ゲーム列による安全性証明の形式化と自動化

真野健，櫻田英樹，
河辺義信，塚田恭章
NTTコミュニケーション科学基礎研究所

はじめに

[目的]数理的技法を用いて，形式的に情報セキュリティの証明を行いたい

[方法]直接的方法と間接的方法

二つの方法の違い：攻撃者の能力の形式化の方法

形式化とは

- セキュリティプロトコルの動作や，それに関する推論などを，記号の操作で定式化（c.f. 厳密）

  例: （確率）オートマトン，（確率）Hoare論理，（確率）プロセス計算
- 計算機による証明の支援の可能性（証明チェッカ，自動証明器）
- 実際には，すべてが形式化されるわけではない

  例：形式的な推論規則の正当性を，（形式的にではないが）厳密に証明する

攻撃者の能力の形式化

［古典的アプローチ］

Dolev-Yao モデル（DY）：形式化に都合がいい，取り扱いが楽，
しかし DY の仮定と実際の暗号システムの間には隔たり
⇒ Abadi-Rogaway 理論

[現在]

間接的方法：DYをもとに発展，形式的な取り扱いが楽，
現状では，依然暗号系に強い仮定
“形式化の方法，はじめにありき”

直接的方法: DY を用いない，形式的な取り扱いが大変，
暗号研究者が行っている議論に，より忠実
“形式化の対象，はじめにありき”

おさらい：Dolev-Yao モデル

絶対にやぶられない暗号系を仮定し，そのメッセージや性質を
代数的な体裁で表現（メッセージ代数）

$E(k, x)$ ： 鍵 k によるデータ x の暗号化

$D(k, m)$ ： 鍵 k によるメッセージ m の復号化

$D(k, E(k, x)) = x$ ： k で暗号化して復号化すると元に戻る

$E(k, x)$ は知っているが，鍵 k は知らない
⇒ $D(k, E(k, x))$ などを作れない
⇒ x （と同値な項）を作れない
： k なしに，x の k による暗号化を復号化できない

直接的方法に関する３つの研究

(1) Corin と Hartog: “A Probabilistic Hoare-style Logic for Game-Based Cryptographic Proofs”. ICALP (2) 2006.

(2) Blanchet と Pointcheval: “Automated Security Proofs with Sequences of Games”. CRYPTO 2006.

(3) Canetti ら: “Using Task-Structured Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol”. MIT CSAIL TR, 2006.

共通点:目的

ゲーム列による
安全性証明の形式化

相異点: 方法

特に用いている形式的体系:
(1) 確率 Hoare論理
(2) 確率 プロセス計算
(3) 確率 I/O オートマトン

本日の話

これらの3つの仕事を，方法に関する各ポイント:

用いている形式的体系，
確率・ランダム性・計算が困難な問題の取り扱い，
記述例，形式的証明の概要，ツールサポートの有無

などに関してヤブニラミすることで，直接的方法研究の現状を概観することを目指す．

目次

ゲーム列による安全性証明とは

V. Shoup.“Sequence of games: a tools for taming complexity in security proofs”.

ゲームとは: 攻撃者と挑戦者との間で行われる攻撃ゲーム

セキュリティがやぶられる ⇔ 攻撃者が勝利する

特に，計算量的安全性　⇔ “効率的な”攻撃者の勝つ確率が十分に小さい

ゲーム列による安全性証明とは:

初期ゲームを次々に変換（変形）しながら，以下を証明

- 攻撃者の勝つ確率は，各ステップで“ほとんど”変わらない
- 最終ゲームでは，攻撃者が勝つ確率が十分ちいさい

例：ElGamal 暗号の安全性

ランダムに選ぶ　　$\{0, 1, \ldots, q-1\}$

鍵生成 (pk, sk)　　：$x \overset{R}{\leftarrow} Z_q, \alpha \leftarrow \gamma^x, p_k \leftarrow \alpha, s_k \leftarrow x$

メッセージ m の暗号化 (β, ζ)：$y \overset{R}{\leftarrow} Z_q, \beta \leftarrow \gamma^y, \delta \leftarrow \alpha^y, \zeta \leftarrow \delta \cdot m$

暗号文 (β, ζ) の復号化　　：$m \leftarrow \zeta / \beta^x$

（γ は位数 q の群の生成元）

$$\zeta/\beta^x = \alpha^y m/\beta^x = (\gamma^x)^y m/(\gamma^y)^x = \gamma^{xy} m/\gamma^{xy} = m$$

安全性の“説明”:

公開鍵 $\alpha\,(= \gamma^x)$ や暗号化メッセージの第一要素 $\beta\,(= \gamma^y)$ から $\beta^x\,(= \gamma^{xy})$ が得られると困る，しかしそれは困難.

ElGamal 暗号の安全性の根拠

Decisional Diffie-Hellman 仮定（DDH）

任意の“効率的な”アルゴリズム D について，

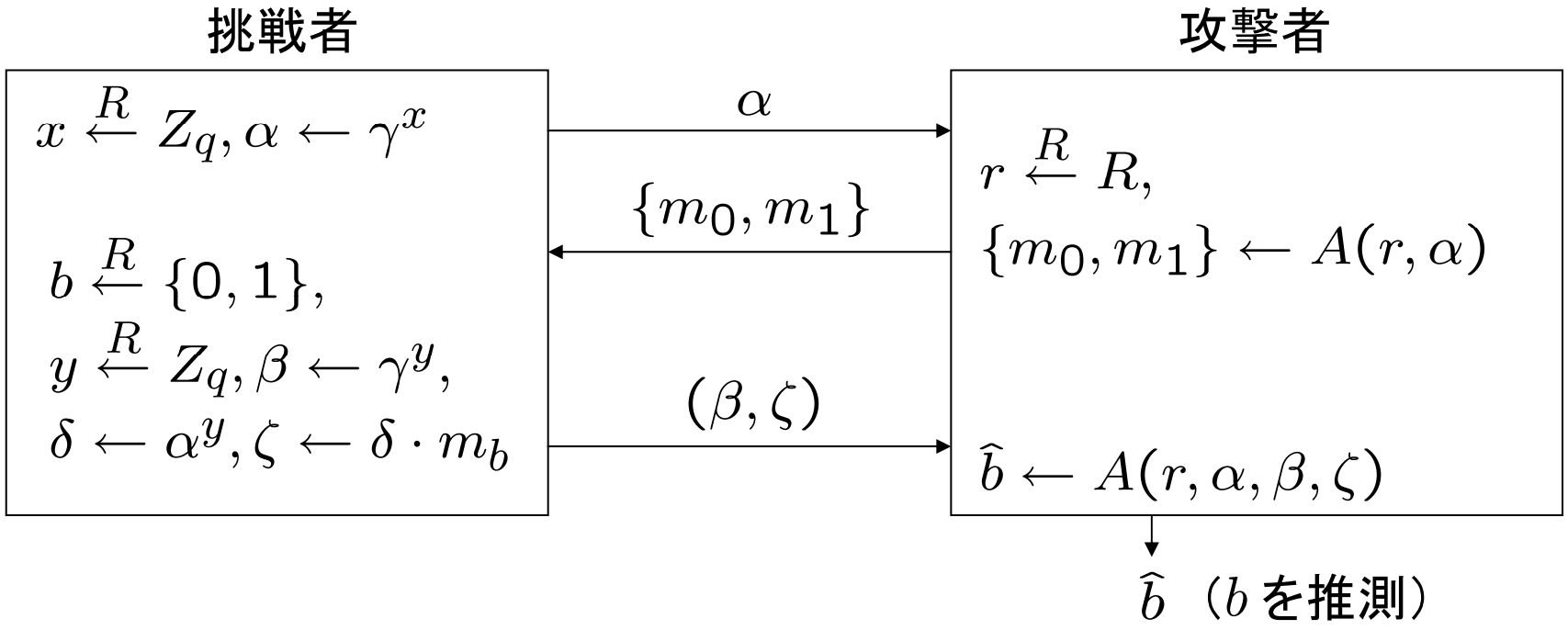
$$|Pr[x, y \xleftarrow{R} Z_q : D(\gamma^x, \gamma^y, \gamma^{xy}) = 1] -$$
$$Pr[x, y, z \xleftarrow{R} Z_q : D(\gamma^x, \gamma^y, \gamma^z) = 1]|$$

