Computer Security Symposium 2019
21 - 24 October 2019

大規模な医療統計に向けた実用的な秘密分析計算

市川 敦謙[1,a)]　須藤 弘貴[1]　竹之内 大地[2]　五十嵐 大[1]　三品 気吹[1]　濱田 浩気[1]　菊池 亮[1]

概要：秘密計算は，データを秘匿したままで統計分析を始めとする様々な演算を行える技術である．近年，大規模データへの分析需要が高まる中で，医療統計を含む様々な場面で分析とプライバシ保護の両立が必要とされており，秘密計算技術の重要性が増している．一方で，秘密計算では通常の計算と比べて精度・速度が低下してしまうことが実用上の課題とされる．本研究では今日の秘密計算が十分な実用性を獲得しつつあることを示す目的で，代表的な分析手法であるログランク検定，オッズ比，フィッシャー正確検定の 3 つを秘密計算する方法を提案し，また実験によりそれらが実用的な精度・速度を達成できていることを示す．

キーワード：秘密計算，医療統計，ログランク検定，オッズ比，フィッシャー正確検定

Practical Secure Computation for Medical Statistics on Big Data

Atsunori Ichikawa[1,a)]　Hiroki Sudo[1]　Daichi Takenouchi[2]　Dai Ikarashi[1]　Ibuki Mishina[1]
Koki Hamada[1]　Ryo Kikuchi[1]

Abstract: Secure computation enables us to make statistical analysis with hiding input data. Recently, there are increasing needs to analyze big data. On the other hand, since we also need to preserve privacy, secure computation becomes important more and more. However, sometimes it is said that there are problems for practical usage of secure computation: Worsening of *accuracy* and *execution time*. In this paper, we aim to show that secure computation is now becoming practical. We propose three schemes which securely compute log-rank test, odds ratio, and Fisher's exact test. Moreover, we evaluate *accuracy* and *execution time* of them to show they are practical.

1. 背景

近年の技術発展に伴い，潤沢な計算資源を利用した大規模データへの統計分析が盛んに行われるようになった．例えばゲノム解析を始めとする医療統計もその 1 つであり，数万を超える遺伝子データを元に疾患の要因となる遺伝子を明らかにする，などのユースケースが考えられる．また分析対象とするデータをより多く集めるといった目的で，複数の医療機関・研究機関が保有する情報を横断的に分析するというニーズも高まっており，十分な量のデータに基づく確度の高い分析結果を得られることが期待できる．

しかしながら，各機関が保有するデータは個々が患者のプライバシに直結するものであり，安易に第三者提供のような形で横断分析を行うことは避けるべきである．このような横断分析需要とプライバシ保護を両立する技術の 1 つとして，秘密計算が知られている．

秘密計算は，暗号化など何らかの形でデータを秘匿化し，その情報を公開することなく様々な計算を行う技術である．Yao [6] により考案された garbled circuit や準同型暗号，秘密分散法 [5], [7] に基づくマルチパーティ計算など，今日までに様々な秘密計算の実現方法が提案されている．秘密計算を用いれば，統計分析を含む様々な演算の結果を高いプライバシ保護の元で得ることができる [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]．

一方で秘密計算による演算は，計算結果の精度や計算速度が平文での計算に比べて低下してしまうことがあり，しばしば実用上の課題とされる．これは情報の秘匿性を保つ

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[2] NTT アドバンステクノロジ株式会社
NTT Advanced Technorogy Corporation
[a)] atsunori.ichikawa.nf@hco.ntt.co.jp

ため，通常の計算機と全く同一のアルゴリズムで秘密計算が行えないことに起因している．

本論文の目的はこうした課題に対し，今日の秘密計算が十分な実用性を獲得しつつある，という事実を示すことである．本論文では医療統計を始めとする様々な統計分析の場面で実際に用いられる，ログランク検定，オッズ比，フィッシャー正確検定という 3 つの分析手法に焦点を当て，これらを秘密計算上で実現する手法を示す．さらに各手法を実機上で実行し，(1) 計算精度，(2) 計算速度，の両面から実用性の評価を行う．

