暗号プロトコルの安全性定義と結合定理

有田正剛

平成 19 年 6 月 22 日

目 次

1 はじめに

暗号技術を産んだ「数学的土壌」とは、単にオイラー関数や楕円曲線などの数学的プリミティブの寄せ集めではない。対象の本質を形式的に定義し、主張するところを明確にステートメントとして表し、そしてそれを証明する、という数学的な方法論は暗号の安全性を考察するためにもとても有効である。

本解説では、暗号プロトコルの安全性という、一見捉えどころのなさそうな課題が、数学的な方法論の下で、どのように定義され、それに対しどのようなステートメントが主張され、そして証明されているのか、見たい。残念ながら網羅的な展開はできない。筆者の好みが反映して選択された、ある断面を紹介するのみである。[1]

まずこの節では、暗号プロトコルとは何か、それが安全とはどういうことか、直感的に考察する。

1.1 暗号プロトコルとは？

暗号の教科書や解説書では、まず、図 1(と似たような図) をよく目にする。Alice と Bob が通信を行う。その通信を敵である Eve が傍受したり、改ざんしたりする。それを防ぐために暗号を使う、というシナリオである。一見問題なく妥当な図だが、この背景には以下のような設定が隠れている。Alice と Bob の二人はお互いを信頼して、何か共通の利益を達成しようとしている。そのような二人を邪魔しようとする敵が外部の世界にいる。

このような古典的状況の下では、暗号技術の保護対象はデータ通信で十分である。Alice と Bob はお互いを信頼しているので、それぞれのもつ秘密情報を暗号を用いて丸投げすれば、あとの計算は Alice と Bob それぞれが独立に行えばよい。

しかし、現代的状況は、むしろ、図 2 のようである。敵は外部ではなく、プロトコル内部にいる。共通の利益のためではなく、各々の利益を求めるプロトコル参加者は、お互いに味方と言うより敵である。外部には、むしろ、信頼できる第 3 者である TTP(Trusted Third Party) を設け、トラブル発生時に調停してくれることを期待する。このような状況の下では、暗号の仕事は古典的状況のときよりもずっと大変である。TTP を設けているとはいっても

[1]ただし、本解説は、いわゆる「証明可能安全性」の紹介ではありません。証明可能安全性では、ランダムオラクルモデルなどのヒューリスティックなモデルに頼ってでも、プロトコルの現実的な効率を落とさずに証明をつけることが重視されます。本解説で紹介するのは、現実的な効率はさておき、しかしヒューリスティックには頼らずに、暗号プロトコルの安全性について理論的な展開を追う、という立場です。

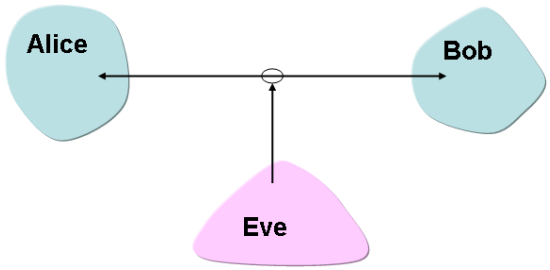

図 1: 暗号通信の古典的モデル

完全に信頼するわけにもいかない。今や相手は Alice ではなく Eve だから、たとえ暗号を用いたとしても、Bob は自分の情報を単純にまる投げできない。Bob は秘密情報にコミットしながら、それでも秘密を隠しつつ、用心深く、ことを進める必要がある。

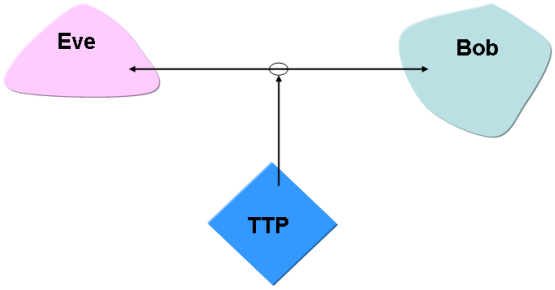

図 2: 暗号通信の現代的状況

暗号プロトコルとは、上にみたようなプロトコル内部に敵を抱えた現代的な状況においても、安全にプロトコルを実行するための技術である。与えられたプロトコルを、それに対応する完璧な TTP を「仮想的に」作り上げることによって、安全なプロトコルに変換する技術である。完璧な TTP を実際に作るのではなく、それが存在するのと変わらない状態をプロトコルとして実現しよう、ということである。このようにして変換された安全なプロトコルも暗号プロトコルと呼ばれる。

1.2 暗号プロトコルが安全とは？

このように、「暗号プロトコル=仮想的な TTP の構成」とするならば、暗号プロトコルの安全性を以下のように捉えるのは自然なことである。

今、(2 者間の) プロトコル Π が与えられているとする。プロトコル Π が安全とはどういうことか、考えたい。まず、現実の世界で Eve と Bob が Π を

実行していると思う (図 3 の左)。Eve は敵を表し、Bob は正直な参加者を表す。Eve と Bob はそれぞれ自身の入力情報を用いて (少なくとも見かけは) 協調してプロトコル Π を実行し、いくつかのメッセージを交換し、その結果、それぞれ何らかの情報を得る。それら Eve と Bob が Π の実行によって得た情報を REAL で表す。

一方、機能 $\mathcal{F}_\Pi$ が存在する仮想的な理想世界を考える (図 3 の右)。ここで、機能 $\mathcal{F}_\Pi$ とは、プロトコル Π が果たすべき機能を完璧に実現してくれる、完全に信頼できる TTP である。機能 $\mathcal{F}_\Pi$ が存在する理想世界では、暗号プロトコルの仕事はたやすい。参加者である iEve と Bob は、それぞれ自身の入力情報を機能 $\mathcal{F}_\Pi$ に送ればよい。あとは、機能 $\mathcal{F}_\Pi$ が必要な計算を行って、iEve が受け取るべき結果は iEve に、Bob が受け取るべき結果は Bob に確実に渡してくれる。機能 $\mathcal{F}_\Pi$ は、iEve が受け取るべきでない結果を iEve に漏らしたりすることはないし、Bob に誤った情報を渡すこともない。このような理想世界で、iEve と Bob が得る情報を IDEAL と書く。

このとき、プロトコル Π が安全であるとは、任意の敵 Eve に対して、ある仮想的な敵 iEve が存在して、

$$\mathsf{REAL} \equiv_c \mathsf{IDEAL} \tag{1}$$

であることと定義される。ここで、「$\equiv_c$」は両辺が実質的には区別できないことを表す。式 (1) は、Bob の視点から見ると、一緒にプロトコル Π を実行している相手 Eve がたとえ悪意をもってどのように振舞おうとも、自分がプロトコル Π の実行によって得るものは、理想的な TTP である $\mathcal{F}_\Pi$ が与えてくれるものと変わらないことを意味する。また、Eve の立場にたつと、どのように画策して何か元来受け取るべき範囲を超えた情報を得ようとしても、所詮、得られる情報はある iEve が完全な $\mathcal{F}_\Pi$ から得るもの、すなわち元々正直に振舞って受け取ることのできる情報のみということを意味する。

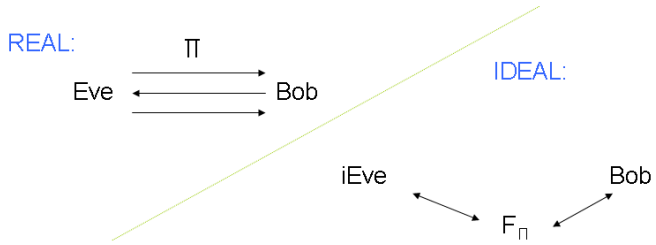

図 3: 暗号プロトコルの直感的な安全性定義

1.3 本章の目的

以下、本章では、上で直感的に説明した暗号プロトコルの安全性を、**対象を 2 者間プロトコルに限定して**、よりフォーマルな形で定義する (定義 1)。そのような定義の実践例として、コイン投げ機能を安全に実現するコイン投げプロトコルを紹介し、それが定義 1 のもとで安全であることを証明する (定理 1)。さらに定義 1 を満たす安全なプロトコルは (直列に) 組み合わされてもその安全性を失わないことをみる (定理 2)。これは、暗号プロトコルのモジュラリティを確保するために非常に重要な定理である。最後に、定理 2 の限界を説明し、それを超えることを目的に提案された、汎用的結合可能性 (Universally Composability) における安全性定義 (定義 3) を紹介し、定義 3 のもとで安全なプロトコルは並行に組み合わされてもその安全性を失わないことをみる (定理 3)。最後に、UC 安全性を満たすプロトコルの構成には困難が存在することをみる。

2 準備

まず、いくつか基礎的な概念を準備する。

2.1 確率変数

有限集合 $\Omega = \{x_1, \ldots, x_n\}$ とその上の 0 以上 1 以下の実数区間 $[0..1]$ に値をとる関数 $p : \Omega \to [0..1]$ で、総和が 1 となるもの、すなわち $\Sigma_{i=1}^{n} p(x_i) = 1$ となるものの組 $(\Omega, p(\cdot))$ を (有限) **確率空間**と呼ぶ。$p(\cdot)$ をその分布という。

