数理解析研究所講究録
第1604巻 2008年 9-18

定理証明ツールによる証明可能安全性

田中三貴
産業技術総合研究所
情報セキュリティ研究センター

概 要

ゲーム列による証明手法 (以下ではゲームの手法と呼ぶ.) は, 検証し易い安全性証明を構成するための方法である. 安全性性質や, 帰着先の計算量的問題はゲームとして記述され, 帰着による証明はゲーム書き換えの列として表現される. とりわけ, 注目すべき最近の提案に, プログラミング言語を用いてゲームを記述し (つまりゲームをプログラムとして記述し), プログラム書き換えというシンタクティックな操作として証明を行うというものがある [5]. 本稿では, 以上の流れについて概説したのち, 上記の提案を実現する安全性証明の構成ツールとして定理証明支援系 Coq 上でのゲーム手法の形式化を行った研究 [10] について概説する. これは, 確率的プログラミング言語を形式的に定義し, 安全性証明に必要な性質を全て Coq 上で表現・証明するというものである. 特にゲーム書き換えステップの根拠となる基本補題と呼ばれるものの証明と, 応用例として PRP/PRF Switching Lemma の証明について説明する.

1 ゲームの手法

ゲームの手法では, 安全性性質を, 攻撃者が解くべき課題あるいは挑戦と考え, それを攻撃者に提示される「ゲーム」という形に記述する. このゲームは攻撃者がやり取りをする外界環境と考えることもできる. このゲームの記述を通じて, 攻撃者に与えられる入力や, 攻撃者に使うことが許されるオラクル等がゲームの記述により決定される. このようにして, 攻撃者の能力を制限したり, 証明すべき安全性性質を規定したりすることが可能である. 例えば, ゲームの中で攻撃者に許される操作の違いにより, CPA(選択平文攻撃), CCA(選択暗号文攻撃) などの攻撃者の能力の違いが表現される. ゲームの実行が終了すると, 攻撃者は何らかの値を出力する. この出力の値により攻撃が成功したか否かが判断される.

今, あるゲーム G_0 について攻撃者の成功確率が不明な場合に, ゲームの内容を $G_1, G_2, \ldots$ と徐々に変更していき, 最終的に成功確率が計算可能なゲーム G_n にすることを考える. さらに, この際に, ゲームを段階的に変更する時の各段階における成功確率の差分の上限を計算し積算することができれば, 元来のゲームにおける攻撃の成功確率についても上限を得ることができる. これがゲームの手法の基本方針である.

よって，ゲームの手法において重要となるのは，ゲームをいかに変更(変換)していくかという点と，それに伴う成功確率の変化を的確に評価することである．ゲームをいかに変換するか(書き換えるか)については，ゲーム変換の各ステップの類型として以下の3種類が考えられている[4].

Bridging steps 確率変化を伴わない単純な書き換えによるステップ．攻撃者の成功確率は不変.

Steps based on indistinguishability 入力の確率分布が異なっているが，その差が無視できる程度であるような二つのゲームから成るステップ．攻撃者の成功確率の差も無視できる程度である.

Steps based on failure events ある失敗事象Fが起らない限りは全く同様の振舞いをする二つのゲームから成るステップ．このような事象Fの生起確率が無視できる値であれば，攻撃者の成功確率の差も無視できる．証明は後節で詳説する補題(基本補題と呼ばれる)による.

本稿では，特に，このうち3番目の種類に注目する．確率変化の評価については，ゲーム1からゲーム2への変換を考えるとし．ゲームiにおいて攻撃者が成功する確率をS_iとすると，ここで問題となるのは$Pr[S_1]$と$Pr[S_2]$の関係，とりわけ，セキュリティパラメタηの関数として$Pr[S_1]$が無視できることと$Pr[S_2]$が無視できることが同値である場合が実用的には重要となる．(実数値の関数fが無視できるとは，任意の整数$c \in \mathbb{Z}$に対し，あるk_0が存在し，いかなる$k \geq k_0$に対しても$|f(k)| \leq k^{-c}$が成立することを言う．)