が無視できるほど小さい．

以上の“説明”を，厳密に証明: ゲーム列

ゲーム列による証明

ゲーム 0:



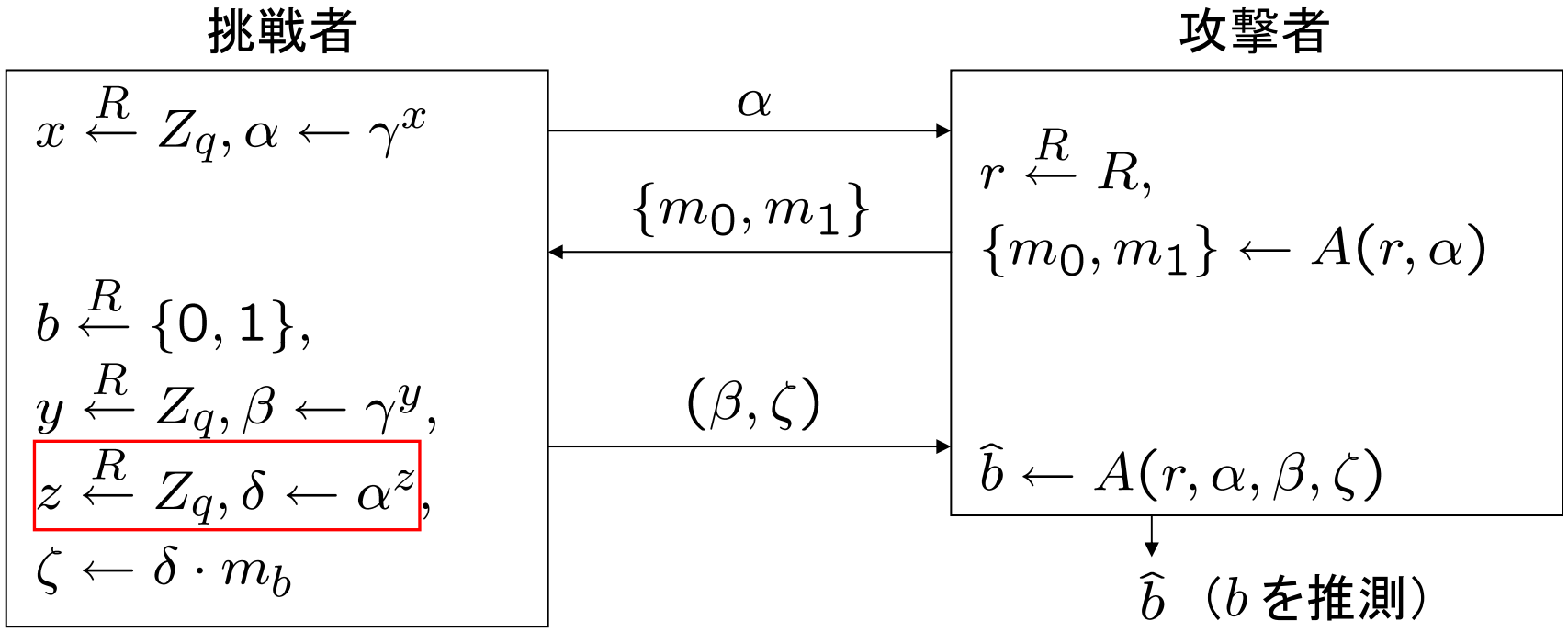
S_0: ゲーム 0 において $b = \hat{b}$ となるイベント（攻撃者の勝利を意味する）

目標: $|Pr[S_0] - 1/2|$ が無視できるほど小さいこと（semantic security）を証明する

攻撃者は，あてずっぽより有意に賢くはない！

ゲーム列による証明（つづき）

ゲーム 1:



S_1: ゲーム 1 において $b = \hat{b}$ となるイベント

［主張1］$Pr[S1] = 1/2$.　∵) δ が y に依存しない乱数 z から作られるから

［主張2］$|Pr[S0] - Pr[S1]|$ は無視できるほど小さい.

（ゲーム書き換えの正当性）

主張2の証明

DDH を適用するためのアルゴリズム S を，ゲーム 0 とゲーム1 を補間することによって構成

Algorithm $S(\alpha, \beta, \delta)$

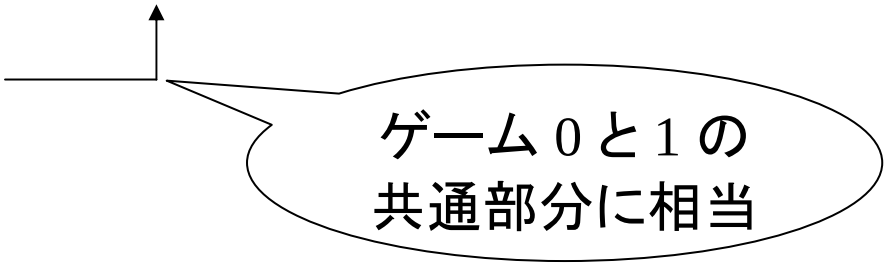
$r \xleftarrow{R} R, \{m_0, m_1\} \leftarrow A(r, \alpha)$

$b \xleftarrow{R} \{0, 1\}, \zeta \leftarrow \delta \cdot m_b$

$\hat{b} \leftarrow A(r, \alpha, \beta, \zeta)$

if $b = \hat{b}$ then output 1

else output 0



なぜ“補間”？

S に $(\gamma^x, \gamma^y, \gamma^{xy})$ を与えるとゲーム 0: $Pr[x, y: S(...) = 1] = Pr[S0]$

S に $(\gamma^x, \gamma^y, \gamma^z)$ を与えるとゲーム 1: $Pr[x, y, z: S(...) = 1] = Pr[S1]$

