TOSHIBA
Leading Innovation

CryptoVerifを用いた RFID向け相互認証プロトコルの 安全性証明の検討

○ 花谷嘉一[1)2)], 大久保美也子[3)], 松尾真一郎[3)], 太田和夫[2)], 崎山一男[2)]

1) (株) 東芝 研究開発センター
2) 電気通信大学大学院 総合情報学専攻
3) 情報通信研究機構

2010/9/6

目次

- 背景と今回の結果
- RFIDシステム
- 安全性要件
- CryptoVerif による検証
- 人手による検証
- まとめ

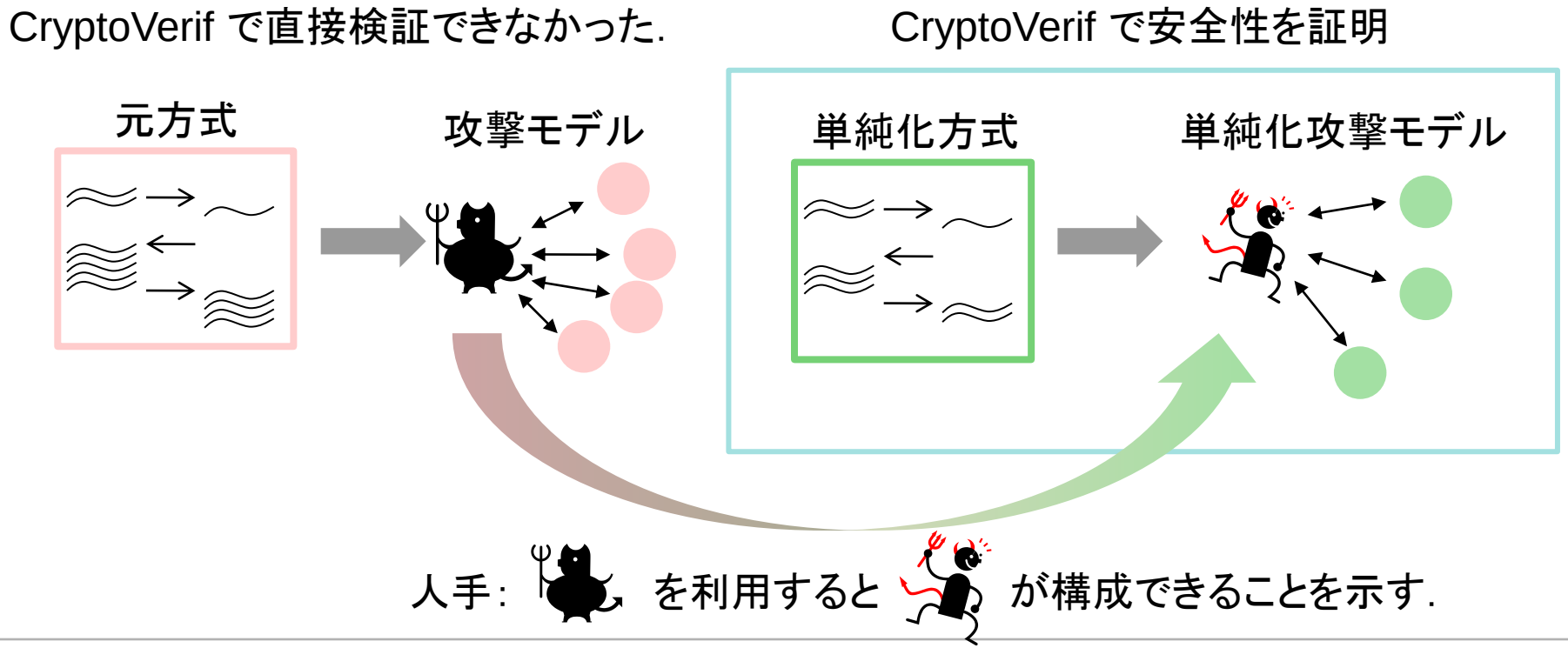
背景と研究の動機

- **電子政府推奨暗号リストの改定**
  - 10年程度安全性が担保される暗号技術をリスト形式でまとめたもの．定期的に見直しを行う．
  - エンティティ認証プロトコルの公募要件：
    - 通信相手が意図した正しい通信相手であることを確認するプロトコル．
    - 安全性の評価は，形式的な手法を用いて行う．
- **RFID向け認証プロトコル**
  - 一般に構成がシンプルであるため，形式検証と相性が良さそう．
  - 公開鍵暗号やデジタル署名と比べると，安全性要件が固まっていない．
- **CryptoVerif**
  - 暗号プロトコルの安全性を半自動的に検証するソフトウェア．