1.1 プログマラブルな秘密計算ライブラリ MEVAL

MEVAL [16] は，加算や乗算といった基本演算からソートなど複雑な演算に至るまで，秘密計算による 100 以上の演算を組み合わせてプログラムが可能な秘密計算ライブラリである．本論文では MEVAL に実装される関数群を用いて秘密計算による統計分析を実現する．本論文で用いる具体的な関数については 3.2 節で述べる．また，実験は全て表 1 に示すマシン上で行った．

OS	CPU	メモリ	ネットワーク
CentOS 7.3	Intel Xeon Gold 6144 ×2 3.50GHz	768GB	10G Ring

表 1 実験環境

1.2 本論文の貢献

本論文では，代表的な統計分析手法であるログランク検定，オッズ比，フィッシャー正確検定を秘密計算上で実行する手法を提案する．

また提案手法について，実機上で 1 万〜1000 万件[*1]のデータセットに対する実験を行い，以下の 2 点を示した．

(1) 3 つの分析全てで，実験した全てのデータにおいて秘密計算が十分実用的な精度で実行可能である．

(2) ログランク検定は 100 万件，オッズ比は 1000 万件，フィッシャー正確検定は 10 万件までのデータセットに対して秘密計算を 5 秒程度で実行可能である．また，それ以上のサイズのデータセットに対しても，5 分以内に実行可能である．

1.3 関連研究

医療統計を中心に多く用いられる分析手法として，例えば Kamm ら [9]，Lu ら [10] はそれぞれ秘密計算による χ^2 検定を提案している．また，本論文でも取り扱うフィッシャー正確検定については，千田ら [13]，濱田ら [14]，長谷川ら [15]，Hamada ら [11] により秘密計算での実現方法が提案されているが，[14] の方式は実機上での評価がなされていない．さらに近年，秘密計算の多機能化・高速化に伴い，平文での計算にも比肩しうる精度・速度を実現したロジスティック回帰が三品ら [17] により提案された．

[*1] フィッシャー正確検定のみ，1 万〜100 万件．

2. 既存の統計分析手法

本章では，本論文で焦点を当てる統計分析手法，ログランク検定・オッズ比・フィッシャー正確検定を紹介する．

2.1 ログランク検定

ログランク検定 [3], [4] は 2 つの群の生存曲線を比較するための検定手法であり，試薬や臨床試験の効果測定などに用いられる．

時刻 t	状態 s	群 g
9	1	A
13	0	A
5	1	B
5	1	B
27	1	B

表 2 生存表の例

表 2 のように，各標本がそれぞれ生存時間 t と 2 値の状態 $s \in \{0,1\}$，および群 $g \in \{A,B\}$ を属性に持つとする．この s, g は例えば，$\{0 =$ 観測打ち切り，$1 =$ 死亡 $\}$ および $\{A =$ 実薬投与，$B =$ 偽薬投与 $\}$ といったものが挙げられる．この時，各時刻における群ごとの生存率を表すグラフが生存曲線と呼ばれ，例えば Kaplan-Meier 法 [2] によって得られることが知られている．

生存曲線は，時刻と生存率との相関を与え，その概形を視覚的にもわかりやすく図示することができる点で非常に有用な統計量である．しかしながら，2 群の生存曲線の差異についてはグラフの概形に基づく直感的な理解を与えるのみであり，統計的に差を示すには至らない．ログランク検定はこのような 2 つの生存曲線について，統計的な差の有無を明らかにする手法である．

ログランク検定の基本的な原理は χ^2-検定である．すなわち，2 つの生存曲線に差がないとした時 (帰無仮説) に，各時刻で死亡 ($s = 1$) となる標本数の期待値を算出し，実際に観測された値とどれほどの隔たりがあるかを計算する．具体的な計算方法を以下に示す．

(1) 表 2 に表されるようなデータセットについて，各時刻 t_i ごとに次の値を計算する．ただし $i = 0, \ldots, k-1 (<$ 標本数 $n)$ とする．

i) A 群の生存数 n_i^A，B 群の生存数 n_i^B．ただし生存数とは，時刻 t_i の直前において死亡も打ち切りも発生していない標本の数とする．

ii) 全体の生存数 $n_i = n_i^A + n_i^B$.

iii) A 群の死亡数 o_i^A，B 群の死亡数 n_i^B．ただし死亡数とは，時刻 t_i において新たに死亡 $(s = 1)$ が観測された標本の数とする．

iv) 全体の死亡数 $o_i = o_i^A + o_i^B$.

v) A 群の死亡数期待値 $e_i^A = n_i^A \cdot \frac{o_i}{n_i}$.