S に DDH を適用

⇒ $|Pr[S0] - Pr[S1]| = |Pr[S0] - 1/2|$ は無視できるほど小さい．□

ゲーム“列”による証明は，一般には複数のステップからなる．

ゲーム変換の分類

Shoup は，ゲームの変換を 3 つに分類

(1) 識別不能性にもとづく変換

2 つのゲームの差が無視できることを，計算量的仮定を用いて証明

（ElGamal の例の変換はこれ）

2 つのゲームが与えられれば，それらを“補間”するアルゴリズムは機械的に作れることが多い

(2) 失敗イベントに基づく変換

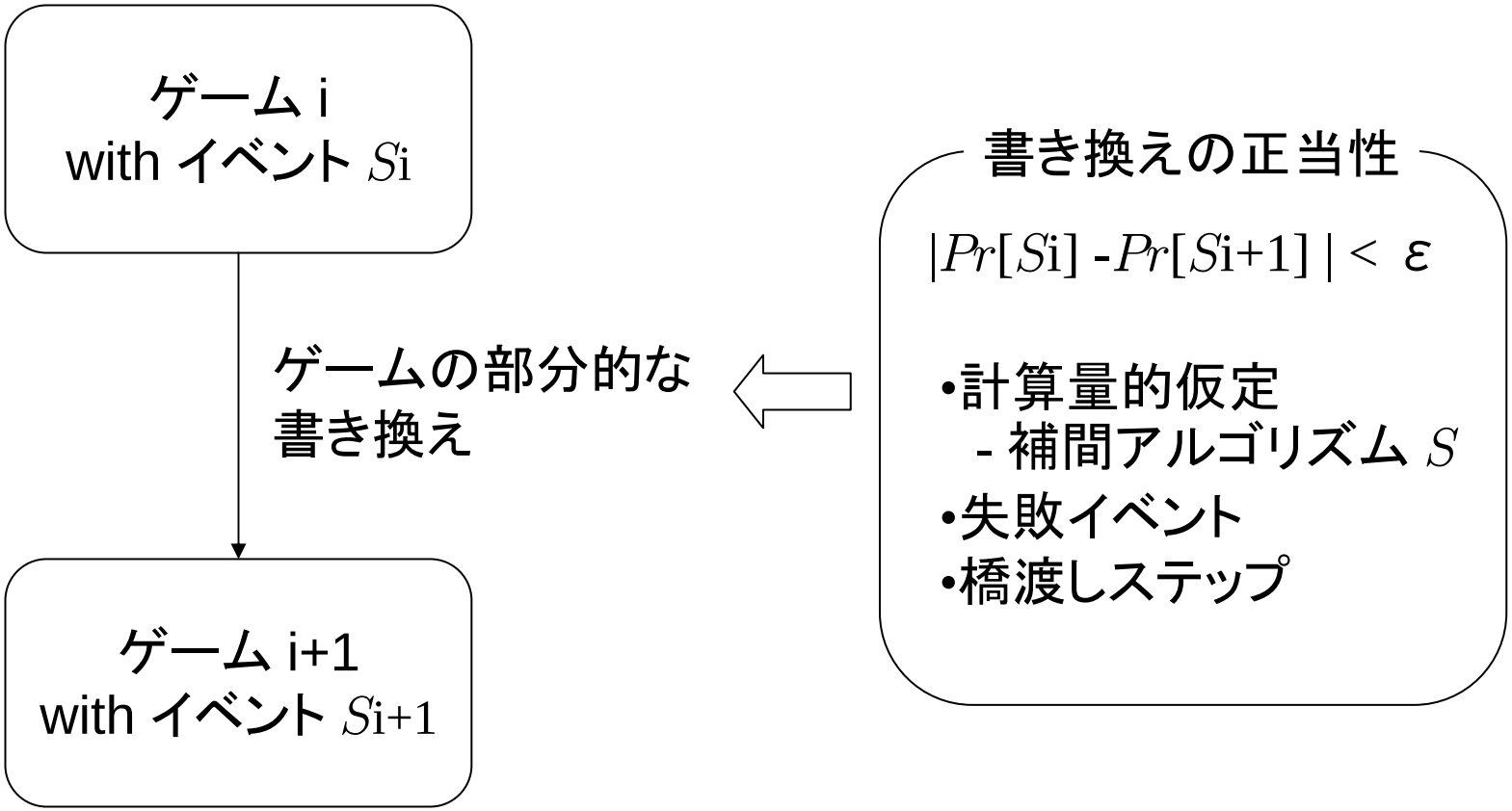
- 2 つのゲームが，失敗イベント F が起こらない限り同一
- F の確率が無視できるほど小さい

を（しばしば組み合わせ論的に）証明

(3) 橋渡しステップ

“純粋に概念的”な変換（等価変換）．例: 条件分岐の条件が常に真なら else 部を取り除く，など．確率には変化なし．

ゲーム列：まとめ


ゲーム i
with イベント Si
ゲームの部分的な
書き換え
ゲーム i+1
with イベント Si+1
書き換えの正当性
$|Pr[S\mathrm{i}] - Pr[S\mathrm{i}+1]| < \varepsilon$
•計算量的仮定
- 補間アルゴリズム S
•失敗イベント
•橋渡しステップ

目次

確率 Hoare 論理を用いた形式化

Hoare 論理: プログラムが仕様を満たしていることを証明するための論理体系

正しさの表明: Hoare の三つ組み

$$\{P\}\ S\ \{Q\}$$

S : プログラム（単純な命令型プログラミング言語で記述）

P, Q : 状態（一階述語論理式で記述）

“事前条件 P が成り立つ時に，プログラム S を実行すると，
その実行後には事後条件 Q が成り立つ”

例: $\{0 \leq x \ \wedge \ x \leq 1\}\, y := x * x\, \{y \leq x\}$

公理/推論規則の例

複合文の規則

$$\frac{\{P\}\, S_1\, \{R\} \quad \{R\}\, S_2\, \{Q\}}{\{P\}\, S_1; S_2\, \{Q\}}$$

代入の公理

$$\{Q[e/x]\}\, x := e\, \{Q\}$$

連言の規則

$$\frac{\{P\}\, S\, \{Q\} \quad \{P\}\, S\, \{R\}}{\{P\}\, S\, \{Q \wedge R\}}$$

帰結の規則

$$\frac{P \to P' \quad \{P'\}\, S\, \{Q'\} \quad Q' \to Q}{\{P\}\, S\, \{Q\}}$$

確率のための拡張

[プログラミング言語]

確率的選択 $\oplus_\rho$

例: $x := 1 \oplus_{1/3} x := 2$

x に 1/3 の確率で 1 が, 2/3 の確率で 2 が代入されるプログラム

[論理式]

確率に関する条件

例: $\mathbb{P}(P) < 1/4$

推論規則も拡張

$$\frac{\{P\}\, S_1\, \{Q_1\} \quad \{P\}\, S_2\, \{Q_2\}}{\{P\}\, S_1 \oplus_\rho S_2\, \{Q_1 \oplus_\rho Q_2\}}$$

Q_1 と Q_2 が表す確率分布の重みつき和

⇒ 確率版での“状態”: 状態の確率分布

“実行”: 実行系列の確率分布

ランダム性の取り扱い

[プログラミング言語]

ランダムアサインメント $x \xleftarrow{R} \{v_1, \ldots, v_n\}$

$$\mathtt{x} := \mathtt{v}_1 \oplus_{1/\mathtt{n}} (\mathtt{x} := \mathtt{v}_2 \oplus_{1/(\mathtt{n}-1)} (\cdots \oplus_{1/2} \mathtt{x} := \mathtt{v}_\mathtt{n}))$$

[論理式]

ランダム性: $e_k \in S_k$ (k = 1, ..., n) が各々独立かつランダム

$$R_{S_1,\ldots,S_n}(e_1, \ldots, e_n) \;=$$
$$\forall i_1, \ldots, i_n : \mathbb{P}(e_1 = i_1 \wedge \ldots \wedge e_n = i_n) = 1/|S_1| \cdot \ldots \cdot 1/|S_n|$$

計算が困難な問題

DDH 仮定は，以下のように書ける:

$$\{\mathbb{P}(\mathtt{true}) = 1\}$$
$$\mathtt{x} \xleftarrow{R} Z_q^*;\ \mathtt{y} \xleftarrow{R} Z_q^*;\ \mathtt{r1} \xleftarrow{R} RND;\ \mathtt{b1} \xleftarrow{R} Bool;\ D(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}; \mathtt{out1});$$
$$\mathtt{z} \xleftarrow{R} Z_q^*;\ \mathtt{r2} \xleftarrow{R} RND;\ \mathtt{b2} \xleftarrow{R} Bool;\ D(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2})$$
$$\{|\mathbb{P}(\mathtt{out1}) - \mathbb{P}(\mathtt{out2})| \leq \varepsilon_{ddh}\}$$