1.1 最近の傾向

ゲームの手法は，暗号理論分野における帰着証明の中で様々な形で使われてきたものであるが，特に最近，Shoup[4], Pointcheval[6], Bellare and Rogaway[5]などの論文で詳しく紹介されており，証明可能安全性における証明手法の一つとして確立されつつある．その背景としては，次のような状況がある．安全性証明は，その目的上，証明の正しさに非常に価値があるにもかかわらず，一般的傾向として非常に複雑であることから，証明の検証が容易でないことが多い．そのため当然の要望として誰もが容易に検証できる形で証明を提供することが求められており，ゲームの手法はその流れに貢献する期待のもとに洗練され，広められてきたと言えるだろう.

なかでも，Bellare and Rogaway[5]では，code-based game-playingと呼ばれるものが提案されている．これは，一般的には疑似コードのような形でアルゴリズムとして記述されるゲームを，厳密に定義された言語によって書かれたプログラムのレベルまで持ち上げようという提案である．形式的に定義されたプログラミング言語を用いて，ゲームをプログラムとして記述すれば，ゲームの変換はプログラムの書き換えとなる．これは，完全にシンタクティックな操作であるため，機械的に行なうことが可能である．このようにして構成された証明は，人の手による証明に比べ

て誤りの可能性が低く，一方コンピュータによる自動化の可能性が高まる．安全性証明への (部分的) 自動化の導入の有用性についてはHalevi[7] においても議論されている．

本稿では，以下において，この Bellare and Rogaway[5] の提案を実現に近づける試みである Affeldt, Tanaka and Marti[10] の内容を紹介する．

2 確率的プログラミング言語

2.1 確率的状態の形式化

Affeldt, Tanaka and Marti[10] における確率的状態 (状態の確率分布) と，それに基く確率の概念の形式化について説明する．

確率的状態 まず，(確率的動作のない) 通常の意味でのプログラム遷移状態を決定性状態と呼ぶ．そして，決定性状態から成る確率空間を考え，この上での決定性状態の確率分布を確率的状態と呼ぶ．形式的には，確率的状態は (正実数値，決定的状態) の対のリストとして定義される．つまり，決定性状態の型を `A` とすると，Coq における確率的状態の型は `distrib = list (R * A)` と表される．ここで，決定的状態の型はパラメータ (変数 `A`) である．

`(s, r)` をある確率的状態 `d`(型は `distrib`) の一要素とすると，状態 `s` が起こりうる確率はリスト `d` 全体の実数値部分の総和 `sum d` に対する `r` の比率として与えられる．ここで，`sum` は確率的状態を引数にとり，リストの実数値部分の総和を返す演算子であり，Coq 上の再帰的関数として定義される．

```
Fixpoint sum (d : distrib) : R :=
  match d with (p, _) :: tl => p + sum tl | nil => 0 end.
```

一般的な確率の扱いでは，この総和が 1 になるように正規化するが，[?] の形式化では便宜上正規化は行なっていない．

確率事象 確率事象 `event e` は，決定性状態上のブール値関数として定義される．確率的状態 `d` において，このように定義された事象 `e` の生起確率を求めるには，`d` に対し `e` をリストのフィルターとして適用してから `sum` を適用すればよい．Coq 上では，関数 `filter` の再帰的定義は次のようになっている．

```
Fixpoint filter (e : event) (d : distrib) : distrib :=
  match d with
    | (p, a) :: tl => (if e a then (p, a) :: nil else nil) ++ filter e tl
    | nil => nil
  end.
```

これを用いて，事象 `e` の生起確率は `sum (filter e d)` と計算できる．これを `Pr e d` と表記する．

確率的状態に対する操作　確率状態に対しては，sum，filterの他に演算scale，map，forkが定義されており，これらの組み合わせでランダムサンプリング等の操作に対する意味論を定義する (Sect. 2.2). mapは，決定性状態上の関数を引数にとり，リストの各要素について，決定性状態に対し引数の関数を適用するという演算である．一方scaleは正の実数値を引数にとり，リストの各要素の確率比部分を実数値倍する演算である．演算forkは，確率状態の各要素を分割し共起確率を導入するものである．これはmapとscaleを用いて次のように定義される．

```
Fixpoint fork (l : list (R * (A → A))) (d : distrib) : distrib :=
  match l with
    | (k, f) :: tl => map f (scale k d) ++ fork tl d
    | nil => nil
  end.
```

