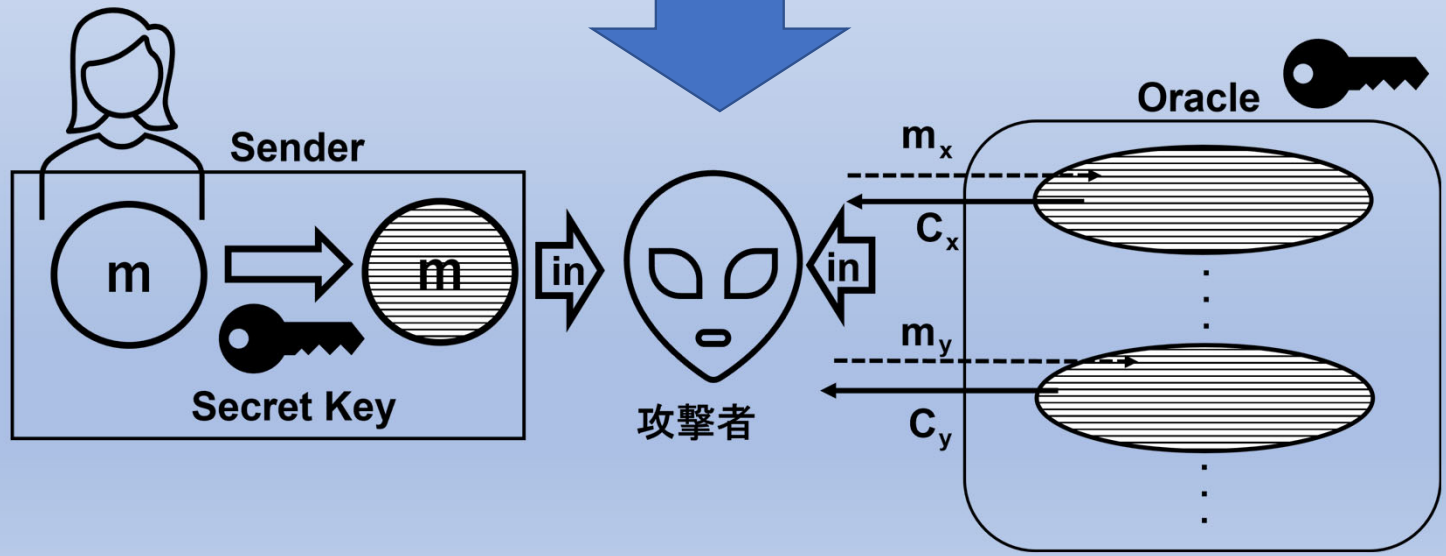
・共有する秘密鍵(一回限り)を作成後,それを使って平文を暗号化

・攻撃者は,暗号化 Oracleを利用して暗号文を得る

・暗号化 Oracleにアクセス出来るタイミングは,任意のタイミング

攻撃発見まで永久に止まらないモデル

CPA 検証model : OW-CPA



5.2 検証コード

一部抜粋

https://github.com/mienotakehiko/fais2024_summer

```
rule Setup:
[Fr(~key)] --[OnlyOnce(), Setup(~key)]-> [!Skey($A, ~key)]

rule attackerkey:
  [ !Skey($A, key) ] --[ Attacker($A) ]-> [ Out(key) ]

rule Attacker_make_message:
 [ Fr(m_)] --[ ]-> [ Out(<$A,m_>),St_Attacker(m_) ]

rule Encrypt_oracle:
 [ In(<$A, m_>), Fr(~key) ,St_Attacker(m_)] --[Send($A, senc(<$A, m_>, ~key)), Secret(m_), Oracle($A), Sender($B),
 Role('Oracle'), Oracle_send_plain(m_) ]->  [ Out(senc(<$A, m_>,~key)), St_ReceiveKeyOracleplain(m_,~key) ]

rule Sender_encrypt:
[ Fr(~m), Fr(~key)  ]  --[Send($B, senc(<$B, ~m>, ~key)), Secret(~m), Sender($B), Oracle($A), Role('Sender'), Sender_send_plain(~m) ]->
[ Out(senc(<$B, ~m>,~key)), St_ReceiveKey(~m,~key) ]

rule Attacker:
let
c_ = senc(m_ , key_)
c  = senc(~m, key)
message = sdec(c_, key)
in
 [St_ReceiveKeyOracleplain(m_,key_), In(senc(<$A, m_>,key_))  , St_ReceiveKey(~m,key), In(senc(<$B, ~m>,key)) ]
 --[ Notequalvalue(<c_, c>), Secretmessage(message)]-> [  ]
```