(2) 死亡数と期待値の差 $U = \sum_{i=0}^{k-1}(o_i^A - e_i^A)$ を得る．

(3) U の分散 $V = \sum_{i=0}^{k-1} \frac{n_i^A n_i^B o_i(n_i - o_i)}{n_i^2(n_i - 1)}$ を計算する．

(4) U^2/V と自由度 1 の χ^2 分布から p 値を得る．

この後，p 値と有意水準との比較をもって生存曲線の差の有無を判定する．

2.2 オッズ比

オッズ比は 2 つの事象の起こりやすさを比較する統計的尺度の一つであり，医療統計の分野でよく用いられる．オッズ比は 2×2 分割表から計算される．2×2 分割表は，2 値の 2 変数 (事象) の観測回数をまとめた表である．分割表が表 2.2 のように表されるとする．この場合，オッズ比

表 3 分割表の例.

	疾患 B あり	疾患 B なし
遺伝子 A	a	b
遺伝子 A'	c	d

は $r = (a \cdot d)/(b \cdot c)$ と計算される．(b または c が 0 の場合は定義できない)

オッズ比 1 は遺伝子 A でも A'(1 塩基変異が入っている) でも疾患 B の発症率は同じということを意味する．逆にオッズ比が 1 より大きくなるほど遺伝子 A のほうが A' であるときよりかかりやすい，つまり遺伝子 A が疾患 B の要因である可能性が高いことを示唆している．

2.3 フィッシャー正確検定

フィッシャー正確検定 [1] は，Fisher により考案された検定手法である．本論文では，表 2.2 に表されるような 2×2 分割表に焦点を絞り，2 属性間の独立性検定としてフィッシャー正確検定を用いる．

分割表の周辺分布，すなわち各属性の小計値を以下のように定義する．

$$n_A = a + b, n_{A'} = c + d, n_1 = a + c, n_0 = b + d,$$
$$m = a + b + c + d.$$

この時，表の分布 (a, b, c, d) が現れる確率 p_* は式

$$p_* = \frac{n_A! n_{A'}! n_1! n_0!}{a! b! c! d! m!}$$

によって得られる．

周辺分布を固定した場合，分割表の自由度が 1 となることから，全ての分布のパターンは整数 $i \in [-\min(a, d), \min(b, c)]$ を用いて $(a + i, b - i, c - i, d + i)$ と表すことができる．その発生確率 p_i は以下に表される．

$$p_i = \frac{n_A! n_{A'}! n_1! n_0!}{(a + i)!(b - i)!(c - i)!(d + i)! m!}.$$

フィッシャー正確検定における p 値は，

$$p = \sum_{p_i \le p_*} p_i$$

により与えられる．これは元の分布 (a, b, c, d) を基準として，より偏りの大きな分布が現れる確率 p_i ; $p_i \le p_*$ の総和である．p が有意水準 α を下回れば帰無仮説を棄却する．

フィッシャー正確検定は，近似的な検定を行う他の手法に比べ正確な確率を得られる反面，階乗を始めとする多くの乗除を必要とし，計算に大きなコストがかかってしまう問題がある．この問題に対する現実的な解法として，確率 p_*, p_i を対数として計算する方法が知られている．

関数 $f(x) := \log(x!)$ とすると，確率 p_* の対数は

$$\begin{aligned}\log p_* =& f(n_A) + f(n_{A'}) + f(n_1), f(n_0) \\ & - (f(a) + f(b) + f(c) + f(d) + f(m))\end{aligned}$$

