Mizarによる離散確率分布の識別不能性の形式化

岡崎裕之（信州大学）

科研費基盤研究(C) 17K00182
形式手法による暗号の安全性証明自動検証システムの開発

目的

■Mizarを用いた暗号の安全性証明システム開発

- 暗号方式そのものの形式化
- アルゴリズムの計算量評価
- 確率分布、確率事象の扱い

暗号の安全性の定義（秘匿の安全性）

- 完全解読
  暗号文から（秘密鍵なしで）平文を計算する
- 強秘匿(semantic security)
  暗号文より平文を解読成功する条件付確率と暗号文なしで平文を解読成功する確率がほとんど違わない
- 識別不能(indistinguishability)
  乱数と暗号文が見分けがつかない

Alice want to send a message to Bob.

Choice $x = C$ or r

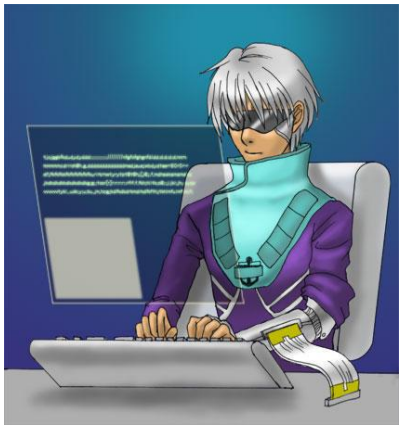
$C = enc(M, K)$

Send x

Adversary guesses whether x is a cipher text or a random nonce.

$M = dec(C, K)$

Pick r from Uniform distribution



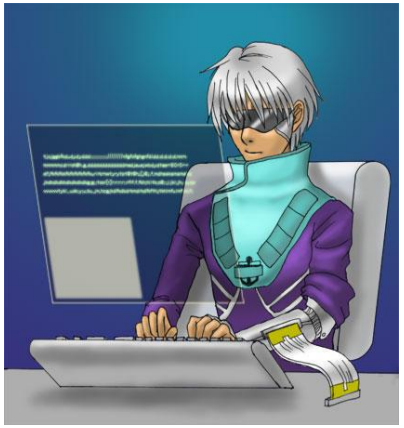
Public channel is shared magnetic tape

$C = enc(M, K)$

$C_0, C_1, \ldots, C_n, \ldots$

Pick r from
Uniform distribution

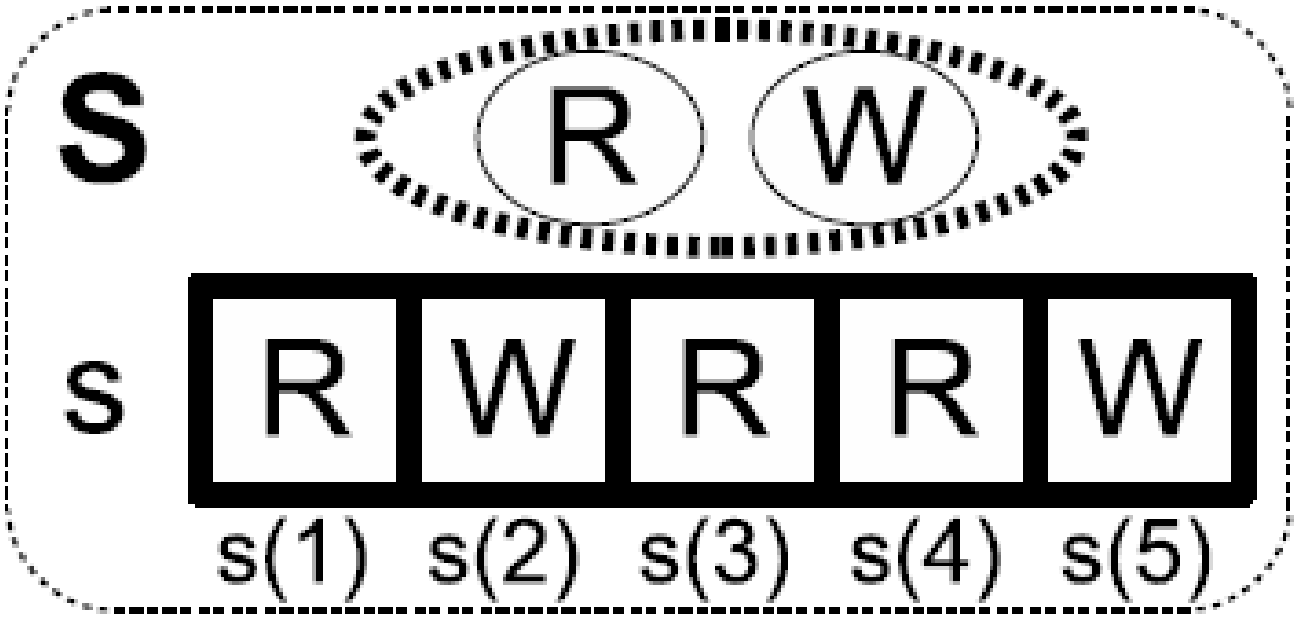
$r_0, r_1, \ldots, r_n, \ldots$

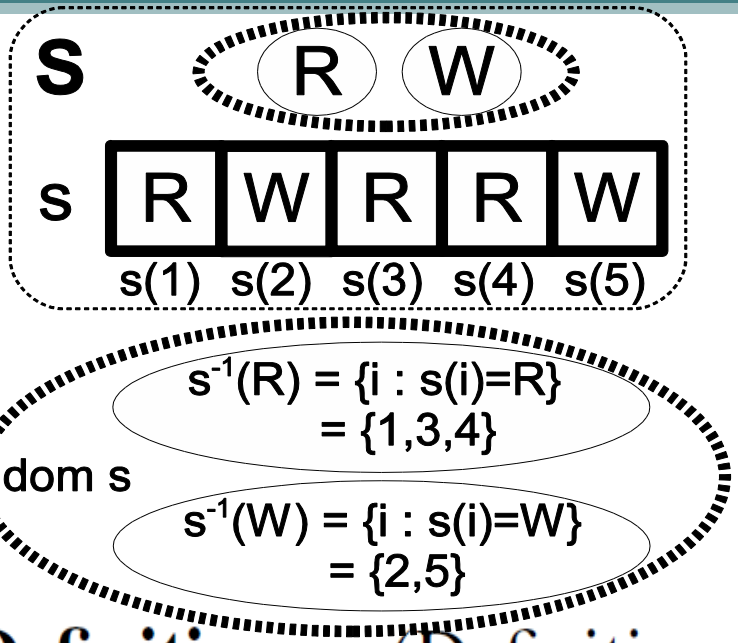


有限列を確率変数とみなす

袋の中に赤玉２個、 白玉3個あります。

赤白を並べた「有限列」





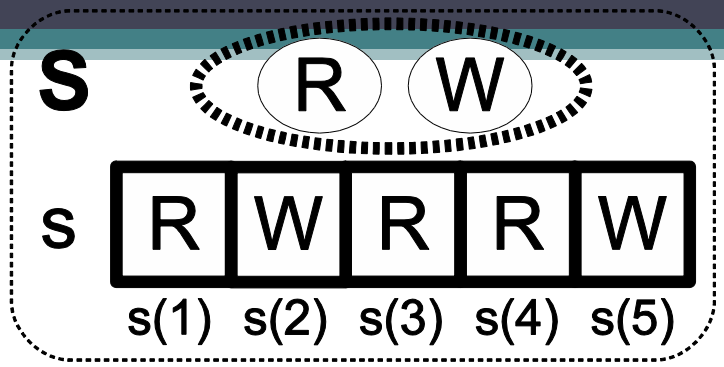
(各要素の確率)

$$s^{-1}(x) = \{i \in dom\ \ s : s(i) = x\}$$

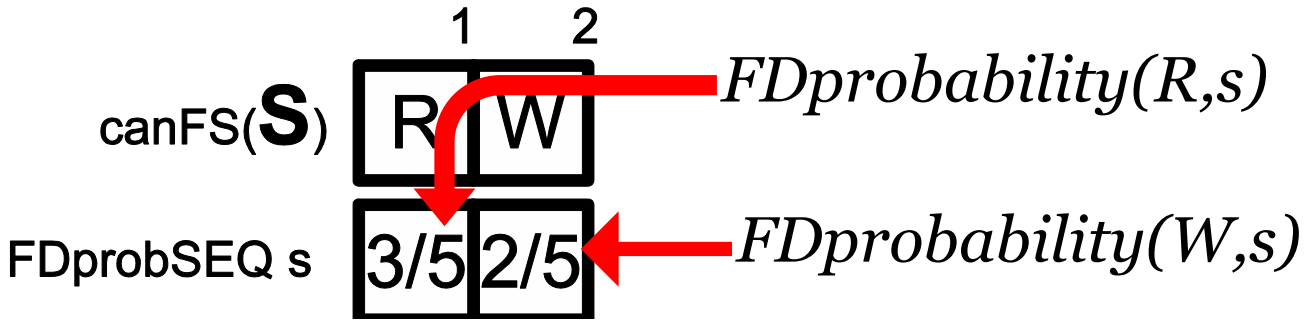
Definition (Definition of FDprobability)