攻撃者作成：秘密鍵のrule形成

攻撃者が任意に作成する平文mのrule形成

暗号化オラクル

送信者が平文から暗号文を作成して送る、プロトコルの内容

攻撃者が通信路から搾取した暗号文と、攻撃者が選んだ平文に対する暗号文(オラクルに問い合わせた暗号文)が一致するか

5.2 検証コード 一部抜粋


```
rule Attacker:
let
c_ = senc(m_ , key_)
c  = senc(~m, key)
message = sdec(c_, key)
in
 [St_ReceiveKeyOracleplain(m_,key_), In(senc(<$A, m_>,key_))  , St_ReceiveKey(~m,key), In(senc(<$B, ~m>,key)) ]
 --[ Notequalvalue(<c_, c>), Secretmessage(message)]-> [  ]
```

送信者の暗号文と、
攻撃者が選んだ平文に対する暗号文,
それぞれの値の不一致を検証する。

検証するlemma

```
lemma value_notequal:
 "All #i #j x. Notequalvalue(x)@i & Notequalvalue(x)@j ==> #i=#j"
```

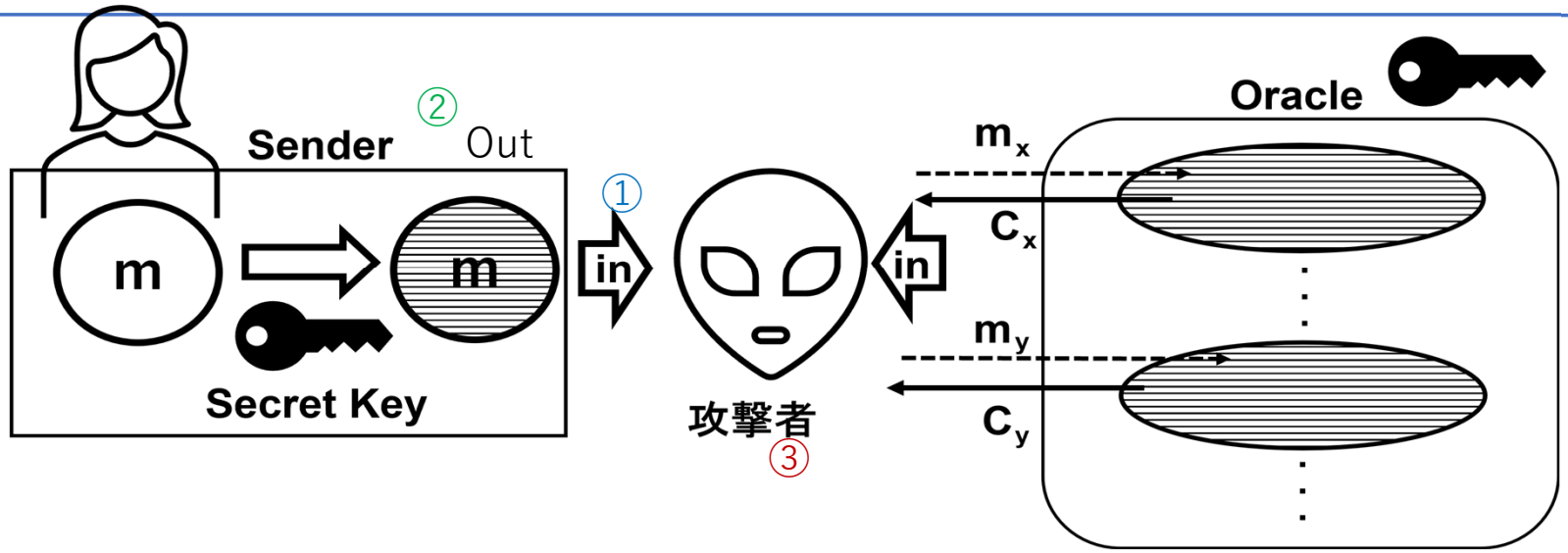
Tamarinはデフォルトで、鍵を特定しようとする。
秘密鍵を特定しようとして、永久に止まらない状況となっている。