D の出力

無視できるほど小さい

“補間”アルゴリズム S は:

$$\mathtt{m0} := A0(\mathtt{v1}, \mathtt{v4}); \mathtt{m1} := A1(\mathtt{v1}, \mathtt{v4});$$
$$\text{if } \mathtt{v5} = \mathtt{false} \text{ then } \mathtt{tmp} := \mathtt{v3} \cdot \mathtt{m0} \text{ else } \mathtt{tmp} := \mathtt{v3} \cdot \mathtt{m1} \text{ fi};$$
$$\mathtt{b} := A2(\mathtt{v1}, \mathtt{v2}, \mathtt{tmp}, \mathtt{v4});$$
$$\text{if } \mathtt{v5} = \mathtt{b} \text{ then } \mathtt{x1} := true \text{ else } \mathtt{x1} := false \text{ fi};$$

セキュリティ証明の概要

S を用いて，ゲーム 1 の安全性を記述/証明（形式的ではない議論含む）

$$\{\mathbb{P}(\mathtt{true}) = 1\}$$

$$\mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;$$

$$\mathtt{z} \leftarrow Z_q^*;\ \mathtt{r2} \leftarrow RND;\ \mathtt{b2} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2})$$

$$\{\mathbb{P}(\mathtt{out2}) = 1/2\}$$

$\{R_{Z_q^{*3}, RND, Bool}(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2})\}$ を証明

DDH 仮定，
プログラムの“直交性”，
連言の規則，
帰結の規則

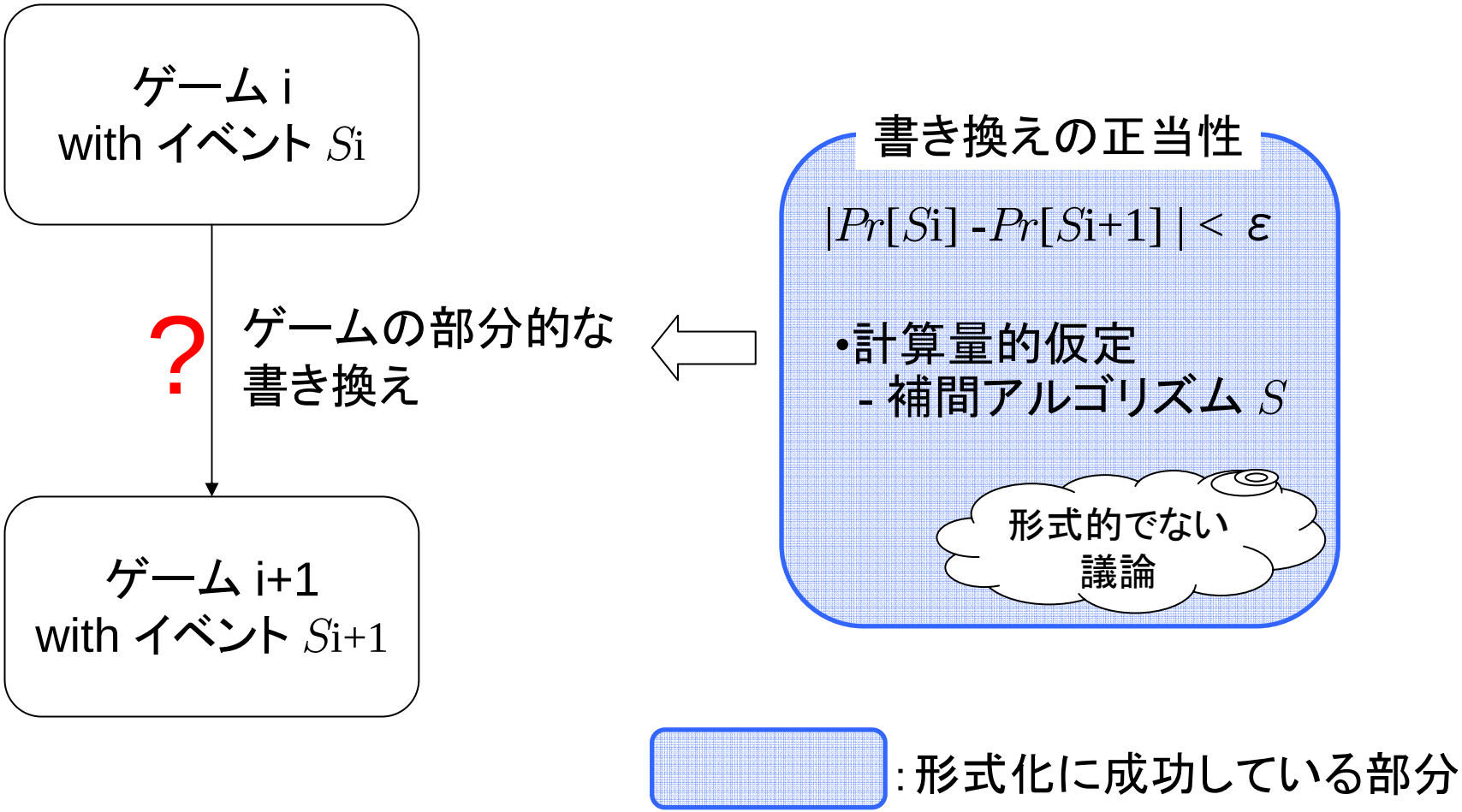
S を用いて記述された，ゲーム 0 の安全性を形式的に導出．

$$\{\mathbb{P}(\mathtt{true}) = 1\}$$

$$\mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{r1} \leftarrow RND;\ \mathtt{b1} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}, \mathtt{out1});$$

$$\{|\mathbb{P}(\mathtt{out1}) - 1/2| \leq \varepsilon_{ddh}\}$$

確率 Hoare 論理：まとめ



：形式化に成功している部分

目次

確率プロセス計算を用いた形式化

プロセス計算とは:

並行システムの基本要素を演算子に対応させ，代数的な体裁で記述

プロセス式

0	: 何もしないプロセス
$P + Q$	: プロセス P と Q を非決定的に選択して実行
$P \mid Q$	: P と Q を並行に実行
$\bar{c}(d).P$	: 通信チャネル c へ値 d を送信
$c(x).P$	: 通信チャネル c から x の値を受信
$(\nu c)P$	: 新しい通信チャネルの生成
$!P$	: replication, P のコピーを無限個生成（ループ構文の代わり）

プロセス式の動作

構造同値関係 ≡（抜粋）

$$P|0 \equiv P,\ P + P \equiv P,$$
$$(\nu c_1)(\nu c_2)P \equiv (\nu c_2)(\nu c_1)P,$$
$$!P \equiv P|!P$$

遷移規則（抜粋）

$$\text{COMM}: \ (\cdots + x(y).P) \mid (\cdots + \overline{x}z.Q) \to P\{^z/_y\} \mid Q$$

$$\text{PAR}: \ \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{STRUCT}: \ \frac{Q \equiv P \quad P \to P' \quad P' \equiv Q'}{Q \to Q'}$$

合同関係に基づく意味論

合同関係 $\approx$:

$P \approx Q$ ならば，任意の文脈 $C[\ ]$ について
$C[P] \approx C[Q]$ となるような同値関係

$\Rightarrow$ P を Q に書き換えても全体の意味が変わらない
（書き換え“規則”の正当性）

本手法のプロセス計算

ゲーム列の表現に便利なように，Blanchet が拡張/変更

- イベント
- replication は有限個に制限
  - 計算時間を見積もるため，構文: **foreach** $i \leq d$ **do** Q
- 配列（ランダムオラクルモデルの形式化に利用）
  - foreach: 配列の宣言，find: 配列要素の検索
- 乱数生成 $x \xleftarrow{R} M$
- オラクル
  - 引数を受け取って値を返す単なるプロセス，“全てのプロセスはオラクルである”

ごめんなさい，ちょっとマニアックな話

配列の形式化

- foreach の中で代入される変数は，すべて配列．すなわち，

  foreach $i \leq N$ **do** $x \leftarrow M(i); \ldots$

  は，暗に 配列 $x[N]$ を宣言している．

- 配列の値は，並行に実行されるプロセス（代入文の“スコープ”の外！）から参照できる．

  (**foreach** $i \leq N$ **do** $x \leftarrow M(i);\ \ldots$)
  | (**find** $j \leq N$ **suchthat defined** $(x[j]) \wedge y = x[j]$ **then** $\ \ldots$)