Let S be a non empty finite set, let s be a finite sequence of elements of S, and let x be a set. Then, the functor FDprobability(x,s) yielding a real number is defined as follows:

$$FDprobability(x,s) = \frac{card\ s^{-1}(x)}{len\ s}.$$



(確率質量関数)



Definition (Definition of FDprobSEQ)

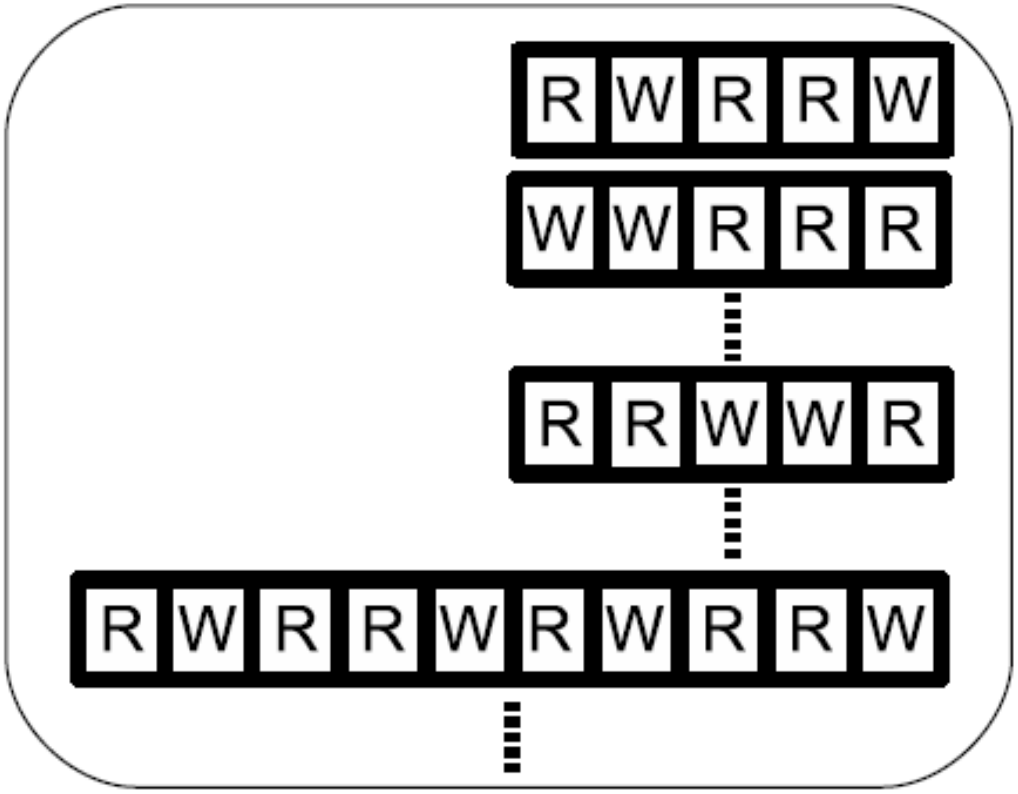
Let S be a non empty finite set and let s be a finite sequence of elements of S. Then, the functor FDprobSEQ s yielding a finite sequence of elements of $\mathbb{R}$ *is defined by:*

dom (FDprobSEQ s) = Seg(card S) and for every natural number n such that $n \in$ *dom (FDprobSEQ s) holds (FDprobSEQ s)(n) = FDprobability((canFS(S))(n),s).*

確率質量関数の等しい有限列の集合

等価な確率分布の族と定義

FDprobability(R,s)=3/5
FDprobability(W,s)=2/5



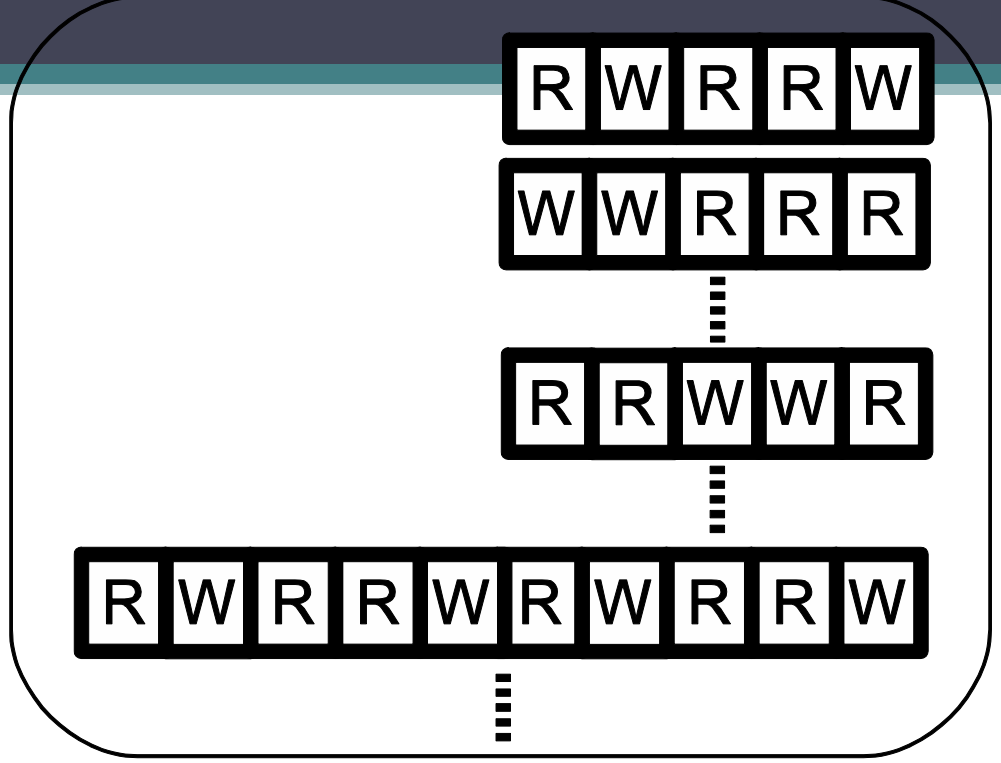
関連MML(Mizar数学ライブラリ)

離散確率分布に関するライブラリ

http://www.mizar.org/version/current/html/dist_1.html
http://www.mizar.org/version/current/html/dist_2.html

確率変数に関するライブラリ

http://www.mizar.org/version/current/html/random_1.html
http://www.mizar.org/version/current/html/random_2.html
http://www.mizar.org/version/current/html/random_3.html



完全一致だと
自明な識別不能
しかないので
使えない

Definition (Definition of equivalence class)

Let S be a non empty finite set and let s be a finite sequence of elements of S. Then, the equivalence class of s yielding a non empty subset of S∗ is defined by:

The equivalence class of s = {t; t ranges over finite sequences of elements of S: s and t are probability equivalent}.

Negligible Functions

ある$\mathbf{N} \to \mathbf{R}$である関数$\mu(\cdot)$について

任意の多項式$p(\cdot)$に対して、

ある自然数Nが存在し、

$N \leq n$なる任意の自然数nについて

$$\mu(n) < \frac{1}{|p(n)|}$$

であるとき$\mu(\cdot)$は無視できるほど小さい関数である

Negligible Functions

```
definition
let f be Function of NAT,REAL;
attr f is negligible
 means
:defneg:
  for c be non empty  positive-yielding XFinSequence of
    REAL
   holds
   ex  N be Element of NAT
       st
    for x be Element of NAT
        st N <=x holds |. f.x .| < 1/((seq_p(c)).x) ;
end;
```

$$\frac{1}{2^x}$$ is negligible

```
theorem
for  f be Function of NAT,REAL st
    for x be Element of NAT holds
            f.x = 1/ (2 to_power x)
holds  f is negligible
```

Polynomially-bounded Functions in Mizar

```
definition
  let p be Real_Sequence;
  attr p is polynomially-bounded means
:: ASYMPT_2:def 1
  ex k be Element of NAT st p in
   Big_Oh(seq_n^(k));
end;
```

Theorem

2^n is non polynomially-bounded

$$\forall x \in \mathbf{N} \text{ s.t. } 1 < x \text{ holds}$$

$$\neg\left(\exists c, N \in \mathbf{N} \text{ s.t. } \forall n \in \mathbf{N} \text{ s.t. } N \leq n \text{ holds } 2^n \leq c \cdot n^x\right)$$

```
theorem :: ASYMPT_2:18
  for x be Element of NAT st 1 < x holds
  not ex N,c be Element of NAT st
  for n be Element of NAT st N <= n holds
  2 to_power n <= c * ( n to_power x);
```