5.3 検証 1/5

止まらないモデルを止めるために、tamarinのheuristic oracleを利用

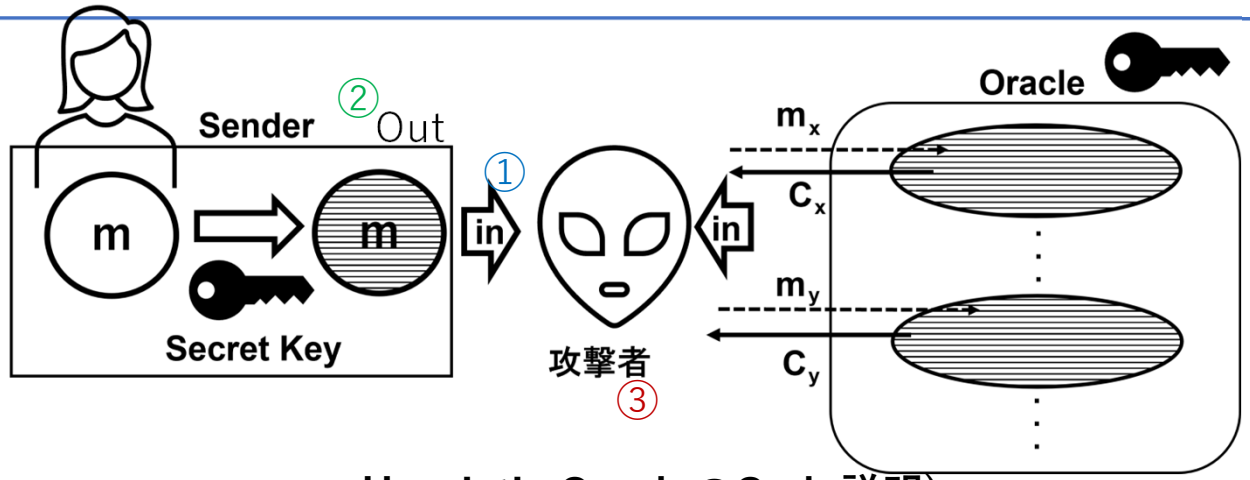
[秘密鍵が特定できる場合]
今回のCPA Modelは,攻撃者への In Fact(①),
Sender側のOut Fact(②),
攻撃者秘密鍵を推論するFact(③),

3つのFactを指定し,そのFactから秘密鍵が洩れるパスが見つからなければ良いことを証明する.



5.3 検証 2/5

やりたいこと）
今回のCPA Modelは,
攻撃者へのIn Fact(①), Sender側のOut Fact(②), 攻撃者秘密鍵を推論するFact(③),
3つのFactを指定し,そのFactから秘密鍵が洩れるパスが見つからなければ良いことを証明する.



