ProVerif と Tamarin prover を開発支援プロセスへ導入するための考察 その2

2024.3.4

Authors
三重野 武彦
岡崎 裕之
布田 裕一
荒井 研一

Contents

1. 目標・モチベーション　何をしたいのか

ProVerifとTamarin prover それぞれの使い所を考察した上で、暗号プロトコルの設計・開発に役立つことを議論する

目標:　設計・開発現場で容易に利用できる

モチベーション

・ProVerifで出力する"cannot be proved"の部分を,
Tamarin prover の命題にして,手動で証明する
・補題の自動抽出を将来実施したい

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. なぜ、暗号プロトコルの安全性に検証ツールを使うのか

暗号プロトコルの安全性は

攻撃の確立や計算量を用いて定式化

👇

数学的証明は煩雑であり間違いも生じる

👇

安全性証明を形式手法で記述し、証明の正しさをPC上で機械的に確かめる

そのための手段

確率的多項式時間チューリング機械のゲームとして暗号プロトコルの安全性を定式化

👇

ゲームの書き換えによって安全性を証明

👇

確立関係ホーア理論を用いた安全性証明の形式化

👇

検証ツール:ProVerif, Tamarin prover, Deep Sec, EasyCrypt, etc を用いて安全性証明

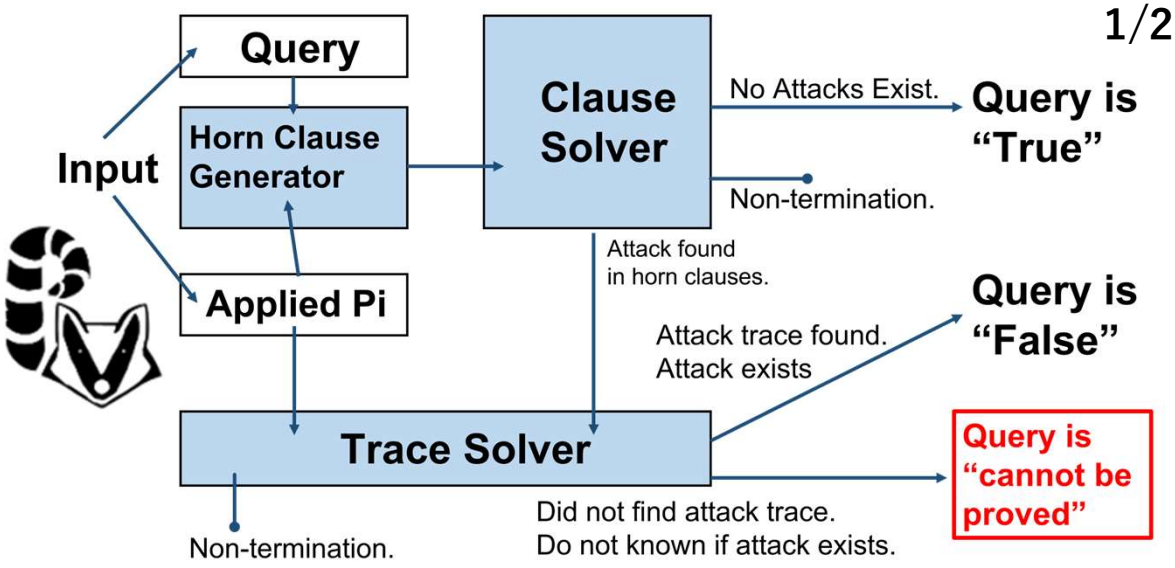
数学的に安全性が保持された暗号:証明可能安全性:provable security
証明可能安全性理論では暗号の秘匿性を攻撃者と挑戦者の間のゲームを用いて定義

3.検証手法

ProVerif(モデル検査器)で出力される
“Cannot be proved”

Tamarin prover(定理証明器) を使用し
厳密に検証

セキュリティ特性や弱点を判断できる
可能性がある
複数の暗号プロトコルのモデル検証
両ツールの検証結果の違いも検証



Tamarin-prover の概要

A multiset rewriting rule is described as follows.

1 *rule Name:*
2 *[Pre(x)] – [Action(x)] –>[Post(x)]*

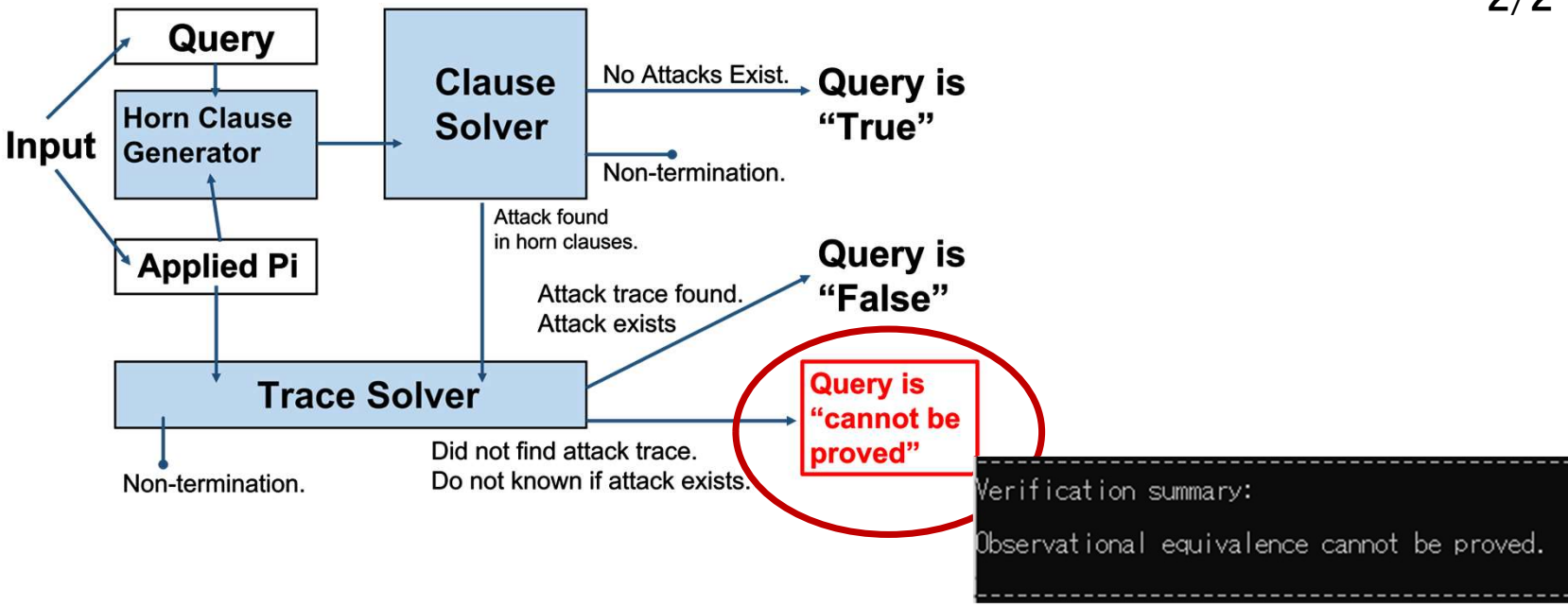
Lemma Annotations

lemma Name [Annotation1,Annotation2]:
e.g.)
lemma nonce_secret:
"All m #i #j. Secret(m) @i & K(m) @j ==> F"

Tamarin-prover query handling

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

3. 検証手法



"ProVerifは判断できないので,ユーザ(検証者)が(攻撃)手順を確認し,
それが正しいか(攻撃として成り立つ)を判断して下さい." というものである.

ProVerifが "cannot be proved" を出力するパターンは下記に分類される.

1. 観測等価性の同値性が成立しない場合: Observational equivalence cannot be proved.
2. 到達可能性が導けない場合: Query not event(・・・) cannot be proved.
3. ProVerifが保持する特性の証明失敗

