[I486S]
暗号プロトコル理論

藤崎 英一郎

北陸先端科学技術大学院大学

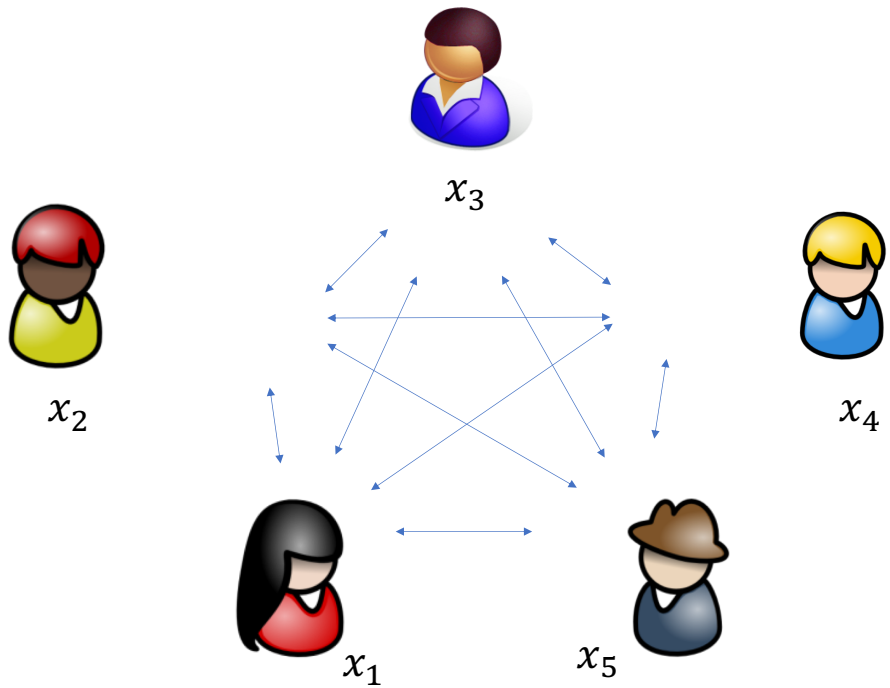
2020 年 6 月 30 日

本日の講義の内容

1. （復習）耐受動的攻撃安全なマルチパーティ計算（$t < n/2$ の場合）
2. 耐能動的攻撃者安全なマルチパーティ計算 ($t < n/3$) に向けて

マルチパーティ計算

- 参加者: $P_1, \ldots, P_n$.
- 各 P_i への秘密の入力: $x_i \in \{0,1\}^\lambda$
- 全参加者への入力（公開情報）: 関数 $F : \{0,1\}^{n\lambda} \to \{0,1\}^*$.
- 各 P_i への出力：$F(x_1, \ldots, x_n)$. より一般的には、参加者ごとに違う出力をすることも許す.
- ネットワーク: Pair-wise private & synchronized.



MPC の構成（準備）

準備

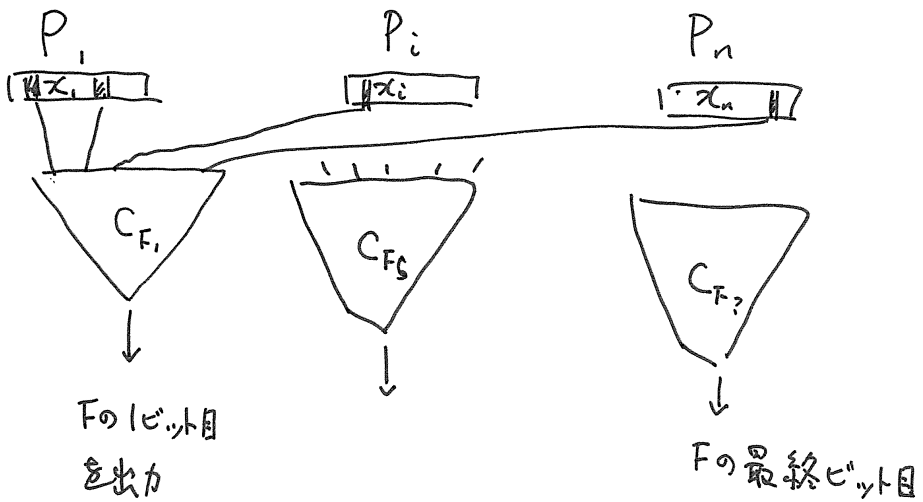
- $F : \{0,1\}^{n\lambda} \to \{0,1\}^*$: 多項式時間計算可能関数
- C_F: F を計算する多項式サイズの AND, OR, NOT で構成された論理回路
- K: $n < \#K$ な有限体
- C_F の論理演算子への入力 $b \in \{0,1\}$ を $b \in \{0,1\} \subset K$ と K の元とみなす。
- AND, OR, NOT の論理ゲートを K 上の演算に置き換える。
  - $b \wedge b' \iff b \cdot b' \in K$.
  - $b \vee b' \iff b + b - b \cdot b' \in K$.
  - $\neg b \iff 1 - b \in K$.