Polynomial is Polynomially-bounded

```
theorem :: ASYMPT_2:54
  for k be Nat,
  c be XFinSequence of REAL
  st len c = k+1 & 0 < c.k
  holds seq_p(c) in Big_Oh( seq_n^(k) );
```

関連MML(Mizar数学ライブラリ)

多項式オーダーに関するライブラリ
http://www.mizar.org/version/current/html/asympt_2.html

無視可能変数に関するライブラリ
http://www.mizar.org/version/current/html/asympt_3.html

あとは確率変数間の距離入れれば完成

- 関数間距離を勝手に定義してしまうと怒られる
- 統計（仮説検定）の手法を採用
- X^2 検定？　　は使えない
- 母集団の確率分布を仮定している検定はダメ
- ノンパラメトリック検定

KS検定

ＫＳ検定（Kolmogorov-Smirnov検定）

- 確率変数（標本）の**適合度**を検定する
- 母集団の確率分布について仮定を必要としない（ノンパラメトリック検定）
- １標本KS検定
  - ある標本Ｘが確率質量関数f(x)に適合するか
- ２標本KS検定
  - ある標本ＸとＹが同じ確率質量関数によるか

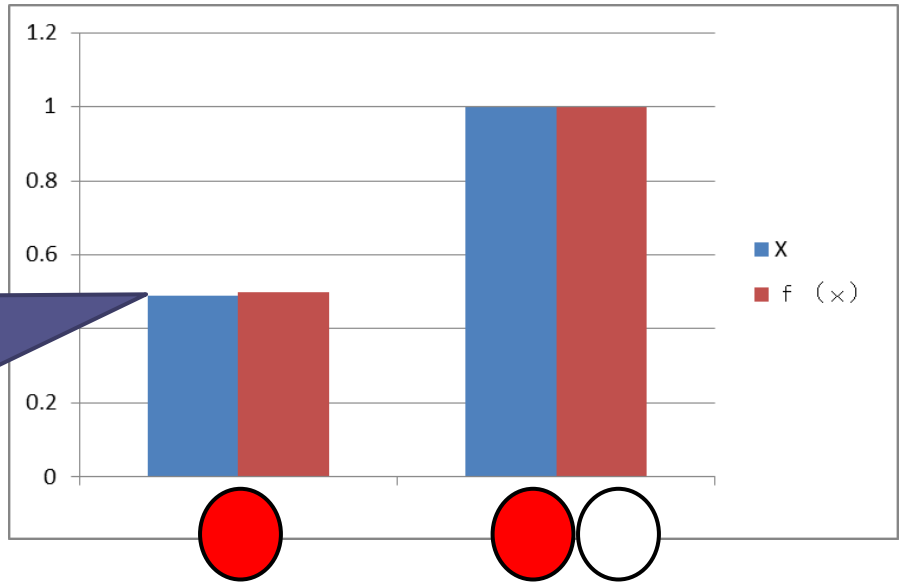
- ＫＳ検定の統計量Ｄを距離として使う

１標本ＫＳ検定

- 累積確率質量関数f(x)と標本X(標本数n)から得られる累積確率質量関数の差の絶対値のMAXが標本量D

- 例（赤玉49個、白玉51個）は一様分布か？

D
=|0.5-0.49|
＝0.01



１標本ＫＳ検定

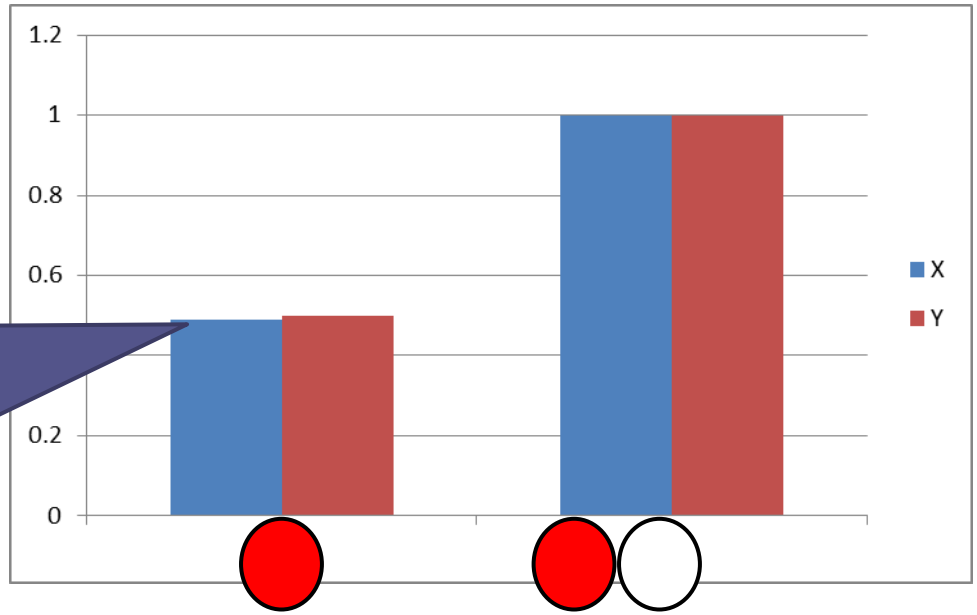
- 有意確率の計算
- 標本数nの場合

- 有意水準５％の場合
  - $D\sqrt{n}$が1.36以上の場合無帰仮説を棄却
  - （分布は一致しない）
- とするらしいが今回は「無視可能」かだけに興味がるのでこの話は後回しにする

２標本ＫＳ検定

- 標本X(標本数n)と標本Y(標本数m)から得られる累積確率質量関数の差の絶対値のMAXが標本量D
- 例（赤玉49個、白玉51個）（赤玉50個、白玉50個）は同じ分布か？

D
=|0.5-0.49|
＝0.01



2標本ＫＳ検定

- 有意確率の計算
- X,Yの標本数がそれぞれnとmの場合

- 有意水準５％の場合
  - $D\sqrt{\frac{nm}{n+m}}$が1.36以上の場合無帰仮説を棄却
  - （分布は一致しない）
- とするらしいが今回は「無視可能」かだけに興味がるのでこの話は後回しにする

１標本ＫＳ検定（Mizarで形式化中）

```
definition
let  Omega,Sigma,X;
 let  S be non empty FinSequence of REAL;
 func KS_p(S,X)-> Real
  means
  :Defo6:
  ex D be  non empty      bounded_above Subset of REAL
    st
D = the set of all |. (Cumulative (X)).x - (Cumulative (S)).x .|
       where   x is Element of REAL
    &
       it = upper_bound  D ;
```

２標本ＫＳ検定（Mizarで形式化中）

```
definition
 let  X,Y be non empty FinSequence of REAL;
 func KS_p(X,Y)-> Real
  means
  :Def07:
  ex D be  non empty      bounded_above Subset of REAL
    st
D = the set of all |. (Cumulative (X)).x - (Cumulative
(Y)).x .|
        where   x is Element of REAL
    &
        it = upper_bound  D ;
```