Oracleコード

```
13 for line in lines:↓
14   num = line.split(':')[0]↓
15 ↓
16   if lemma == "value_notequal":↓
17       if ": St_ReceiveKey" in line:↓
18         la.append(num)↓
19       elif "senc(<$B, ~m>,key" in line↓
20        or "senc(<$B, ~m>, key" in line:↓
21         lb.append(num)↓
22       elif "KU(key" in line ↓
23        or "KU(~key" in line:↓
24         lc.append(num)↓
25       else:↓
26         ld.append(num)↓
27   else:↓
28 ↓
29     exit(0)↓
```

①
②
③

Heuristic OracleのCode説明）

モデルが満たすべき性質を記述した lemmaを、heuristic oracleを利用して証明する

Oracleの概要）
攻撃者がSenderからの暗号文を受け取るステートファクト(St_ReceiveKey(~m,~key))が実行された際
1.攻撃者がSenderからの暗号文Inの際(受け取り時)に秘密鍵を取得できる
2.攻撃者がSenderが暗号部Outの際(送信時)に秘密鍵を取得できる
3.攻撃者が1,2以外で秘密鍵を推論できる
※Sender側だけのIn, Outパターンで必要十分条件を満たせる.
(Oracle側のIn, Outパターンは,そもそもOracleなので証明しなくてもよいと考える)

この順序(パターン)を証明することで,攻撃者が,送信者の暗号文とオラクルから受け取る暗号文が一致しない,
つまり,秘密鍵を取得できるパスが見つからない or 秘密鍵を推論できないことが証明できる.

5.3 検証

Tamarin CPA Code

```
                    :
                (omitted)
                    :

rule Encrypt_oracle:
 [ In(<$A, m_>), Fr(~key) ,St_Attacker(m_)] --[Send($A, senc(<$A, m_>, ~key)),
Secret(m_), Oracle($A), Sender($B), Role('Oracle'), Oracle_send_plain(m_) ]->
[ Out(senc(<$A, m_>,~key)), St_ReceiveKeyOracleplain(m_,~key) ]

rule Sender_encrypt:
[ Fr(~m), Fr(~key)  ]  --[Send($B, senc(<$B, ~m>, ~key)), Secret(~m),
Sender($B), Oracle($A), Role('Sender'), Sender_send_plain(~m) ]->
[ Out(senc(<$B, ~m>,~key)), St_ReceiveKey(~m,~key) ]

rule Attacker:
let
c_ = senc(m_ , key_)
c  = senc(~m, key)
message = sdec(c_, key)
in
 [St_ReceiveKeyOracleplain(m_,key_), In(senc(<$A, m_>,key_)),
St_ReceiveKey(~m,key), In(senc(<$B, ~m>,key)) ]
 --[ Notequalvalue(<c_, c>), Secretmessage(message)]-> [  ]
                    :
                (omitted)
                    :
restriction choiceplain:
 "All x #i. Oracle_send_plain(x)@i
   ==> (Ex y #j. Oracle_send_plain(y)@j & #j < #i) |
  (Ex y #j. Sender_send_plain(y)@j & #j < #i)"

lemma value_notequal:
 "All #i #j x. Notequalvalue(x)@i & Notequalvalue(x)@j ==> #i=#j"
                    :
                (omitted)
                    :[EOF]
```

My oracle

```
                :
            (omitted)
                :

lines = sys.stdin.readlines()

la = []
lb = []
lc = []
ld = []
lemma = sys.argv[1]

for line in lines:
  num = line.split(':')[0]

  if lemma == "value_notequal":
      if ": St_ReceiveKey" in line:
        la.append(num)
      elif "senc(<$B, ~m>,key" in line
       or "senc(<$B, ~m>,~key" in line:
        lb.append(num)
      elif "KU(key" in line
       or "KU(~key" in line:
        lc.append(num)
      else:
        ld.append(num)
  else:

    exit(0)

ranked = la + lb + lc + ld

for i in ranked:
  print(i)
[EOF]
```