関数 F の回路

$F : \{0,1\}^{\lambda} \times \cdots \times \{0,1\}^{\lambda} \to \{0,1\}^{*}$ ← 任意の有限ビット列の集合

(n 個)

回路に分解



MPC の構成

- Input sharing phase: 各参加者 P_i は、x_i の各ビット b を $(t+1, n)$-Shamir SS を使い $[b]$ の状態にする。この結果、全ての参加者の秘密の全てのビットが（線型）秘密分散される。
- Computation phase: 各論理ゲートを秘密分散した形で実行
  - $[a] + [b] = [a + b]$
  - $1 - [a] = [1 - a]$
  - $[ab] = \sum_{i=1}^{n} \lambda_{i,n} [a_i b_i]$
- Output reconstruction phase: 出力ゲート結果が秘密分散された状態になっているので、各 P_i は自分の出力ゲート結果に関するシェアを他の参加者に公開（全員に送る）。各 P_i は、出力結果を復元する。

本日の講義の内容

1. （復習）耐受動的攻撃安全なマルチパーティ計算（$t < n/2$ の場合）

2. 耐能動的攻撃者安全なマルチパーティ計算 ($t < n/3$) に向けて

能動的攻撃者存在の元での Shamir 秘密分散

- 参加者数は n、攻撃者数は t で、$n > 3t$.
- $a \to [a]$: a の秘密分散（コミット）。[BGW88] を使うことで、プロトコルが受理された時は正しく $[a]$ が行われたことが保証される。
- $[a] \Rightarrow a$: $a = (f(\alpha_1), \ldots, f(\alpha_n))$ を復元するとき、攻撃者が自分のシェアを正しく提示しなくても Reed-Solomon 符号の (Welch/Berlekamp) 復号によりもとの多項式 $f(X)$ が復元できる。

$$\hat{f}(\alpha_i) = f(\alpha_i) + e_i \text{ s.t. } W_H(\boldsymbol{e}) \leq t \quad \Longrightarrow \quad \text{Reconstruct } f(X)$$

用語等の定義

- $[a]_i$:　秘密 a が、参加者間で正しく線型秘密分散されており、a を P_i が保持している状態。
- $[a]_\emptyset$: 誰も a を 1 人で保持していない状態
- $[a]$:　誰が a を 1 人で保持しているか興味ないので明記していない場合
- $P_i : a \to [a]_i$:　P_i が、秘密 a を正しく秘密分散する行為（コミットメント）。
- $P_i : [a]_i \Rightarrow a$:　P_i が、秘密 a を各参加者に配ったシェアを証拠として broadcast して開示する行為。
- $[a]_i \Rightarrow a$:　全参加者が a のシェアを broadcast して、全員が a を認識する行為。
- $P_i : a \leftarrow [a]_j$:　参加者が各自の $[a]_j$ のシェアを P_i に秘密通信で送り、P_i に a を教える行為。この行為により、$[a]_i$ という状態にもなることに注意。

不正者が全体の 1/3 未満であれば、BGW VSS の Distribution は、$[a]$ を正しくコミットし、Reconstruction は、開示を正しく行うこと常にできるので、上記の具体的な実現法の一つである。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a + b]$: 線形性から問題なし。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a + b]$: 線形性から問題なし。
- $\lambda, [a] \to [\lambda a]$: 線形性から問題なし。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a + b]$: 線形性から問題なし。
- $\lambda, [a] \to [\lambda a]$: 線形性から問題なし。
- $\lambda, [a] \to [a + \lambda]$: Shamir SS なら簡単。$[a + \lambda] = [a] + \lambda := (a_1 + \lambda, \ldots, a_n + \lambda)$.