引数のリストlは，各分岐についての，分岐に伴う状態の変化と対応する確率比のペアを要素とする．このリストの要素数が一決定性状態からの分岐の数を表す．

ブール値サンプリング，ランダムサンプリングはforkを用いて定義される．簡単な具体例として，自然数値の変数割り当てについての確率的状態dが

$$(p_0, x{=}0) :: (p_1, x{=}1) :: \mathtt{nil}$$

と定義されている場合を考える．これは，変数xが確率$p_0/(p_0+p_1)$で値0を，確率$p_1/(p_0+p_1)$で値1を取るという状態である．ここに，新たなブール値変数yを導入し，確率pで値0を選んだ場合の状態を考える．x=0の場合は，yの値によって，$(pp_0, x{=}0 \wedge y{=}0)$と$((1-p)p_0, x{=}0 \wedge y{=}1)$という新たな二つの状態に分岐する．x=1の場合についても同様であり，結局全体として，

$$(pp_0, x{=}0 \wedge y{=}0) :: ((1-p)p_0, x{=}0 \wedge y{=}1) :: (pp_1, x{=}1 \wedge y{=}0) :: ((1-p)p_1, x{=}1 \wedge y{=}1) :: \mathtt{nil}$$

という，4つの要素からなる確率的状態が得られる．この状態はforkを使って次のように表される．

```
fork ((p, update y 0)::(1-p, update y 1)::nil) d
```

ここで，update y nは現在の変数割り当てに変数yを導入し値をnとするという関数である．

2.2　プログラミング言語の定義

Affeldt, Tanaka and Marti[10] で定義するゲーム手法のための言語は，ランダムサンプリングと関数呼び出しの機能を持つ命令型プログラミング言語である．さらに，この言語に対し，前節で定義した確率分布の形式化を応用して操作的意味論が定義されている．ここでの確率分布における決定性状態は，変数割り当てとランダムオラクルの対として定義される．ここで，ランダムオラクルとは疑似ランダム関数やハッシュ関数を理想化して実現するためのデータ構造である．確率的プログラムの実行は，確率状態上の遷移，つまり確率分布の変化に相当する．

ランダムオラクル　暗号理論におけるハッシュ関数とは，任意の長さのメッセージを引数にとり，一定長の出力を返す関数であって，いくつかの特殊な性質を満たすものである．このような関数は理想的にはランダム関数のように振舞い，かつ効率的に計算ができる必要がある．暗号理論における証明では，このような関数の抽象化としてランダムオラクルというものが考案されてきた．ランダムオラクルとは，与えられた入力に対し，値域から一様に選ばれた値を出力として返すような関数である．そのような関数の存在を仮定して，それを疑似ランダム関数やハッシュ関数として使って構成された証明では,「ランダムオラクル仮定のもとで安全」などという表現が使われる．

Coq上ではランダムオラクルはハッシュ表のようなデータ型として実現される．関数 f をランダムオラクルとして実現する場合，f の入力 (引数) がキー，出力が値となる．ランダムオラクルは以下のような振舞いを繰り返す．

- 入力 x が与えられたら，まず表中にキー x が既に登録されているか否かを調べる．
  - Yes →対応する値 y を返す．
  - No →値域からランダムにサンプリングした値 y を $f(x)$ として返し，同時に表にキー x と値 y を登録する．

このような構造を [10] ではCoqモジュールによる抽象データ型として実現している．

実行状態　ここで定義される確率的プログラミング言語では，プログラムの実行が進むにつれ，オラクルのエントリと変数割り当ての内容が変化する．したがってプログラムの実行状態はオラクルと変数割り当てのペアの確率分布として定義される．前節の定義に照らせば，決定性状態の型を `dstate` とすると `dstate = (list (var * nat)) * oracle.t` と定義され，これを使って確率的状態 `pstate` を `distrib dstate` と定義する．