確率空間 $(\Omega, p(\cdot))$ 上の**確率変数** X とは、Ω に値をとる変数 X であって、その値が $x(\in \Omega)$ となる確率が $p(x)$ となるもの、すなわち $\Pr[X = x] = p(x)$ となるもののことをいう。確率変数 X から (分布 $p(\cdot)$ にしたがって) サンプル x を 1 つ取り出すことを

$$x \leftarrow X$$

と書く。確率変数 X とそれが定義されている確率空間 (Ω, p) はしばしば同一視される。

n ビットのバイナリ列 (0 と 1 からなる文字列) 全体を $\{0,1\}^n$ と書く。$\{0,1\}^n$ と定数関数 $1/2^n$ の組 $U_n = (\{0,1\}^n, 1/2^n)$ は確率空間である。U_n からサンプル x を 1 つ取り出すこと、すなわち、n ビットのバイナリ列を 1 つ一様ランダムに選択することを

$$x \xleftarrow{\$} \{0,1\}^n$$

と書く。

2.2 確率的多項式時間アルゴリズム

A をアルゴリズムとする。以下、アルゴリズムの入力や出力は任意長の (それぞれ 1 つの) バイナリ列と見なす。アルゴリズム A が**多項式時間アルゴリズム**であるとは、A を実行するのに要するステップ数が常に、入力 x の長さ $|x|$ のある多項式で上から押さえられることをいう。すなわち、(A に依存して) ある多項式 $f(\cdot)$ があって、

$$\mathrm{Time}(A(x)) \leq f(|x|) \quad (\forall x \in \{0,1\}^*)$$

であることをいう。$\{0,1\}^*$ は有限任意長のすべてのバイナリ列の集合を表す。

アルゴリズム A が**確率的アルゴリズム**であるとは、A は内部で適当な長さの乱数 (すなわち、確率変数 U_n) を用いることができ、実行結果 (出力) がその乱数に依存して決まることをいう。確率的アルゴリズムでないアルゴリズムは決定的アルゴリズムと呼ばれる。

アルゴリズム A が**確率的多項式時間アルゴリズム**であるとは、A が多項式時間アルゴリズムであり、かつ確率的アルゴリズムであることをいう。

(計算量的な) 暗号理論では、

$$\text{確率的多項式時間アルゴリズム} = \text{効率的で実行可能なアルゴリズム}$$

とみなして議論することが通常である。パーティや敵は確率的多項式時間アルゴリズムでモデル化される。

2.3 negligible な関数

自然数全体の集合を $\mathbb{N}$ とかく。実数区間 $[0..1]$ に値をとる $\mathbb{N}$ 上の関数で、どんな多項式関数の逆数関数 (逆関数ではない) よりも漸近的に小さな関数は、**negligible** であると呼ばれる。すなわち、関数 $\epsilon : \mathbb{N} \to [0..1]$ が negligible であるとは、

$$\forall c > 0,\ \exists k,\ \forall n \geq k,\ \epsilon(n) < 1/n^c$$

であることである (どんな (大きな)c に対しても、ある k があって n が k 以上ならば、$\epsilon(n)$ は $1/n^c$ より小さくなる)。確率的多項式時間アルゴリズムにとって、negligible な関数は漸近的に 0 に等しい。

実数区間 $[0..1]$ に値をとる $\mathbb{N}$ 上の関数 κ に対し、$1-\kappa$ が negligible な関数であるとき、関数 κ は **overwhelming** であるといわれる。

2.4 一方向関数

順方向は効率的に計算できて、逆方向は決して効率的に計算できない関数を**一方向関数**と呼ぶ。すなわち、関数 $f : \{0,1\}^* \to \{0,1\}^*$ が一方向関数であるとは、

1. ある多項式時間アルゴリズム A があって、

$$A(x) = f(x) \quad (\forall x \in \{0,1\}^*)$$

2. 任意の確率的多項式時間アルゴリズム A' に対して、(A' に依存して) ある negligible な関数 ϵ があって、

$$\Pr[u \xleftarrow{\$} \{0,1\}^n; v \leftarrow A'(f(u)) \mid f(v) = f(u)] < \epsilon(n)$$

(n ビットのバイナリ列 u を一様ランダムに選択し、アルゴリズム A' を入力 $f(u)$ で実行して出力 v を得たとき、イベント $f(v) = f(u)$ が起きる確率」と読む。)

という、2 条件を満たすことである。

一方向関数は (計算量的な) 暗号理論の礎である。その候補として、RSA 関数や有限体上のべき乗関数、その他 NP 完全問題にもとづく関数などが知られている。(一方向関数の存在自体は証明されていない。)

f を一方向関数とすると、その定義より、$f(u)$ から u を求めることはできない。しかし、u のある部分情報を求めることは容易である場合が多い。とはいっても、$f(u)$ からどうしても求められない u の 1 ビットの部分情報が必ず存在することが知られている。これを f の hard-core predicate と呼ぶ。

すなわち、一方向関数 $f : \{0,1\}^* \to \{0,1\}^*$ と多項式時間で計算可能な predicate(出力が 1 ビットの関数) $b : \{0,1\}^* \to \{0,1\}$ に対して、b が f の **hard-core predicate** であるとは、任意の確率的多項式時間アルゴリズム A に対して、ある negligible な関数 ϵ が (A に依存して) 存在して、

$$\Pr[u \xleftarrow{\$} \{0,1\}^n \mid A(f(u)) = b(u)] < 1/2 + \epsilon(n)$$

となることをいう。

2.5 識別不可能性

$X = \{X_w\}_{w \in S}$ と $Y = \{Y_w\}_{w \in S}$ を同じパラメータ集合 S をもつ確率変数の族 (集まり) とする。どのような効率的なアルゴリズムにとっても、w(の長さ) を十分大きくとられると、X_w と Y_w が区別できないとき、X と Y は識別不可能であるといわれる。

すなわち、X と Y が**識別不可能 (indistinguishable)** であるとは、任意の確率的多項式時間アルゴリズム A に対し、ある negligible な関数 ϵ があって、任意の $w \in S \cap \{0,1\}^n$ に対し、

$$|\Pr[x \leftarrow X_w \mid A(x) = 1] - \Pr[y \leftarrow Y_w \mid A(y) = 1]| < \epsilon(n)$$

であることをいう。

以下、$X = \{X_w\}_{w\in S}$ と $Y = \{Y_w\}_{w\in S}$ が識別不可能であることを

$$X \equiv_c Y$$

と書く。単に、$X_w \equiv_c Y_w$ と書くことも多い。確率的多項式時間アルゴリズムにとって、識別不可能な X と Y は漸近的に同一である。

3 プロトコルの安全性

1 節でプロトコルの安全性を直感的にみた。ここでは、プロトコルの安全性をフォーマルに定義する。まずプロトコルの実行モデルを定める。

3.1 プロトコル実行モデル

(2 者間) プロトコル Π とは、お互いに通信テープを共有した、2 つの対話的な確率的多項式時間 Turing 機械[2]の組 (P_1, P_2) である。入力 x_1, x_2 がプロトコル $\Pi = (P_1, P_2)$ にセットされると (x_1 は P_1 の、x_2 は P_2 のローカルな入力)、(P_1, P_2) は動作を開始し、そのプログラムに従って、計算を行ったり、メッセージを交換したりして、出力 y_1, y_2 を出す (y_1 は P_1 の、y_2 は P_2 のローカルな出力)。

プロトコル $\Pi = (P_1, P_2)$ への攻撃は P_1 もしくは P_2 を敵 A に置き換えることで表す[3]。(P_1 も P_2 も敵となる場合は、守るべきものがないので考えない。) ここでは、P_1 を敵 A に置き換えたプロトコル (A, P_2) を考える (A が P_2 のときも同様)。敵 A は入力 x_1 を、P_2 は入力 x_2 を得てプロトコルを実行する。P_2 は一緒にプロトコルを実行している相手は P_1 と思い、自身のプログラムに正直に従って、A へのメッセージを計算したり、A から送られたメッセージを処理して、出力 y_2 を得る。敵 A は P_1 のプログラムには従わず、勝手な方法で計算したメッセージを P_2 に送ったり、P_2 から送られたメッセージを P_2 が期待していないような方法で処理して、出力 y_1 を得る。ただし、敵 A は見かけはプロトコルに従うとする。すなわち、P_2 にメッセージを送るべきところで送らなかったり、プロトコルの実行中、一方的にアボートすることはないとする。

このように、敵 A によって攻撃されたプロトコル (A, P_2)(または (P_1, A)) の出力 (y_1, y_2) を表す確率変数を $\mathsf{REAL}_{\Pi,A}(x_1, x_2)$ とし、さらに確率変数 $\mathsf{REAL}_{\Pi,A}(x_1, x_2)$ の、入力 (x_1, x_2) をパラメータとする族を $\mathsf{REAL}_{\Pi,A}$ とおく:

$$\mathsf{REAL}_{\Pi,A} = \{\mathsf{REAL}_{\Pi,A}(x_1, x_2)\}_{x_1,x_2} = \{y_1, y_2\}_{x_1,x_2}.$$

[2]「対話的な確率的多項式時間 Turing 機械」とは、ここでは、確率的多項式時間アルゴリズムのプログラムを読み込んで実行可能な状態になっている、通信テープすなわち通信インタフェースを備えた計算機のことと思っていただければ十分と思います。

[3]第 3 者的な敵は考察の範囲外におきます。

これら $\mathsf{REAL}_{\Pi,A}$ に含まれる、y_1, y_2 は、敵 A の悪意ある振る舞いにより、本来そうあるべきものとはまるで異なっている可能性がある。(P_1 に置き換わった) 敵 A の出力 y_1 には、P_2 の知らないうちに (P_2 だけが知るべき)x_2 や y_2 の部分情報が含まれているかも知れないし、P_2 の出力 y_2 は敵 A によってゆがめられているかも知れない。