4. SCIS2024 発表内容

観測等価性の同値性が成立しないモデルの形式化 -AEAD model- Log

[ProVerif]

```
(* Code.3-Log: ProVerif-AEAD model *)
1 ・・・
2 (omitted)
3 ・・・
4 Biprocess 0 (that is, the initial process):
5 (
6 {1}!
7 {2}let Kaead_1: key = Kaead in
8 {3}in(c, (na: bitstring,ada: bitstring,pa: bitstring));
9 {4}out(c, AEAD_MtE_Enc(Kaead_1,na,pa,ada))
10 ) | (
11 {5}!
12 {6}let Kaead_2: key = Kaead in
13 {7}in(c, (na_1: bitstring,ada_1: bitstring,pa_1: bitstring));
14 {8}out(c, AEAD_EtM_Enc(Kaead_2,na_1,pa_1,ada_1))
15 ) | (
16 {9}new t: bitstring;
17 {10}new non: bitstring;
18 {11}new ad: bitstring;
19 {12}let CMtE: bitstring = AEAD_MtE_Enc(Kaead,non,t,ad) in
20 {13}let CEtM: bitstring = AEAD_EtM_Enc(Kaead,non,t,ad) in
21 {14}out(c, choice[CMtE,CEtM])
22 )
23 -- Observational equivalence in biprocess 1
        (that is, biprocess 0, with let
24 moved downwards):
25 ・・・
26 (omitted)
27 ・・・
28 goal reachable: bad
29 Derivation:
30
31 1. The message enc((t[],mac((non[],ad[],t[]),Kaead[])),Kaead[],non[]) (resp.
32 (enc(t[],Kaead[],non[]),mac((non[],ad[],enc(t[],Kaead[],non[])),Kaead[]))) may
33 be sent to the attacker at output {14}.
34 attacker2(enc((t[],mac((non[],ad[],t[]),Kaead[])),Kaead[],non[]),(enc(t[],Kaead[],
35 non[]),mac((non[],ad[],enc(t[],Kaead[],non[])),Kaead[]))).
36
37 2. By 1, the attacker may know enc((t[],mac((non[],ad[],t[]),Kaead[])),Kaead[],non[])
38 (resp. (enc(t[],Kaead[],non[]),mac((non[],ad[],enc(t[],Kaead[],non[])),Kaead[]))).
39 Using the function 1-proj-2-tuple the attacker may obtain fail-any_type (resp.
40 enc(t[],Kaead[],non[])).
41 attacker2(fail-any_type,enc(t[],Kaead[],non[])).
42
43 3. By 2, the attacker may know fail-any_type (resp. enc(t[],Kaead[],non[])).
44 So the attacker may test the failure of this term, which may allow it to
45 distinguish cases.
46 bad.
47 Initial state
48 Additional knowledge of the attacker:
49 ・・・
50 (omitted)
51 ・・・
52 1-proj-2-tuple(~M) =
       choice[fail-any_type,enc(t_1,Kaead,non_1)]
53 ~M =
       choice[AEAD_MtE_Enc(Kaead,non_1,t_1,ad_1),
    AEAD_EtM_Enc(Kaead,non_1,t_1,ad_1)]
54 ----------------------------------------------
55 3rd process: Reduction 0
56 2nd process: Reduction ! 0 copy(ies)
57 1st process: Reduction ! 0 copy(ies)
58 New processes:
59
60 The attacker tests whether 1-proj-2-tuple(~M) =
       choice[fail-any_type,
61 enc(t_1,Kaead,non_1)] is fail.
62 This holds on one side and not on the other.
63 A trace has been found.
64 RESULT Observational equivalence cannot be
proved.
65 Looking for simplified processes ...
66 No simplified process found.
67 ----------------------------------------------
68 Verification summary:
69 Observational equivalence cannot be proved.
70 ----------------------------------------------
```

Biprocess 0とBiprocess 1 とのそれぞれで, 観測等価性を検証

それぞれの項の区別ができる可能性がある

攻撃導出手順

4. SCIS2024 発表内容

観測等価性の同値性が成立しないモデルの形式化 -AEAD model- Log

(* Code.4-Log: Tamarin prover-AEAD model *)

1 ・・・

2 (omitted)

3 ・・・

4 // Function signature and definition of the equational theory E

5

等式論に従う規則の設定

```
6 functions: enc/3, fst/1, left/1, mac/2, pair/2, right/1, snd/1
7 equations: fst(<x.1, x.2>) = x.1, snd(<x.1, x.2>) = x.2
8 rule (modulo E) Setup:
9 [ Fr( ~k1 ), Fr( ~k2 ) ]
10 --[ OnlyOnce( ), Setup( ~k1, ~k2 ) ]->
11 [ !Key( ~k1 ), !Key( ~k2 ) ]
12
13 rule (modulo E) Alice_makemac_then_enc:
14 [ Fr( ~m ), !Key( ~k1 ), !Key( ~k2 ), Fr( ~nonce ), Fr( ~data ) ]
15 -->
16 [
17 Out( <enc(<~m, mac(<~m, ~nonce, ~data>, ~k1)>, ~k2,
     ~nonce), ~nonce,
18 ~data, mac(<~m, ~nonce, ~data>, ~k1)>
19 )
20 ]
21
22 rule (modulo E) Alice_enc_then_mac:
23 [ Fr( ~m ), !Key( ~k1 ), !Key( ~k2 ), Fr( ~nonce ), Fr( ~data ) ]
24 -->
25 [
26 Out( <enc(~
27 mac(<enc(~
28 )
29 ]
30
```

```
31 rule (modulo E) diff_check:
32 [
33 In( <c1, ~nonce, ~data,
    alicetag> ), In( <c2, ~nonce,
    ~data, alicetag> )
34 ]
35 -->
36 [ Out( diff(c1, c2) ) ]
37
38 restriction unique [right]:
39 "∀ #i #j. ((OnlyOnce( ) @ #i) ∧
    (OnlyOnce( ) @ #j)) ⇒ (#i = #j)"
40 // safety formula
41
47
48 diffLemma Observational_equivalence:
```

[Tamarin prover]

```
49 rule-equivalence
50   case Rule_Equality
51   backward-search
52     case LHS
53     step( simplify )
54     step( solve( !KD( x ) ▶₁ #i ) )
55       case diff_check
56       step( solve( (#vl, 0) ~~> (#i, 1) ) )
57         case Var_msg_t
58         step( solve( !KU( ~nonce ) @ #vk.5 ) )
59           case Alice_enc_then_mac_case_1
60           step( solve( !KU( ~data ) @ #vk.6 ) )
61             case Alice_enc_then_mac_case_1
62             by ATTACK // trace found
63             qed
64           qed
65         qed
66       qed
67   qed
68 qed
69
70 end
71 ==========================================
72 summary of summaries:
73 DiffLemma: Observational_equivalence : falsified - found trace
74 ==========================================
```

後方探索に従いルールの等価性を左右で検証し、falsifiedと判定

ProVerifが出力した攻撃導出手順の内容と同じ

4. SCIS2024 発表内容

観測等価性の同値性が成立しない場合:
Observational equivalence cannot be proved.

検証結果の違い

Tool Name	Simple	AEAD
ProVerif モデル検査器	Cannot be proved	Cannot be proved
Tamarin-prover 定理証明器	falsified	falsified

Tamarin prover は明確に“falsified”であると示す

ProVerif が出力するLog を詳細に検証すると, Tamarin prover と同じ検証結果の内容

ProVerifで出力する"cannot be proved"の部分を,
Tamarin prover の命題にして,手動で証明することが出来なかった

5. 関連研究1

[タイトル]
“SAPIC+: Protocol Verifiers of the World, Unite!,"
In USENIX Security Symposium (USENIX Security), 2022.

ProVerifで証明出来ない仮定をTamarinで証明

[作者]
Vincent Cheval, Charlie Jacomme, Steve Kremer, Robert Künnemann

6.2.3 Tamarin proofs of ProVerif axioms

Since its recent update [18] ProVerif allows specifying axioms that can be used to prove a protocol by adding an extra assumption that may hold but that ProVerif cannot prove. An example of such a case is given in Example 6 of [18], where an axiom is used to specify that an action can only occur once. Consider the following subprocess

```
new stamp; in(c,xr); event S(stamp,xr); new r;
out(c,sign(aenc(vote,(r,xr),pk (sk e),sk ))
```

イベントSが与えられたタイムスタンプで一度だけ発生することを証明することができない。

On such a process, ProVerif cannot prove that the event S can only occur once with a given timestamp, due to its abstraction. As this fact is required to prove the expected security properties, the following axiom is needed:

```
All stamp xr xr2 #i #i2. S(stamp,xr)@i & S(stamp,xr2)@i2
      ⟹ i=i2 & x=xr2
```

ProVerifはこれを証明できないが、Tamarinが典型的にうまく処理する種類の式である。

While ProVerif cannot prove this, this is the sort of formula that Tamarin typically handles well. We thus ported this example to SAPIC$^+$, successfully using Tamarin to prove the axiom, and ProVerif to prove with the axiom the expected security property, both being formally combined.

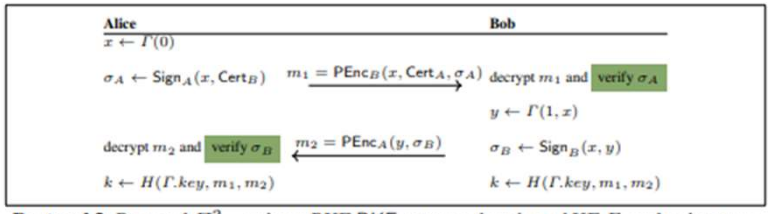
Axioms are also typically how GSVerif [21] functions: its adds axioms about states, axioms that were proven correct by hand. Once again using SAPIC$^+$, we proved for the previous 5G example that the axiom generated by GSVerif did hold thanks to Tamarin.

5. 関連研究2

ProVerifと Tamarin proverで認証プロトコルの検証比較

[タイトル]
Privacy Preserving Authenticated Key Exchange　Modelling, Constructions, Proofs and Formal Verification

[作者]
Andreas Weninger



Protocol 3: Protocol Π^2_{PKE} using a PKE PKE, an unauthenticated KE Γ, and a signature scheme Σ. where Certs contain Σ and PKE public keys.

論文内容)
PPAKEプロトコルのモデリング、構成、証明、形式的検証
PPAKE（Privacy Preserving Authenticated Key Exchange）プロトコルは、通信相手の身元を第三者から隠すことを目的とした認証鍵交換（AKE）プロトコル

結論)
・Tamarin proverを使用してプロトコルを評価したが、
　与えられたリソース（約60GBのメモリ）では証明を終了することも矛盾を見つけることもできなかった。
・ProVerifを活用し、プロトコルの安全性を証明することができた。
　特にif/then/elseステートメントが書けるため、結果の信頼性を高めるために有益であることを論証

課題

#モデル化する際の主な課題は、Tamarin proverの言語に if/else構造がないこと

Tamarin proverは単純な Lemmaの証明には有効であるが、
　Strong MITM privacy の観測等価性証明は実行不可能であることが判明

（→ProVerifは strong MITM privacyと前方秘匿性を証明することに成功）

5. 関連研究3

[タイトル]
ProVerif with Lemmas, Induction, Fast Subsumption, and Much More
https://doi.org/10.1109/SP46214.2022.9833653

[作者]
Bruno Blanchet, Vincent Cheval and Veronique Cortier

Algorithm 2: $\texttt{prove}(\mathcal{C}, \mathcal{A}x, \mathcal{L}, \mathcal{L}_i, \mathcal{R}, \mathcal{Q})$: Verification procedure for correspondence queries

input: An initial configuration $\mathcal{C}$, sets of axioms $\mathcal{A}x$, proved lemmas $\mathcal{L}$, inductive lemmas $\mathcal{L}_i$, restrictions $\mathcal{R}$ and queries $\mathcal{Q}$.