CryptoVerif を用いて，実用的な認証プロトコルの検証が可能か検討する．

今回の結果

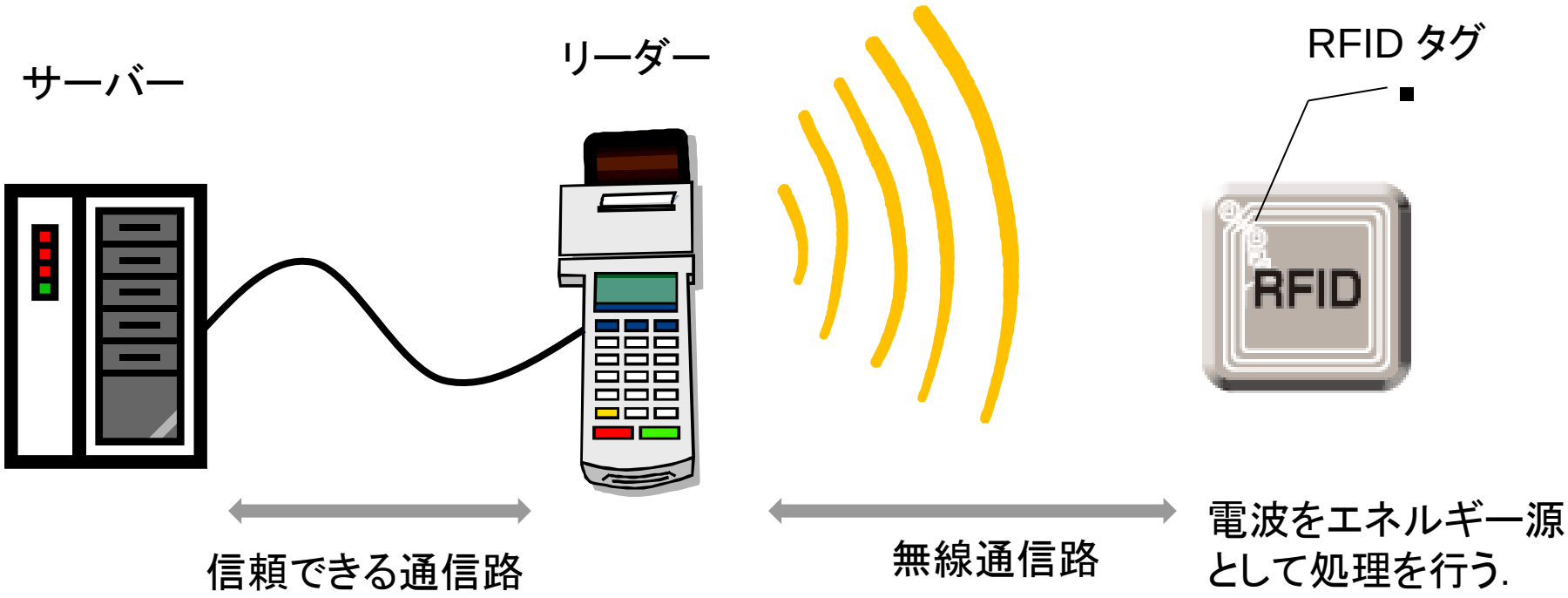
- 大久保さんらのRFID相互認証プロトコルを検証の対象とした.
- CryptoVerif の文法規則では，検証対象とした方式を定式化できなかった.
- 単純化した方式を考えて，単純化した安全性を証明.
- 単純化した安全性と元の安全性の等価性を示した.



対象とするパッシブ RFID システム

Radio Frequency IDentification

電波による個体認識. ➡ バーコード, 在庫管理, 位置情報の把握 などへの適用



タグは安価に製造出来ないとダメ. ➡ タグの構成に使えるゲート数は少ないため, 使える技術が限られる.