となる (各 p_i についても同様)．$f(x) = f(x - 1) + \log x$ より，階乗の計算よりも容易に計算できる．

3. 準備

3.1 記法，定義

以下に，本論文で用いる記号の定義を記す．$\mathbb{F}_p$ を位数 p の有限体とし，以下で扱う値は全て $\mathbb{F}_p$ の要素であるとする．$\mathbb{F}_p$ 上の十分大きな数を $\perp$ $(\in \mathbb{F}_p)$ と表す．

長さ t の配列やベクトルは $\mathbf{a}(\in \mathbb{F}_p^t)$ と表し，その i 番目の要素を $\mathbf{a}[i]$ と表記する．また，$\mathcal{D}_n \subseteq \mathbb{F}_p^n$ を n 次元データ空間 (または n 属性〜) と呼び，$\mathbf{D} \in \mathcal{D}_n^m$ を長さ m の n 次元データセットと呼称する．$\mathbf{D} \in \mathcal{D}_n^m$ に含まれる i 番目のデータを $\mathbf{D}[i] \in \mathcal{D}_n$ で表し，またその中の第 j 次元の値を $\mathbf{D}[i]_{(j)}(\in \mathbb{F}_p)$ と表記する．全てのデータに含まれる第 j 次元の値の列を $\mathbf{D}_{(j)}(\in \mathbb{F}_p^m)$ と表記する．

3.2 秘密計算上の演算

本論文で提案するプロトコルは，秘密計算上の演算の組み合わせにより実現される．以下に本論文で用いる演算を紹介する．なお実装においては [16] に示される関数群に加えて，五十嵐による除算と指数関数 [18]，制限付き剰余，濱田らによるローテーション [12] を新規に追加した．

3.2.1 秘匿化

$a \in \mathbb{F}_p$ を秘匿化した値を，本論文では便宜上「分散値」と呼び $[\![a]\!]$ と表記する．また a を $[\![a]\!]$ の平文値と呼ぶ．配列 (またはベクトル)$\mathbf{a} = (\mathbf{a}[0], \ldots, \mathbf{a}[n-1]) \in \mathbb{F}_p^n$ の各要素

を秘匿化した $([\![\mathbf{a}[0]]\!],\ldots,[\![\mathbf{a}[n-1]]\!])$ を $[\![\mathbf{a}]\!]$ と表記する．同様に，n 次元データ $D \in \mathcal{D}_n$ および長さ m のデータセット $\mathbf{D} \in \mathcal{D}_n^m$ についても，各要素を秘匿化したものを $[\![D]\!]$，$[\![\mathbf{D}]\!]$ と表記する．

3.2.2 算術演算

加算，減算，乗算，除算の各演算は，$a, b \in \mathbb{F}_p$ の分散値 $[\![a]\!], [\![b]\!]$ を入力とし，それぞれ $a+b, a-b, ab, a/b$ の計算結果 c_1, c_2, c_3, c_4 の分散値を出力する．これらの演算の実行を以下のように記述する．

$$[\![c_1]\!] \leftarrow \textsc{Add}([\![a]\!],[\![b]\!]),\ [\![c_2]\!] \leftarrow \textsc{Sub}([\![a]\!],[\![b]\!]),$$
$$[\![c_3]\!] \leftarrow \textsc{Mul}([\![a]\!],[\![b]\!]),\ [\![c_4]\!] \leftarrow \textsc{Div}([\![a]\!],[\![b]\!]).$$

特に誤解を招く恐れがなければ，それぞれを $[\![a]\!]+[\![b]\!], [\![a]\!]-[\![b]\!], [\![a]\!]\times[\![b]\!], [\![a]\!]/[\![b]\!]$ と略記する．

3.2.3 等号・不等号判定

等号判定，および不等号判定の演算は，$a, b \in \mathbb{F}_p$ の分散値 $[\![a]\!], [\![b]\!]$ を入力とし，述語 $a =_? b$ および $a \leq_? b, a <_? b$ の真偽値 c_1, c_2, c_3 の分散値を出力する．これらの演算の実行を以下のように記述する．

$$[\![c_1]\!] \leftarrow [\![a]\!] =_? [\![b]\!],\ [\![c_2]\!] \leftarrow [\![a]\!] \leq_? [\![b]\!],\ [\![c_3]\!] \leftarrow [\![a]\!] <_? [\![b]\!].$$