ハフマン木の形式化

http://www.mizar.org/version/current/html/huffman1.html

以降はおまけ

- SCIS2018発表分のスライド

- ProVerifは以下から入手可能

http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

ProVerifを用いたCTの形式化

2018年1月25日

○荒井研一＊，岡崎裕之†，布田裕一‡

＊長崎大学 大学院工学研究科
†信州大学 大学院理工学系研究科
‡東京工科大学 コンピュータサイエンス学部

はじめに

- 本発表について
  - ProVerifを用いたCTの形式化
    - マークルハッシュ木の形式化
    - 再帰的な呼び出しの形式記述

暗号プロトコル評価技術コンソーシアム(CELLOS)勉強会における成果

CELLOS

Cryptographic protocol Evaluation toward Long-Lived Outstanding Security

安全な暗号プロトコルの普及促進

https://www.cellos-consortium.org/jp/index.php

形式検証ツールを用いた評価技術の普及を目指す！

ProVerifについて

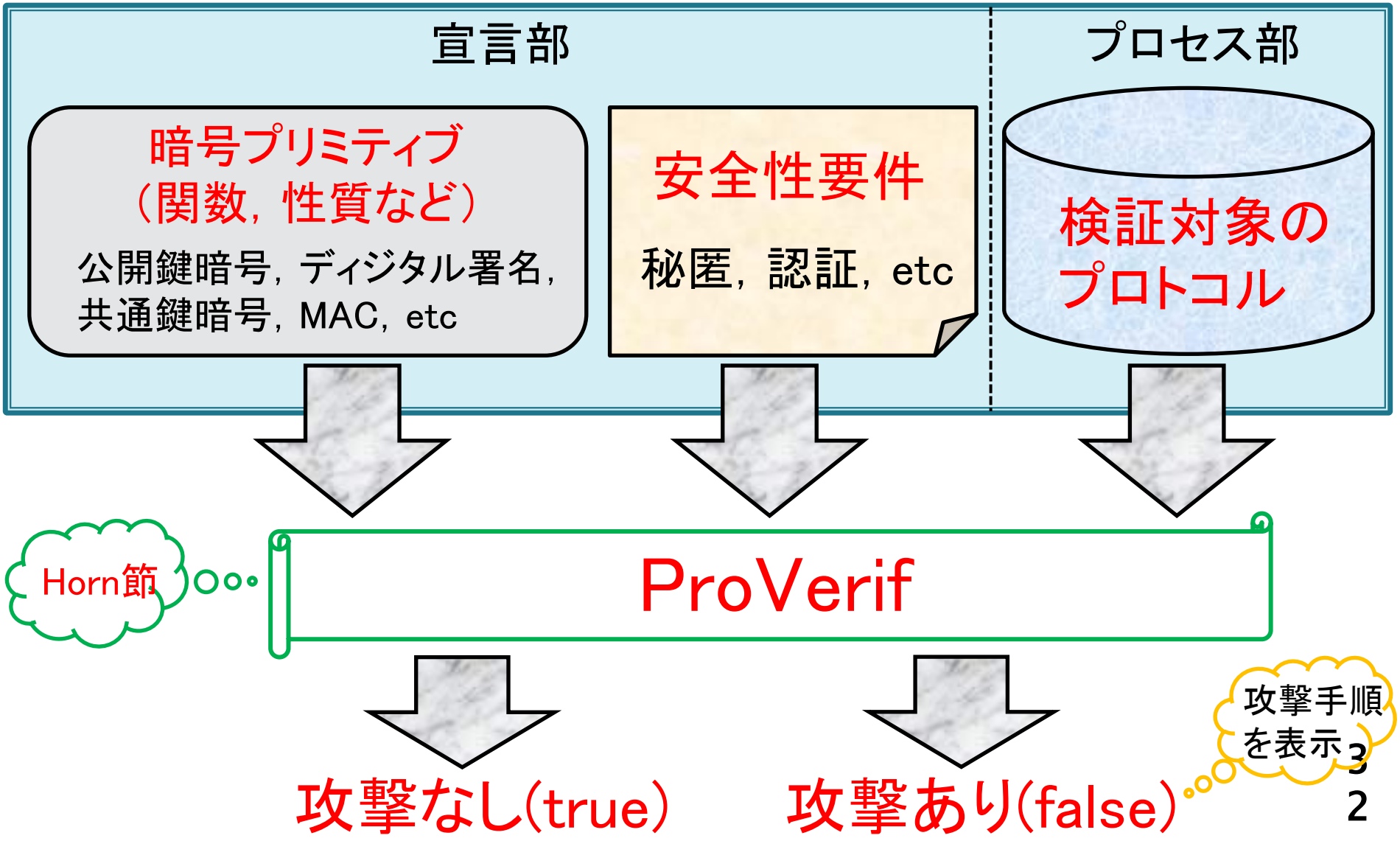
- 形式モデル(Dolev-Yaoモデル)での暗号プロトコルの安全性自動検証ツール
- B.Blanchetらが開発(自身のサイトで公開，Ver.1.98pl1)
- Horn節によるプロトコル表現
- 様々な暗号プリミティブを関数として与え，その性質を等式を用いて定義(理想的)
- 秘匿，認証などの安全性要件を検証可能

2017/12/19 更新

POINT!

✓ 様々な暗号プロトコルの検証が可能
✓ 多くの暗号プロトコルに対してセキュリティ上の欠陥を発見することに成功

ProVerifによる検証



CTについて

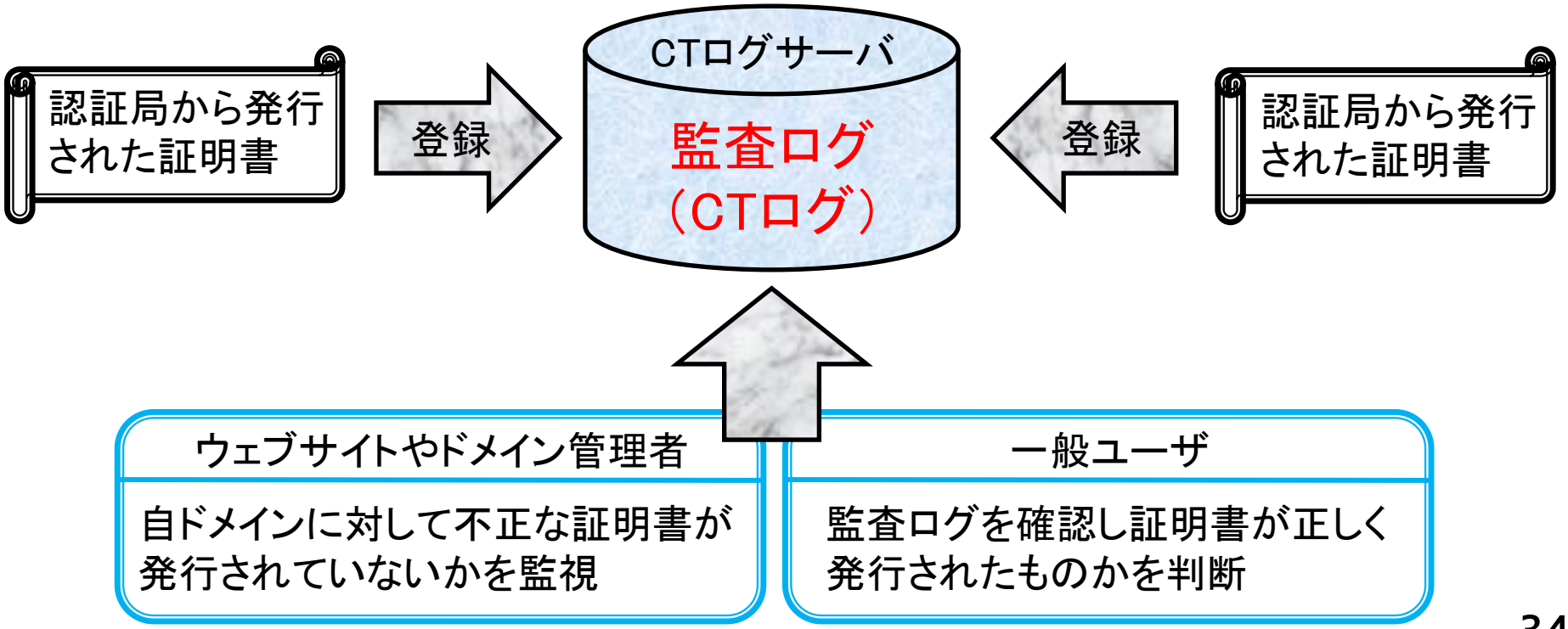
- CTについて
  - CT(Certicate Transparency：証明書の透明性)は，Googleによって提唱された証明書の不正発行を防止する監査・監視の仕組み（i.e. 証明書発行の“透明性”を確保する仕組み）
  - 2013年にRFC6962として策定
  - 認証局から発行された不正な証明書の早期発見を目的としており，不正な証明書の発行自体を防ぐものではない