RFIDシステムに求められる安全性の例

- 成りすまし耐性
- プライバシー保護
- 追跡不可能性

- Forward-Security
- DOS攻撃耐性
- 同期ズレ攻撃耐性

などなど

今回の検証対象のプロトコルの特徴

- メインの部品はハッシュ関数のみ.
- ランダムオラクルモデルで安全性証明が可能.

- 鍵更新機能を備えている.
  - タグを物理的に解析して，秘密鍵を盗られたとしても，解析以前の認証の正当性は保たれる.
  - DoS攻撃に対しても安全.
- 秘密鍵を知らない人は，タグから有用な情報を得られない.
  - タグのID や 認証の成功/失敗 など.
- 同期ズレ対策も考慮して設計している.

検証対象プロトコル（相互認証部）

サーバー		タグ
$sk_{ID,i}, sk_{ID,i-1}, ID, i$		$sk_{ID,i'}, ID, i'$
$X \xleftarrow{R} \{0,1\}^t$	$\xrightarrow{X}$	$\alpha' \xleftarrow{R} \{0,1\}^t$ $\beta' \leftarrow H_0(sk_{ID,i'}, ID, i', X, \alpha')$
	$\xleftarrow{\alpha', \beta'}$	
if $\beta' = H_0(sk_{ID,i}, ID, i, X, \alpha')$ accept $Z \leftarrow H_1(sk_{ID,i}, ID, i, X, \alpha')$	同期している (i = i')	
if $\beta' = H_0(sk_{ID,i-1}, ID, i-1, X, \alpha')$ accept $Z \leftarrow H_1(sk_{ID,i-1}, ID, i-1, X, \alpha')$	同期がずれている (i-1 = i')	
else reject $Z \leftarrow H_1(rnd)$	$\xrightarrow{Z}$	if $Z = H_1(sk_{ID,i'}, ID, i', X, \alpha')$ accept else reject
認証失敗		