3.2 安全性定義

このようなプロトコル実行モデルのもとで、プロトコル $\Pi = (P_1, P_2)$ が安全であるとはどういうことだろうか。それは、「プロトコル Π が理想的に安全である」ことと区別がつかないことと考える。では、プロトコル Π が理想的に安全であるとはどういうことか。

プロトコル Π が実現しようとしている機能を $\mathcal{F}$ とおく。すなわち、$\mathcal{F}$ は、それ自身、2 つの入力 (x_1, x_2) と 2 つの出力 (y_1, y_2) をもつ確率的 Turing 機械であって、正直な P_1, P_2 の入力 x_1, x_2 に対する出力 $P_1(x_1), P_2(x_2)$ を再現するものである:

$$\mathcal{F}(x_1, x_2) \equiv (P_1(x_1), P_2(x_2)).$$

今、機能 $\mathcal{F}$ を完全に公平公正に実行してくれる、セキュリティ上完璧なサードパーティ (これも $\mathcal{F}$ で表す) が存在するする。そのような世界ではプロトコル Π の安全な実行はたやすい。パーティ P_1, P_2 はそれぞれ自身の入力 x_1, x_2 を $\mathcal{F}$ に渡し、結果 y_1, y_2 をもらうだけである。敵 B が P_1 として登場しても、完全に信頼できる存在である $\mathcal{F}$ は、たとえ B が何をいっても、B にはそれが受け取るべき情報 $\mathcal{F}_1(x_1', x_2)$ しか渡さないし、P_2 にはそれが受け取るべき情報 $\mathcal{F}_2(x_1', x_2)$ をきちんと渡す。

以上のことを踏まえて、機能 $\mathcal{F}$ と任意の敵 B に対し、確率変数 $\mathsf{IDEAL}_{\mathcal{F},B}(x_1, x_2)$ を以下のように定義する:

$$\mathsf{IDEAL}_{\mathcal{F},B}(x_1, x_2) = \begin{cases} (B(x_1, \mathcal{F}_1(x_1', x_2)), \mathcal{F}_2(x_1', x_2)) & (B \text{ が } P_1 \text{ として振舞うとき}) \\ (\mathcal{F}_1(x_1, x_2'), B(x_2, \mathcal{F}_2(x_1, x_2'))) & (B \text{ が } P_2 \text{ として振舞うとき}) \end{cases}$$

ここで、$\mathcal{F}_i(\cdot,\cdot)$ は $\mathcal{F}(\cdot,\cdot)$ の (出力の) 第 i 成分を表す $(i = 1, 2)$。$\mathsf{REAL}_{\Pi,A}$ の場合と同様に、入力 (x_1, x_2) をパラメータとする、確率変数 $\mathsf{IDEAL}_{\mathcal{F},B}(x_1, x_2)$ の族を $\mathsf{IDEAL}_{\mathcal{F},B}$ と書く:

$$\mathsf{IDEAL}_{\mathcal{F},B} = \{\mathsf{IDEAL}_{\mathcal{F},B}(x_1, x_2)\}_{x_1,x_2}.$$

$\mathsf{IDEAL}_{\mathcal{F},B}$ は、プロトコル Π を実現する、セキュリティ上完璧なサードパーティ $\mathcal{F}$ が存在する世界で、敵 B が得る情報と正当なパーティが得る情報の組を表す。実際、上の定義において、$B(x_1, \mathcal{F}_1(x_1', x_2))$ は x_1 と $\mathcal{F}_1(x_1', x_2)$ だけから (B が) 効率的に計算できる情報を表す。これは、敵 B は、(P_1 として振舞うとき)$\mathcal{F}$ に入力 x_1' を渡して、結果 $\mathcal{F}_1(x_1', x_2)$ をもらうだけで、$\mathcal{F}$ の実

行する計算には介入できないことを示す。また、P_2 も $\mathcal{F}_2(x_1', x_2)$ を正しく受け取っている。(B が P_i として振舞うとき、B が $\mathcal{F}$ に対し、自身の入力 x_i を x_i' と偽ることは防ぎようがない。)

定義 1 *[プロトコルの安全性]* プロトコル Π が**安全である**とは、より正確には、プロトコル Π が機能 $\mathcal{F}$ を**安全に実現する**とは、Π に対する任意の敵 A に対して、$\mathcal{F}$ に対するある敵 B が存在して、

$$\mathsf{REAL}_{\Pi,A} \equiv_c \mathsf{IDEAL}_{\mathcal{F},B}$$

となることである。

すなわち、プロトコル Π が安全であるとは、どのような敵 A が、(例えば) パーティ P_1 の立場で、プロトコル Π をどのように攻撃したとしても、所詮、敵 A が獲得する情報 $A(x_1)$ は、自身の入力 x_1 と元々正直に振る舞って受け取ることのできる情報 $\mathcal{F}_1(x_1', x_2)$ から効率的に計算できる情報 $B(x_1, \mathcal{F}_1(x_1', x_2))$ に過ぎないということである。また、敵 A が P_2 にもたらす情報も $\mathcal{F}_2(x_1', x_2)$ なので、敵 A が正直に振舞う場合と本質的に同じである。このようなプロトコルであれば、敵 A にとってわざわざ攻撃するメリットはない。

4 コイン投げプロトコル

この節では、定義 1 を満たすプロトコルの例として、コイン投げ機能を安全に実現するコイン投げプロトコルをみる。

コイン投げ機能 $\mathcal{F}_{coin}$ とは、その名前の通り、1 ビットの乱数 σ を生成し、各パーティに与える機能である:

$$\mathcal{F}_{coin}(1^n, 1^n) = (\sigma, \sigma) \quad (\sigma \xleftarrow{\$} \{0,1\}).$$

(ここで、1^n は 1 を n 個ならべたものを表す。$\mathcal{F}_{coin}$ にセキュリティパラメータが n であることを伝える。) サッカーの試合で最初に主審がコインを振るようなものである。単純な機能だがプリミティブとして重要な機能であり、またプロトコルによって安全に実現することは自明でない。敵 A は、コイン投げの結果が自分に有利に偏るようにバイアスをかけてくるからである。

問題 1 以下の「コイン投げもどきプロトコル」はコイン投げ機能を安全には実現しないことを示せ。

コイン投げもどきプロトコル (P_1, P_2)

1° P_1 は *1* ビットの乱数 σ を生成し、σ を P_2 に送る。

2° P_2 は *1* ビットの乱数 σ' を生成し、σ' を P_1 に送る。

3° P_1 は $b = \sigma \oplus \sigma'$ を出力する。

4° P_2 は $b = \sigma \oplus \sigma'$ を出力する。

略解 P_2 として振舞う A が、上記の 2° で P_1 から受け取った σ をそのまま σ' として P_1 に送ると、P_1 の出力は *0* に固定されてしまう。□

4.1 コミットメントスキーム

コイン投げ機能 $\mathcal{F}_{coin}$ をプロトコルとして安全に実現するために、コミットメントスキームを用いる。コミットメントスキームは、送信者 S と受信者 R との間で、コミットフェーズとオープンフェーズの 2 段階で実行されるプロトコル (S, R) である:

1. コミットフェーズ:
   送信者 S は、入力 $\sigma(\in \{0,1\})$ を受け取ると、乱数 r を生成し、“コミットメント”$c = com(\sigma, r)$ を計算して、c を受信者 R に送る。

2. オープンフェーズ:
   送信者 S は受信者 R に σ と r を送信する。受信者 R は $com(\sigma, r)$ を計算し、それが (コミットフェーズで受け取った)c に等しければ σ を出力し、そうでなければ $\perp$ を出力する。

ここで、$\perp$ はプロトコルの実行が失敗したことを表す。また、$com(\cdot,\cdot)$ は (スキームが定める)2 入力の関数を表し、以下の条件を満たすものとする:

条件 1 (守秘性):
どのような確率的多項式時間アルゴリズムも、$c(= com(\sigma, r))$ を与えられて、それが 0 に対するコミットメントなのか 1 に対するコミットメントなのか分からない。すなわち、$com(0, r_0)$ と $com(1, r_1)$ は確率変数として識別不可能である。

条件 2 (束縛性):
どのような確率的多項式時間アルゴリズムも、negligible な確率の例外を除いて、$c = com(0, r_0) = com(1, r_1)$ となる c, r_0, r_1 を作ることはできない。

守秘性と束縛性は相反する性質である。守秘性を達成するには、$c = com(\sigma, r)$ には表面上なるべく σ の情報を残さないようにしたい。しかし、そのようにすると、c を σ 以外の σ' にもオープンできてしまいそうである。幸いなことに、一方向置換 (1 対 1 の一方向関数) を用いると守秘性と束縛性を両立することができる。

問題 2 f を一方向置換、b をその *hardcore predicate* とすれば、

$$com(\sigma, r) = (f(r), b(r) \oplus \sigma)$$

は守秘性と束縛性をともに満たすことを示せ。

略解 *(*守秘性*)* *hardcore predicate* の性質から $f(r)$ をみても $b(r)$ が *0* なのか *1* なのか分からない。
*(*束縛性*)* f は置換なので、$f(r)$ から r が定まる。よって、$b(r) \oplus \sigma$ より σ が定まる。□

結局、コミットメントスキームでは以下のことが行われる。まず、コミットフェーズで、送信者は、ある値 b を選択し、選択した証拠をコミットメント c として、受信者に送る。受信者は c をみても、送信者が 0 を選択したのか、1 を選択したのかわからない (守秘性)。オープンフェーズで、送信者は実はその値は b でしたと明かす。このとき、送信者は b を偽ることはできない (束縛性)。このように、コミットメントスキームはそれ自体で意味をもつプロトコルではないが、他のプロトコルと組み合わせるととても役にたつ、最も重要な要素プロトコルの一つである。