3.2.4 選択

選択は，真偽値 $c \in \{0,1\}$ と $a, b \in \mathbb{F}_p$ の分散値 $[\![c]\!], [\![a]\!], [\![b]\!]$ を入力とし，

$$d = \begin{cases} a & \text{if } c = 1, \\ b & \text{otherwise} \end{cases}$$

を満たす d の分散値を出力する．この演算の実行を以下のように記述する．

$$[\![d]\!] \leftarrow \textsc{IfElse}([\![c]\!],[\![a]\!],[\![b]\!]).$$

3.2.5 秘密一括写像

秘密一括写像は，写像 $f : \mathbb{F}_p \to \mathbb{F}_p$ に関する定義域ベクトル $\mathbf{x} := (x_0,\ldots,x_{m-1})$ と値域ベクトル $\mathbf{f_x} := (f(x_0),\ldots,f(x_{m-1}))$，およびベクトル $\mathbf{a} \in \mathbb{F}_p^n$ の分散値 $[\![\mathbf{f_x}]\!], [\![\mathbf{x}]\!], [\![\mathbf{a}]\!]$ を入力とし，ベクトル $\mathbf{b}$; $\mathbf{b}[i] = f(\mathbf{a}[i])$ の分散値を出力する．この演算の実行を以下のように記述する．

$$[\![\mathbf{b}]\!] \leftarrow \textsc{Map}([\![\mathbf{a}]\!],[\![\mathbf{f_x}]\!],[\![\mathbf{x}]\!]).$$

3.2.6 指数関数

e をネイピア数とする．指数関数は，$a \in \mathbb{F}_p$ の分散値 $[\![a]\!]$ を入力とし，e^a の近似値 b の分散値を出力する．この演算の実行を以下のように記述する．

$$[\![b]\!] \leftarrow \textsc{Exp}([\![a]\!]).$$

3.2.7 グループ和

グループ和は，配列 $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n$ の分散値 $[\![\mathbf{a}]\!], [\![\mathbf{b}]\!]$ を入力とし，グループごとの総和を格納した配列 $\mathbf{c}$ の分散値を出力する．ここでは，ある値 $\mathbf{b}[i]$ について，$\mathbf{b}[i] = \mathbf{b}[j]$ を満たす全ての $j \in \{0,\ldots,n-1\}$ の集合をグループ $G_{\mathbf{b}[i]}$ と呼ぶものとする．ソートされ，重複する値を削除された各 $\mathbf{b}[i]$ に関して，$\mathbf{c}$ は $\sum_{j \in G_{\mathbf{b}[i]}} \mathbf{a}[j]$ を格納する配列である．この演算の実行を以下のように記述する．

$$[\![\mathbf{c}]\!] \leftarrow \textsc{GroupSum}([\![\mathbf{a}]\!],[\![\mathbf{b}]\!]).$$

3.2.8 制限付き剰余

制限付き剰余は，$a \in \mathbb{F}_p$ の分散値 $[\![a]\!]$ と，平文値 $n \in \mathbb{F}_p$ を入力とし，$c = a \mod n$ を満たす c の分散値を出力する．ただし $-n \leq a < n$ であるとする．この演算の実行を以下のように記述する．

$$[\![c]\!] \leftarrow \textsc{LMod}([\![a]\!], n).$$

この処理は，$[\![c]\!] \leftarrow [\![a]\!] + ([\![a]\!] <_? 0) \times n$ により実現できる．

3.2.9 ローテーション

ローテーションは，配列 $\mathbf{a} \in \mathbb{F}_p^n$ の分散値 $[\![\mathbf{a}]\!]$ と $b \in \mathbb{F}_p$ の分散値 $[\![b]\!]$ を入力とし，$\mathbf{c}[i] = \mathbf{a}[i+b \mod n]$ となる配列 $\mathbf{c}$ の分散値を出力する．ただし $0 \leq b \leq n-1$ とする．この演算の実行を以下のように記述する．