記述例: 理想的なハッシュ関数

ハッシュ関数を，本当の乱数表だとみなす（ランダムオラクルモデル）

$$\begin{aligned}
&\textbf{foreach } i_{\mathrm{h}} \leq n_{\mathrm{h}} \textbf{ do } OH(x : bitstring) := \textbf{return}(\mathsf{hash}(x)) \\
\approx_0\ &\textbf{foreach } i_{\mathrm{h}} \leq n_{\mathrm{h}} \textbf{ do } OH(x : bitstring) := \\
&\quad \textbf{find } u \leq n_{\mathrm{h}} \textbf{ suchthat } (\textbf{defined}(x[u], r[u]) \wedge x = x[u]) \textbf{ then return}(r[u]) \\
&\quad \textbf{else } r \xleftarrow{R} D; \textbf{return}(r)
\end{aligned}$$

すでに問い合わせがあった値なら，それを返す

さもなければ新たに乱数を生成

ゲームの書き換え規則

$P_1 \approx_p P_2$：（確率的）観測等価性

- 時間 t の実行では，いかなる文脈も $P1$ と $P2$ を確率 $p(t)$ 以上で区別できない．
- “合同的”，ただし同値関係ではない（推移律が成り立たない）．

本手法のプロセス計算（つづき）

非決定性の排除

- + は用いない
- 乱数生成，通信，find: すべての可能性が等しい確率分岐と解釈

ちょと脇道にそれますが，

非決定性な分岐と，確率的な分岐

確率に関する共通点：確率が入っていない体系に，確率を導入

非決定性（どちらを選ぶか規定されていない分岐）をどう取り扱うか：
取り扱いは三者三様．

- 確率 Hoare 論理: 決定的な体系に，確率分岐を導入
  → 純粋に確率的
- Blanchet の確率プロセス計算: 非決定性を確率的に解釈
  （非決定的分岐は，一様な確率の分岐だと思う）
  → 純粋に確率的
- task-PIOA: 非決定的な体系に，確率を導入
  → 両方の分岐がある

他の 2 つに比べて複雑

セキュリティ証明の例

対象: Full Domain Hash（FDH）署名スキーム

ハッシュ関数: hash

落し戸付き一方向性置換: f(pk, m)

およびその逆関数: invf(sk, m)

公開鍵暗号

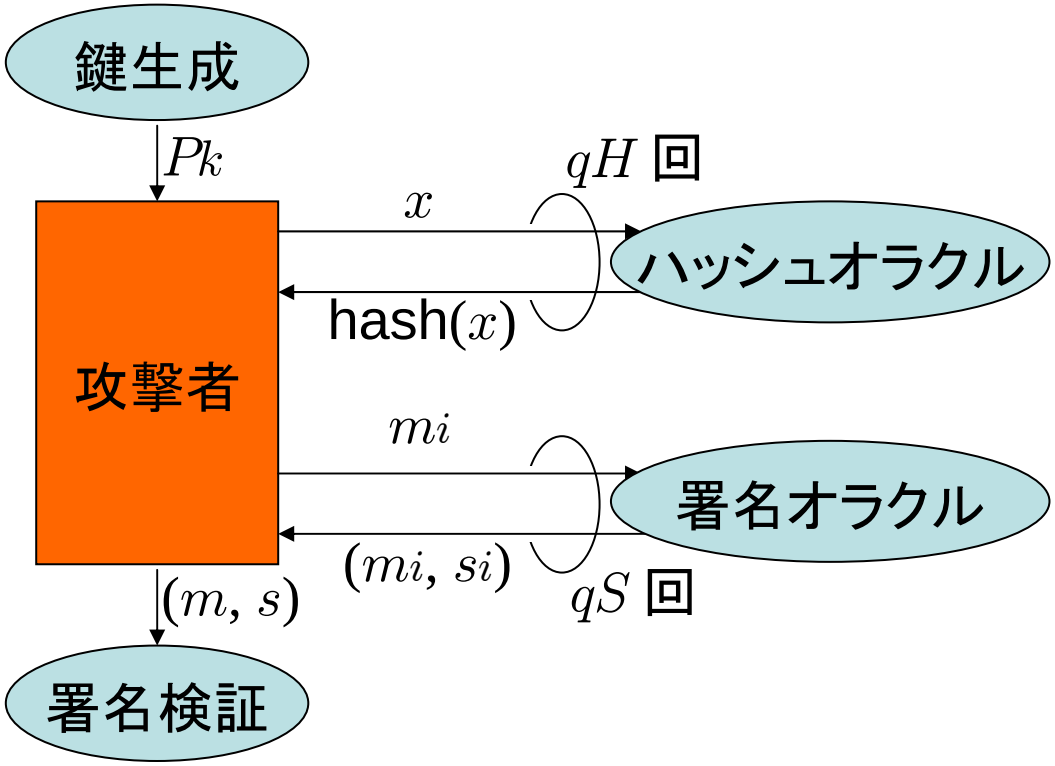
署名 sign(m, sk) $\stackrel{\text{def}}{=}$ invf(sk, hash(m))

検証 check(m, pk, s) $\stackrel{\text{def}}{=}$ f(pk, s) = hash(m) ?

署名の安全性の定式化

FDH の安全性： existential UnForgeability against Chosen Message Attack（UF-CMA）

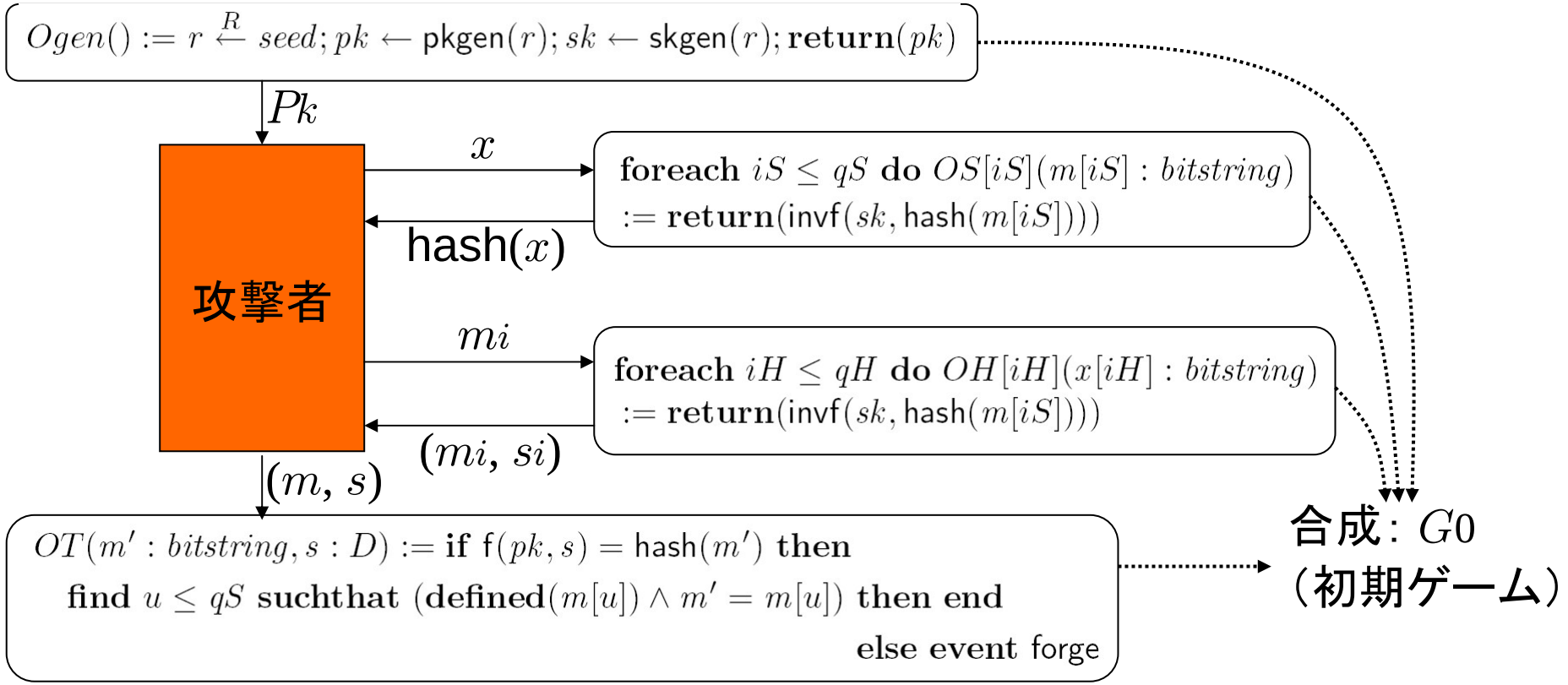
攻撃者は，公開鍵，ハッシュオラクル，署名オラクルをあたえられて，
未知の署名 s を生成しようとする．生成できたら攻撃者の勝ち．