シンタックス　式 (expression) は型 `expr` を持ち次のように帰納的に定義される．

```
Inductive expr : Set :=
  var_e : var → expr | int_e : nat → expr | neg_e : expr → expr.
```

ある決定性状態における式の値は関数 `eval` により得ることができる．命令文 (型 `cmd`) の定義は次のとおりである．

```
Definition fun_id := nat.
Inductive cmd : Set :=
| skip : cmd
| assign : var → expr → cmd          (* Notation: _ <- _ *)
| sample_n : var → nat → cmd         (* Notation: _ <-$- _ *)
| sample_b : var → R → cmd           (* Notation: _ <-b- _ *)
| find_value : var → expr → cmd
| insert : expr → expr → cmd
| ifte : expr → cmd → cmd → cmd
| seq : cmd → cmd → cmd              (* Notation: _ ; _ *)
| call : fun_id → cmd.
```

命令文 sample_n x n は 0 から $n-1$ の自然数の中から一様な確率でランダムに選んだ数を x の値とする，というものである．これを x <-$-n と表す．命令文 sample_b x p はブール値 (自然数の 0 と 1) のサンプリングであり，確率 p で 0 を x の値とする，というものである．これを x <-p-n と表す．insert k v find_value x e はオラクル操作のための命令文で，前者はエントリ (k,v) をオラクルに追加するもの，後者は e がオラクルに既に登録されているかどうかを調べ，その結果を変数 x の値とするものである．seq c1 c2 は c1; c2 という表記を用いる．ifte b c1 c2 は条件分岐を表し，b が分岐条件，c1 が真の場合の分岐，c2 が偽の場合の分岐である．

例：PRF ゲーム　以下，具体的なプログラムの例を簡単に説明する．

```
Definition PRF' bad (A:nat→nat) i :=
 x <- int_e (A i) ;
 y <-$- n ;
 find_value z (var_e y) ;
 ifte (var_e z)
          (bad <- int_e 1) skip ;
 insert (var_e x) (var_e y).
```

このゲームは，オラクルを利用して (疑似) ランダム関数を構成している．一行目は変数割り当て．ここで x の値を入力と考える．二行目はサイズ n の集合からのランダムサンプリング．サンプリングした値を入力に対する出力 y の値としている．三行目は，サンプリングした値が既にオラクルに登録されいるかを調べている．四行目では，三行目の結果が真ならば bad が 1 になる．(偽ならば何もしない．) 五行目では，入力 x と出力 y のペアを (x, y) をオラクルに登録する．

操作的意味論　操作的意味論は "exec" という，1) 関数環境，2) 始状態，3) プログラム，4) 終状態の 4 項関係として帰納的に定義され，prg ||- st -- c --> end と表記される．2.1 節で例として示したブール値サンプリングについては，意味論は次のようになる．

```
| exec_sample_b : ∀ x p st, 0 < p < 1 →
  prg ||- st -- x <-b- p -->
                        fork ((p, update x 1)::(1-p, update x 0)::nil) st
```

3　ゲームの手法の基本補題

本節では，基本補題 (Fundamental lemma)，あるいは Difference lemma と呼ばれる補題について説明する．基本補題は，1 節で述べた 3 種類のゲーム変換ステップのうちの一つ，失敗事象に基づくステップに対し，確率変化の上限を求めるために利用されるものである．

確率的証明　基本補題の主張と証明は以下のとおりである.

Lemma. 四つの確率事象 A, B, F_1, F_2 について, $Pr(A \wedge \overline{F_1}) = Pr(B \wedge \overline{F_2})$ と $Pr(F_1) = Pr(F_2)$が成立するならば, $|Pr(A) - Pr(B)| \leq Pr(F_1)$ が成り立つ.