2011年：英国の認証局 Comodo
2011年：オランダの認証局 DigiNotar
} 攻撃者により不正アクセスを受け不正な証明書が発行された問題

2011年：マレーシアの認証局 DigiCert
2013年：トルコの認証局 TURKTRUST
} 運用ミスにより不正な証明書が発行された問題

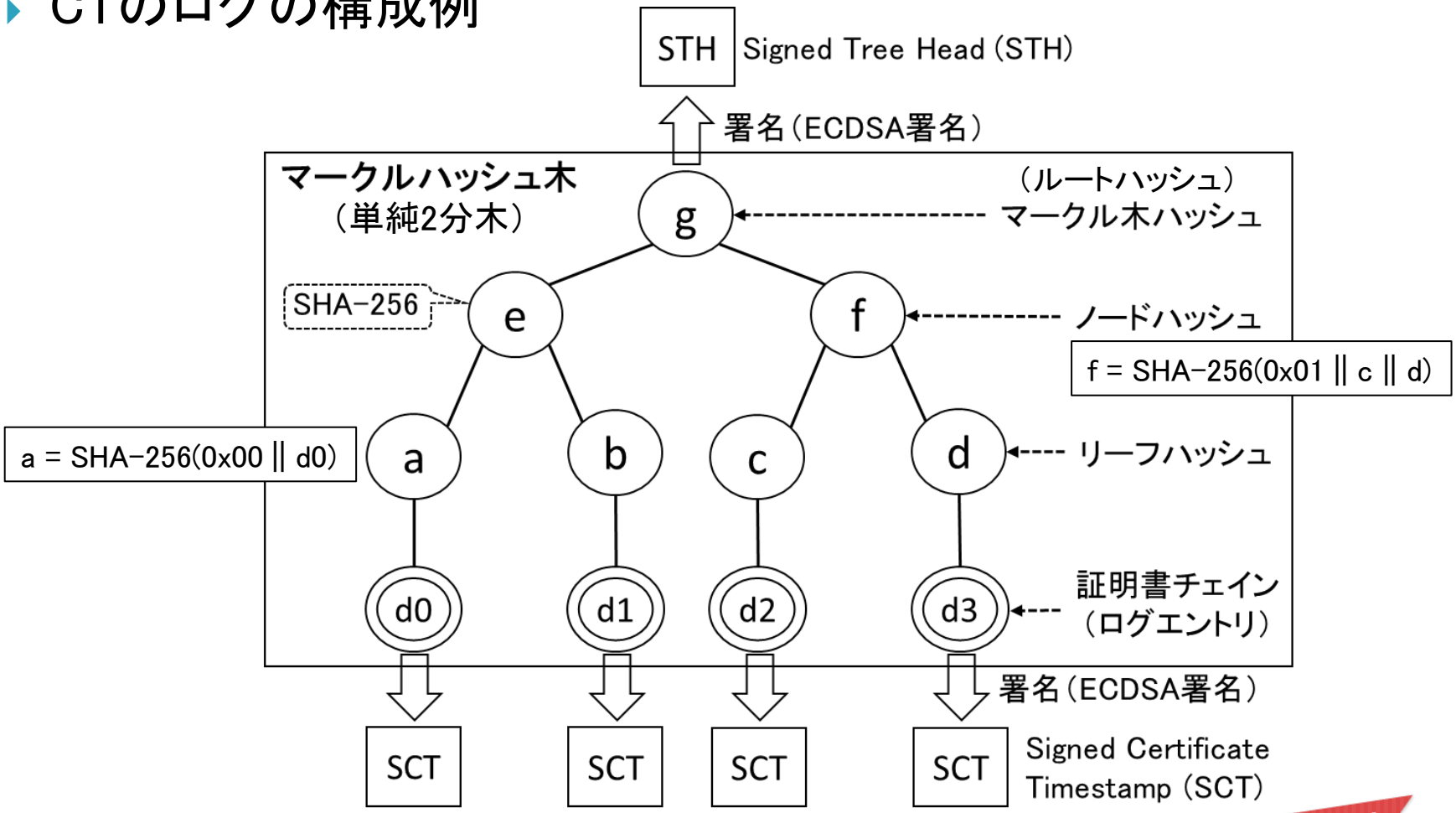
CTの仕組み（概要）

- CTの仕組み（概要）
  - すべての認証局から発行される証明書に対して認証局が正規に発行したものかどうかを監査ログとして(第三者が) 公開



CTログの構成

- CTのログの構成例

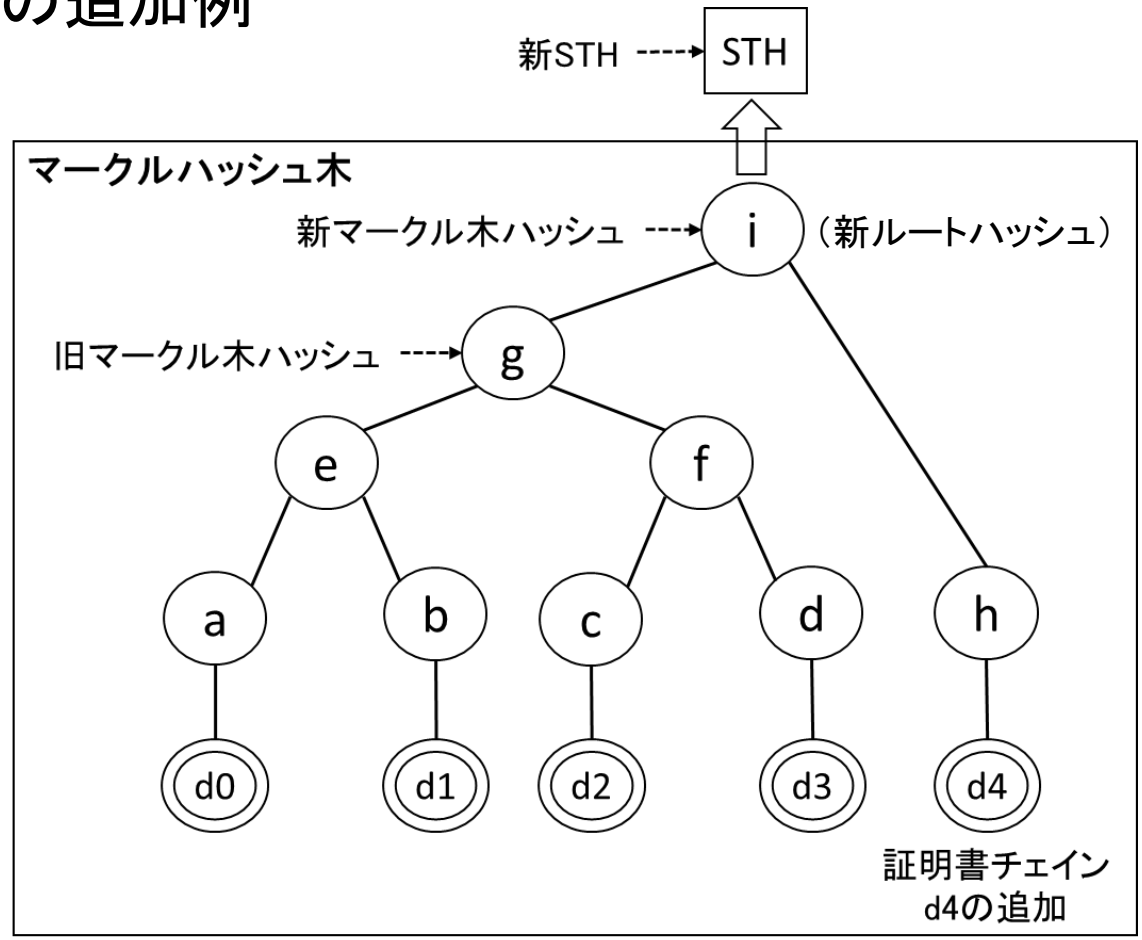


POINT!

効率的な監査のために，マークルハッシュ木が用いられている

CTログの追加

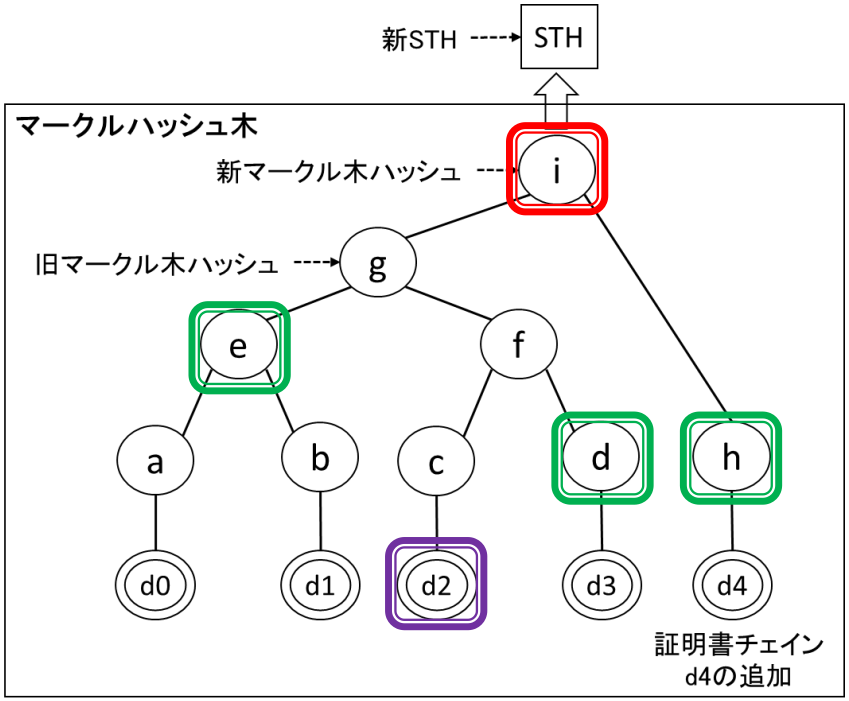
- CTのログの追加例



POINT!