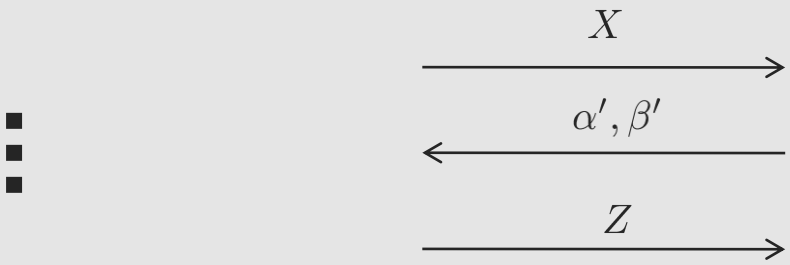
検証対象プロトコル（鍵更新部）

サーバー

$sk_{ID,i}, sk_{ID,i-1}, ID, i$

タグ

$sk_{ID,i'}, ID, i'$



if $\beta' = H_0(sk_{ID,i}, ID, i, X, \alpha')$

$sk_{ID,i-1} \leftarrow sk_{ID,i}$

$sk_{ID,i} \leftarrow H_2(sk_{ID,i}, ID, i)$

$i \leftarrow i + 1$

else keep status

if $Z = H_1(sk_{ID,i'}, ID, i', X, \alpha')$

$sk_{ID,i'} \leftarrow H_2(sk_{ID,i'}, ID, i')$

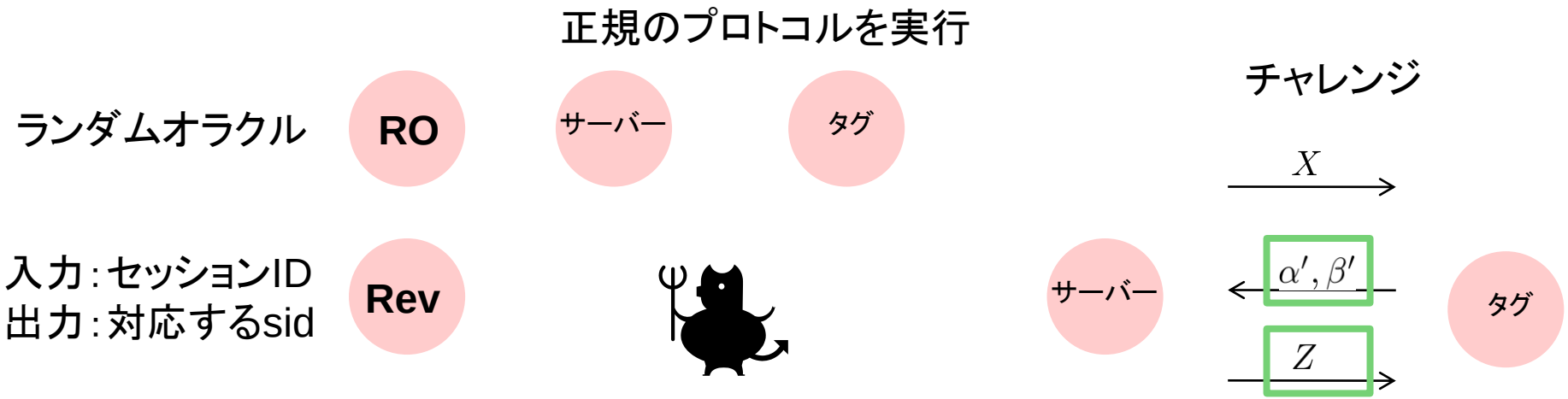
$i' \leftarrow i' + 1$

else keep status

同期していて，認証に合格した場合にのみ
秘密鍵を更新する.

安全性要件

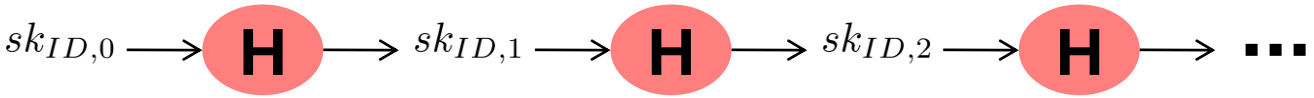
- **Forward-secure Client Indistinguishability (FS-C-IND)**
  - 秘密鍵を知らない人は，タグが本物か識別できない．
- **Forward-secure Client Unforgeability (FS-C-UF)**
  - 秘密鍵を知らない人は，タグに成りすませない．
- **Forward-secure Server Unforgeablity (FS-S-UF)**
  - 秘密鍵を知らない人は，サーバーに成りすませない．



CryptoVerif のための定式化の問題点

- **ハッシュ連鎖を利用した鍵更新**

$$sk_{ID,i} \leftarrow H_2(sk_{ID,i}, ID, i)$$



CryptoVerif： セッション i で定義された sk は，sk[i] に記録される.

sk[i] を 別のセッションで使う場合は，

find i such that defined(sk[i]) & X = sk[i] & ・・・

条件を満たすインデックスを見つけてきて，
そこに記録されている値を使って何かする.

のように，間接的にしか指定できない.