$$[\![\mathbf{c}]\!] \leftarrow \textsc{Rotate}([\![\mathbf{a}]\!],[\![b]\!])$$

4. 秘密ログランク検定

本章では，まずログランク検定をシミュレートする秘密計算プロトコルを提案し，次に提案プロトコルの計算精度・計算速度を実機を用いて評価する．

4.1 秘密計算プロトコル

Protocol 1 Count survivor

Input: 時刻ベクトル $[\![\mathbf{t}]\!]$，群 A, B のデータ位置 $[\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$.
Output: 各群の時刻ごとの生存数ベクトル $[\![\mathbf{n^A}]\!], [\![\mathbf{n^B}]\!]$.

procedure CountSurvive($[\![\mathbf{t}]\!], [\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$)
  $[\![\mathbf{s^A}]\!] \leftarrow \textsc{GroupSum}([\![\mathbf{g^A}]\!],[\![\mathbf{t}]\!])$
  $[\![\mathbf{s^B}]\!] \leftarrow \textsc{GroupSum}([\![\mathbf{g^B}]\!],[\![\mathbf{t}]\!])$
    ▷ ソートされた各時刻ごとのデータ数集計.
  $n \leftarrow \textsc{Length}([\![\mathbf{s^A}]\!])$
  $[\![\mathbf{n^A}]\!], [\![\mathbf{n^B}]\!] \leftarrow \textsc{Fill}([\![0]\!], n)$
  $[\![\mathbf{n^A}[0]]\!] \leftarrow \sum_{i=0}^{n-1} [\![\mathbf{s^A}[i]]\!]$
  $[\![\mathbf{n^B}[0]]\!] \leftarrow \sum_{i=0}^{n-1} [\![\mathbf{s^B}[i]]\!]$ ▷ 各群の初期生存数 (総データ数).
  for $i = 1$ **to** $n-1$ **do**
    $[\![\mathbf{n^A}[i]]\!] \leftarrow [\![\mathbf{n^A}[i-1]]\!] - [\![\mathbf{s^A}[i-1]]\!]$
    $[\![\mathbf{n^B}[i]]\!] \leftarrow [\![\mathbf{n^B}[i-1]]\!] - [\![\mathbf{s^B}[i-1]]\!]$
  end for ▷ 各群の時刻ごとの生存数.
  return $[\![\mathbf{n^A}]\!], [\![\mathbf{n^B}]\!]$
end procedure

本論文における秘密ログランク検定の問題設定は以下．

(1) 入力は 3 次元データセット $\mathbf{D} \in \mathcal{D}_3^m$ の分散値とする．各データ $D := (t, s, g) \in \mathcal{D}_3$ の要素は，時刻 $t \in \mathbb{F}_p$，

状態 $s \in \{0,1\}$，群 $g \in \{A,B\}$ とする．

(2) 出力は 2.1 節における U,V の分散値とする．以降の計算は平文値を取得した者が行う．

本論文では各演算を有限体 $\mathbb{F}_p$ 上に定義しているため，全ての処理を小数部 u ビットの固定小数点演算とする．

Protocol 2 Count dead

Input: 時刻ベクトル $[\![\mathbf{t}]\!]$，状態ベクトル $[\![\mathbf{s}]\!]$，群 A,B のデータ位置 $[\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$.
Output: 各群の時刻ごとの死亡数ベクトル $[\![\mathbf{o^A}]\!], [\![\mathbf{o^B}]\!]$.