4.2 コイン投げプロトコル

コミットメントスキームを用いれば安全なコイン投げプロトコル Π_{coin} を構成することができる:

コイン投げプロトコル $\Pi_{coin} = (P_1(1^n), P_2(1^n))$

1° P_1 は 1 ビットの $\sigma(\in \{0,1\})$ と n ビットの $s(\in \{0,1\}^n)$ を一様ランダムに生成し、$c = com(\sigma, s)$ を計算して、c を P_2 に送る。

2° $(c$ を受け取った$)P_2$ は 1 ビットの $\sigma'(\in \{0,1\})$ を一様ランダムに生成し、σ' を P_1 に送る。

3° $(\sigma'$ を受け取った$)P_1$ は P_2 に対し c をオープンする。すなわち、σ と s を P_2 に送る。加えて、$b = \sigma \oplus \sigma'$ を出力する。

4° P_2 は $c = com(\sigma, s)$ が成り立つかどうかチェックする。成立すれば $b = \sigma \oplus \sigma'$ を、しなければ $\perp$ を出力する。

P_1、P_2 がともに正直に振舞うとき、このコイン投げプロトコルがコイン投げ機能 $\mathcal{F}_{coin}$ を実現していることは明らかである。問題は P_1 または P_2 が敵 A である場合である。

まず、敵 A が P_2 として振舞う場合を考える。この、P_2 として振舞う A を $A(P_2)$ と表す。敵 $A(P_2)$ による攻撃手段として「覗き見」が考えられる。も

し、ステップ 1 で P_1 から送られた c から、そのコミット対象である σ を覗き見ることができれば、$A(P_2)$ は好きな値 $\hat{b}$ に対し、$\sigma' = \hat{b} \oplus \sigma$ とし、素知らぬ顔でステップ 2 でこの σ' を送れば、$A(P_2)$ はプロトコルの実行結果として、好きな値 $\hat{b}(= \sigma \oplus \sigma')$ を得ることができる。このとき、P_1 も実行結果として $\hat{b}$ を得ていること、また、この $A(P_2)$ がズルをしたプロトコル実行は、P_1 からは全く正常に見えていること (上の不正な σ' も $\{0,1\}$ 上一様分布していること) に注意する。しかし、$c = com(\sigma, s)$ は守秘性を持つので、このような $A(P_2)$ による「覗き見」攻撃は不可能である。

次に、敵 A が P_1 として振舞う場合を考える。$A(P_1)$ による攻撃手段として「後出し」が考えられる。もし、$A(P_1)$ が σ' を受け取った後で、(ステップ 1 ですでに送った)c を (ステップ 1 で選んだ)σ とは異なる値 $\hat{\sigma}$ にオープンすることができれば、今度は $A(P_1)$ がプロトコルの出力を支配できる。すなわち、$A(P_1)$ はステップ 3 で、好きな値 $\hat{b}$ に対し、$\hat{\sigma} = \hat{b} \oplus \sigma'$ とし、$c = com(\hat{\sigma}, \hat{r})$ となる $\hat{r}$ を計算し、$(\hat{\sigma}, \hat{r})$ を送る (後だし)。これで、$A(P_1)$ はプロトコルの実行結果として、好きな値 $\hat{b}$ を得ることができる。この場合もやはり、P_2 も実行結果として $\hat{b}$ を得ていて、プロトコル実行は P_2 からは正常に見えている。しかし、$c = com(\sigma, s)$ は束縛性を持つので、このような $A(P_1)$ による「後だし」攻撃も不可能である。

以上のように、コミットメントスキームの守秘性と束縛性がうまく機能することで、コイン投げプロトコルの安全性が達成されているように見える。しかし、上でみた「覗き見」や「後出し」以外に何か巧妙な攻撃方法がないといいきれるだろうか。

4.3 コイン投げプロトコルの安全性

ほかに何か巧妙な攻撃方法は、3.2 節の安全性定義のもとでは、存在しないといいきれる。すなわち、

定理 1 コイン投げプロトコル Π_{coin} はコイン投げ機能 $\mathcal{F}_{coin}$ を安全に実現する。

定理 1 を証明する前に、この種の、プロトコルの安全性証明で用いられる一般的な手法についてインフォーマルに説明する。対象とするプロトコルを $\Pi = (P_1, P_2)$ とし、それが実現しようとしている機能を $\mathcal{F}$ とする。Π が定義 1 のもとで安全であることをいうには、Π に対する任意の敵 A に対して、機能 $\mathcal{F}$ に対する (ideal な) ある敵 B があって、任意の入力 x_1, x_2 に対して、

$$\mathsf{REAL}_{\Pi,A}(x_1, x_2) \equiv_c \mathsf{IDEAL}_{\mathcal{F},B}(x_1, x_2) \tag{2}$$

であることを示せばよい。

A は P_1 として振舞うとする (P_2 として振舞う場合も同様である)。このとき、式 (2) は

$$(A(x_1), P_2(x_2)) \equiv_c (B(x_1, \mathcal{F}_1(x_1', x_2)), \mathcal{F}_2(x_1', x_2))$$

となる。これを満たす B は図 4 のような具合で、Π が安全ならば、構成される。B は A の出力 $A(x_1)$ と識別不可能な情報を生成するために、その内部で A(のコピー) を x_1 を入力として起動し、A に対して P_2 の振る舞いをシミュレートする。このとき、B の用いることのできる情報は、自身の入力 x_1 と、(x_1 から効率的に生成される情報)x_1' を尋ねることで $\mathcal{F}$ からもたらされる $\mathcal{F}_1(x_1{}', x_2)$ だけである。x_2 は知らないことに注意する。もし P_2 のシミュレーションに成功すれば、B の内部の A は REAL のときと同様に振舞い、REAL のときと識別不可能な出力 $\tilde{A}(x_1)$ をもたらすはずである。B はそれを自身の出力 $B(x_1, \mathcal{F}_1(x_1', x_2))$ としてそのまま出力すればよい。ただし、この $B(x_1, \mathcal{F}_1(x_1', x_2))(= \tilde{A}(x_1))$ は B の知ることのできない、$\mathcal{F}_2(x_1', x_2)$ と対にされて、本当の $(A(x_1), P(x_2))$ と比較されることになる。

証明の鍵は、$\mathcal{F}$ からもたらされる情報だけを用いて、A に対して正直な P_2 を、その見えない出力と辻褄のあう形で、演じることができるか、という点である。

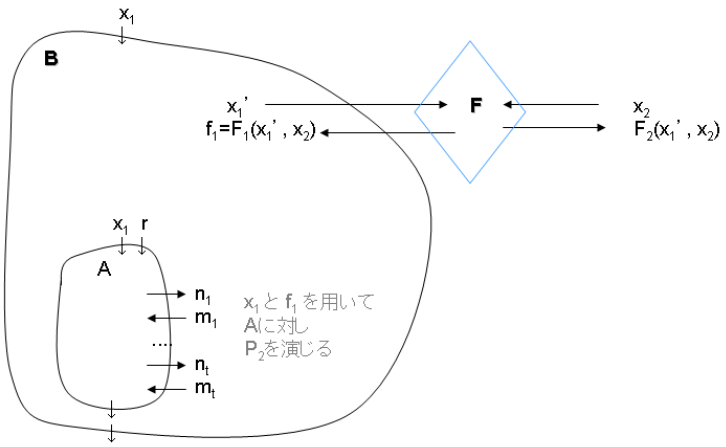

図 4: 安全性証明で用いられる一般的な手法:シミュレータ B の構成

定理 1 の証明をはじめる。

証明 (定理 1) コイン投げプロトコル $\Pi_{coin} = (P_1, P_2)$ に対する任意の敵を A とする。以下では、敵 A は確率的ではなく、決定的に振舞うとしてよい *(*任意のランダムテープを *fix* したと思う*)*。このとき、機能 $\mathcal{F}_{coin}$ に対するある

敵 B があって、定義 *1* の条件を満たすこと、すなわち、

$$\mathsf{REAL}_{\Pi_{coin},A}(1^n,1^n) \equiv_c \mathsf{IDEAL}_{\mathcal{F}_{coin},B}(1^n,1^n) \tag{3}$$

であることを示したい。

まず、A が P_2 として振舞うときを扱う。

$\mathsf{REAL}_{\Pi_{coin},A}(1^n,1^n)$ をみる。P_1 は正直にプロトコルにしたがって、$\{0,1\}$ から σ を $\{0,1\}^n$ から s をそれぞれ一様ランダムに選択し、$c = com(\sigma,s)$ を計算し、A に送る。A は何らかの計算を行い、σ' を P_1 に送る。P_1 は (σ,s) を A に送り、加えて $b = \sigma \oplus \sigma'$ を出力する。A はこの実行によって得た情報 c,σ,s を用いて何らかの計算を行いある情報を得る。この情報を $A(1^n,c,(\sigma,s))$ と書く。このとき、$\mathsf{REAL}_{\Pi_{coin},A}(1^n,1^n)$ は

$$\mathsf{REAL}_{\Pi_{coin},A}(1^n,1^n) = (b, A(1^n,c,(\sigma,s))$$

となる。

この A に対し、機能 $\mathcal{F}_{coin}$ に対する敵 B を以下のように構成する。B も P_2 の立場で振舞う。B はまず、入力 1^n を $\mathcal{F}_{coin}$ に送る。$\mathcal{F}_{coin}$ は、その仕様に従って、*(*P_1 からも 1^n を得て*)1* ビット乱数 b を生成し、*(*P_1 と*)B* に送る。b を得た B は以下を高々 n 回繰り返す*:*