Proof.

$$
\begin{aligned}
&|Pr(A) - Pr(B)| \\
&= |Pr(A \wedge F_1) + Pr(A \wedge \overline{F_1}) - Pr(B \wedge F_2) - Pr(B \wedge \overline{F_2})| \\
&= |Pr(A \wedge F_1) - Pr(B \wedge F_2)| \\
&\leq Pr(F_1)
\end{aligned}
$$

二つ目の等号は条件 $Pr(A \wedge \overline{F_1}) = Pr(B \wedge \overline{F_2})$ による. 不等号は, $Pr(A \wedge F_1)$ と $Pr(B \wedge F_2)$ が共に 0 と $Pr(F_1)(= Pr(F_2))$ の 間の実数値であることから示せる. □

[10] では, 前節での確率の定義の上でこの補題を証明している. Coq 上では本補題は以下のように記述される.

```
Lemma abstract_fundamental_lemma : ∀ d1 d2 e f r, 0 ≤ r →
  sum d1 = sum d2 → coeff_pos d1 → coeff_pos d2 →
    Pr f d1 = Pr f d2 = r → Pr (e ∩ f̄) d1 = Pr (e ∩ f̄) d2 →
      Rabs(Pr e d1 - Pr e d2) ≤ r.
```

ゲームについての基本補題　基本補題を適用してゲームの書き換えに伴う攻撃者の成功確率変化の上限を計算するするためには, ゲームの構成について以下のような準備をしておく.

まず両方のゲームに失敗事象に対応したフラグ変数 bad を導入する. ここで, bad の値はゲーム開始時点では 0 としておき, 失敗事象が起きたら bad の値を 1 に変更する. さらに, 次の条件が満たされるようにゲームを構成する : 失敗事象が起きない場合 (bad の値が 1 に変らない場合) は, 両ゲームは全く同一の振舞いをする. つまり, プログラムとしてのゲームについて

1. 失敗事象に依存した分岐点までは全く同一である
2. bad が 0 のまま (失敗事象が起きなかった場合) の分岐ブランチは全く同一である

という条件が成立するということである. 例えば, 次の二つのゲームは上の条件を満たす形になっている. ただし, c, c1, c2, c2' はそれぞれ command である.

```
prg ||- _ -- c;
      ifte b
      c1
      (bad <- int_e 1; c2) --> _
```

```
prg ||- _ -- c;
      ifte b
      c1
      (bad <- int_e 1; c2') --> _
```

上記の条件を満たすような二つのゲーム, ゲーム 1 とゲーム 2 について, 攻撃者が攻撃に成功するという事象をそれぞれ S_1, S_2 とし, bad が 1 に変更されるという

事象をそれぞれ F_1, F_2 とすると，上の条件 1 は $Pr(F_1) = Pr(F_2)$ が成り立つこと，条件 2 は $Pr(S_1 \wedge \overline{F_1}) = Pr(S_2 \wedge \overline{F_2})$ が成り立つことに相当する．よって，基本補題を適応することができ，$|Pr(S_1) - Pr(S_2)| \leq Pr(F_1)$ が成り立つ．

よって，以上のような状況において，失敗事象 F_1 の生起確率が無視できることが示せれば，ゲーム変換に伴なう成功確率の変化も無視できることが分かる．[10] では，ゲームの有限回ループについて，ゲームが上記の条件を満たす場合には基本補題が適用でき，成功確率の差についての上限が求められることを証明している．

4 PRP/PRF Switching Lemma

これまでに解説した枠組みを利用した証明の例として，[10] では PRP/PRF Switching Lemma(以下 PRP/PRF 補題とする) と呼ばれる補題の証明を示している．

PRP/PRF 補題は，暗号理論におけるブロック暗号に関する安全性証明の中でよく用いられる補題である．本来，ブロック暗号は疑似ランダム置換としてふるまうべきものであるが，この仮定を少し緩めて疑似ランダム関数として扱うと証明が容易になる．PRP/PRF 補題は，任意の確率事象について，この仮定の変更に伴う生起確率の変化に上限を与えるものである．この補題については誤った証明が複数発表された経緯がある [5]．[10] では，ゲームの手法の形式化に基いた，厳密に正確な証明を与えている．

この証明では二つのゲームに着目する．一つは，2.2 節で示した PRF ゲームである．もう一つは次に示す PRP ゲームである．

```
Definition PRP' bad (A:nat→nat) i :=
 x <- int_e (A i) ;
 y <-$- n ;
 find_value z (var_e y) ;
 ifte (var_e z)
          (bad <- int_e 1; resample) skip ;
 insert (var_e x) (var_e y).
```