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a + b]$: 線形性から問題なし。
- $\lambda, [a] \to [\lambda a]$: 線形性から問題なし。
- $\lambda, [a] \to [a + \lambda]$: Shamir SS なら簡単。$[a + \lambda] = [a] + \lambda := (a_1 + \lambda, \ldots, a_n + \lambda)$.
- $[a], [b] \to [ab]$ をどうするか。$ab = \sum \lambda_{i,n} a_i b_i$ なので、各 P_i に a_i, b_i から自主的に $a_i b_i$ を作ってもらわなければいけない。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a + b]$: 線形性から問題なし。
- $\lambda, [a] \to [\lambda a]$: 線形性から問題なし。
- $\lambda, [a] \to [a + \lambda]$: Shamir SS なら簡単。$[a + \lambda] = [a] + \lambda := (a_1 + \lambda, \ldots, a_n + \lambda)$.
- $[a], [b] \to [ab]$ をどうするか。$ab = \sum \lambda_{i,n} a_i b_i$ なので、各 P_i に a_i, b_i から自主的に $a_i b_i$ を作ってもらわなければいけない。
- 回路への入力をビット $a \in \{0, 1\}$ にしなければいけない。

能動的攻撃者 (active adversary) 対策方針

（大方針）能動的攻撃者を受動的攻撃者と同じことしかできないよう強制する。それでも従わない場合は、プロトコルから排除され、攻撃者の秘密は開示されプロトコルが続行される。

- $P_i : a \to [a]$: (BGW) VSS に変更して、不正な P_i でも正しく秘密分散せざるを得ないようにする。
- $[a] \Rightarrow a$: 全参加者が a のシェアを broadcast して、a を復元するとき、能動的攻撃者が正しくないシェアを broadcast しても、誤り訂正符号の技術を使って、正しく a を復元する。
- $[a], [b] \to [a+b]$: 線形性から問題なし。
- $\lambda, [a] \to [\lambda a]$: 線形性から問題なし。
- $\lambda, [a] \to [a+\lambda]$: Shamir SS なら簡単。$[a+\lambda] = [a] + \lambda := (a_1 + \lambda, \ldots, a_n + \lambda)$.
- $[a], [b] \to [ab]$ をどうするか。$ab = \sum \lambda_{i,n} a_i b_i$ なので、各 P_i に a_i, b_i から自主的に $a_i b_i$ を作ってもらわなければいけない。
- 回路への入力をビット $a \in \{0, 1\}$ にしなければいけない。
  $\Rightarrow [a(a-1)] = [0]$ を証明する！