> $\sigma \xleftarrow{\$} \{0,1\}$
> $s \xleftarrow{\$} \{0,1\}^n$
> $c = com(\sigma,s)$
> $A(1^n)$ を B の内部で起動し、c を A に送る。
> A から σ' を得る。
> もし、$\sigma \oplus \sigma' = b$ ならば
> 　　(σ,s) を A に送り、$A(1^n,c,(\sigma,s))$ を出力して終了。
> そうでないならば *(*失敗*)*、繰り返しの最初に戻る。

上で、もしも n 回とも失敗すれば、B は $\perp$ を出力して終了する。以上が B の記述である。*(*上の繰り返しにおいて、B も *(*A も*)* 失敗からは何も学習しない設定であることに注意する。すなわち、失敗しても、そのことはすっかり忘れて、新たにランダムに σ と s を選択する。失敗した σ や s を避けようとするような工夫は何もしていない。*)*

今、上の B の出力が $\perp$ ではなかったとする。このとき、

$$\mathsf{IDEAL}_{\mathcal{F}_{coin},B}(1^n,1^n) = (b, A(1^n,c,(\sigma,s))$$

であり、この b,c,σ,s は完全に $\mathsf{REAL}_{\Pi,A}(1^n,1^n)$ 中のそれと同じ分布である。実際、(b,σ,s) は $\mathsf{REAL},\mathsf{IDEAL}$ ともに、$\sigma \oplus A(1^n,com(\sigma,s)) = b$ を満たす

範囲で一様ランダムである *(*$A(1^n, com(\sigma, s))$ は σ' を表す*)*。以上より、*(A* が P_2 のときに*)* 式 *(3)* を示すには、あとは、上の B が $\perp$ を出力する、すなわち上の繰り返しにおいて B が n 回とも全て失敗する確率が *negligible* であることをいえばよい。ところが、A が B から受け取る $c = com(\sigma, s)$ には com の守秘性より *negligible* な例外を除いて、σ の情報は全く含まれない。よって、A が出力する σ' は σ と独立である。よって、σ が一様ランダムなことより、$\sigma \oplus \sigma'$ も一様ランダムであり、それが b と一致する確率は、n 回の繰り返し中、常に *1/2* である *(b* はこの繰り返しにおいては定数であることに注意*)*。よって、n 回の繰り返しを全て失敗する確率は $1/2^n$ となり、*negligible* である。

つぎに、A が P_1 として振舞うときを扱う。$\mathsf{REAL}_{\Pi_{coin},A}(1^n, 1^n)$ をみる。A は入力 1^n に対し、何らかの計算をして c を生成し、P_2 に送る。P_2 は正直にプロトコルにしたがって、$\{0,1\}$ から σ' をランダムに選択し、σ' を A に送る。σ' を受け取った A は何らかの計算を行って σ, s を生成し、これを P_2 に送る。P_2 は $c = com(\sigma, s)$ が成り立つことを確認し、$b = \sigma \oplus \sigma'$ を出力する。このとき、$\mathsf{REAL}_{\Pi_{coin},A}(1^n, 1^n)$ は

$$\mathsf{REAL}_{\Pi_{coin},A}(1^n, 1^n) = (A(1^n, \sigma'), \sigma \oplus \sigma')$$

となる。ここで、σ は $com(\cdot, \cdot)$ の束縛性より、*negligible* な例外を除いて、c のみによって決定されていること、σ' は $\{0,1\}$ 上一様に分布していることに注意する。*(*本章では、敵 A はプロトコルの実行中、一方的にアボートすることはないとしている。よって、上の A も P_2 に対して最後に必ず $c = com(\sigma, s)$ が成り立つような σ, s を送ることになる。*)*

上の A に対し、機能 $\mathcal{F}_{coin}$ に対するある敵 B を以下のように構成する。B も P_1 として振舞う。B はまず、入力 1^n に対し、B の内部で *A(*のコピー*)* を 1^n を入力として起動する。A は REAL のときと同様に c を *(B* の内部でシュミレートされた P_2 に向けて*)* 送信する。これに対し、B は A に P_2 からのメッセージとして *0* を送り、A から (σ_0, s_0) を受け取る。B は $c = com(\sigma_0, s_0)$ が成り立つことを確認し、$\sigma = \sigma_0$ とおく。ここで、B は一旦 A を終了させ、もう一度最初から A を 1^n を入力として起動する。A は確定的に振舞うので、A はやはり先と全く同一の c を P_2 に向けて送信するが、これに対し、B は 1^n を $\mathcal{F}_{coin}$ に送る。$\mathcal{F}_{coin}$ は、その仕様に従って、*(*P_2 からも入力 1^n を得て*)1* ビット乱数 b を生成し、*(*P_2 と*)B* に b を送る。b を得た B は、$\sigma' = b \oplus \sigma$ とし、σ' を A に送る。これに対し、A は σ, s を P_2 に向けて送信し、何らかの情報を出力する。この情報は、A がこのプロトコル実行によって得た全ての情報である $1^n, \sigma'$ によって定まるので、$A(1^n, \sigma')$ と書く。B はこの $A(1^n, \sigma')$ を自身の出力として出力する。以上が B の記述である。上で、B は A を *2* 度実行している。最初の実行で、A が c を将来 σ にオープンすることを先読みしている。この先読みによって、*(B* にとって制御しようがない*)* 機能 $\mathcal{F}_{coin}$

の出力 b と辻褄のあう σ' を作っている。B はずるいことをしているようだが、A は自分が B の内部で *2* 度も実行されているとは、*(*前世の記憶は存在しないから*)* 知りようがない。

上の B に対し、

$$\mathsf{IDEAL}_{\mathcal{F}_{coin},B}(1^n, 1^n) = (A(1^n, \sigma'), b)$$

となる。B の定義より、$b = \sigma' \oplus \sigma$ である。*com* の束縛性より、*negligible* な例外を除いて、σ は c によって決定されていて、c は REAL の場合と全く同一なので、σ も REAL の場合と全く同一である。さらに、b の乱数性より、$\sigma' = b \oplus \sigma$ は $\{0,1\}$ 上一様に分布している。以上より、A が P_1 として振舞う場合にも、式 *(3)* が成り立つことが分かった。□

5 結合定理

前節で、例としてコイン投げ機能を取り上げ、それがコイン投げプロトコルによって定義 1 の意味で安全に実現されることをみた。しかし、現実の機能はコイン投げ機能より遥かに複雑で、コイン投げ機能のようなプリミティブな機能をいつくも組み合わせて構成される。それらについて、そのつどはじめから証明をつけていくのは大変である。

幸いなことに、暗号プロトコルの (定義 1 における) 安全性には結合性 (composability もしくは modularity) が成立する。すなわち、安全であることが証明された複数のプロトコルを直列に組み合わせて、より高次の機能を実現するプロトコルを作成すると、それは、いちいち証明を付け直さなくても、安全であることが自動的に保証される。その保証を与えるのが結合定理である。

5.1 ハイブリッドプロトコル

まず、ハイブリッドプロトコルを導入する。機能 $\mathcal{F}$ に対応したハイブリッドプロトコル $\Pi^{\mathcal{F}}$ とは、機能 $\mathcal{F}$(を実現する TTP) が与えられた世界を考え、その世界で $\mathcal{F}$ の助けを借りて実行されるプロトコルのことである (図 5)。ハイブリッドプロトコルを実行する対話的 Turing 機械 P_1, P_2 は、通常の入出力手段や通信手段のほかに、機能 $\mathcal{F}$ とやりとりするための特殊な通信手段 (オラクルテープ) を持つ。P_1, P_2 は通常のプロトコルの実行過程で、オラクルテープに必要な情報をセットし、それらに対し機能 $\mathcal{F}$ が計算してくれた結果をオラクルテープから読み出してその後の計算に用いることができる。機能 $\mathcal{F}$ の計算は常に正しく、どのような敵にも干渉されない。ただし、機能 $\mathcal{F}$ が計算してくれているからといって、その間に P_1, P_2 が別の計算を進めることは許されず、P_1, P_2 は機能 $\mathcal{F}$ の返答をジッと待っていなければならない。すなわち、機能 $\mathcal{F}$ の呼び出しは直列的である。

このようなハイブリッドプロトコル $\Pi^{\mathcal{F}}$ に対しても、通常のプロトコルと同様、敵 A とパーティが入力 (x_1, x_2) について $\mathcal{F}$ の助けを借りてハイブリッドプロトコル $\Pi^{\mathcal{F}}$ を実行した結果を $\mathsf{REAL}^{\mathcal{F}}_{\Pi^{\mathcal{F}},A}(x_1, x_2)$ とかく。

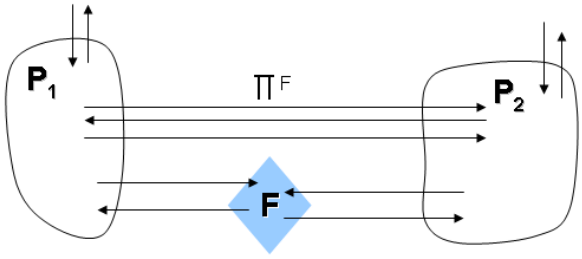

図 5: ハイブリッドプロトコル

ハイブリッドプロトコルによって、機能の間に帰着関係が定義される。

定義 2 *$\mathcal{F}$,$\mathcal{G}$ を機能とする。機能 $\mathcal{G}$ が機能 $\mathcal{F}$ に安全に帰着するとは、ある (機能 $\mathcal{F}$ に対応した) ハイブリッドプロトコル $\Pi^{\mathcal{F}}$ があって、それが $\mathcal{G}$ を安全に実現することである。すなわち、$\Pi^{\mathcal{F}}$ に対する任意の敵 A に対して、$\mathcal{G}$ に対するある敵 B が存在して、*

$$\mathsf{REAL}^{\mathcal{F}}_{\Pi^{\mathcal{F}},A} \equiv_c \mathsf{IDEAL}_{\mathcal{G},B}$$