このゲームでは，PRF ゲームの時と同様に，与えられた入力 x に対して，値 y をサンプリングし，その値が既にオラクルに登録されているかを調べている．その結果が真ならば，bad の値を 1 に変更した上で，命令文 resample を実行する．ここで，resample は，オラクルに登録されていない値が y の値となるまでサンプリングを繰り返し，その結果を y の値とすることを表す任意のコードである．(ただし，bad の値は変更しないものとする．) つまり，このゲームは，異なる入力に対しては出力値の衝突することがないように，(衝突した場合には)y を選びなおしながら関数を構成するゲームである．定義域が値域と同一であれば，このように構成された関数は置換 (permutation) となる．

このように定義された PRP ゲームと PRF ゲームは，3 の条件を満たしていることは容易に観察できる．よって，両者を有限回繰り返して実行する形のゲームについて基本補題を適用することができる．よって，任意の確率事象 e について，PRP ゲームの場合の生起確率 Pr e d1 と PRF ゲームの場合の生起確率 Pr e d2 は，サンプリ

ングの結果衝突が起る確率 `Pr (sets bad 1) d1` で抑えられることが証明できる．これまでが証明の前半となる．

証明の後半は，確率 `Pr (sets bad 1) d1` を具体的に評価すればよい．ループの回数を q，ランダムサンプリングの値域のサイズを n とすると，この確率は正規化された値で $\frac{q(q-1)}{2n}$ で抑えられる．[10] では，前節までに概説した構成を使って，以上の証明を Coq 上で行なっている．

5 まとめ

本稿では，ゲームの手法による安全性証明について，最近の研究動向について概説した．特に，ゲームの手法にプログラミング言語理論の手法を適用する最近の研究 [10] について紹介した．類似の研究としては，同じく Coq を用いて ElGamal 暗号スキームの安全性を証明した [9] や，プロセス計算に基づく形式化を使って独自のツールを構成した [8] などがある．

参考文献

[1] The LogiCal Project, INRIA. The Coq proof assistant: `http://coq.inria.fr`.

[2] Mihir Bellare and Phillip Rogaway. Random Oracle are Practical: A Paradigm for Designing Efficient Protocols. In *1st ACM Conference on Computer and Communications Security (CCS 1993)*, p. 62–73. ACM Press.

[3] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures—How to Sign with RSA and Rabin. In *Advances in Cryptology (EuroCrypt 1996)*, volume 1070 of *Lecture Notes in Computer Science*, p. 399–416. Springer.

[4] Victor Shoup. Sequence of Games: A Tool for Taming Complexity in Security Proofs. Manuscript. Available at `http://www.shoup.net/papers/games.pdf`. 2004. Revised 2006.

[5] Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. In *Advances in Cryptology (EuroCrypt 2006)*, volume 4004 of *Lecture Notes in Computer Science*, p. 409–426. Springer. Extended version: Cryptology ePrint Archive: Report 2004/331.

[6] David Pointcheval. Provable Security for Public Key Schemes. In *Contemporary Cryptology, Advanced Courses in Mathematics CRM Barcelona*, p. 133–189. Birkhäuser Publishers, 2005.

[7] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive: Report 2005/181.

[8] Bruno Blanchet and David Pointcheval. Automated Security Proofs with Sequences of Games. In *26th Annual International Cryptology Conference (CRYPTO 2006)*, volume 4117 of *Lecture Notes in Computer Science*, p. 537-554. Springer. Extended version: Cryptology ePrint Archive: Report 2006/069.

[9] David Nowak. A Framework for Game-Based Security Proofs. Cryptology ePrint Archive: Report 2007/199.

[10] Reynald Affeldt, Miki Tanaka, and Nicolas Marti. Formal Proof of Provable Security by Game-playing in a Proof Assistant. In *1st International Conference on Provable Security (ProvSec 2007)*, volume 4784 *Lecture Notes in Computer Science*, p. 151–168. Springer.