Code execution

:
(omitted)
:

```
>>>>>>>>>>>>>>>>>>>>>>> START INPUT
0:  (#i < #j)  ∨  (#j < #i)
1: St_ReceiveKeyOracleplain( m_, key_ ) ▶0 #i
2: !KU( senc(<$A, m_>, key_) ) @ #vk
3: St_ReceiveKey( ~m, key ) ▶2 #i
4: !KU( senc(<$B, ~m>, key) ) @ #vk.1
5: St_ReceiveKeyOracleplain( m_, key_ ) ▶0 #j
6: !KU( senc(<$A.1, m_>, key_) ) @ #vk.2
7: St_ReceiveKey( ~m, key ) ▶2 #j
8: !KU( senc(<$B.1, ~m>, key) ) @ #vk.3

>>>>>>>>>>>>>>>>>>>>>>> START OUTPUT
1
3
5
7
4
0
2
6
8
>>>>>>>>>>>>>>>>>>>>>>> END Oracle call
```

:
(omitted)
:

```
value_notequal (all-traces): verified (5 steps)
secret_sender_message (all-traces): verified (6 steps)
m_secret (all-traces): verified (34 steps)
```

攻撃者は, In Fact と Out Factを介して Heuristic Oracleにアクセス

証明完了

(1)特定の事象(In, Outで起きている, sencによる任意の平文の暗号化)の制約優先順位を優先する.
(2)攻撃者が秘密鍵を推論する制約の優先順位を下げる.

5.3 検証

Proof scripts

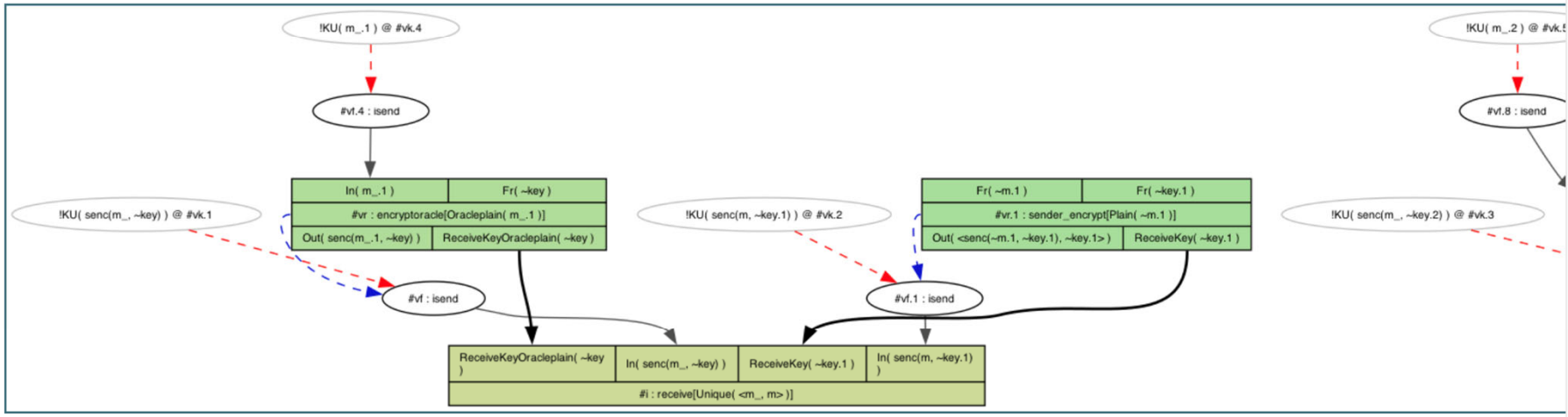
```
theory CPA_symmetrickey_20240906 begin

Message theory

Multiset rewriting rules and restrictions (

Tactic(s)

Raw sources (7 cases, deconstructions compl

Refined sources (7 cases, deconstructions c

lemma uniqueness:
  all-traces
  "∀ #i #j x.
        ((Unique( x ) @ #i) ∧ (Unique( x )
simplify
solve( ReceiveKeyOracleplain( key_ ) ▶₀ #i
  case encryptoracle
  solve( ReceiveKey( key.1 ) ▶₂ #i )
    case sender_encrypt
    solve( ReceiveKeyOracleplain( key_ ) ▶
      case encryptoracle
      solve( ReceiveKey( key.3 ) ▶₂ #j )
        case sender_encrypt
        solve( !KU( senc(m, ~key.3) ) @ #vk
          case c_senc
          by sorry
        next
          case sender_encrypt
          by sorry
        qed
      qed
    qed
  qed
qed

end
```