であることである。

5.2 結合定理

$\Pi^{\mathcal{F}}$ を、機能 $\mathcal{F}$ を用いることのできるハイブリッドプロトコルとする。$\Pi^{\mathcal{F}}$ は機能 $\mathcal{G}$ を機能 $\mathcal{F}$ に帰着させる、すなわち $\Pi^{\mathcal{F}}$ はオラクル $\mathcal{F}$ のもとで $\mathcal{G}$ を安全に実現しているとする。一方、機能 $\mathcal{F}$ を安全に実現するプロトコル $\Pi_{\mathcal{F}}$ が与えられているとする。このとき、$\Pi^{\mathcal{F}}$ におけるオラクル $\mathcal{F}$ への呼び出しをプロトコル $\Pi_{\mathcal{F}}$ への呼び出しに置き換えることで、$\Pi^{\mathcal{F}}$ と $\Pi_{\mathcal{F}}$ を「結合した」プロトコル Π が得られる (図 6)。

このプロトコル Π が、機能 $\mathcal{F}$ の助けのもとで $\Pi^{\mathcal{F}}$ が実現しようとしていた機能 $\mathcal{G}$ を、自立的に実現していると期待するのは自然なことである。実際、そのことは以下の結合定理によって保証される。しかし、これは決して自明なことではないことに注意する。一般に、安全性はこの種の結合操作に対して脆弱であって、実際、各種プロトコルへの攻撃は攻撃者が複数のプロトコルを自分に都合のよい形で結合させることによって実現されることが多い。定義 1 の安全性定義があってこそ成り立つ結合性である。

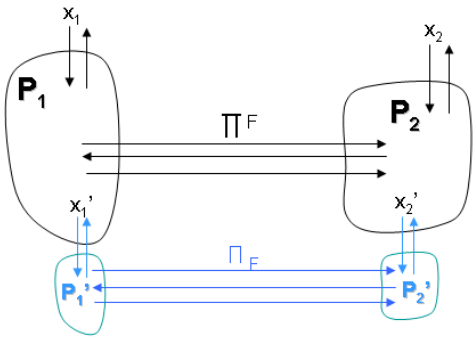

図 6: ハイブリッドプロトコル $\Pi^{\mathcal{F}}$ と $\mathcal{F}$ を安全に実現するプロトコル $\Pi_{\mathcal{F}}$ を結合したプロトコル Π

また、ハイブリッドプロトコルは定義より機能 $\mathcal{F}$ を直列に呼び出すので、上の $\Pi^{\mathcal{F}}$ と $\Pi_{\mathcal{F}}$ の結合もまた直列的でなければならない。すなわち、$\Pi^{\mathcal{F}}$ は $\Pi_{\mathcal{F}}$ を呼び出したならば、それが出力を返すまで、プロトコルの実行を進めず、ジッと待たなければならない。

定理 2 (結合定理) 機能 $\mathcal{G}$ が機能 $\mathcal{F}$ にハイブリッドプロトコル $\Pi^{\mathcal{F}}$ によって安全に帰着し、プロトコル $\Pi_{\mathcal{F}}$ が機能 $\mathcal{F}$ を安全に実現しているとする。このとき、$\Pi^{\mathcal{F}}$ と $\Pi_{\mathcal{F}}$ を直列に結合したプロトコル Π は機能 $\mathcal{G}$ を安全に実現する。

証明　プロトコル $\Pi = (P_1, P_2)$ に対する任意の敵を A とおく。一般性を失わず、A は P_1 として振舞うとしてよい。機能 $\mathcal{G}$ に対するある敵 B が存在して、入力 (x_1, x_2) に対して、

$$(A(x_1), P_2(x_2)) \equiv_c (B(x_1, \mathcal{G}_1(x_1'', x_2)), \mathcal{G}_2(x_1'', x_2)) \tag{4}$$

であることを示したい。

プロトコル Π の実行 $(A(x_1), P_2(x_2))$ は部分プロトコルとして $\Pi_{\mathcal{F}}$ を実行する。この $(A(x_1), P_2(x_2))$ 内の、任意の $\Pi_{\mathcal{F}}$ の実行を $(A'(x_1'), P_2'(x_2'))$ とおく *(*図 7 の左*)*。このとき、仮定より、$\Pi_{\mathcal{F}}$ は $\mathcal{F}$ を安全に実現しているので、$\mathcal{F}$ に対するある敵 B' があって、

$$(A'(x_1'), P_2'(x_2')) \equiv_c (B'(x_1', \mathcal{F}_1(x_1''', x_2')), \mathcal{F}_2(x_1''', x_2')) \tag{5}$$

となる。

B' を用いて、ハイブリッドプロトコル $\Pi^{\mathcal{F}}$ に対する敵 $\tilde{A}$ を以下のように構成する *(*図 7 の右*)*。敵 $\tilde{A}$ は、以下の点を除いて A と同一である*:*

- A において A' への呼び出し $A'(x'_1)$ が発生するときはいつでも、$\tilde{A}$ は $A'(x'_1)$ の代わりに、$B'(x'_1)$ を呼び出す。

この $\tilde{A}$ は部分プロトコルとして機能 $\mathcal{F}$ に対する敵 B' を用いるので、ハイブリッドプロトコル $\Pi^{\mathcal{F}}$ に対する敵となっていることに注意する。

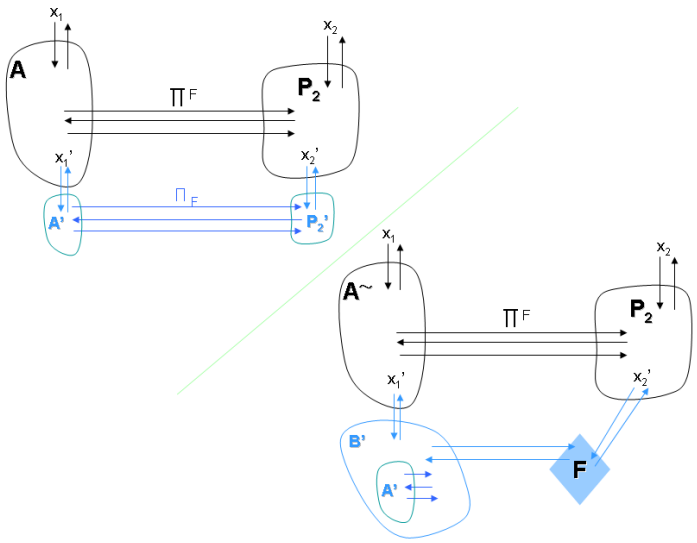

図 7: 敵 $\tilde{A}$ の構成

主張 ***1***

$$(A(x_1), P_2(x_2)) \equiv_c (\tilde{A}(x_1), P_2(x_2))$$

証明　$(A(x_1), P_2(x_2))$ の実行と $(\tilde{A}(x_1), P_2(x_2))$ の実行で異なる点は、$\tilde{A}$ の定義より、部分プロトコルとして、$(A'(x'_1), P'_2(x'_2))$ を呼び出すか、それとも $(B'(x'_1), P'_2(x'_2))$ を呼び出すかという点のみである。ところが、式 *(5)* より、これらの実行結果は識別不可能である。よって、$(A(x_1), P_2(x_2))$ と $(\tilde{A}(x_1), P_2(x_2))$ も識別できない。□

一方、仮定より、プロトコル $\Pi^{\mathcal{F}}$ は、機能 $\mathcal{F}$ のもとで、機能 $\mathcal{G}$ を安全に実現する。よって、上の $\tilde{A}$ に対して、機能 $\mathcal{G}$ に対する敵 B が存在して *(*図 *8* の右*)*、

$$(\tilde{A}(x_1), P_2(x_2)) \equiv_c (B(x_1, \mathcal{G}_1(x''_1, x_2)), \mathcal{G}_2(x''_1, x_2)) \qquad (6)$$

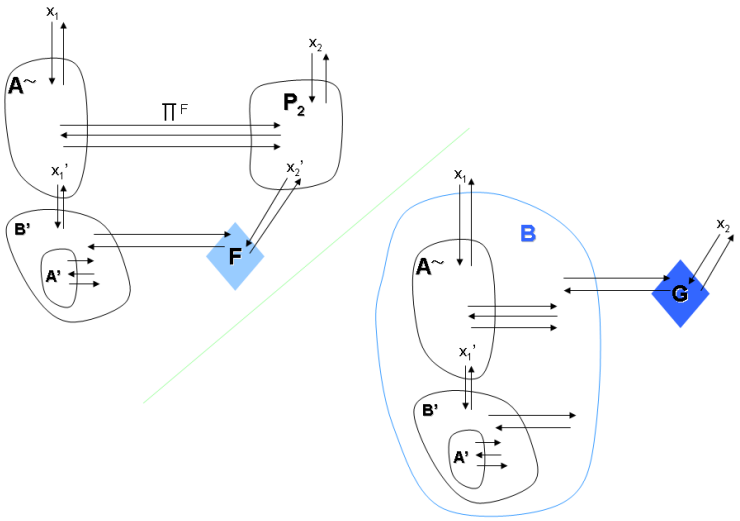

図 8: 敵 B の構成

となる。

主張 *1* と式 *(6)* より、式 *(4)* が得られる。□

定理 2 より、定義 1 のもとで安全なプロトコルどうしを直列に結合しても安全性は損なわれないことが分かった。すなわち、安全なプロトコルの構成において modularity が保証されたわけである。この modularity を活用して、与えられた任意の機能を定義 1 の意味で安全に実現するプロトコルが必ず存在することが証明されている。