$\mathbb{C} := \mathbb{C}_{\mathcal{P}}(\mathcal{C}) \cup \mathbb{C}_{\mathcal{A}}(\mathcal{C})$
$\mathbb{C}_{sat} := \mathsf{saturate}_{\mathcal{L} \cup \mathcal{A}x \cup \mathcal{R}, \mathcal{L}_i}(\mathbb{C})$
return $\forall \varrho \in \mathcal{Q} \cup \mathcal{L}_i$
  suppose $\varrho = F_1 \wedge \cdots \wedge F_n \Rightarrow \psi$
  $G_1 := \lceil F_1 \rceil^m, \ldots, G_n = \lceil F_n \rceil^m$
  $R_q := G_1@t_1 \wedge \ldots \wedge G_n@t_n \rightarrow and_{G_1,\ldots,G_n}@(t_1, \ldots, t_n)$
  if $\varrho \in \mathcal{L}_i$ **then**
    $\mathbb{C}_s := \mathsf{saturateS}_{\mathcal{L} \cup \mathcal{A}x \cup \mathcal{R}, \mathcal{L}_i}(\{R_q\}, \mathbb{C}_{sat})$
  else
    $\mathbb{C}_s := \mathsf{saturateS}_{\mathcal{L} \cup \mathcal{L}_i \cup \mathcal{A}x \cup \mathcal{R}, \emptyset}(\{R_q\}, \mathbb{C}_{sat})$
  $\forall R \in \mathbb{C}_s.\ R \models \varrho$

論文内容）
2つの主要な貢献
1. ProVerifをLemma、公理、帰納法による証明、自然数、時間的クエリを拡張
2. ProVerifで使用されているアルゴリズムの多くを手直しし、最適化することで、目覚ましいスピードアップを実現

詳細）
・ProVerifの精度、効率、表現力を改善し、ProVerifが証明を完了するのを助ける中間的な性質をユーザが宣言できるようにすることで、ある程度の対話性を導入
・時間クエリーのサポート

・ProVerifはより多くのプロトコル、特に到達性と等価性の両方についてグローバル・ステートを持つプロトコルをサポートできるようになった。
・GSVerifというツールが導入され（到達可能性プロパティの文脈で）、ProVerifでは証明できないが成立することが証明されたプロパティを自動的に生成する。
・GSVerifによって生成された特性は、公理として記述することができるようになり、手順の早い段階で使用することができるようになった。

6. 検証

ProVerifが "cannot be proved" を出力するパターンは下記に分類される.

1. 観測等価性の同値性が成立しない場合: Observational equivalence cannot be proved.
2. 到達可能性が導けない場合: Query not event(・・・) cannot be proved.
3. ProVerifが保持する特性の証明失敗

2.到達可能性が導けない場合:
Query not event(・・・) cannot be proved.

複数の論文にて,cannot be proved となる結果が示されているが,
いずれもLogの検証を手動で詳細に行っている.