追加(append-only)しかできず，途中改竄ができないといった
特徴があり，これにより，証明書とログの監査を容易にしている

マークル検査証明



POINT!

マークル検査証明を用いることにより，
特定の証明書チェインがCTログに
含まれていることを検証することが可能

（例）d2 がCTログに含まれていることを検証したい場合・・・

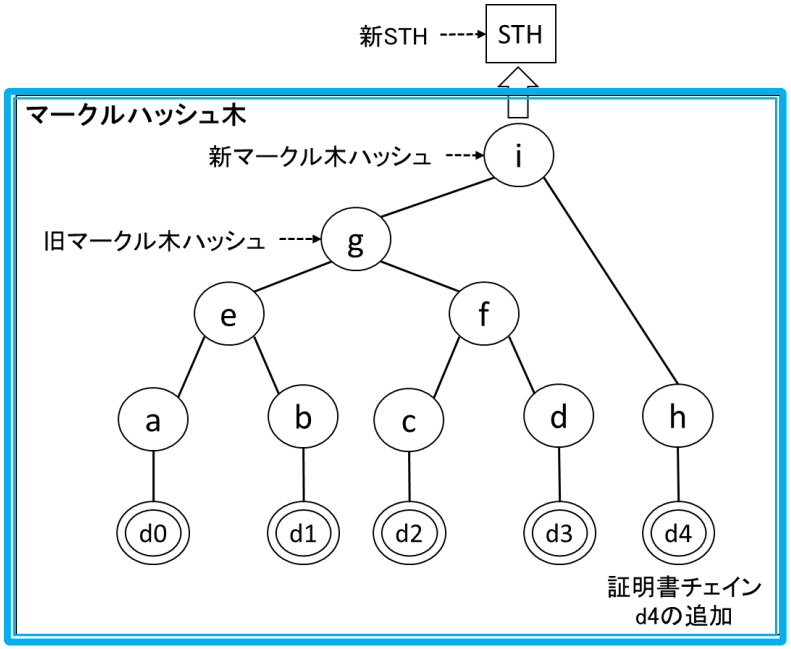
（ハッシュ値のリスト）

d2 ＋ 監視パス [d,e,h] ⇒ マークル木ハッシュ i が計算可能

“計算で求めた i の値”と“別途与えられるマークル木ハッシュの値”が一致するかを確認することにより，d2 がCTログに含まれているかどうかを検証可能

ProVerifでのCTの形式化

- ProVerifを用いてCTを形式化するためには・・・



- マークルハッシュ木の形式化（木のモデル化）
- マークル検査証明の形式化

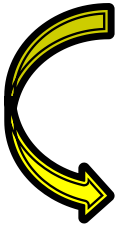
が必要

通信路とテーブルの併用

POINT!

木のモデル化
～通信路とテーブルの併用の理由～

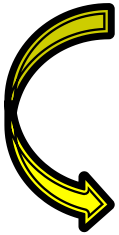
- 木のモデル化の条件
  - 木は，複数のノードとそれらの（親子）関係性から構成
    - テーブルを使うと，関係性は表現可能だが，ノードは表現できない
    - 通信路だけだと，ノードしか表現できない



POINT!

通信路とテーブルを併用することで，木のモデル化を実現

- Proverifにおける通信路の定義
  - 初期の宣言時点では，空集合として扱われ，要素をOutするごとに集合に追加され，上書きや削除ができない

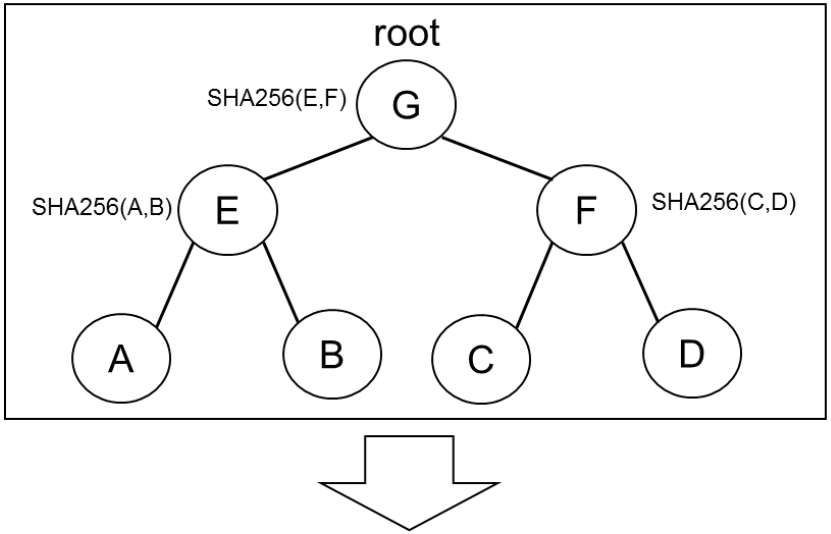


POINT!

ノードを追加するだけで上書きしない，マークルハッシュ木に最適

マークルハッシュ木の形式化（木のモデル化）

マークルハッシュ木（簡略図）



集合　{A,B,C,D,E,F,G}

表引き

A	B	C	D	E	F	G
E	E	F	F	G	G	root

POINT!

通信路を木（ノードの集合）とみなす（項が追加できる集合）

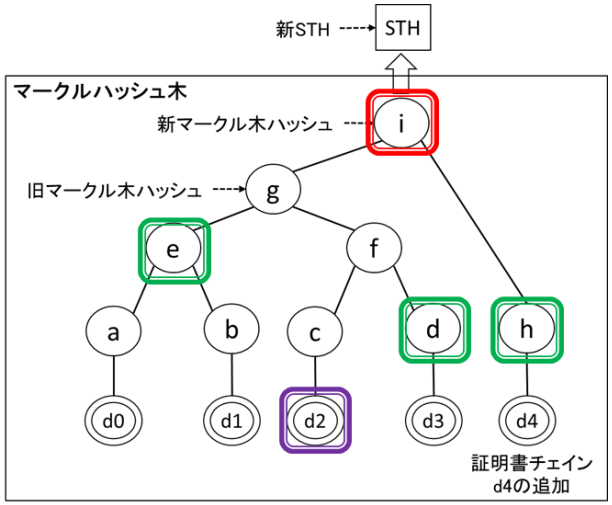
- 木専用の通信路を用意
  - ✓ 通信路を用いることでノードを公開

POINT!

ノード同士の親子関係をテーブルで表現

- テーブルを親子の対を定める集合とみなす
  - ✓ 攻撃者のテーブルへのアクセスは許可されない

マークル検査証明の形式化



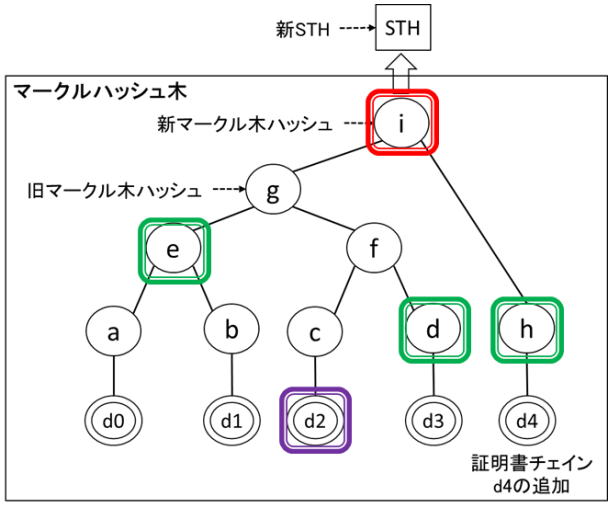
マークル検査証明を用いることにより，
特定の証明書チェインがCTログに
含まれていることを検証することが可能

マークル検査証明を形式化するためには再帰のような繰り返し呼び出しの形式化が必要

ProVerifでは再帰のような繰り返し呼び出しは
基本機能としてはサポートされていない

再帰呼び出しを形式記述する必要がある

マークル検査証明の形式化



マークル検査証明を用いることにより，
特定の証明書チェインがCTログに
含まれていることを検証することが可能

- 再帰呼び出しを実現するにあたり・・・

POINT!