Visualization display

7. solve((∃ y #j. (Oracleplain(y) @ #j) ∧ #j < #vr.2) ∥
(∃ y #j. (Plain(y) @ #j) ∧ #j < #vr.2)) // nr. 19

a. autoprove (A. for all solutions)
b. autoprove (B. for all solutions) with proof-depth bound 5
s. autoprove (S. for all solutions) for all lemmas

Constraint system



last: none

formulas:
((#i < #j) ∨ (#j < #i))

((∃ y #j. (Oracleplain(y) @ #j) ∧ #j < #vr) ∨
(∃ y #j. (Plain(y) @ #j) ∧ #j < #vr))

((∃ y #j. (Oracleplain(y) @ #j) ∧ #j < #vr.2) ∨

==========
summary of summaries:
analyzed: cpamodel_verification_symmetric-encryption_out.spthy
value_notequal (all-traces): verified (5 steps)
secret_sender_message (all-traces): falsified - found trace (6 steps)
m_secret (all-traces): falsified - found trace (16 steps)
==========

証明完了

5.3 検証

秘匿性の検証

参考: 意図的に秘密鍵を漏らすCode

rule Sender_encrypt:
[Fr(~m), Fr(~key)] --[Send($B, senc(<$B, ~m>, ~key)), Secret(~m), Sender($B), Oracle($A), Role('Sender'),
Sender_send_plain(~m)]-> [Out(<senc(<$B, ~m>,~key),~key>), St_ReceiveKey(~m,~key)]

secret_sender_message (all-traces): **falsified - found trace (6 steps)**

秘密鍵を漏らさないモデル

```
rule Sender_encrypt:
[ Fr(~m), Fr(~key) ] --[Send($B, senc(<$B, ~m>, ~key)), Secret(~m), Sender($B), Oracle($A), Role('Sender'), Sender_send_plain(~m) ]->
[ Out(senc(<$B, ~m>,~key)), St_ReceiveKey(~m,~key) ]

rule Attacker:
let
c_ = senc(m_ , key_)
c  = senc(~m, key)
message = sdec(c_, key)
```

送信者が平文から暗号文を作成して送る、プロトコルの内容

secret_sender_message (all-traces): **verified (6 steps)**

5.4 検証結果

Heuristic Oracle 有無による検証結果			
無し	有り	証明したい補題: lemma	検証結果
検証が止まらない	検証が止まる	value_notequal (送信者の暗号文と、攻撃者が選んだ平文に対する暗号文)	Verified (値は一致しない)

・Tamarinは入力と出力部分で秘密鍵を探そうとする(デフォルトの探索で)
→ 検証が止まらない

・鍵を探す特定の条件をHeuristic Oracle で書く→ 検証が止まる

(鍵が洩れない限り、送信者の暗号文と、攻撃者が選んだ平文に対する暗号文のそれぞれの値は不一致)

⇒One-Way CPA(OW-CPA)の安全性検証

6. 考察

Interactive Modeを使用して、手作業で証明Goalを選択するよりも、Heuristic Oracle を書いて証明した方が、便利である場合がある

・どの証明Goalを最初に解決すべきかを示す番号の順序付きリストを生成する
・元のファクトの順序を直接変更することができる
⇒任意に証明すべきことを記述することができる