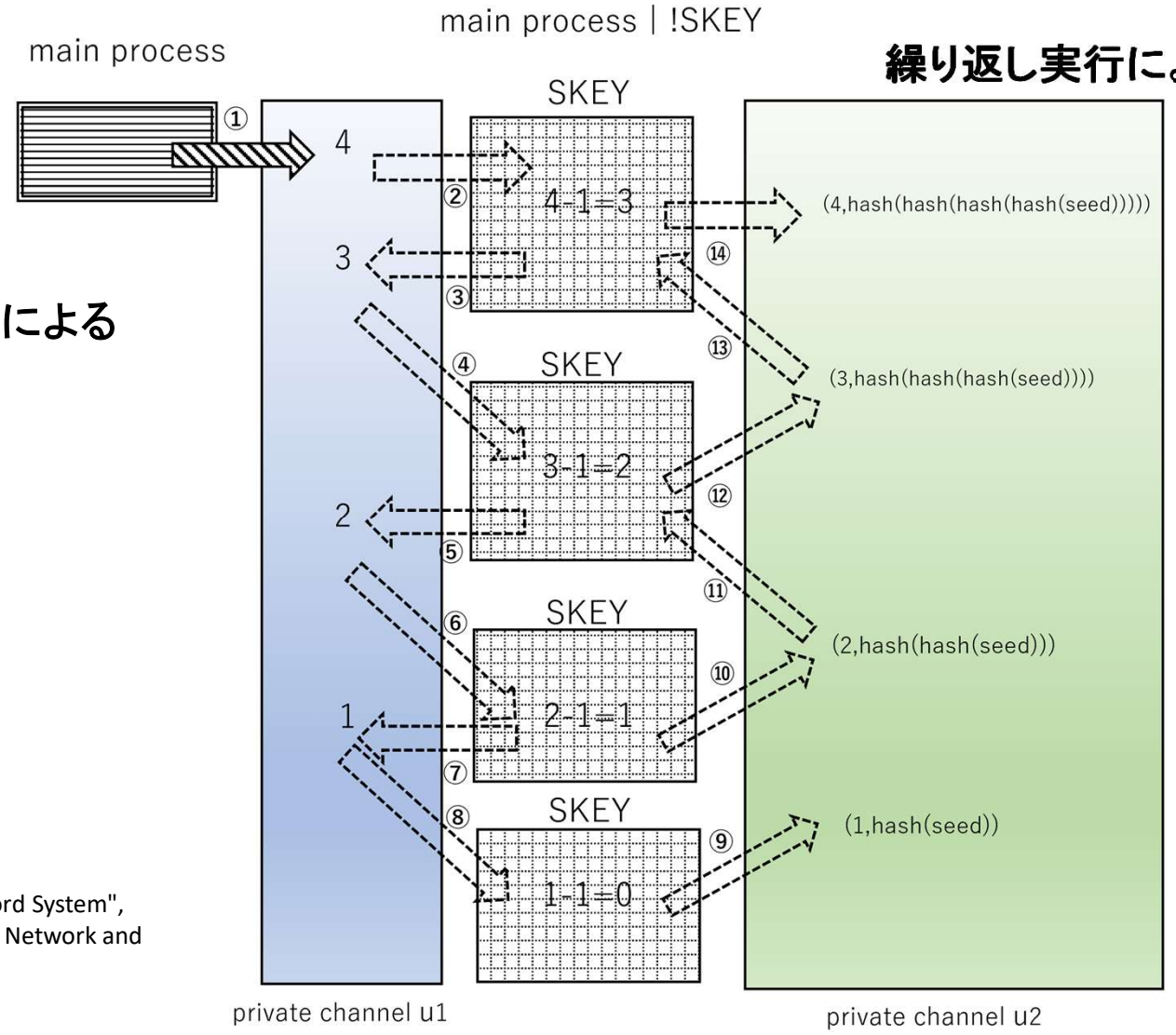
6.1 検証 S/Key model の形式化

到達可能性が導けない場合

S/Key

RFC1670

ワンタイムパスワードによる
認証方式の一つ



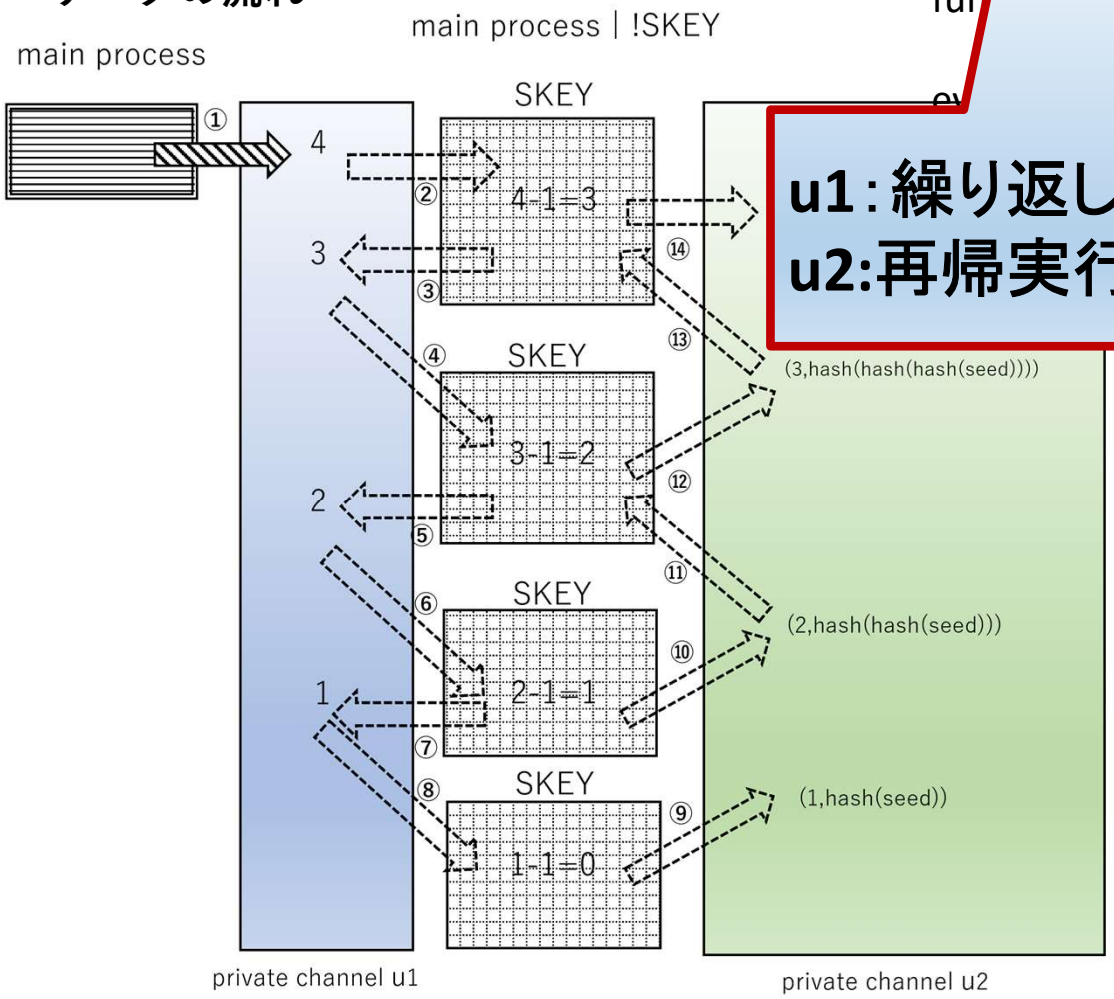
繰り返し実行によって“hash”関数を計算

[参考文献]
Haller, N., "The S/KEY One-Time Password System",
Proceedings of the ISOC Symposium on Network and
Distributed System Security,
February 1994, San Diego, CA

6.1 検証 S/Key model

到達可能性が導けない場合

データの流れ




```
free u1:channel[private].
free u2:channel[private].
free s ed:bitstring [private].

fun          string):bitstring.
```

```
let SKEY(s:bitstring) =
  in(u1, x3:nat);
  let x4:nat = x3-1 in
  if (x4 <
  (
    out(u1
    in(u2,(x5:nat,sk:bitstring));
    if(otp4=hash(hash(hash(hash(seed))))) then
      event COL
    )
  )
  else
    out(u2, (x3,hash(seed))).

process
  let a1:nat = 4 in
  !(
   out(u1, a1)
  )
  | !SKEY(seed)
```

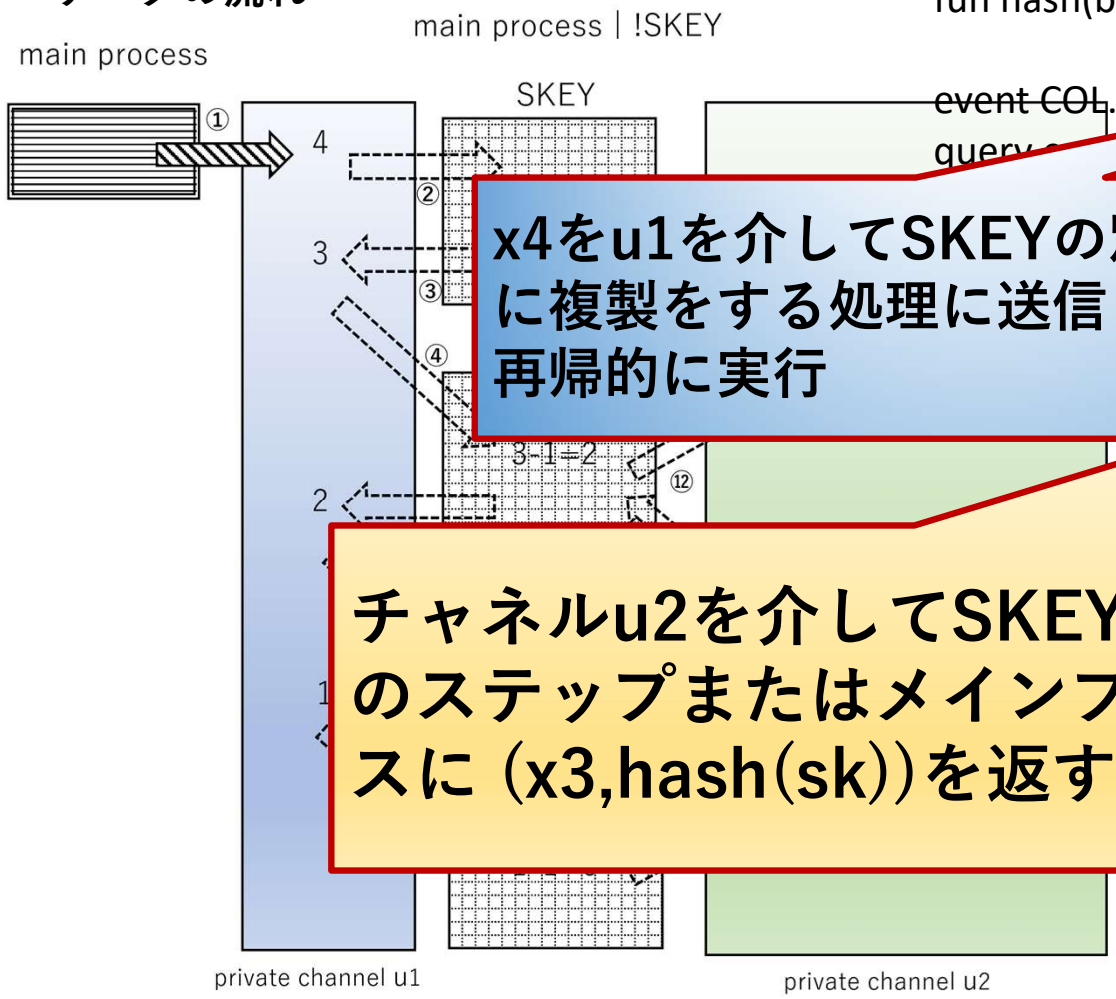
u1を介してNAT項x3を受け取る

u1：繰り返しのカウンターのチャネル
u2:再帰実行中の一時データのチャネル

6.1 検証 S/Key model

到達可能性が導けない場合

データの流れ



x4をu1を介してSKEYの別に複製をする処理に送信し，再帰的に実行

チャネルu2を介してSKEYの前のステップまたはメインプロセスに（x3,hash(sk)）を返す

```
free u1:channel[private].
free u2:channel[private].
free seed:bitstring [private].

fun hash(bitstring):bitstring.

event COL.
query ...COL).
```

```
let SKEY(s:bitstring) =
  in(u1, x3:nat);
  let x4:nat = x3-1 in
  if (x4 <> 0) then
  (
   out(u1, x4);
   in(u2,(x5:nat,sk:bitstring));
   if(x5=x4) then
   (
    let otp4 = hash(sk) in
    out(u2,(x3,otp4));
    if(otp4=hash(hash(...h(hash(seed))))) then
      event COL
   )
  )
  else
   out(u2, (x3,hash...).
```

```
process
  let a1:nat =
  !(
   out(u1, a1
  )
  | !SKEY(seed)
```

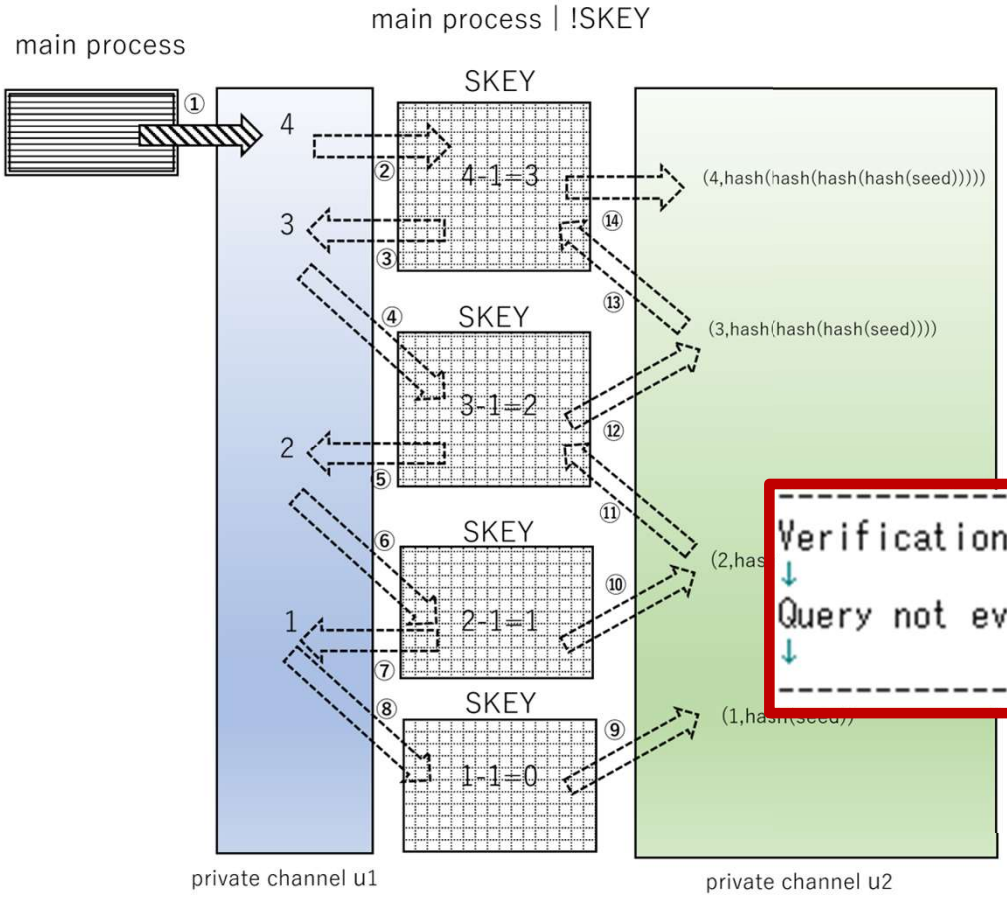
SKEYの別の複製する処理からプライベートチャネルu2を介して項のペア（x5,sk）を受け取る