能動的攻撃者モデルで使われる秘密分散

能動的攻撃者モデルで使われる秘密分散（コミットメント）：

$$[[a]]_D := ([a_1]_1, \ldots, [a_n]_n)$$

D の持つ秘密 a が参加者間で秘密分散され、さらに参加者 P_i の持つ a のシェア a_i がさらに参加者間で秘密分散されている状態。

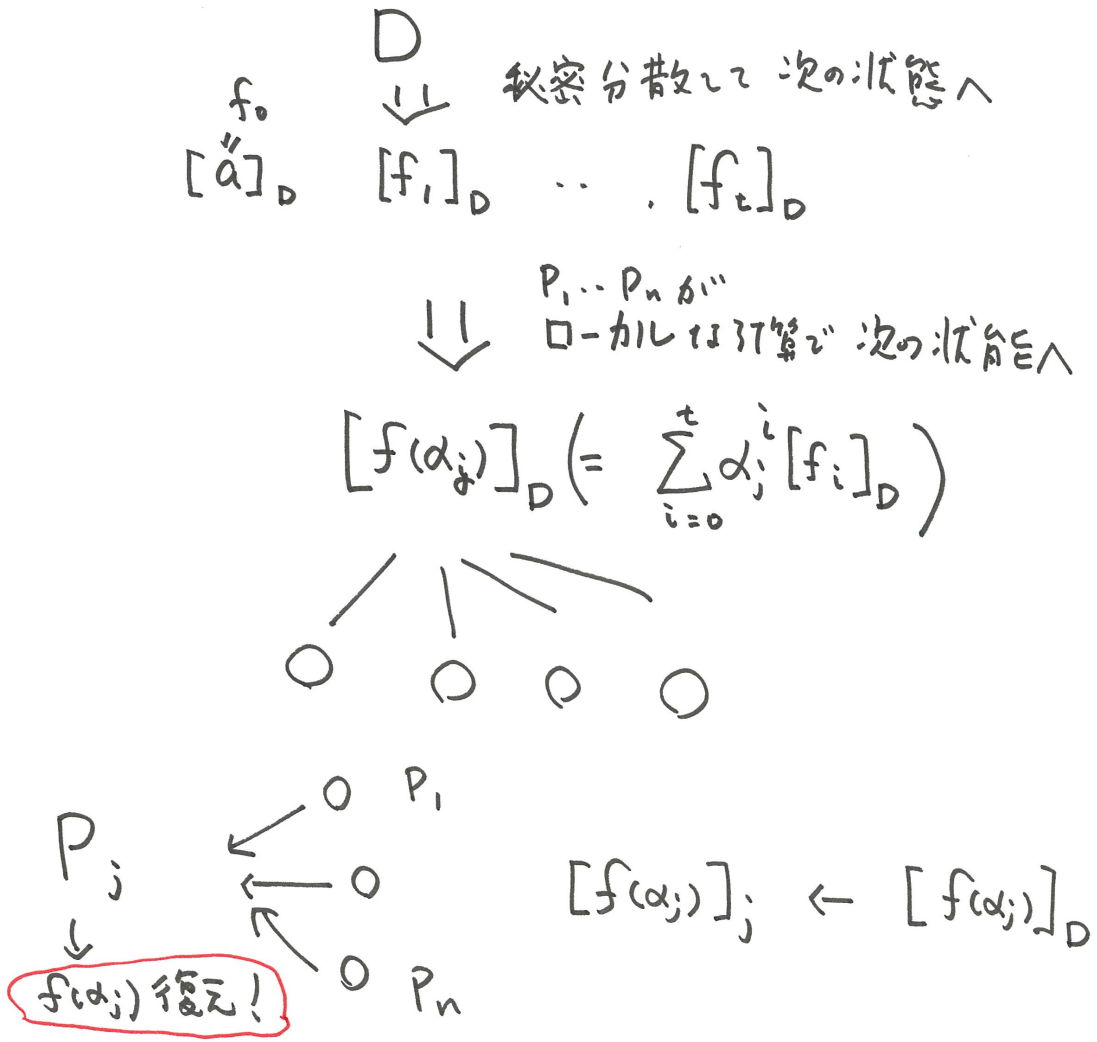
$D : a \to [[a]]_D$ は基本コマンドを使い次のように実現される。

1. $D : a \to [a]_D$.
2. D は、ランダムな $f_1, \ldots, f_t \in K$ を選び、$f_0 = a$ として多項式 $f(X) = \sum_{i=0}^{t} f_i X^i$ を定義する。
3. $D : f_1, \ldots, f_t \to [f_1]_D, \ldots, [f_t]_D$.
4. $P_1, \ldots, P_n$ は、線形性を生かしローカルな（互いの通信なし）計算で次の状態を作る。

$$[f(\alpha_1)]_D, \ldots, [f(\alpha_n)]_D \quad \text{where } [f(\alpha_j)]_D = \sum_{i=0}^{t} \alpha_j^i [f_i]_D.$$

5. 全ての P_i に対して、$P_i : f(\alpha_i) \leftarrow [f(\alpha_i)]_D$.
6. 各 P_i は、$a_i = f(\alpha_i)$ と設定する。

図で


D
秘密分散して次の状態へ
f_0
$[a]_D$
$[f_1]_D$ .. . $[f_t]_D$
$P_1 .. P_n$ が
ローカルな計算で次の状態へ
$[f(\alpha_j)]_D (= \sum_{i=0}^{t} \alpha_j^i [f_i]_D)$
P_j
P_1
P_n
$f(\alpha_j)$復元!
$[f(\alpha_j)]_j \leftarrow [f(\alpha_j)]_D$

足し算

次のように定義。

$$[[a]] + [[b]] := \Big([a_1]_1 + [b_1]_1, \ldots, [a_n]_n + [b_n]_n\Big)$$

$[\cdot]$ の線形性から、

$$[[a + b]] = [[a]] + [[b]]$$

が成立し、参加者間のローカルな計算で済む。実際の手順は以下の通り。

- Input: $[[a]] = ([a_1]_1, \ldots, [a_n]_n)$, $[[b]] = ([b_1]_1, \ldots, [b_n]_n)$.
- Output: $[[a + b]] = \Big([a_1 + b_1]_1, \ldots, [a_n + b_n]_n\Big)$.