ログ検証用のプライベート通信路を利用

- 再帰呼び出しは，プロセス間通信として表現するしかない
- 検証（検査）対象が攻撃者に見えない（分からない）ようにしなければならない

マークルハッシュ木の形式化（宣言部）

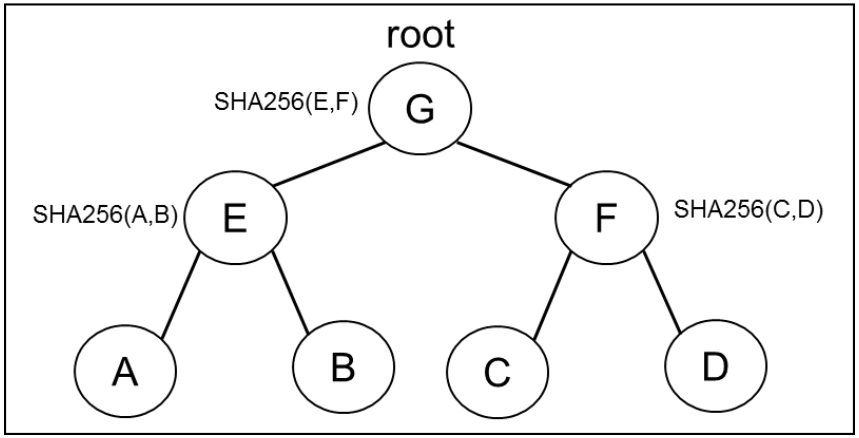
```
type pkey. (* 検証（公開）鍵の型*)
type skey. (* 署名（秘密）鍵の型*)

free c: channel. (* 一般通信路 *)
free s: channel [private]. (* ログ検証用通信路*)
free t: channel. (* 木専用通信路 *)

free A: bitstring. (* リーフハッシュA *)
free B: bitstring. (* リーフハッシュB *)
free C: bitstring. (* リーフハッシュC *)
free D: bitstring. (* リーフハッシュD *)
free H: bitstring. (* 追加用リーフハッシュH *)

free sigk: skey [private]. (* 署名鍵 *)

const root: bitstring.
```

```
(* Signature (理想的なディジタル署名) *)
fun pk(skey): pkey.
fun sign(bitstring,skey): bitstring [private].
reduc forall x:bitstring, y:skey;
  verif(sign(x,y),pk(y),x) = true.

(* Hash (理想的なハッシュアルゴリズム) *)
fun SHA256(bitstring,bitstring): bitstring.
equation forall x:bitstring, y:bitstring;
  SHA256(x,y) = SHA256(y,x). (* 可換性 *)

(* テーブル *)
table SUBROOT(bitstring,bitstring).

(* クエリ *)
event PROOF.
query event(PROOF).
```

ProVerifでの形式化（コード）

マークルハッシュ木の形式化（メインプロセス）

```
(* Main Process *)
process
  let PK = pk(sigk)
  in out(c,PK);

  (* A,B -> E *)
  let E = SHA256(A,B) in
  out(t,A);
  out(t,B);
  out(t,E);
  insert SUBROOT(A,E);
  insert SUBROOT(B,E);

  (* C,D -> F *)
  let F = SHA256(C,D) in
  out(t,C);
  out(t,D);
  out(t,F);
  insert SUBROOT(C,F);
  insert SUBROOT(D,F);

  (* E,F -> G *)
  let G = SHA256(E,F) in
  out(t,G);
  insert SUBROOT(E,G);
  insert SUBROOT(F,G);

  insert SUBROOT(G,root);
  let abcdSIG = sign(G,sigk) in
  out(c,abcdSIG);

(* マークル検査証明の実行 *)
  out(s,A) | !MAPROOF
```



表引き

A	B	C	D	E	F	G
E	E	F	F	G	G	root

マークル検査証明の形式化（プロセス部）

```
(* マークル検査証明 *)
let MAPROOF =
    in (s,(z:bitstring));
    in (t,(p:bitstring));
    in (t,(b:bitstring));
    in (t,(sz:bitstring));
    in (c,(pSIG:bitstring));
    get SUBROOT(=sz,Rz) in
    (
      get SUBROOT(=b,Rb) in
      get SUBROOT(=p,Rp) in
      if (sz = z && SHA256(sz,b) = p &&
          Rz = p && Rb = p) then out(s,p)
    )
    else
    (
      if (verif(pSIG,pk(sigk),z) = true) then
      event PROOF
    ).
```

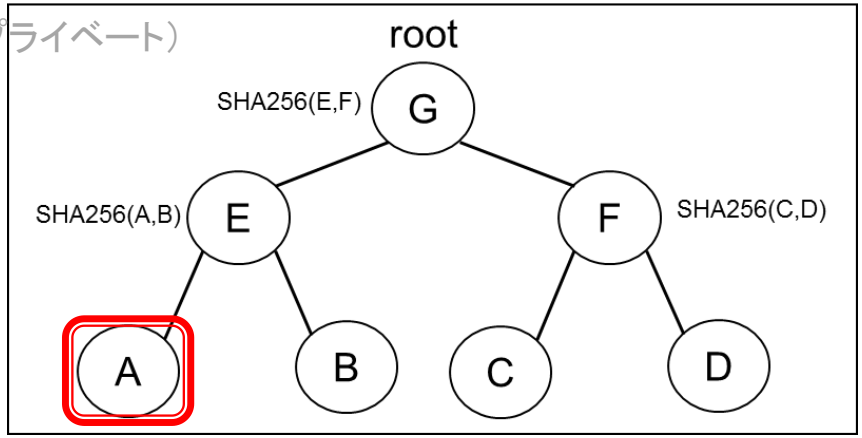
s: ログ検証用通信路（プライベート）

t: 木専用通信路

イベントPROOFの実行＝マークル検査証明の成功

表引き

A	B	C	D	E	F	G
E	E	F	F	G	G	root



s	z	A	
t	p	E (=SHA256(A,B))	G (=Rp)
t	b	B	E (=Rb)
t	sz	A	E (=Rz)
c	pSIG	適当な値（署名値）	

通信路

テーブル参照値

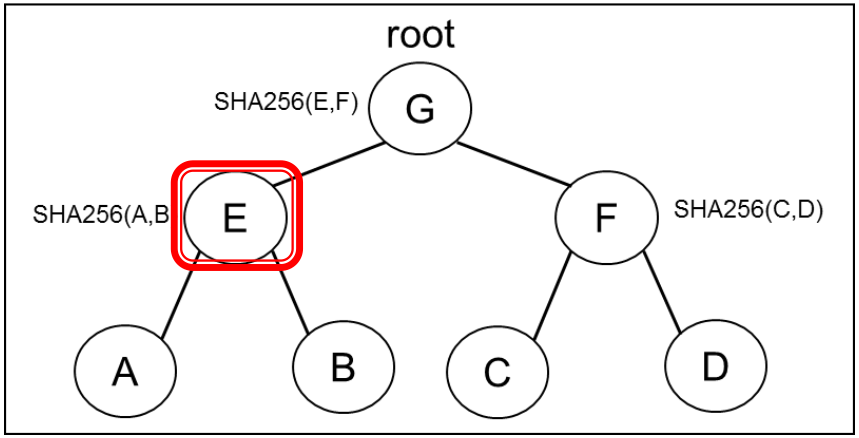
マークル検査証明の形式化（プロセス部）

```
(* マークル検査証明 *)
let MAPROOF =
    in (s,(z:bitstring));
    in (t,(p:bitstring));
    in (t,(b:bitstring));
    in (t,(sz:bitstring));
    in (c,(pSIG:bitstring));
    get SUBROOT(=sz,Rz) in
    (
      get SUBROOT(=b,Rb) in
      get SUBROOT(=p,Rp) in
      if (sz = z && SHA256(sz,b) = p &&
          Rz = p && Rb = p) then out(s,p)
    )
    else
    (
      if (verif(pSIG,pk(sigk),z) = true) then
      event PROOF
    ).
```

イベントPROOFの実行＝マークル検査証明の成功

表引き

A	B	C	D	E	F	G
E	E	F	F	G	G	root



s	z	E	
t	p	G (=SHA256(E,F))	root (=Rp)
t	b	F (=SHA256(C,D))	G (=Rb)
t	sz	E	G (=Rz)
c	pSIG	適当な値（署名値）	

通信路

テーブル参照値

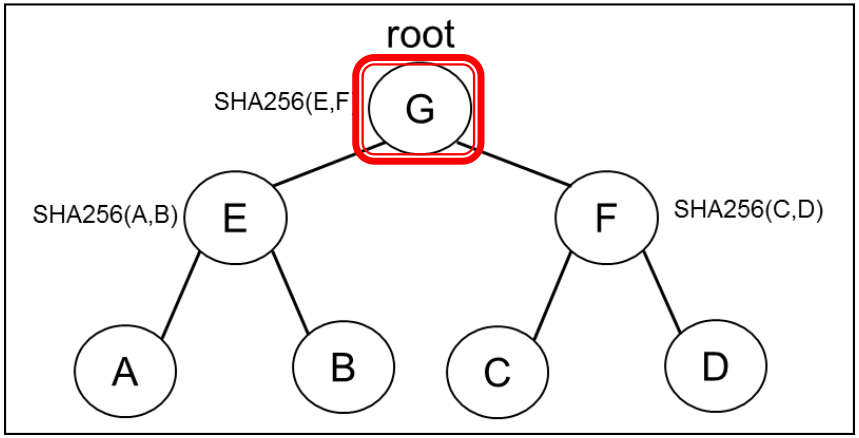
マークル検査証明の形式化（プロセス部）

```
(* マークル検査証明 *)
let MAPROOF =
    in (s,(z:bitstring));
    in (t,(p:bitstring));
    in (t,(b:bitstring));
    in (t,(sz:bitstring));
    in (c,(pSIG:bitstring));
    get SUBROOT(=sz,Rz) in
    (
      get SUBROOT(=b,Rb) in
      get SUBROOT(=p,Rp) in
      if (sz = z && SHA256(sz,b) = p &&
          Rz = p && Rb = p) then out(s,p)
    )
    else
    (
      if (verif(pSIG,pk(sigk),z) = true) then
      event PROOF
    ).
```