6.1 検証 S/Key model の形式化

検証結果

RFC1670

繰り返し実行によって“hash”関数を計算



到達可能性が導けない場合

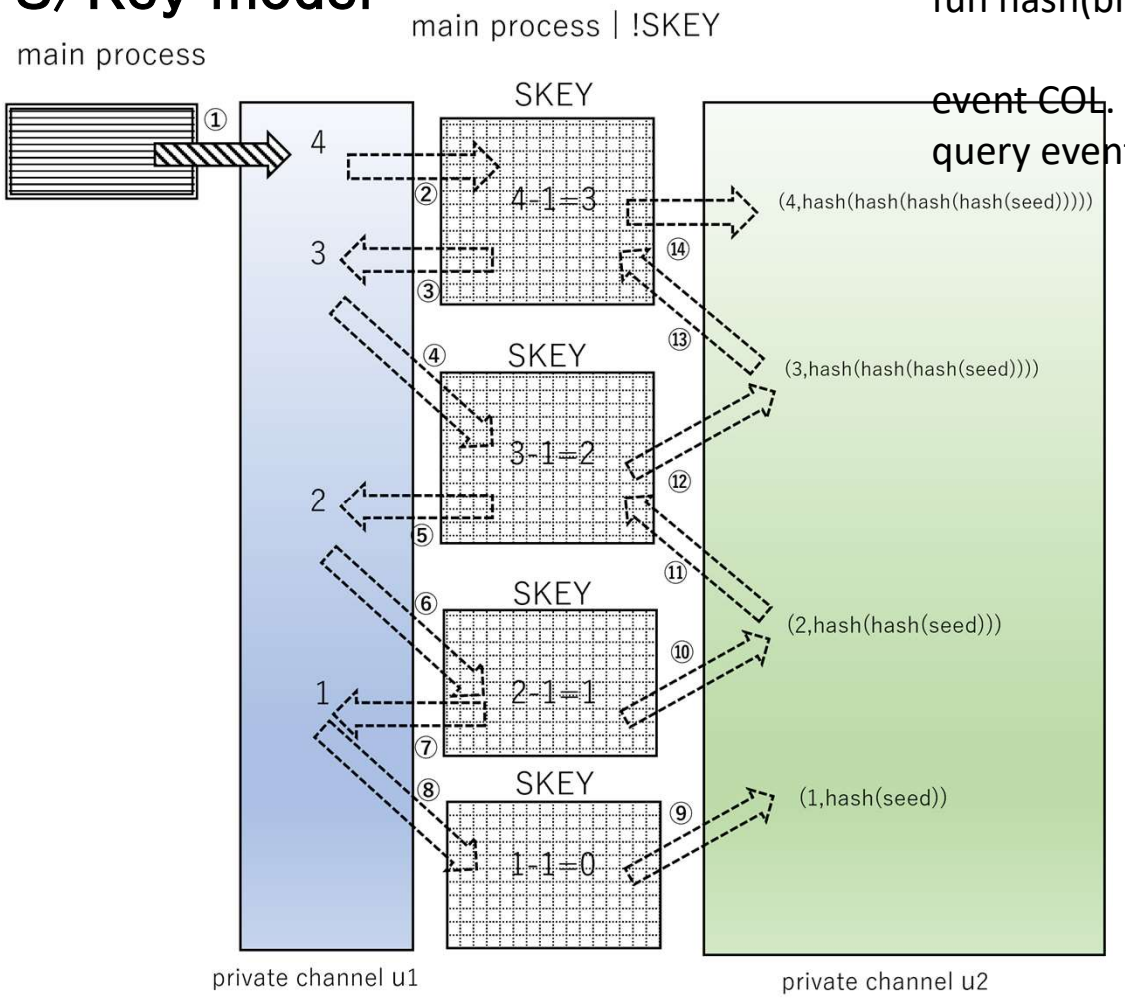
```
4. The message 3 that may be sent on channel u1[] by 3 may be received at input {5}.
We have 2 ≠ 0.
So the message 2 may be sent on channel u1[] at output {8}.
mess(u1[],2).

5. The message 4 that may be sent on channel u1[] by 1 may be received at input {5}.
We have 3 ≠ 0.
So the message 3 may be sent on channel u1[] at output {8}.
mess(u1[],3).

6. The message 3 that may be sent on channel u1[] by 5 may be received at input {5}.
We have 2 ≠ 0.
So the message 2 may be sent on channel u1[] at output {8}.
mess(u1[],2).

7. The message 2 that may be sent on channel u1[] by 6 may be received at input {5}.
We have 1 ≠ 0.
So the message 1 may be sent on channel u1[] at output {8}.
mess(u1[],1).

8. The message 1 that may be sent on channel u1[] by 7 may be received at input {5}.
So the message (1,hash(seed[])) may be sent on channel u2[] at output {15}.
mess(u2[],(1,hash(seed[]))).

9. The message 2 that may be sent on channel u1[] by 4 may be received at input {5}.
The message (1,hash(seed[])) that may be sent on channel u2[] by 8 may be received at input {9}.
We have 1 ≠ 0.
So the message (2,hash(hash(seed[]))) may be sent on channel u2[] at output {12}.
mess(u2[],(2,hash(hash(seed[])))).

10. The message 3 that may be sent on channel u1[] by 2 may be received at input {5}.
The message (2,hash(hash(seed[]))) that may be sent on channel u2[] by 9 may be received at input {9}.
We have 2 ≠ 0.
So the message (3,hash(hash(hash(seed[])))) may be sent on channel u2[] at output {12}.
mess(u2[],(3,hash(hash(hash(seed[]))))).

11. The message 4 that may be sent on channel u1[] by 1 may be received at input {5}.
The message (3,hash(hash(hash(seed[])))) that may be sent on channel u2[] by 10 may be received at input {9}.

------------------------------------------------------------
Verification summary:

Query not event(COL) cannot be proved.

------------------------------------------------------------
```

```
----------------------------------------------------------------↓
Verification summary:↓
↓
Query not event(COL) is false.↓
↓
----------------------------------------------------------------↓
```

6.2 プライベート通信路の有無による検証結果の違い

到達可能性が導けない場合

S/Key model

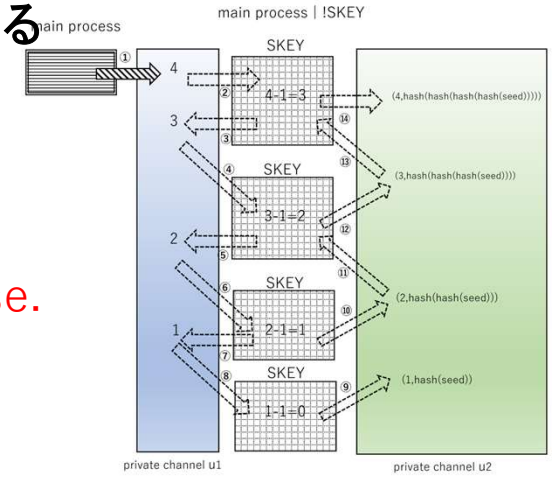



```
free u1:channel[private].
free u2:channel[private].
free seed:bitstring [private].

fun hash(bitstring):bitstring.

~~event COL~~.
query event(COL).
```

```
let SKEY(s:bitstring) =
    in(u1, x3:nat);
    let x4:nat = x3-1 in
    if (x4 <> 0) then
    (
     out(u1, x4);
     in(u2,(x5:nat,sk:bitstring));
     if(x5=x4) then
     (
      let otp4 = hash(sk) in
      out(u2,(x3,otp4));
      if(otp4=hash(hash(hash(hash(seed))))) then
        event COL
     )
    )
    else
     out(u2, (x3,hash(seed))).

process
    let a1:nat = 4 in
    !(
     out(u1, a1)
    )
    | !SKEY(seed)
```