1. $i = 1, \ldots, n$ に対して、$[a_i] + [b_i]$ を各参加者が実行。すなわち、各参加者が、$[a_i]$ と $[b_i]$ の自分のシェアをローカルに足し算し、$[a_i] + [b_i]$ の状態にする。$[\cdot]$ の線形性から、これは $[a_i + b_i]$ の状態と同じ。
2. $i = 1, \ldots, n$ に対して、各 P_i が $a_i + b_i$ をローカルに計算し保持する。その結果、$[a_i + b_i]_i$ の状態になる。

図で

$$[\![a]\!] + [\![b]\!] := ([a_1]_1 + [b_1]_1, \ldots, [a_n]_n + [b_n]_n)$$

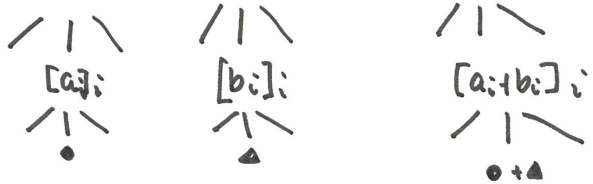
$$= \quad = \qquad \overset{\text{linearity}}{=} \qquad \overset{\text{linearity}}{=}$$

$$([a_1]_1, \ldots [a_n]_n) \quad ([b_1]_1, \ldots [b_n]_n) \quad [a_1+b_1]_1 \quad [a_n+b_n]_n$$

$P_i : a_i, b_i \to a_i + b_i$ 保持

$P_1, \ldots P_n : [a_i]_i + [b_i]_i \overset{\text{linearity}}{=} [a_i + b_i]_i$

$$\Rightarrow [\![a]\!] + [\![b]\!] = [\![a+b]\!]$$



ちょっとした注意

$[[a]]$ は、特に誰が秘密 a を保持しているか気にしていないことを示す表現。実際、

- $[[a]]_i + [[b]]_i = [[a+b]]_i$
- $[[a]]_i + [[b]]_j = [[a+b]]_\emptyset$ for $i \neq j$.
- $[[a]]_\emptyset + [[b]]_j = [[a+b]]_\emptyset$.
- $[[a]]_\emptyset + [[b]]_\emptyset = [[a+b]]_\emptyset$.

などのバリエーションが可能であるが、しかしいずれの場合も、

$$[[a+b]] = \Big([a_1 + b_1]_1, \ldots, [a_n + b_n]_n\Big)$$

は成り立っている。つまり、秘密を 1 人で保持している参加者がいるかどうかに関わらず、各シェア $a_i + b_i$ は、P_i により保持され、$a_i + b_i$ も参加者間で正しく秘密分散されている。

スカラー倍

$\lambda \in K$ に対して、次のように定義。

$$\lambda[[a]] := \left(\lambda[a_1]_1, \ldots, \lambda[a_n]_n\right)$$

$[\cdot]$ の線形性から、

$$[[\lambda a]] = \lambda[[a]]$$

が成立し、参加者間のローカルな計算で済む。実際の手順は以下の通り。

- Input: λ, $[[a]] = ([a_1]_1, \ldots, [a_n]_n)$
- Output: $[[\lambda a]] = \left([\lambda a_1]_1, \ldots, [\lambda a_n]_n\right)$.

1. $i = 1, \ldots, n$ に対して、$\lambda[a_i]$ を各参加者が実行。すなわち、各参加者が、$[a_i]$ の自分のシェアとスカラー λ をローカルに掛け算し、$\lambda[a_i]$ 状態を作る。$[\cdot]$ の線形性から $[\lambda a_i]$ の状態になる。
2. $i = 1, \ldots, n$ に対して、各 P_i が λa_i をローカルに計算し保持。その結果 $[\lambda a_i]_i$ の状態になる。

図で

$$\lambda\,[\![a]\!] \;:=\; (\lambda[a_1]_1, \ldots \lambda[a_n]_n)$$

$$[a_1]_1 \ldots [a_n]_n$$

$$P_i : \lambda, a_i \to \lambda a_i \;(\text{secret})$$

$$P_1 \ldots P_n : \lambda\,[a_i] = [\lambda a_i] \;(\text{linearity})$$

Commitment Multiplication Protocol (CMP) I

このプロトコル CMP は、$[[a]]$, $[[b]]$ から、$[[ab]]$ を構成するときに使われるサブプロトコルである。

- Input: $[a]_D$, $[b]_D$, $[c]_D$
- Output: accepts if $c = ab$; otherwise, rejects.