・Tamarinとは別に実行されるCodeの一部
・証明Goalの番号付きListを入力として受け取れる

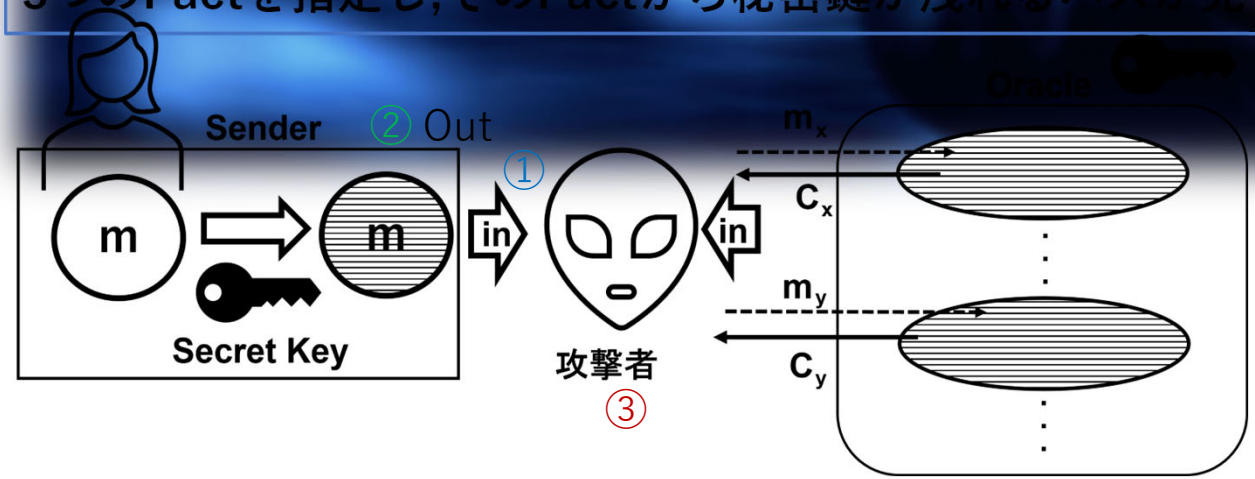
Heuristic Oracleを使用したからと言って、
必ずしも検証が止まるわけではない

6. 考察

今回のCPA Model(OW-CPA)は止まる。

・送信側だけの In, Outファクトと、攻撃者が何らかのActionで秘密鍵を取得できる、3つのActionの存在を解く証明のパスが見つからなければ良いこと証明

工夫したこと)
今回のCPA Modelは,
攻撃者へのIn Fact(①), Sender側のOut Fact(②), 攻撃者秘密鍵を推論するFact(③),
3つのFactを指定し,そのFactから秘密鍵が洩れるパスが見つからなければ良いことを証明する.



(1)特定の事象(In, Outで起きている,
sencによる任意の平文の暗号化)の制約優先順位を優先する.
(2)攻撃者が秘密鍵を推論する制約の優先順位を下げる.

まとめ　Tamarin prover のHeuristic Oracleを使用して，OW-CPAモデルの安全性検証を実施

検証が終わらないモデルを Heuristic Oracleを使用し，
証明したいlemmaを効率的に検証できることを示した.

Tamarin proverのHeuristicオラクルは、
ファクトの順序を好きなように直接変更するpython スクリプト

#Tamarinは入力と出力部分で秘密鍵を探そうとする(デフォルトの探索で)
→ 検証が止まらない
#鍵を探す特定の条件をHeuristic Oracleで書く → 検証が止まる
鍵が洩れない限り、送信者の暗号文と攻撃者が選んだ平文に対する暗号文、それぞれの値は不一致 (逆に、鍵を漏らせば、暗号文は一致)

・OW-CPAの安全性検証は出来た。

今後　・今回のフレームワークを利用し、公開鍵暗号の IND-CCA等

ご清聴ありがとうございました。