定理 2 の証明において、プロトコル $\Pi^{\mathcal{F}}$ とプロトコル $\Pi_{\mathcal{F}}$ の結合が直列的であることが本質的に効いている。もしもこれら 2 つのプロトコルが並行に結合されると、敵 A' はその実行過程で A と対話することができるようになる。こうなると、上の敵 B' の性能は保証できず、主張 1 が成立しない。B' は A' に対して呼び出し元の A のシミュレートまではできないからである。

現実にはもちろん、プロトコルは直列的ではなく、並行に実行されているので、定理 2 に満足することはできない。しかし、並行結合に対してプロトコルの安全性を保証することは難しく、いまだ満足できる結果は得られていないが、これに対し、R. Canetti によって、汎用的結合可能性と呼ばれる、プロトコルの安全性定義の枠組みが提案されている。次節ではこれを見る。

6　汎用的結合可能性 (UC)

並行結合に対してプロトコルの安全性を保証するためには、定理 2 の直後で見たように、定義 1 の安全性のもとでは無理のようである。何か、より適切な安全性定義を求めなければならない。以下では、そのような安全性定義として汎用的結合可能性 (Universal Composability、以下 UC) を紹介する。UC そのものは一般のマルチパーティプロトコルを対象とするものだが、本解説では対象を 2 者間プロトコルに限定しているので、以下の記述でも 2 者間プロトコルに限定した場合の (static adversary model における)UC となる。

6.1　UC におけるプロトコル実行モデル

UC では、プロトコルの実行モデルに、プロトコル Π を実行する正直なパーティ P、敵 A に加えて、新たな存在である**環境** $\mathcal{Z}$ が導入される。環境 $\mathcal{Z}$ といっても、パーティ P や敵 A と同様、一つの対話的多項式時間 Turing 機械であることに変わりはない。

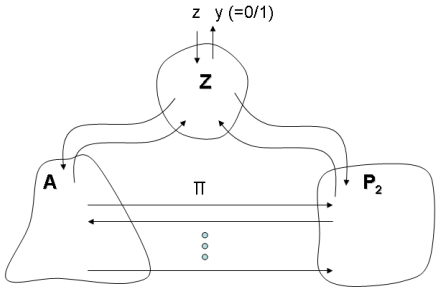

図 9: UC におけるプロトコル実行モデル

以下、UC におけるプロトコルの実行モデル (図 9) を述べる。Π をプロトコルまたは機能 $\mathcal{F}$ であるとする。まず、環境 $\mathcal{Z}$ が入力 z について起動される。環境 $\mathcal{Z}$ は Π をなんらかの入力のもと起動する。ただし、Π は敵 $\mathcal{A}$ と正直なパーティ P との間で実行され、3.1 節の実行モデルのときと同様、$\mathcal{A}$ は任意にプロトコルから逸脱できる (ただし、プロトコル・シンタックスには従う) とする。Π の実行過程で $\mathcal{Z}$ **と** $\mathcal{A}$ **はいつでも好きなときに対話する**ことができる。さらに、$\mathcal{Z}$ は Π の出力を受け取ることができる。$\mathcal{Z}$ は 1 ビット (0 or 1) の y を出力して終了する。この $\mathcal{Z}$ の出力 y を表す確率変数の族を $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ とかく:

$$\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} = \{y\}_z.$$

6.2 UC 安全性

Π がプロトコルのとき、$\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ は $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ と書かれ、Π が機能 $\mathcal{F}$ であるとき、$\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ は $\mathsf{IDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}$ と書かれる。環境 $\mathcal{Z}$ の目的は、$\mathcal{A}$ を実行部隊として用いて、自分が相手にしている世界が REAL なのか IDEAL なのかを判定することである (その判定結果が 1 ビットの出力 y となる)。ここで、プロトコルの実行において、$\mathcal{Z}$ は、$\mathcal{A}$ から間接的な報告は受けるものの、直接には $\Pi = (A, P_2)$ そのもののメッセージのやりとりを見ているわけではないことに注意する。$\mathcal{Z}$ にとって対象が REAL なのか IDEAL なのかは自明ではない。

定義 3 (UC 安全性) Π をプロトコルとし、それが実現しようとしている機能を $\mathcal{F}$ とおく。Π が $\mathcal{F}$ を **UC 安全に実現する**とは、Π に対する任意の敵 $\mathcal{A}$ に対し、$\mathcal{F}$ に対するある敵 $\mathcal{B}$ があって、任意の $\mathcal{Z}$ に対し、

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \equiv_c \mathsf{IDEAL}_{\mathcal{F},\mathcal{B},\mathcal{Z}}$$

となることである。

この UC 安全性において、本質的なのは、$\mathcal{Z}$ と $\mathcal{A}$ が対話しつつ、環境 $\mathcal{Z}$ とプロトコル $(\mathcal{A}, P)$ が同時並行に動作するという点である。これらが直列に動作するだけならば、定義 3 は判定者を明示的に表現しただけの定義 1 に過ぎず、両者の安全性は等価である。

UC 安全性においても、5 節の場合と同様に、ハイブリッドプロトコルを定義する。すなわち、機能 $\mathcal{F}$ に対応したハイブリッドプロトコル $\Pi^{\mathcal{F}}$ とは、機能 $\mathcal{F}$(を (超越的に) 実現する TTP) が与えられたものとし、プロトコル参加者である A や P が機能 $\mathcal{F}$ を用いることができる点を除けば、6.1 節の実行モデルと同一である。機能 $\mathcal{F}$ の計算は常に正しく、どのような敵にも干渉されない。環境 $\mathcal{Z}$ からはその存在は見えない。ただし、5 節の場合と異なり、機能 $\mathcal{F}$ が計算してくれる間に A や P は別の計算を進めてもよい。

環境 $\mathcal{Z}$ が、ハイブリッドプロトコル $\Pi^{\mathcal{F}}$ をオラクル $\mathcal{F}$、敵 $\mathcal{A}$ とともに実行したときの出力を (表す確率変数の族を) $\mathsf{EXEC}^{\mathcal{F}}_{\Pi^{\mathcal{F}},\mathcal{A},\mathcal{Z}}$ と書く。

定義 4 (UC 帰着) $\mathcal{F}$,$\mathcal{G}$ を機能とする。機能 $\mathcal{G}$ が機能 $\mathcal{F}$ に **UC 帰着**するとは、機能 $\mathcal{F}$ に対応するあるハイブリッドプロトコル $\Pi^{\mathcal{F}}$ があって、それが $\mathcal{G}$ を UC 安全に実現することである。すなわち、$\Pi^{\mathcal{F}}$ に対する任意の敵 A に対して、$\mathcal{G}$ に対するある敵 B が存在して、任意の環境 $\mathcal{Z}$ に対し、

$$\mathsf{EXEC}^{\mathcal{F}}_{\Pi^{\mathcal{F}},\mathcal{A},\mathcal{Z}} \equiv_c \mathsf{IDEAL}_{\mathcal{G},\mathcal{B},\mathcal{Z}}$$

であることである。

6.3 UC 結合定理

UC 安全性のもとでは、並行結合に対する安全性が成立する。

定理 3 (UC 結合定理) 機能 $\mathcal{G}$ が機能 $\mathcal{F}$ にハイブリッドプロトコル $\Pi^{\mathcal{F}}$ によって UC帰着し、プロトコル $\Pi_{\mathcal{F}}$ が機能 $\mathcal{F}$ を UC安全に実現しているとする。このとき、$\Pi^{\mathcal{F}}$ と $\Pi_{\mathcal{F}}$ を任意に結合したプロトコル Π は機能 $\mathcal{G}$ を安全に実現する。

$\Pi^{\mathcal{F}}$ と $\Pi_{\mathcal{F}}$ を任意に結合するとは、結合したプロトコル Π を実行するさいに、$\Pi_{\mathcal{F}}$ を呼び出している間に同時並行に、$\Pi^{\mathcal{F}}$ を実行してよいことを意味する。

定理 3 の証明の前に、定理 2 の証明が並行結合の場合にどこでつまずいたか思い出すと、それは、図 7 の敵 A と A' が実行過程で対話する部分を B' がシュミレートできない点だった。ところが、UC 安全性のもとでは、図 7 を図 10 に書き換えることができる。

図 10 では $\Pi^{\mathcal{F}} = (A, P_2)$ が環境 $\mathcal{Z}_1$ に取り込まれている。環境 $\mathcal{Z}_1$ が $\Pi^{\mathcal{F}} = (A, P_2)$ の $\Pi_{\mathcal{F}} = (A', P_2')$ への呼び出しをシュミレートしているわけである。このとき、$\Pi^{\mathcal{F}}$ の UC 安全性より、A' に対し、ある B' があって、(A', P_2') と (B', P_2') は、どのような環境からも区別できない。よって、とくに上の $\mathcal{Z}_1$ にも、ということは $\Pi^{\mathcal{F}} = (A, P_2)$ にも区別がつかない。つまり、B' は、A との対話まで込めて、A' をシュミレートできていることになり、定理 2 の証明中の主張 1 の正当性が復活するわけである。