procedure CountDead($[\![\mathbf{t}]\!], [\![\mathbf{s}]\!], [\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$)
  $m \leftarrow$ Length($[\![\mathbf{s}]\!]$)
  $[\![\mathbf{d^A}]\!], [\![\mathbf{d^B}]\!] \leftarrow$ Fill($[\![0]\!], m$)
  for $i = 0$ **to** $m-1$ **do in parallel**
    $[\![\mathbf{d^A}[i]]\!] \leftarrow [\![\mathbf{s}[i]]\!] \times [\![\mathbf{g^A}[i]]\!]$
    $[\![\mathbf{d^B}[i]]\!] \leftarrow [\![\mathbf{s}[i]]\!] \times [\![\mathbf{g^B}[i]]\!]$
  end for ▷ 各群の死亡 ($s=1$) データ位置抽出
  $[\![\mathbf{o^A}]\!] \leftarrow$ GroupSum($[\![\mathbf{d^A}]\!], [\![\mathbf{t}]\!]$)
  $[\![\mathbf{o^B}]\!] \leftarrow$ GroupSum($[\![\mathbf{d^B}]\!], [\![\mathbf{t}]\!]$)
    ▷ 各群の時刻ごとの死亡数.
  return $[\![\mathbf{o^A}]\!], [\![\mathbf{o^B}]\!]$
end procedure

Protocol 3 Secure log-rank test

Input: 3 次元データセット $\mathbf{D} \in \mathcal{D}_3^m$ の分散値 $[\![\mathbf{D}]\!]$.
Output: 分散値 $[\![U]\!], [\![V]\!]$.

$[\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!] \leftarrow$ Fill($[\![0]\!], m$)
for $i = 0$ **to** $m-1$ **do in parallel**
  $[\![\mathbf{g^A}[i]]\!] \leftarrow [\![\mathbf{D}[i]_{(3)}]\!] =_? A$
  $[\![\mathbf{g^B}[i]]\!] \leftarrow [\![\mathbf{D}[i]_{(3)}]\!] =_? B$
end for ▷ 各群 A,B のデータ位置を $\{0,1\}$ 値で抽出.
$[\![\mathbf{n^A}]\!], [\![\mathbf{n^B}]\!] \leftarrow$ CountSurvive($[\![\mathbf{D}_{(1)}]\!], [\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$)
$[\![\mathbf{o^A}]\!], [\![\mathbf{o^B}]\!] \leftarrow$ CountDead($[\![\mathbf{D}_{(1)}]\!], [\![\mathbf{D}_{(2)}]\!], [\![\mathbf{g^A}]\!], [\![\mathbf{g^B}]\!]$)
$f(x) := 0$ (if $x=0$), $1/x$ (otherwise)
  ▷ $x=0$ を定義域に含む逆数の写像.
$[\![\mathbf{x}]\!] \leftarrow ([\![0]\!], \ldots, [\![m]\!])$
$[\![\mathbf{f_x}]\!] \leftarrow ([\![f(0)]\!], \ldots, [\![f(m)]\!])$
$n \leftarrow$ Length($[\![\mathbf{n^A}]\!]$)
$[\![\mathbf{n}]\!], [\![\mathbf{n'}]\!], [\![\mathbf{o}]\!] \leftarrow$ Fill($[\![0]\!], n$)
for $i = 0$ **to** n **do in parallel**
  $[\![\mathbf{n}[i]]\!] \leftarrow [\![\mathbf{n^A}[i]]\!] + [\![\mathbf{n^B}[i]]\!]$
  $[\![\mathbf{n'}[i]]\!] \leftarrow [\![\mathbf{n}[i]]\!] - 1$
  $[\![\mathbf{o}[i]]\!] \leftarrow [\![\mathbf{o^A}[i]]\!] + [\![\mathbf{o^B}[i]]\!]$
end for ▷ 全体の生存数 $\mathbf{n}$，死亡数 $\mathbf{o}$ と，$\mathbf{n-1}$ を算出.
$[\![\mathbf{n}_{inv}]\!] \leftarrow$ Map($[\![\mathbf{n}]\!], [\![\mathbf{f_x}]\!], [\![\mathbf{x}]\!]$)
$[\![\mathbf{n'}_{inv}]\!] \leftarrow$ Map($[\![\mathbf{n'}]\!], [\![\mathbf{f_x}]\!], [\![\mathbf{x}]\!]$)
$[\![U]\!] \leftarrow \sum_{i=0}^{n-1} \left([\![\mathbf{o^A}[i]]\!] - [\![\mathbf{n^A}[i]]\!] \times [\![\mathbf{o}[i]]\!] \times [\![\mathbf{n}_{inv}[i]]\!] \right)$
$[\![V]\!] \leftarrow \sum_{i=0}^{n-1} \Big([\![\mathbf{n^A}[i]]\!] \times [\![\mathbf{n^B}[i]]\!] \times [\![\mathbf{o}[i]]\!] \times ([\![\mathbf{n}[i]]\!] - [\![\mathbf{o}[i]]\!]) \times [\![\mathbf{n}_{inv}[i]]\!] \times [\![\mathbf{n}_{inv}[i]]\!] \times [\![\mathbf{n'}_{inv}[i]]\!] \Big)$
return $[\![U]\!], [\![V]\!]$