直接 i を指定して，sk[i] を使うような記述は禁止！

前のセッションで作成された秘密鍵を 次のセッションで使うことができない・・・

単純化した方式

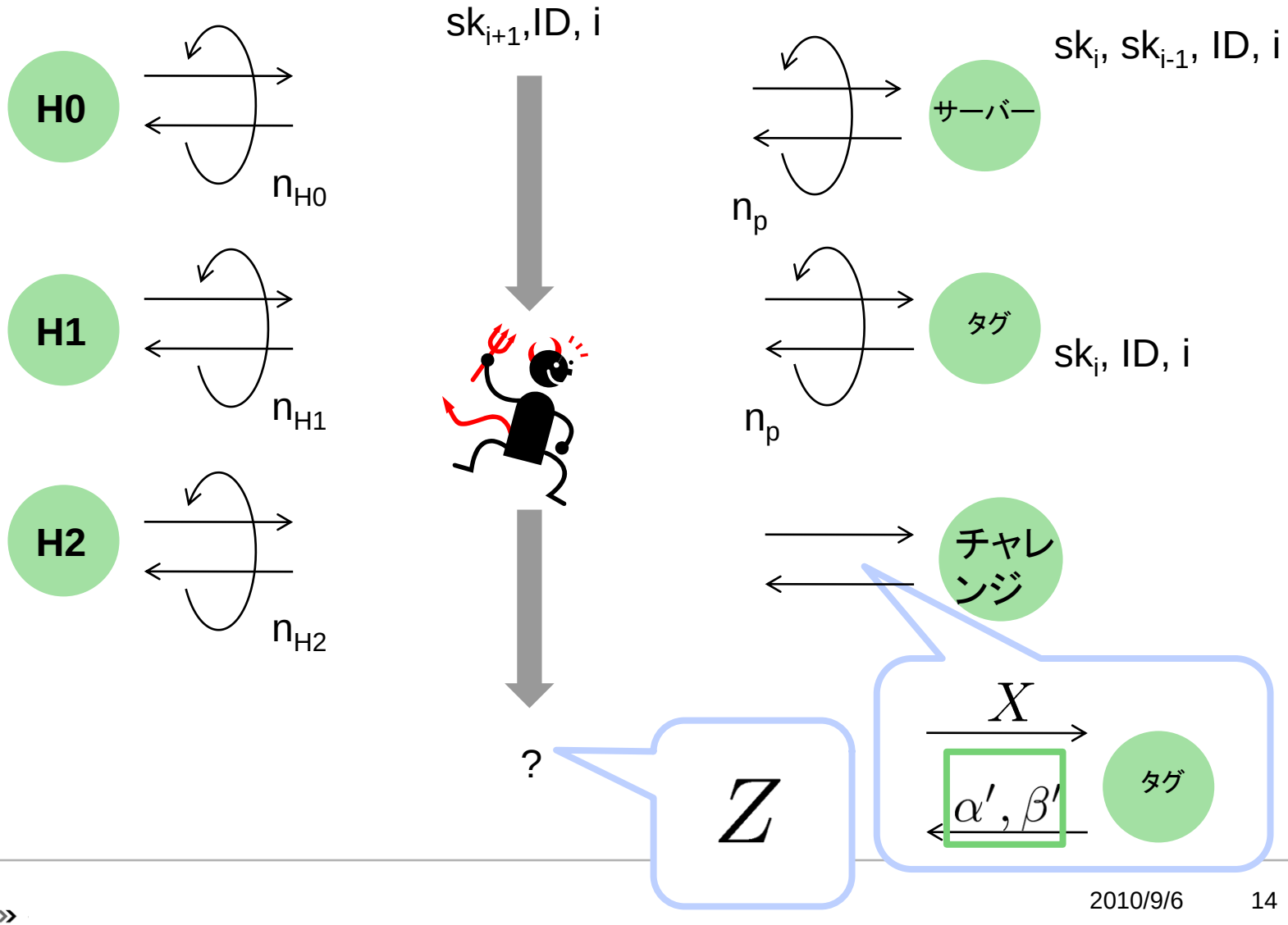
鍵更新機能を省略した方式

サーバー		タグ
$sk_{ID,i}, sk_{ID,i-1}, ID, i$		$sk_{ID,i'}, ID, i'$
$X \xleftarrow{R} \{0,1\}^t$	X →	
		$\alpha' \xleftarrow{R} \{0,1\}^t$
		$\beta' \leftarrow H_0(sk_{ID,i'}, ID, i', X, \alpha')$
	← α', β'	
if $\beta' = H_0(sk_{ID,i}, ID, i, X, \alpha')$ accept $Z \leftarrow H_1(sk_{ID,i}, ID, i, X, \alpha')$	同期している (i = i')	
if $\beta' = H_0(sk_{ID,i-1}, ID, i-1, X, \alpha')$ accept $Z \leftarrow H_1(sk_{ID,i-1}, ID, i-1, X, \alpha')$	同期がずれている (i-1 = i')	
else reject $Z \leftarrow H_1(rnd)$	Z →	if $Z = H_1(sk_{ID,i'}, ID, i', X, \alpha')$ accept else reject
認証失敗		