署名の安全性の定式化

FDH の安全性：existential UnForgeability against Chosen Message Attack（UF-CMA）

攻撃者は，公開鍵，ハッシュオラクル，署名オラクルをあたえられて，
未知の署名 s を生成しようとする．生成できたら攻撃者の勝ち．



初期ゲーム $G0$

$$
\begin{aligned}
G_0 = {} & \textbf{foreach } iH \leq qH \textbf{ do } OH(x : bitstring) := \textbf{return}(\mathsf{hash}(x)) \\
& | \; Ogen() := r \xleftarrow{R} seed; pk \leftarrow \mathsf{pkgen}(r); sk \leftarrow \mathsf{skgen}(r); \textbf{return}(pk); \\
& \quad (\textbf{foreach } iS \leq qS \textbf{ do } OS(m : bitstring) := \textbf{return}(\mathsf{invf}(sk, \mathsf{hash}(m))) \\
& \quad | \; OT(m' : bitstring, s : D) := \textbf{if } \mathsf{f}(pk, s) = \mathsf{hash}(m') \textbf{ then} \\
& \qquad \textbf{find } u \leq qS \textbf{ suchthat } (\textbf{defined}(m[u]) \wedge m' = m[u]) \textbf{ then end} \\
& \qquad\qquad \textbf{else event } \mathsf{forge})
\end{aligned}
$$

形式的証明の概要

セキュリティ証明の仮定

- 理想的なハッシュ関数
- f, invf の一方向性

$$\begin{aligned}
&\textbf{foreach } i_{\mathrm{k}} \le n_{\mathrm{k}} \textbf{ do } r \xleftarrow{R} seed; (Opk() := \textbf{return}(\mathsf{pkgen}(r)) \\
&\quad | \textbf{ foreach } i_{\mathrm{f}} \le n_{\mathrm{f}} \textbf{ do } x \xleftarrow{R} D; (Oy() := \textbf{return}(\mathsf{f}(\mathsf{pkgen}(r), x)) \\
&\quad\quad | \textbf{ foreach } i_1 \le n_1 \textbf{ do } Oeq(x' : D) := \textbf{return}(x' = x) \\
&\quad\quad | Ox() := \textbf{return}(x))) \\
\approx_{p^{\mathsf{ow}}} &\textbf{foreach } i_{\mathrm{k}} \le n_{\mathrm{k}} \textbf{ do } r \xleftarrow{R} seed; (Opk() := \textbf{return}(\mathsf{pkgen}'(r)) \\
&\quad | \textbf{ foreach } i_{\mathrm{f}} \le n_{\mathrm{f}} \textbf{ do } x \xleftarrow{R} D; (Oy() := \textbf{return}(\mathsf{f}'(\mathsf{pkgen}'(r), x)) \\
&\quad\quad | \textbf{ foreach } i_1 \le n_1 \textbf{ do } Oeq(x' : D) := \\
&\quad\quad\quad \textbf{if defined}(k) \textbf{ then return}(x' = x) \textbf{ else return}(\mathsf{false}) \\
&\quad\quad | Ox() := k \leftarrow \mathsf{mark}; \textbf{return}(x)))
\end{aligned}$$

ただし $p^{\mathsf{ow}}(t) = n_{\mathrm{k}} \times n_{\mathrm{f}} \times \mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t + (n_{\mathrm{k}} n_{\mathrm{f}} - 1)t_{\mathsf{f}} + (n_{\mathrm{k}} - 1)t_{\mathsf{pkgen}})$

$\mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t)$ は，時間 t で一方向性が破られる確率

（この変形規則の正統性や多項式は，人間が手で導く）

形式的証明の概要（つづき）

(1) 識別不能性にもとづく変換

ランダムオラクル，一方向性の書き換え規則（形式的）

(2) 失敗イベントに基づく変換

形式的な規則として明示的には現れないが，2 つの書き換え規則の正当性証明の中で利用

(3) 橋渡しステップ

simplification と呼ばれる形式的規則と，それを適用する戦略によって実現

変形の終了判定: 攻撃者の勝利（この場合は署名の偽造）を表すイベント forge が，ゲームから構文的に消えたら終了

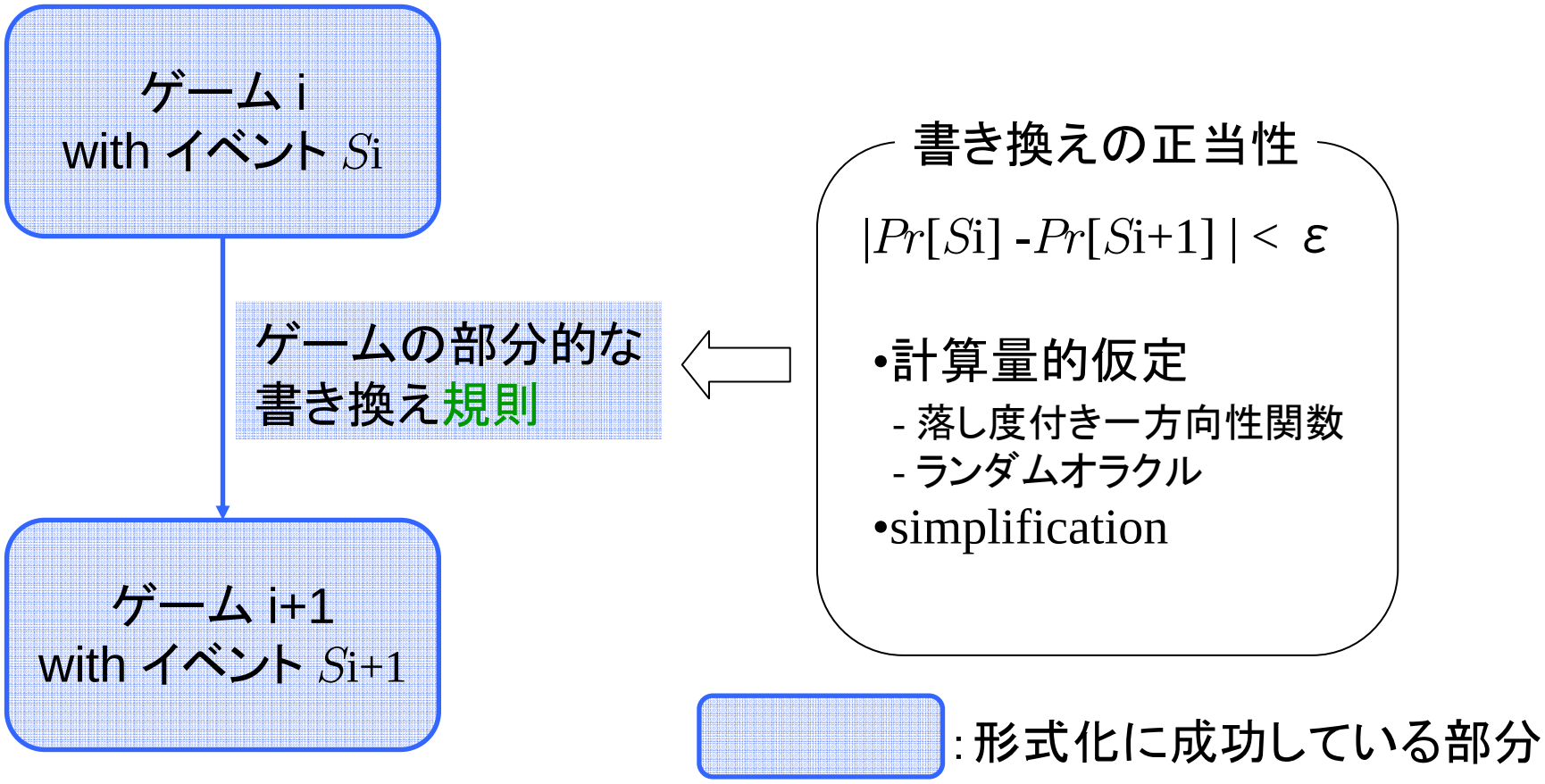
結果: 偽造される確率の上限: $(qH + qS + 1)\mathsf{Succ}^{\mathsf{ow}}_{\mathcal{P}}(t + (qH + qS)t_{\mathsf{f}})$