1. D は、ランダムな $f_1, \ldots, f_t, g_1, \ldots, g_t \in K$ を選び、多項式 $f(X) = a + \sum_{i=1}^{t} f_i X^i$, $g(X) = b + \sum_{i=1}^{t} g_i X^i$ を定義。D は、$h(X) = f(X)g(X) = \sum_{i=0}^{2t} h_i X^i$ を計算する。$h(0) = f(0)g(0) = c$ に注意。
2. $D : f_1, \ldots, f_t \to [f_1]_D, \ldots, [f_t]_D$.
3. $D : g_1, \ldots, g_t \to [g_1]_D, \ldots, [g_t]_D$.
4. $D : h_1, \ldots\ldots, h_{2t} \to [h_1]_D, \ldots\ldots, [h_{2t}]_D$.
5. $P_1, \ldots, P_n$ は、線形性を生かしローカルな計算で次の状態を作る。

$$[f(\alpha_1)]_D, \ldots, [f(\alpha_n)]_D, \quad [g(\alpha_1)]_D, \ldots, [g(\alpha_n)]_D, \quad [h(\alpha_1)]_D, \ldots, [h(\alpha_n)]_D$$

6. 全ての P_i に対して、$P_i : f(\alpha_i), g(\alpha_i), h(\alpha_i) \leftarrow [f(\alpha_i)]_D, [g(\alpha_i)]_D, [h(\alpha_i)]_D$.

Commitment Multiplication Protocol (CMP) II

7. P_i は、$f(\alpha_i)g(\alpha_i) = h(\alpha_i)$ が成り立つかチェックをして、もし成り立たなければ (accuse, P_i, D) を broadcast する。
8. 全ての (accuse, P_i, D) に対して、参加者全員が協力して $f(\alpha_i), g(\alpha_i), h(\alpha_i)$ を開示する。すなわち、

$$[f(\alpha_i)]_i, [g(\alpha_i)]_i, [h(\alpha_i)]_i \Rightarrow f(\alpha_i), g(\alpha_i), h(\alpha_i)$$

9. accuse していない P_j は、一つでも、$f(\alpha_i)g(\alpha_i) \neq h(\alpha_i)$ となるものがあれば、(accuse, P_j, D) を broadcast する。
10. Step 7 と Step 9 の合計 accuse 数が $t+1$ 以上であれば、プロトコルは拒否 (reject) される。そうでなければ、受理 (accept) される。

(Correctness) D が正直な場合

- 正直な P_i は、$f(\alpha_i)g(\alpha_i) = h(\alpha_i)$ が成り立つので、accuse を broadcast することはない。
- 不正直な P_i は、accuse をする可能性があるが、accuse の合計数は高々 t.
- (accuse, P_i, D) に対して、全参加者が協力して $f(\alpha_i)$, $g(\alpha_i)$, $h(\alpha_i)$ を開示した場合、$f(\alpha_i)g(\alpha_i) = h(\alpha_i)$ が成り立つことが分かるので、正直な参加者が新たに accuse することは無い。
- よって、プロトコルは accept される。