単純化した方式の安全性要件

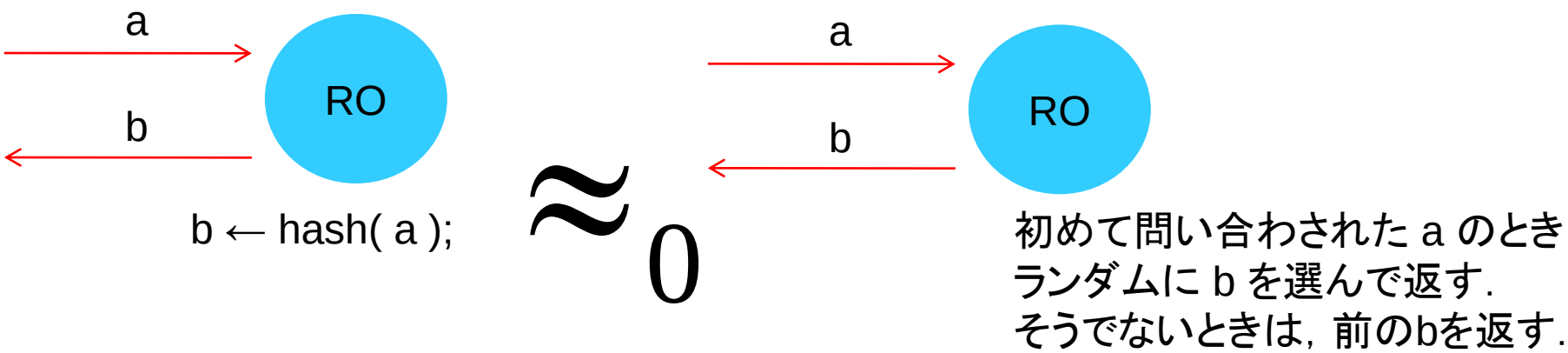
- ~~Forward-secure~~ **Client Indistinguishability (C-IND)**
  - 秘密鍵を知らない人は，タグが本物か識別できない．
- ~~Forward-secure~~ **Client Unforgeability (C-UF)**
  - 秘密鍵を知らない人は，クライアントに成りすませない．
- ~~Forward-secure~~ **Server Unforgeablity (S-UF)**
  - 秘密鍵を知らない人は，サーバーに成りすませない．

CryptoVerif による検証

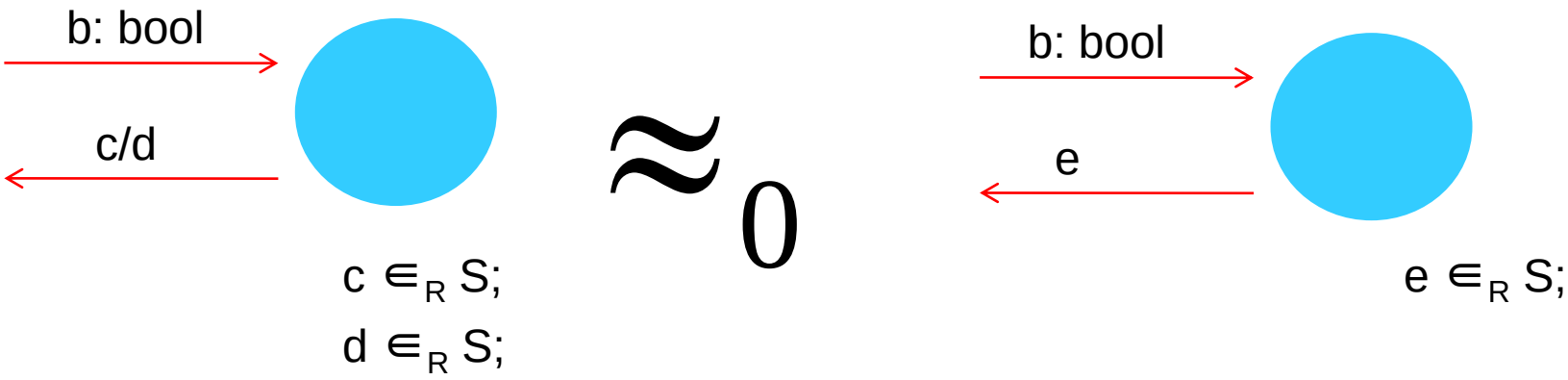


検証に用いた書き換え規則

規則1 : C-IND, C-UF, S-UF の検証で使う.