イベントPROOFの実行＝マークル検査証明の成功

表引き

A	B	C	D	E	F	G
E	E	F	F	G	G	root



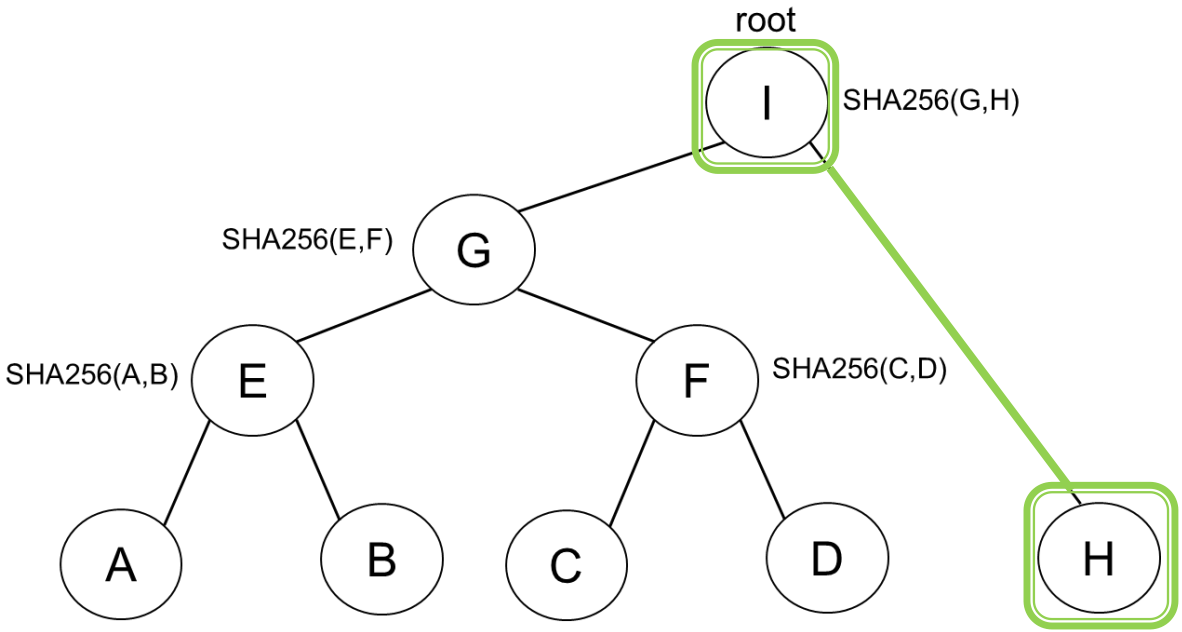
s	z	G	
t	p	適当な値	—
t	b	適当な値	—
t	sz	適当な値	—
c	pSIG	Sign(G,sigk)	

通信路

テーブル参照値

マークルハッシュ木の形式化（メインプロセス）～ログの追加～

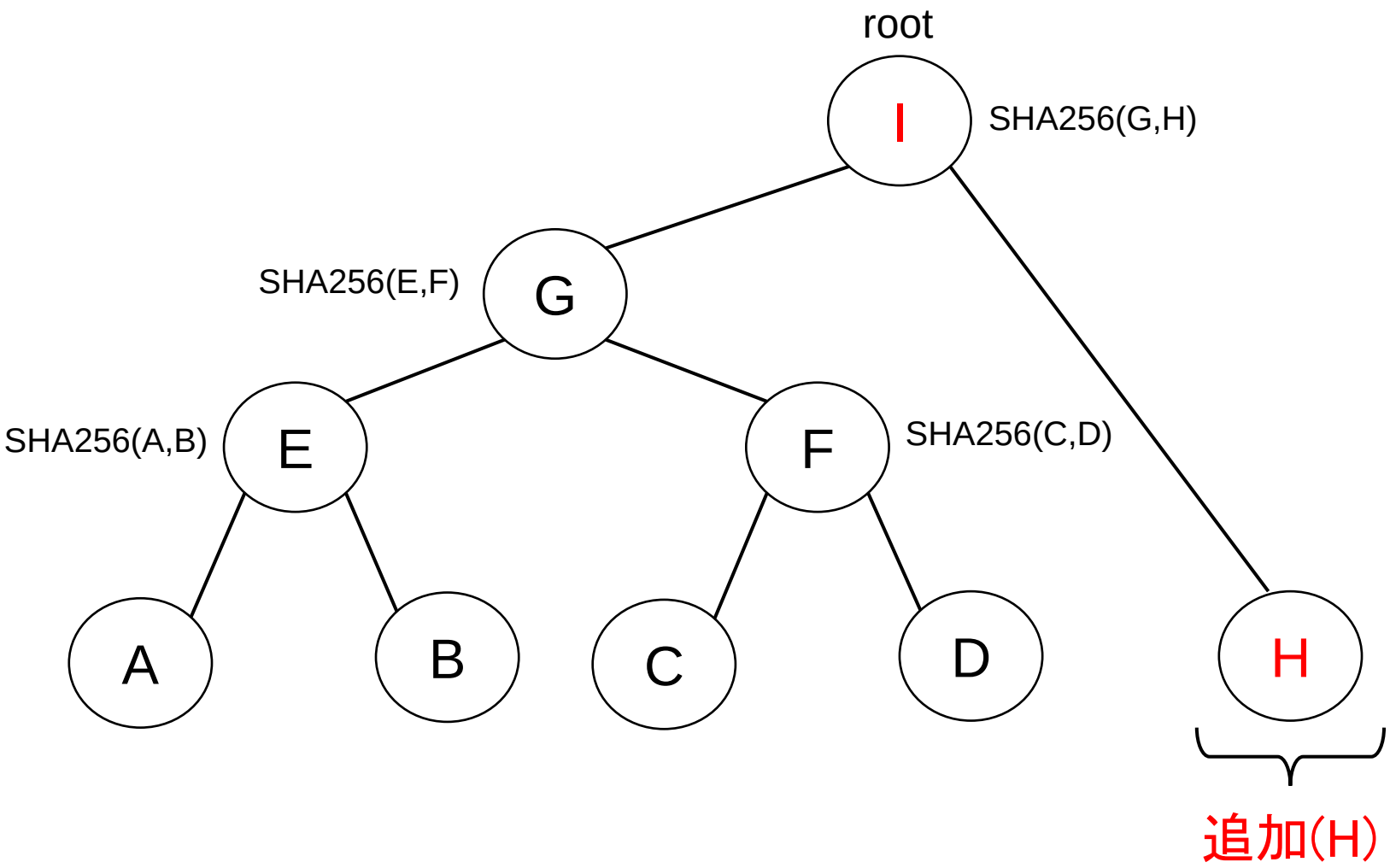
```
(* E,F -> G *)
  let G = SHA256(E,F) in
  out(t,G);
  insert SUBROOT(E,G);
  insert SUBROOT(F,G);

(* ログの追加 *)
  (* G,H -> I *)
  let I = SHA256(G,H) in
  out(t,H);
  out(t,I);
  insert SUBROOT(H,I);
  insert SUBROOT(G,I);

  insert SUBROOT(I,root);
  let ghSIG = sign(I,sigk) in
  out (c,ghSIG);

(* マークル検査証明の実行 *)
  out(s,A) | !MAPROOF
```



まとめと今後の課題

- まとめ
  - ProVerifを用いたCTの形式化
    - マークルハッシュ木の形式化（木のモデル化）
    - マークル検査証明の形式化
      - 通信路とテーブルを併用した木のモデル化
      - プライベート通信路を用いた再帰呼び出しの実現
- 今後の課題
  - CTにおける満たすべき安全性の検討及びその安全性検証
  - 本形式化はブロックチェーンの形式検証への応用も想定しているため，ブロックチェーンの形式検証の検討

御清聴ありがとうございました


root
I
SHA256(G,H)
SHA256(E,F)
G
SHA256(A,B)
E
F
SHA256(C,D)
A
B
C
D
H
追加(H)