(Correctness) D が不正直な場合

- プロトコルが accept されたにも関わらず、$c \neq ab$ であったと仮定する。
- 正直な参加者の数は $n - t \geq 2t + 1$. もし、全ての正直な参加者の各自のシェアが、$f(\alpha_i)g(\alpha_i) = h(\alpha_i)$ が成立してしまったとする。すると $\deg(h) \leq 2t$ より、$[c]_D = [ab]_D = \sum_{i \in H} \lambda_{i,H}[f(\alpha_i)g(\alpha_i)]_D$ が成立してしまい、$c = ab$ に反する。
- よって、正直な参加者のシェアの中に $f(\alpha_i)g(\alpha_i) \neq h(\alpha_i)$ となるものを紛らわせなければならない。
- しかし、正直な P_i に $f(\alpha_i)g(\alpha_i) \neq h(\alpha_i)$ となる $f(\alpha_i)$, $g(\alpha_i)$, $h(\alpha_i)$ を送ると必ず Step 7 で $(\mathsf{accuse}, P_i, D)$ を broadcast され、Step 9 で、他の正直な参加者も全員 accuse を broadcast するため、accuse 数が t を超え、プロトコルは必ず reject されてしまう。
- よって、正直な参加者に不正なシェアを送った場合、プロトコルは必ず reject されるので、矛盾。

Commitment Sending Protocol (CSP)

このプロトコル CSP は、$D : a \to [[a]]_D$ にするプロトコルの中で使われていたサブプロトコル。必要なので定義しておく。

- Input: $[a]_D$,
- Output: $[[a]]_D$; otherwise, rejects.

- D は、ランダムな $f_1, \ldots, f_t \in K$ を選び、$f_0 = a$ として多項式 $f(X) = \sum_{i=0}^{t} f_i X^i$ を定義する。
- $D : f_1, \ldots, f_t \to [f_1]_i, \ldots, [f_t]_i$.
- $P_1, \ldots, P_n$ は、線形性を生かしローカルな（互いの通信なし）計算で次の状態を作る。

$$[f(\alpha_1)]_i, \ldots, [f(\alpha_n)]_i \quad \text{where } [f(\alpha_j)] = \sum_{i=0}^{t} \alpha_j^i [f_i].$$

- 全ての P_i に対して、$P_i : f(\alpha_i) \leftarrow [f(\alpha_i)]_D$.
- 各 P_i は、$a_i = f(\alpha_i)$ と設定する。
- $[[a]]_D = ([a]_1, \ldots, [a]_n)$ の状態になる。

掛け算（考え方）

$[[a]], [[b]]$ から、$[[ab]]$ を構成する掛け算プロトコルを考える。$[[a]] = \left([a_1]_1, \ldots, [a_n]_n\right)$, $[[b]] = \left([b_1]_1, \ldots, [b_n]_n\right)$ とする。$\hat{c}_i = a_i b_i$ の関係を満たす $\hat{c}_i$ を秘密分散して、$[[\hat{c}_i]]$ の状態にした参加者 P_i の集合を S とする。正直な人は正しく $[[\hat{c}_i]]$ をつくるので、$\#S \geq n - t = 2t + 1$. よって

$$ab = \sum_{i \in S} \lambda_{i,S} a_i b_i$$

$[[\cdot]]$ の線形性より、

$$[[ab]] = \sum_{i \in S} \lambda_{i,S} [[a_i b_i]]$$

となり、$[[ab]]$ を実現できることがわかる。

掛け算

- Input: $[[a]] = \left([a_1]_1, \ldots, [a_n]_n\right)$, $[[b]] = \left([b_1]_1, \ldots, [b_n]_n\right)$.
- Output: $[[c]] = ([c_1]_1, \ldots, [c_n]_n)$ where $c = ab$.

1. 各 P_i は、$a_i b_i$ を計算し、$\mathrm{CMP}([a_i]_i, [b_i]_i, [a_i b_i]_i)$ を実行する。このプロトコルが受理されると、$[a_i b_i]_i$ という状態になる。
2. 各 P_i は、$[a_i b_i]_i$ を $[[a_i b_i]]_i$ の状態にするために、$P_1, \ldots, P_n$ の間でサブプロトコル $\mathrm{CSP}([a_i b_i]_i)$ を行う。
3. CMP プロトコルを受理された P_i の集合を S とする。各参加者は各自がローカルに計算し
$$\sum_{i \in S} \lambda_{i,S} [[a_i b_i]]$$
という状態を作る。

$\#S \geq n - t \geq 2t + 1$ と、$[[\cdot]]$ の線形性より、

$$[[ab]] = [[\sum_{i \in S} \lambda_{i,S} a_i b_i]] = \sum_{i \in S} \lambda_{i,S} [[a_i b_i]]$$

という状態になっており目的を達している。

参考文献

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson.

Completeness theorems for non-cyptographic fault-tolerant distributed computation.

In Janos Simon, editor, STOC '88, pages 1–10. ACM, 1988.