6.2 S/Key model
プライベート通信路の有無による検証結果の違い

公開・非公開チャンネルにおける
繰り返し実行方法について
動作が異なる



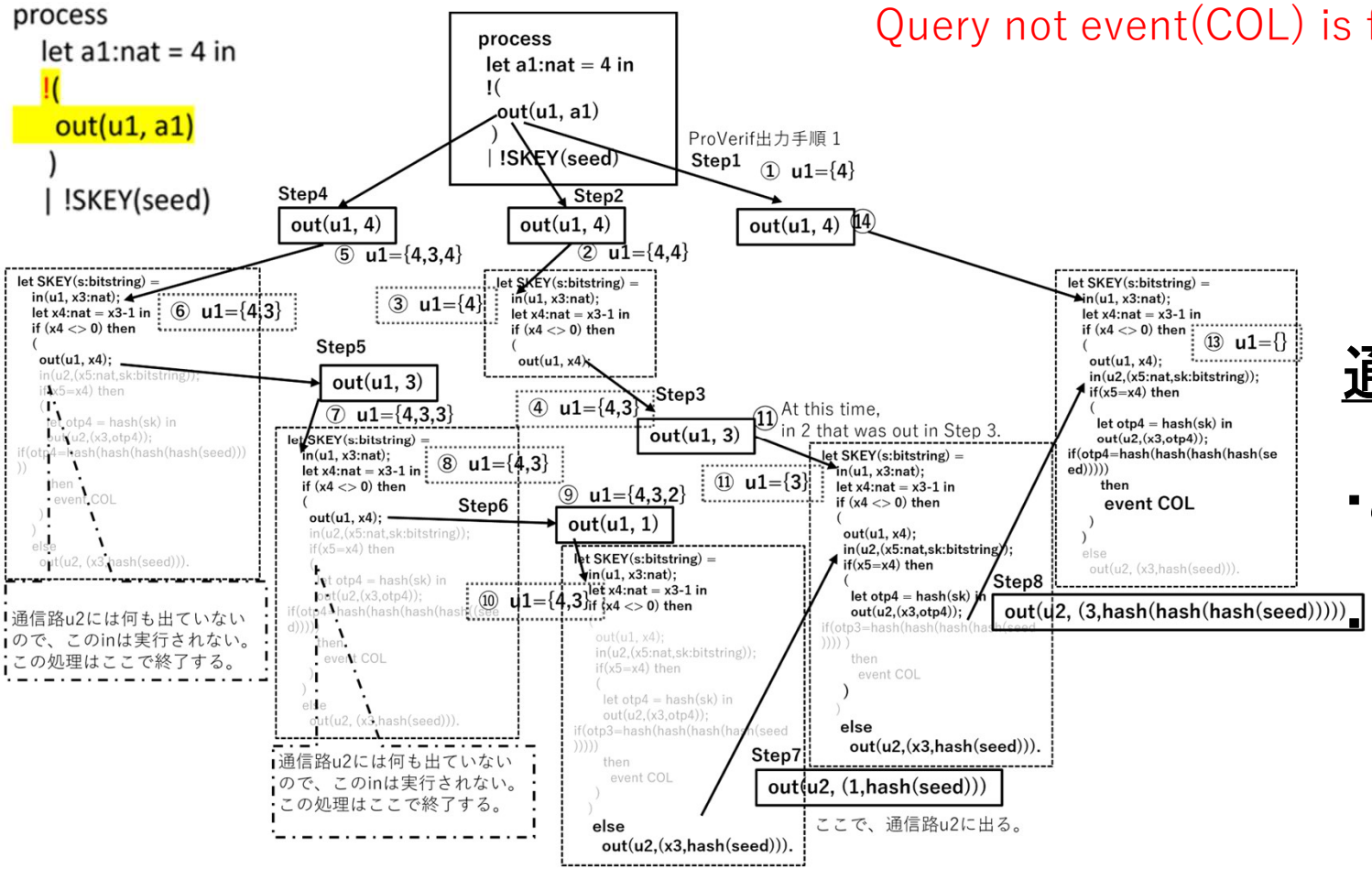
検証結果:

Query not event(COL) is false.

“!” あり Ver.

!SKEY(seed)

```
process
  let a1:nat = 4 in
  !(
    out(u1, a1)
  )
  | !SKEY(seed)
```



通信路を集合と見立てる

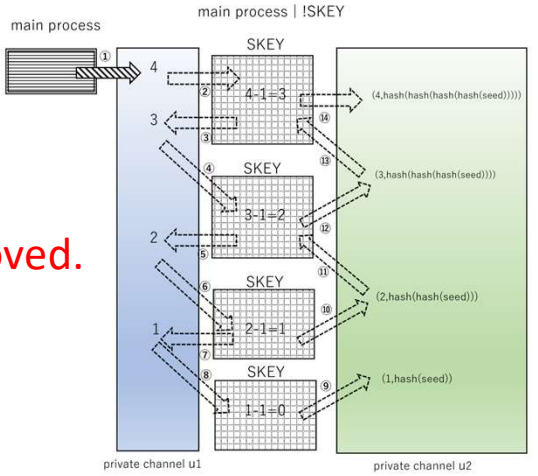
- ある項を outするとその項が集合に追加
  inするとその項が集合から取り出される

6.2 S/Key model

プライベート通信路の有無による検証結果の違い !SKEY(seed)

到達可能性が導けない場合

“!” なし の場合.



検証結果:

Query not event(COL) cannot be proved.

```
process
  let a1:nat = 4 in
  (
   out(u1, a1)
  )
  | !SKEY(seed)
```

```
process
 let a1:nat = 4 in
 !(
  out(u1, a1)
 )
 | !SKEY(seed)
```

Step1

out(u1, 4)

Step2

out(u1, 3)

Step3

out(u1, 3)

Step4

out(u1, 2)

Step5

out(u2, (2,hash(hash((seed))))

ここで、通信路u2に出る。

Step6

out(u2, (3,hash(hash(hash((seed))))))

```
let SKEY(s:bitstring) =
  in(u1, x3:nat);
  let x4:nat = x3-1 in
  if (x4 <> 0) then
  (
   out(u1, x4);
   in(u2,(x5:nat,sk:bitstring));
   if(x5=x4) then
   (
    let otp4 = hash(sk) in
    out(u2,(x3,otp4));
if(otp4=hash(hash(hash(hash((seed))))))
    then
     event COL
   )
  )
  else
   out(u2, (x3,hash(seed))).
```

通信路u2には何も出ていないので、このinは実行されない。この処理はここで終了する。

通信路u2には何も出ていないので、このinは実行されない。この処理はここで終了する。

パブリック通信路の場合は,
“out(u1,a1)"を1回実行するだけで,
攻撃者が “out(u1,a1)"を何度でも実行できるため,
“out(u1,a1)”の複製は必要ない

S/Key model

6.2 プライベート通信路の有無による検証結果の違い

到達可能性が導けない場合

検証結果: Query not event(COL) cannot be proved.

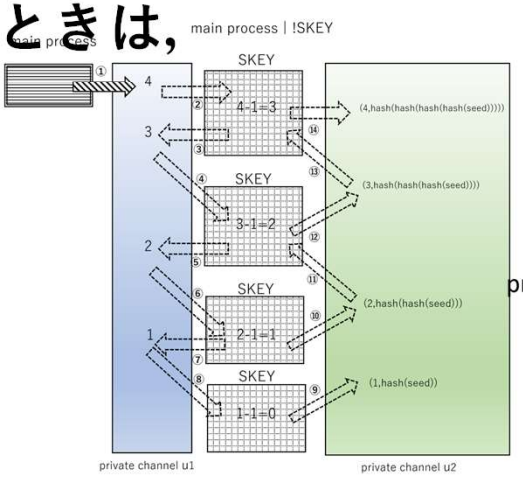
例) **free c:channel[private].**

"プライベート通信路"を使うと，cannot be proved になることがよくある．

通信路は集合とみたてることができる

通信路に in すると集合に要素が追加され，out するすると，集合から削除される

通信路がパブリックならら問題にならないが，通信路がプライベートだと，何かを in したいときは，事前に out されている必要がある




```
let SKEY(s:bitstring) =
    in(u1, x3:nat);
    let x4:nat = x3-1 in
    if (x4 <> 0) then
    (
     out(u1, x4);
     in(u2,(x5:nat,sk:bitstring));
     if(x5=x4) then
     (
      let otp4 = hash(sk) in
      out(u2,(x3,otp4));
      if(otp4=hash(hash(hash(hash(seed))))) then
       event COL
     )
    )
    else
     out(u2, (x3,hash(seed))).

process
  let a1:nat = 4 in
  !(
   out(u1, a1)
  )
  | !SKEY(seed)
```

Tamarin prover の場合のS/Key検証は、
特にこのような検証結果の違いはない。

6.3 検証が止まらないモデル

CBC model

代表的なブロックチェイニングであるCBC(Cipher Block Chaining)

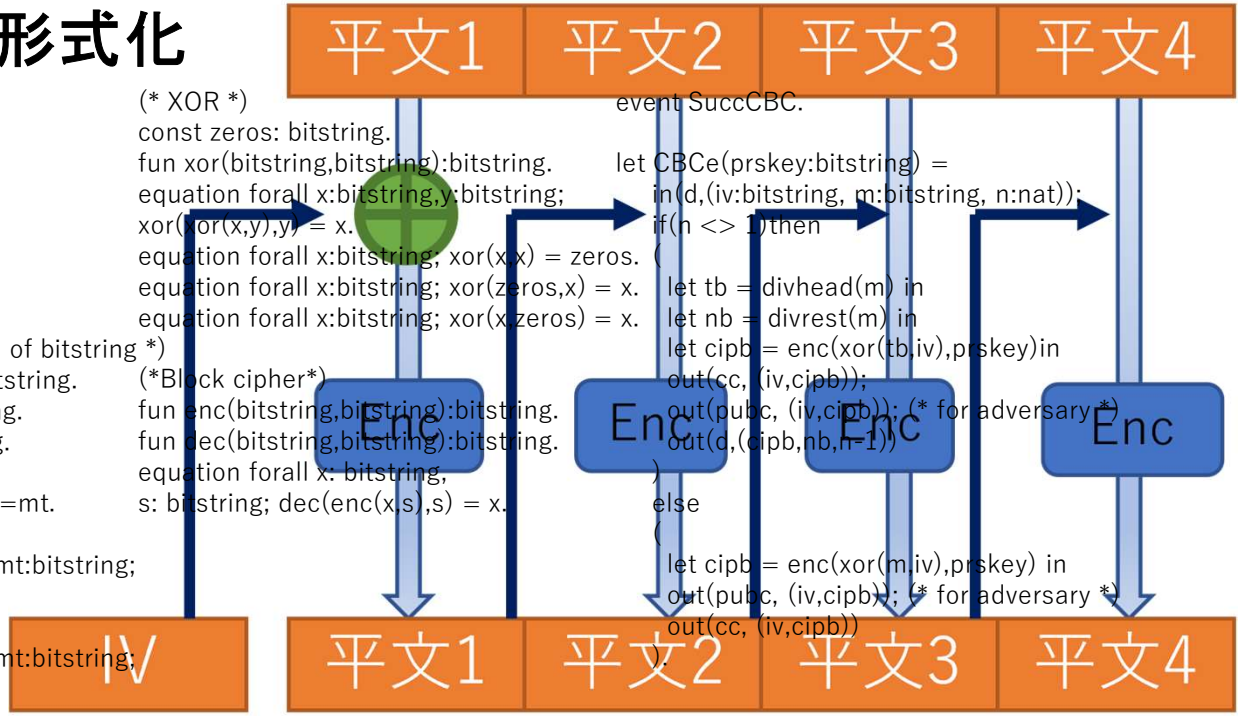
- CBC Modeでの暗号化および復号処理が正しく動作
- CBC Modeは選択平文攻撃に対して弱い