規則2 : C-IND の検証で使う.



実験結果

	C-IND	C-UF	S-UF
変換回数	34回	14回	13回
要した時間	約 45 秒	約 40 秒	約 15 秒
安全性	○	○	○
コマンド	auto simplify remove_assign all auto	auto simplify auto	auto

実験環境：CPU: Core2Duo 1.2 GHz
RAM 2.85GB
CryptoVerif 1.10pl

CryptoVerif による検証結果

C-IND $$\frac{(2+n_p)n_{H0}+3n_{H1}+7n_{H2}+6}{|SK|}+\frac{3n_p}{|h0|}+\frac{(1+n_p)n_{H0}+5n_p+4n_p^2}{|R|}$$

C-UF $$\frac{(3+n_p)n_{H0}+3n_{H1}+7n_{H2}+6}{|SK|}+\frac{3n_p+2}{|h0|}+\frac{n_p n_{H0}+3n_p+4n_p^2}{|R|}$$

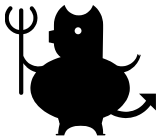
S-UF $$\frac{(2+2n_p)n_{H0}+n_{H1}+7n_{H2}+6}{|SK|}+\frac{n_p}{|h0|}+\frac{1}{|h1|}+\frac{(1+n_p)n_{H0}+4n_p+2n_p^2}{|R|}$$

$\log_2 \lvert SK \rvert$	秘密鍵のビット長
$\log_2 \lvert h0 \rvert$	H0の出力のビット長
$\log_2 \lvert h1 \rvert$	H1の出力のビット長
$\log_2 \lvert R \rvert$	X, α のビット長

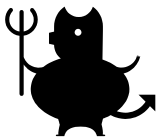
元の方式の単純化した方式の違い

攻撃対象とする 秘密鍵を指定される.

sk 1　　sk 2　　sk 3　　sk 4　　-----　　sk n_p

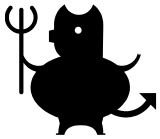


攻撃対象とする 秘密鍵を任意に選択できる.