（FDH の原論文で得られたものと本質的に同じ）

証明ツール Cryptoverif

- ゲーム変換を（半）自動で実行
  - FDH の例については自動で検証
  - 実行時間は 14ms（Pentium M 1.8GHz）
- 確率も自動で計算
- Web で公開:
  http://www.di.ens.fr/~blanchet/cryptoc-eng.html

確率プロセス計算：まとめ



ε の定量的評価，書き換え規則の
適用戦略などもツール化

国内の関連研究

J. Almansa, T. Okamoto (NTT) “Probability Space Transformations and Cryptographic Proofs”. 日仏コンピュータセキュリティシンポジウム.

- 識別不能性を，きちんと同値関係として解釈

目次

task-PIOA を用いた形式化

FAIS 第一回研究集会で O. Pereira が発表

［対象］

- Goldreich, Micali, Wigderson:87 の忘却転送（oblivious transfer）プロトコルの安全性
- 汎用的結合可能性（UC）のスタイルに乗っ取って，安全性を定式化

［証明方法］

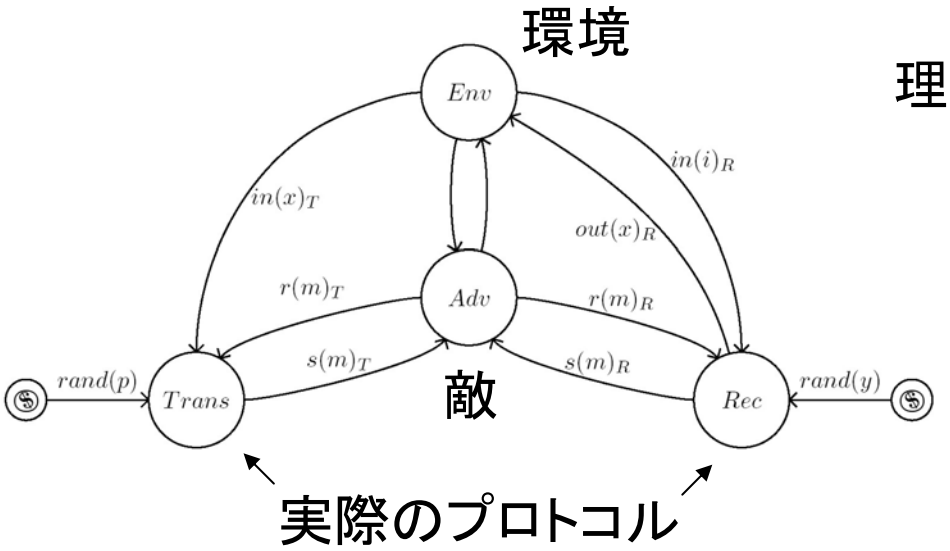
- task-PIOA を用いて形式化
- 確率シミュレーション証明技法によって証明
- ポイント: 安全性の根拠 hard-core predicate（HCP）に関する仮定を，オートマトンの書換えで表現

安全性の定式化と形式化

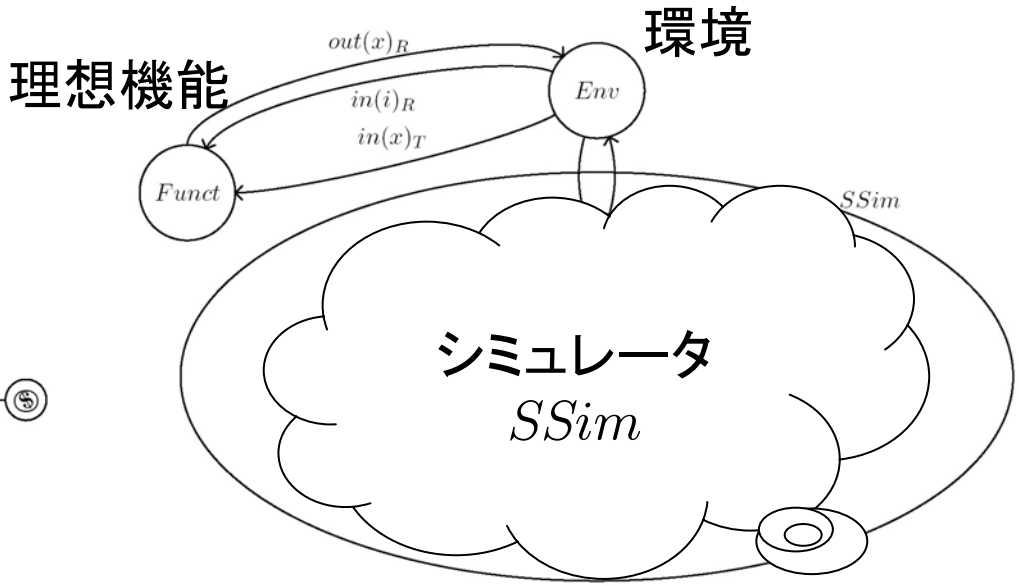
UC では，プロトコルの仕様を“理想機能” $Funct$ として記述

任意の敵 Adv に対して，シミュレータ $SSim$ が存在して，任意の環境 Env は，現実世界（実際のプロトコル実行）と，理想世界（理想機能＆シミュレータ）を計算量的に識別できない．