証明 (定理 3) プロトコル $\Pi = (P_1, P_2)$ に対する任意の敵を A とおき、任意の環境を $\mathcal{Z}$ とおく。一般性を失わず、A は P_1 として振舞うとしてよい。機能 $\mathcal{G}$ に対するある敵 B が存在して、

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \equiv_c \mathsf{IDEAL}_{\mathcal{G},\mathcal{B},\mathcal{Z}} \tag{7}$$

であることを示したい。

$\mathcal{Z}$ によるプロトコル Π の実行は、$\Pi^{\mathcal{F}} = (A, P_2)$ の実行と $\Pi_{\mathcal{F}} = (A', P_2')$ の実行とからなる。この $\Pi^{\mathcal{F}} = (A, P_2)$ と環境 $\mathcal{Z}$ を合成して新たに環境 $\mathcal{Z}_1$ とおく *(図 10)*。すなわち、$\mathcal{Z}_1$ は自身の入力を $\mathcal{Z}$ に与え、$\mathcal{Z}$ に対して $\Pi^{\mathcal{F}} = (A, P_2)$ をシュミレートし、$\mathcal{Z}$ の出力を自身の出力とする。このとき、明らかに、

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \equiv \mathsf{REAL}_{\Pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}_1} \tag{8}$$

である。

仮定より、$\Pi_{\mathcal{F}}$ は $\mathcal{F}$ を UC安全に実現しているので、$\mathcal{F}$ に対するある敵 B' があって、$\mathcal{Z}_1$ に対して、

$$\mathsf{REAL}_{\Pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}_1} \equiv_c \mathsf{IDEAL}_{\mathcal{F},\mathcal{B}',\mathcal{Z}_1} \tag{9}$$

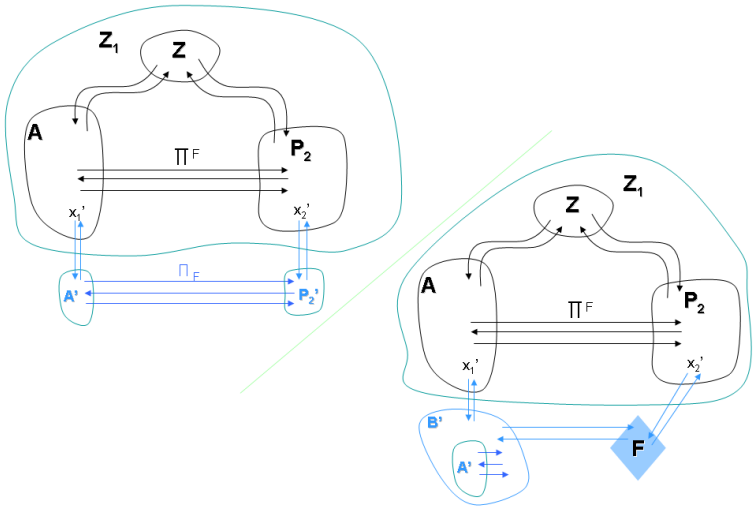

図 10: 敵 B' の構成

となる。

つぎに、式 *(9)* の右辺の $\mathcal{Z}_1$ を再び $\Pi^{\mathcal{F}} = (A, P_2)$ と環境 $\mathcal{Z}$ とに分解し、敵 A と B' を合成して $\tilde{A}$ とおく *(*図 *11)*。$\tilde{A}$ は機能 $\mathcal{F}$ に対応するハイブリッドプロトコル $\Pi^{\mathcal{F}}$ への敵に他ならず、

$$\mathsf{IDEAL}_{\mathcal{F},\mathcal{B}',\mathcal{Z}_1} \equiv \mathsf{EXEC}^{\mathcal{F}}_{\Pi^{\mathcal{F}},\tilde{A},\mathcal{Z}} \tag{10}$$

である。

仮定より、プロトコル $\Pi^{\mathcal{F}}$ は、オラクル $\mathcal{F}$ のもとで、機能 $\mathcal{G}$ を安全に実現するので、上の $\tilde{A}$ に対して、機能 $\mathcal{G}$ に対する敵 B が存在して、

$$\mathsf{EXEC}^{\mathcal{F}}_{\Pi^{\mathcal{F}},\tilde{A},\mathcal{Z}} \equiv_c \mathsf{IDEAL}_{\mathcal{G},\mathcal{B},\mathcal{Z}} \tag{11}$$

となる。

式 *(8),(9),(10),(11)* より、目標の式 *(7)* を得る。□

6.4 UC 安全なプロトコルの実現困難性

定理 3 より、プロトコルを UC 安全にしておけば (セキュリティの天敵である) 並行結合に対する安全性が保証されること、すなわち、それを他のプロトコルとどのように並行に組み合わせて実行しても安全性が損なわれないことが分かった。では、UC 安全なプロトコルはどのように作れるだろうか？そもそも、どのような機能に対しても、存在するといえるのだろうか？

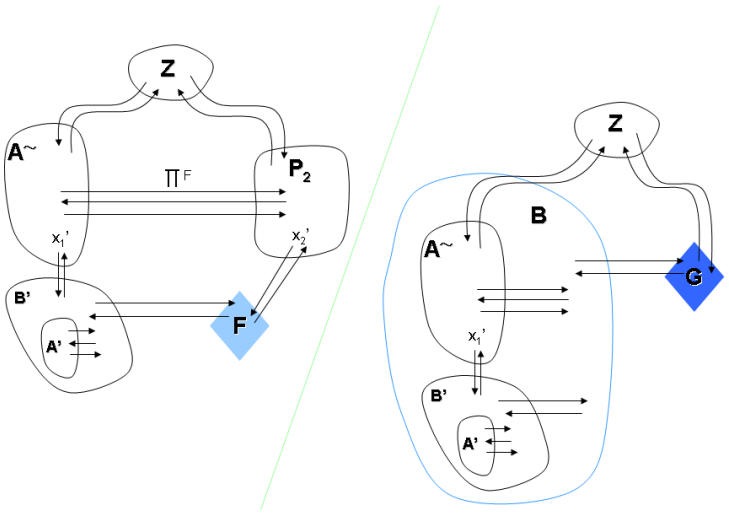

図 11: 敵 B の構成

実は、UC 安全なプロトコルの構成には問題がある。何らかの setup 仮定[4]を設けることなしには、与えられた任意の機能を実現するような UC 安全なプロトコルの存在は保証されない。

ここで、定理 1 の証明を振り返る。定理 1 は、コイン投げプロトコル Π_{coin} が定義 1 のもとで安全であることを主張するものだった。残念ながら、定理 1 の証明は UC 安全性のもとでは、妥当性を失う。定理 1 は、プロトコル Π_{coin} に対する任意の敵 A に対して、それと同等の効果を持つ、機能 $\mathcal{F}_{coin}$ に対する敵 B を構成することで証明された。A と同等の効果を発揮するために、B はその内部で A を完全にコントロールして A の能力を自身に都合のよい形で引き出している。より具体的には、例えば、B は、A が P_1 として振舞うとき、自身の内部で A を 2 度実行することにより、A が発したコミットメント c のコミット対象 σ を、A がオープンする以前に引き出している。(つまり、B は A を完全にコントロールできる立場を活用して、コミットメントスキームの秘匿性を破っているわけである。同様に、A が P_2 として振舞うときには、コミットメントスキームの束縛性を破っているとみなすことができる。) ところが、UC 安全性の枠組みでは、A はプロトコルの実行中に環境 Z と対話することができるので、B が A を 2 度も実行していると、環境 Z がそれに気づいてしまい、シミュレーションは失敗してしまう。このため、4.3 節の証明は UC 安全性のもとでは、妥当性を失うわけである。

[4] ランダムオラクルや Common Reference String など、プロトコルとして安全に実現しきれない機能を Trusted Third Party として仮定すること。暗号プロトコルの部分的な自己否定。

極端な場合、A の中身は空っぽでよい。A は、必要な計算はすべて環境 Z に頼み、結果だけをもらえばよい。このような場合、B は事実上、A に対してコントロールを完全に失っている。これで、B が例えば A の用いるコミットメントスキームの秘匿性を破ろうものなら、それはコミットメントスキームの安全性を否定することに他ならない。

以上の考察から、何らかの setup 仮定を設けることなしには、UC 安全なプロトコルは構成できないことが直感的に了解されると思う。定義の枠組みが素晴らしいだけに、実際にプロトコルを構成するには setup 仮定が必要というのは、残念な事態である。

7 文献紹介

最後に、暗号プロトコルの安全性に関する、主要な文献をいくつか紹介します。

暗号および暗号プロトコルの (UC 以前の) 安全性定義の枠組みを学ぶには、Goldreich の教科書が必読と思います。

- O. Goldreich, *Foundations of Cryptography: Volume 1 - Basic Tools*, Cambridge University Press, 2001.
- O. Goldreich, *Foundations of Cryptography: Volume 2 - Basic Applications*, Cambridge University Press, 2004.

Universal Composability については、Canetti の論文が IACR の eprint アーカイブにあります:

- R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, `http://eprint.iacr.org/2000/067`.

また、R. L. Rivest のホームページに Canetti の集中講義の講義録が UP されています。

- `http://theory.lcs.mit.edu/classes/6.897/spring04/materials.html`

UC 安全なプロトコルの構成に関しては

- Y. Lindell, *Composition of Secure Multi-Party Protocols*, LNCS 2815, Springer, 2003.

に詳しく書かれています。