を利用した

の構成を示すためには・・・



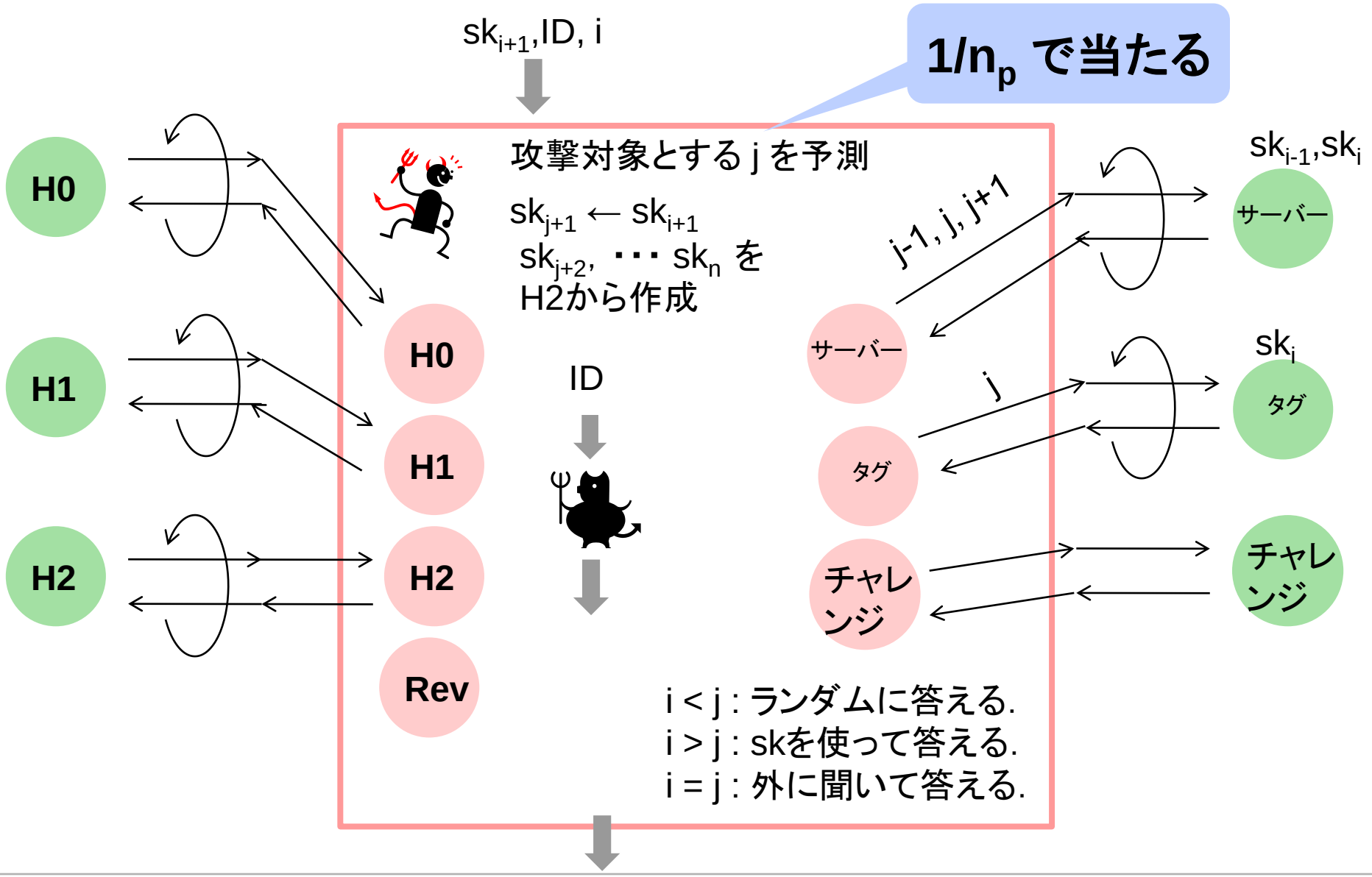
が攻撃対象とする秘密鍵を予想して,

が指定された秘密鍵を埋め込んで, 解かせればいい.

元の方式の安全性証明の概要


を利用して

を構成する.



(CryptoVerif+人手) と 人手のみ の結果

FS-C-IND

$$n_p\left(\frac{(2+n_p)n_{H0}+3n_{H1}+7n_{H2}+6}{|SK|}+\frac{3n_p}{|h0|}+\frac{(1+n_p)n_{H0}+5n_p+4n_p^2}{|R|}\right)$$

$$\frac{n_{H0}+n_{H1}+n_{H2}}{2(|SK|-n_{H0}-n_{H1}-n_{H2})}$$

FS-C-UF

$$n_p\left(\frac{(3+n_p)n_{H0}+3n_{H1}+7n_{H2}+6}{|SK|}+\frac{3n_p+2}{|h0|}+\frac{n_pn_{H0}+3n_p+4n_p^2}{|R|}\right)$$

$$\frac{|SK|^2}{2|h_0|(|SK|^2-2(n_{H0}+n_{H1}+n_{H2})|SK|+(n_{H0}+n_{H1}+n_{H2})^2)}$$

FS-S-UF

$$n_p\left(\frac{(2+2n_p)n_{H0}+n_{H1}+7n_{H2}+6}{|SK|}+\frac{n_p}{|h0|}+\frac{1}{|h1|}+\frac{(1+n_p)n_{H0}+4n_p+2n_p^2}{|R|}\right)$$

$$\frac{|SK|}{|h_1|(|SK|-(n_{H0}+n_{H1}+n_{H2}))}$$

まとめ

- **単純化した相互認証プロトコル安全性をCryptoVerifで検証した.**
- **単純化した安全性と元の安全性の等価性を人手で証明した.**
- **安全性証明として問題は無いが，CryptoVerifを利用した証明は，攻撃者のアドバンテージを多めに見積もっている.**

- **今後の課題**
  - 元の安全性を直接の証明.
  - 今回は検証の対象としなかった性質の検証.
    - 同期ズレ耐性など
  - 人手の証明と同じレベルの評価に近づけることは可能か？