現実世界



理想世界



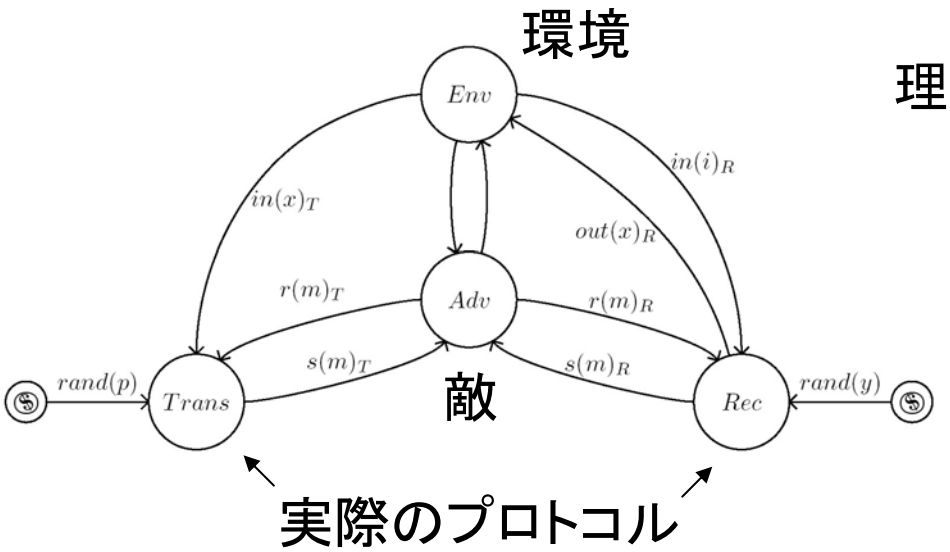
安全性の定式化と形式化

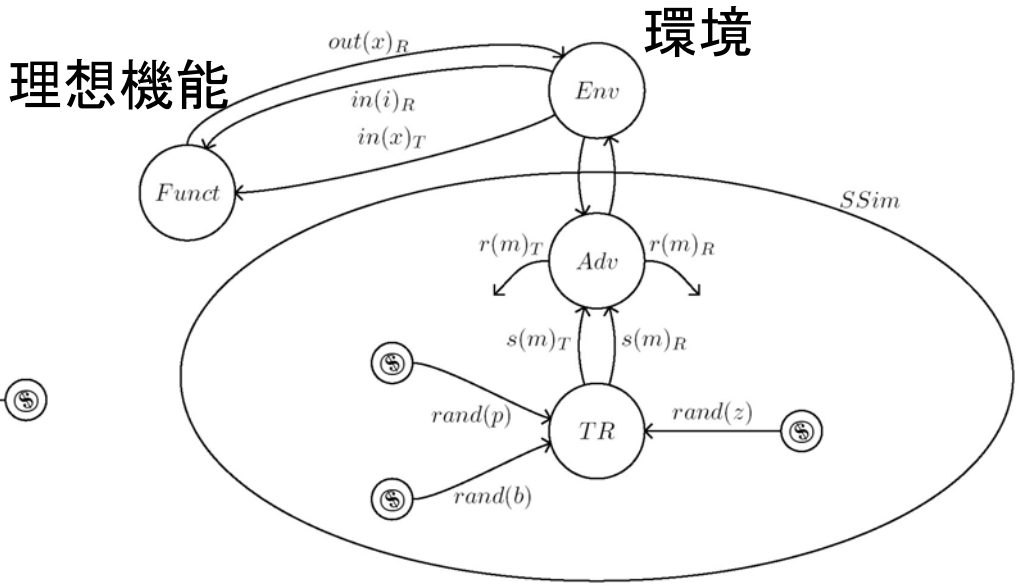
UC では，プロトコルの仕様を“理想機能” $Funct$ として記述

任意の敵 Adv に対して，シミュレータ $SSim$ が存在して，任意の環境 Env は，
現実世界（実際のプロトコル実行）と，理想世界（理想機能＆シミュレータ）を
計算量的に識別できない．

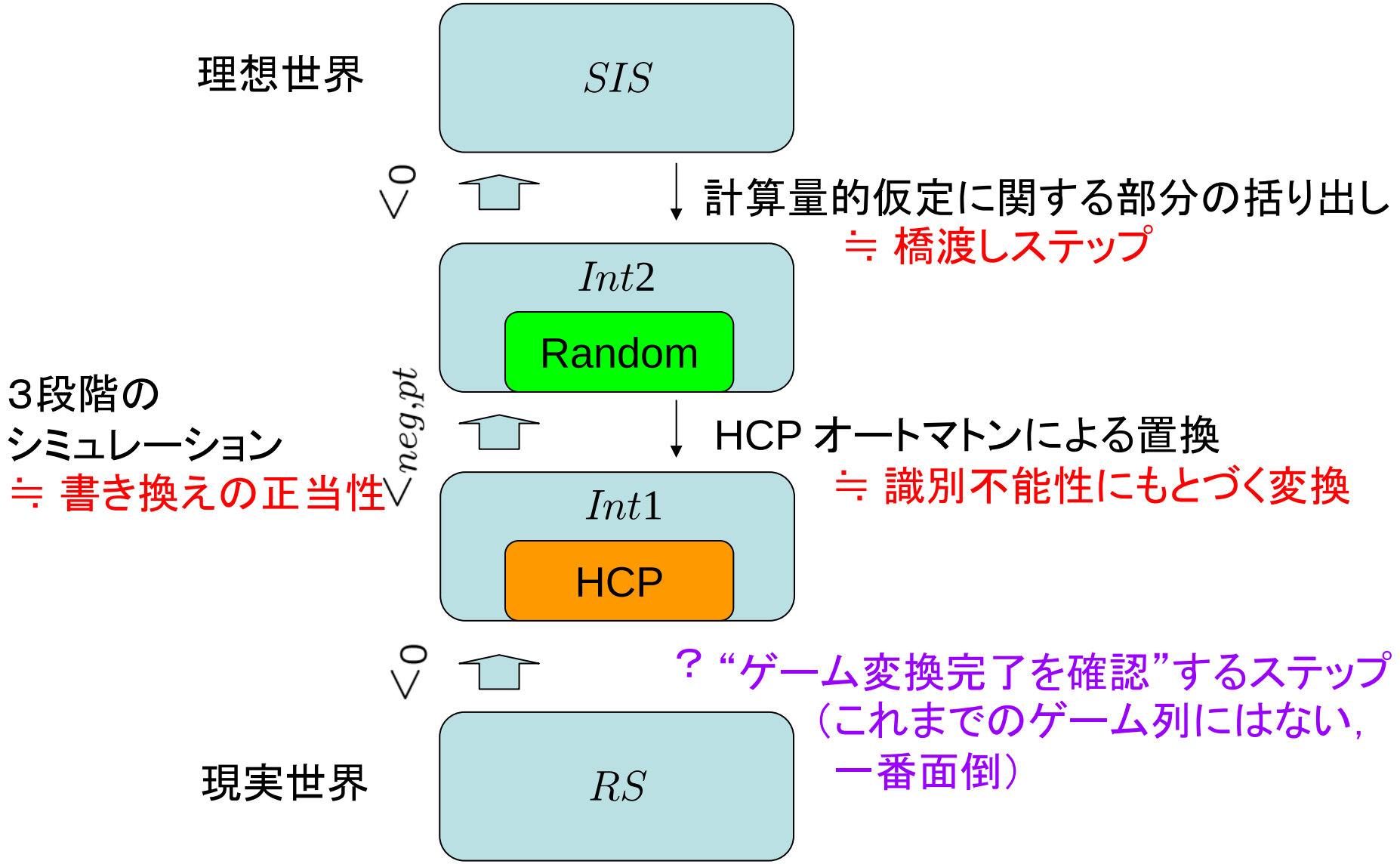
“implement”

現実世界: task-PIOA RS $\leq_{neg,pt}$ 理想世界: task-PIOA SIS





シミュレーションによる証明≒ゲーム列



国内の関連研究

櫟粛之（NTT），“Task-PIOA に基づく Ideal Functionality 実現の証明の自動化”．信学会 AI 研究会．

- “ゲーム変換完了を確認”するステップ（$<_0$）を自動化する手法

まとめ

ゲーム列による安全性証明の形式化に関する 3 つの研究

(1) Corin と Hartog:06（確率 Hoare論理）
(2) Blanchet と Pointcheval:06（確率 プロセス計算）
(3) Canetti ら:06（確率 I/O オートマトン）

について，形式化の方法を中心に概説.