CBC Modeの形式化

```
free d:channel[private].
free dd:channel[private].
free ee:channel[private].
free cc:channel[private].
free pubc:channel.
free ts:bitstring[private].
free sskey:bitstring[private].

(* concatenation and division of bitstring *)
fun con(bitstring,bitstring):bitstring.
fun divhead(bitstring):bitstring.
fun divrest(bitstring):bitstring.
equation forall mt:bitstring;
con(divhead(mt),divrest(mt))=mt.

equation forall lmt:bitstring,rmt:bitstring;
divhead(con(lmt,rmt))=lmt.

equation forall lmt:bitstring,rmt:bitstring;
divrest(con(lmt,rmt))=rmt.
```

```
(* XOR *)
const zeros: bitstring.
fun xor(bitstring,bitstring):bitstring.
equation forall x:bitstring,y:bitstring;
xor(xor(x,y),y) = x.
equation forall x:bitstring; xor(x,x) = zeros.
equation forall x:bitstring; xor(zeros,x) = x.
equation forall x:bitstring; xor(x,zeros) = x.

(*Block cipher*)
fun enc(bitstring,bitstring):bitstring.
fun dec(bitstring,bitstring):bitstring.
equation forall x: bitstring,
s: bitstring; dec(enc(x,s),s) = x.
```

```
event SuccCBC.

let CBCe(prskey:bitstring) =
  in(d,(iv:bitstring, m:bitstring, n:nat));
  if(n <> 1)then
  (
    let tb = divhead(m) in
    let nb = divrest(m) in
    let cipb = enc(xor(tb,iv),prskey)in
    out(cc, (iv,cipb));
    out(pubc, (iv,cipb)); (* for adversary *)
    out(d,(cipb,nb,n-1))
  )
  else
  (
    let cipb = enc(xor(m,iv),prskey) in
    out(pubc, (iv,cipb)); (* for adversary *)
    out(cc, (iv,cipb))
  ).
```




```
let CBCd(prskey:bitstring) =
  in(dd, x2:nat);
  in(cc, (ciphb1:bitstring,ciphb2:bitstring));
  let plainb =dec(ciphb2,prskey) in
  if (ciphb2 = enc(plainb,prskey)) then
  let plaina = xor(plainb,ciphb1) in
  if (plainb = xor(plaina,ciphb1)) then
  let x3:nat = x2-1 in
  if (x3 <> 0) then
  (
    out(dd, x3);
    in(ee,(x4:nat,tailce:bitstring));
    if(x4=x3) then
    (
      let conb = con(plaina,tailce) in
      out(ee,(x2,conb));
      if(conb=ts) then event SuccCBC
    )
  )
  else
    out(ee, (x2,plaina)).
(* queries *)
query attacker(ts).
query event(SuccCBC).

(*execution part*)
process
  (
    new iv:bitstring;
    out(cc,(iv,4));
    out(d,(iv,ts,4));
    !(
      out(dd, 4)
    )
```

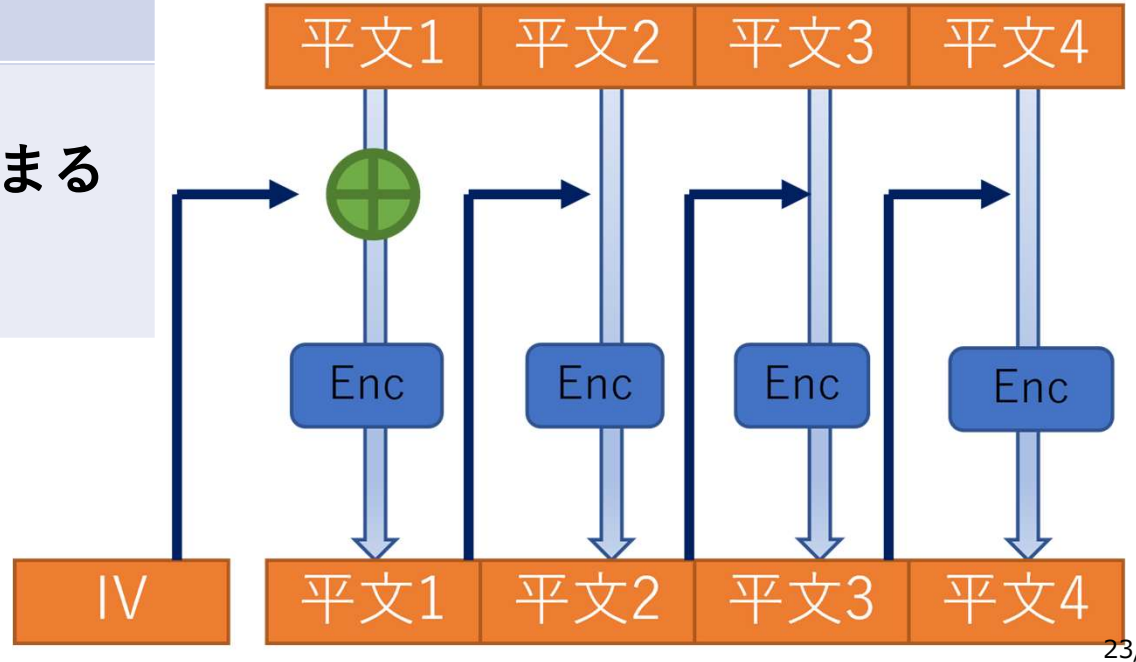
6.3 検証が止まらないモデル

PC Spec	Result 実行時間
Intel i7-9700, 48GB, OS : Ubuntu	×: 1時間実施しても 止まらない
Intel i9, 64GB, OS : Win11	○: 10分かかって止まる

参考)
古いPCでも128GBのメモリを積むと20分で止まる

CBC model

```
process
 (
  (* 初期値 *)
  new iv:bitstring;

  (* 暗号化処理 *)
  out(pric,(iv,iv,n3,ts));

  (* 復号処理 *)
  in(pubc, encbs:bitstring);
   (* 暗号文ブロックを繋いだものを公開通信路pubcよりin *)
  out(pric2,(iv,encbs,n3,true))
   (* ivではなくemptyなどでも良い *)
   (* trueは初回処理フラグ *)
 )
 |!CBCe(sskey)
 |!CBCd(sskey)
```



6.3 検証が止まらないモデル

Markle Damgård model

Feature

暗号学的ハッシュ関数

- **一方向性(one-wayness)**
- **衝突困難性(collision-resistance)**

衝突困難性(collision resistance)

$H(M) = H(M')$

ハッシュ値が一致する

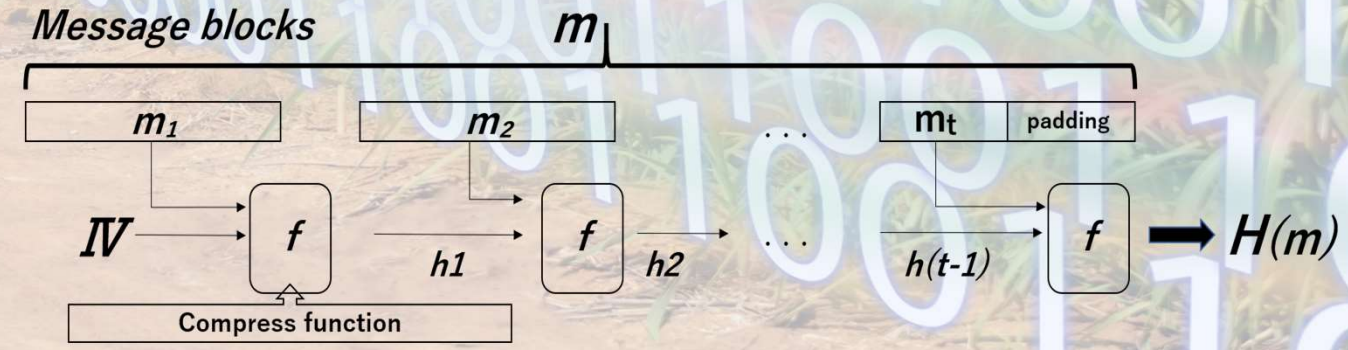
上記を満たす相異なる2つのM,M'を見つけることが計算量的に困難

Component

繰り返し型ハッシュ関数では

(1) パディング
(2) 固定長への分割
(3) 圧縮関数による繰り返し演算
(4) 出力

の順序で実行される。



6.3 検証が止まらないモデル

Markle Damgård model

```
free c:channel.
free s:channel[private].
free t:channel[private].
free m:bitstring[private].
const iv:bitstring.

fun con(bitstring,bitstring):bitstring.
fun divhead(bitstring):bitstring.
fun divrest(bitstring):bitstring.
equation forall mt:bitstring;
        con(divhead(mt),divrest(mt))=mt.

(* Compress function *)
fun comp(bitstring,bitstring):bitstring.

(* Padding function *)
fun pad(bitstring):bitstring.

table MD(bitstring,bitstring).

fun bindm(bitstring,nat):bitstring.

fun nofblocks(bitstring):nat
reduc forall stm:bitstring,blocks:nat;
      nofblocks(bindm(stm,blocks))=blocks.

fun unbindm(bitstring):bitstring.
equation forall stm:bitstring,n:nat;
        unbindm(bindm(stm,n))=stm.
```

```
(* Query *)
event COL.
query event(COL).

(* MD construction for adversary *)
let makeMDadv(mm':bitstring) =
    in(t,(rm:bitstring,ha:bitstring,mm:bitstring));
    if mm = mm' then
    let bl = nofblocks(rm) in
    if(bl <> 1) then
    (
      let newbl = divhead(rm) in
      let newstream = divrest(rm) in
      let bindns = bindm(newstream,bl-1) in
      let newha = comp(ha,newbl) in
      out(t,(bindns,newha,mm))
    ) else (
      let MDh = comp(ha,rm) in
      insert MD(mm',MDh);
      out(c,MDh)
    ).

(* Indicating-Process for Collision Resist      *)
let Collision =
    in(c,(m1:bitstring,m2:bitstring));
    get MD(=m1,MDm1) in
    get MD(=m2,MDm2) in
    if(m1 <> m2 && MDm1 = MD
    then event COL.

(* Main process *)
process
    !(
      in(c,(mm:bitstring,ni:nat));
      out(t,(bindm(pad(mm),ni),iv,mm))
      | !makeMDadv(mm)
    )
```

in(c,(mm:bitstring,ni:nat));
out(t,(bindm(pad(mm),ni),iv,mm))

処理が止まらない.

in(c,mm:bitstring);
out(t,(bindm(pad(mm),5),iv,mm))

数字を指定すると,
例えば,"5"だとすぐに止まり，trueとなる

我々の研究成果参照先

T. Mieno, T. Yoshimura, H. Okazaki, Y. Futa and K. Arai,
“Formal Verification of Merkle–Damgård Construction in ProVerif,”
2020 International Symposium on Information Theory and Its Applications (ISITA),
Kapolei, HI, USA, 2020, pp. 602-606.
https://ieeexplore.ieee.org/abstract/document/9366163

7. 考察

開発支援プロセスへの導入

検討案:

**初期段階でモデル検査器 ProVerif による検証を行う,
その後,検証結果に基づいて, 更に詳細な検証や
セキュリティ特性の証明を定理証明器 Tamarin prover で行う**

目標:

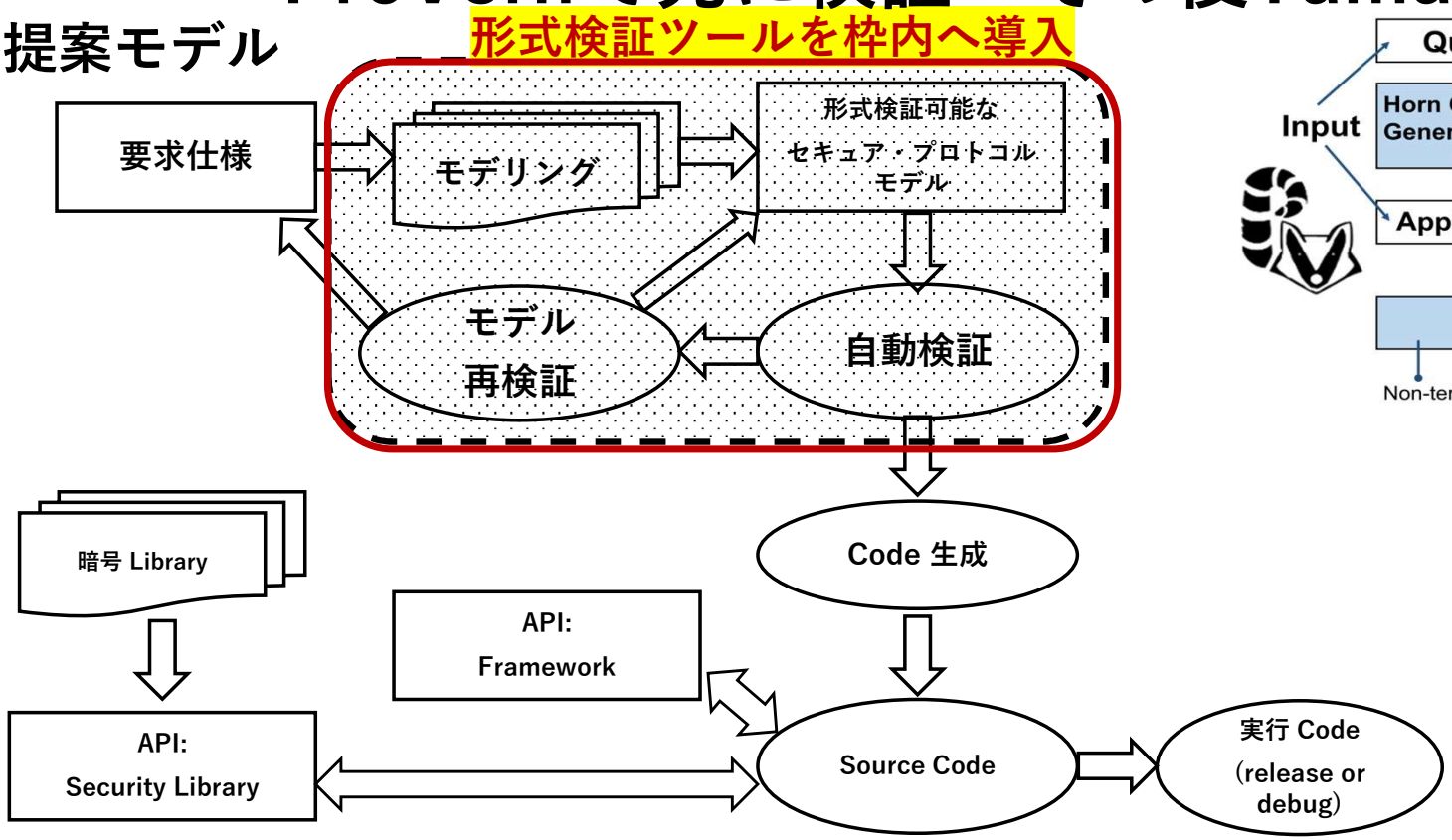
**セキュア・プロトコルの詳細な形式検証で,
安全なセキュア・プロトコルの設計に繋がることを目指す**

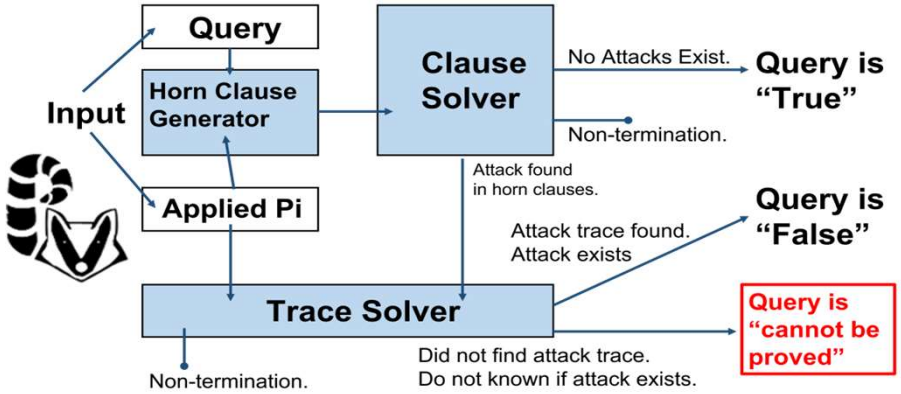
実装段階や運用段階でセキュリティ対策を行うのではなく, 開発設計段階での対策

実装や運用に移る前に対策が施されていることを確認, 検証できる仕組み

ProVerifで先に検証→その後Tamarin で詳細検証

提案モデル





- 両ツールの検証結果からセキュア・プロトコルのセキュリティ特性や弱点が明らかにできる
- 必要に応じてセキュア・プロトコルの改善や 修正を行い, 再度検証を繰り返せる

7. 考察

**初期段階でモデル検査器 ProVerif による検証を行う,
その後,検証結果に基づいて, 更に詳細な検証や
セキュリティ特性の証明を定理証明器 Tamarin prover で行う**

・ProVerifで書けないモデル(Syntax error 等)
・ProVerifでモデル検証に時間がかかる
(リソースが莫大に必要 or 現実的な時間で探索が終わらない)

↓

Tamarin prover を使って補完できる

↓

開発支援プロセスへの導入

まとめ

検証結果の違いによる明確な使い分けまでを検討できる根拠が不足

ProVerif は, 形式化したモデルによって, 検証結果が判断できないので, 検証者に攻撃導出手順の判断を委ねることがある.

　この場合にTamarin prover を用い, 同じモデルを検証することで, 厳密にセキュリティ特性や弱点を判断することができる可能性があるため, 両ツールの検証結果の違いを検証し, 開発支援プロセスへ導入するための考察を行った.

今後

・ProVerifで出力する"cannot be proved"の部分を,Tamarin prover の命題にして,手動で証明する
・補題の自動抽出を将来実施したい

	ProVerif	Tamarin prover
観測等価性の同値性が成立しない場合	cannot be proved	falsified
到達可能性が導けない場合	継続実施	継続実施
ProVerifが保持する特性の証明失敗	継続実施	継続実施

ご清聴ありがとうございました