Protocol 1 はソートされた各時刻 t_i における，各群の生存数 $n_i^{\{A,B\}}$ を得るプロトコルである．入力には生存表から抽出された時刻と，各群のデータ位置 (後述) の分散値をとり，各時刻の生存数[*2]を格納する配列 $\mathbf{n^A n^B}$ の分散値を出力する．ただし，Fill($[\![x]\!], y$) は $[\![x]\!]$ を複製し長さ y の配列を作成する処理を表し，また Length($[\![\mathbf{a}]\!]$) は配列の長さを取得する処理を表す．

Protocol 2 はソートされた各時刻 t_i における，各群の死亡数 $o_i^{\{A,B\}}$ を得るためのプロトコルである．入力には生存表から抽出された時刻，状態，および各群のデータ位置の分散値を取る．出力は各時刻での死亡数を格納する配列 $\mathbf{o^A}, \mathbf{o^B}$ の分散値である．

Protocol 3 に，本論文で提案する秘密ログランク検定プロトコルを示す．これは Protocol 1,2 をサブルーチンとして利用しており，それぞれの入力となる「各群のデータ位置」とは 1〜6 行目に示される，各データの群が A,B のどちらであるかという判定値である．この判定値を集計することで，各群・各時刻に所属するデータ数を得られる．

このプロトコルでは入力の秘匿性のため，本来ならば存在しない時刻での集計値 ($=0$) もベクトル $\mathbf{n}^{\{\mathbf{A},\mathbf{B}\}}, \mathbf{o}^{\{\mathbf{A},\mathbf{B}\}}$ に含まれることとなる[*3]．そのような時刻での統計量を 0 とするため，ここでは通常の秘密除算ではなく，$x=0$ を定義域に含む写像 f を用いて逆数の計算を行っている．

データ件数	p 値 (R)	p 値 (秘密計算)
10,000	0.24621591	0.24621591
100,000	0.8199852**5**	0.8199852**3**
1,000,000	0.3874979**1**	0.3874972**6**
10,000,000	0.813**81462**	0.813**72708**

表 4 R と秘密計算によるログランク検定 p 値.

データ件数	実行時間 (秒)
10,000	0.6
100,000	1.1
1,000,000	6.5
10,000,000	86.7

表 5 秘密計算によるログランク検定実行時間.

4.2 秘密ログランク検定の性能評価実験

本節では，Protocol 3 を実機上で実行した際の，精度・速度の評価を行う．本論文の性能評価では，1 万件〜1000 万件までのダミーデータを用いた．

4.2.1 計算精度

表 4 に，R のライブラリに実装されているログランク検定を実行した結果 (p 値) と，Protocol 3 の実行により得られた U,V より算出した p 値との誤差を比較する．この実験では，秘密計算における小数精度 $u=34$ ビットとした．

結果から，本論文で提案する秘密ログランク検定はデータ件数の増加に合わせて結果の誤差も大きくなることがわかる．しかしながら，1000 万件という巨大なデータに対しても p 値の誤差を小数点以下第 4 位までに抑えられてお

[*2] その時刻直前まで打ち切りも死亡も起きていないデータ数.
[*3] この動作は演算 GroupSum により実現される.

り，検定結果の有意性は十分保証できると考えられる[*4]．

以上により，本論文の秘密ログランク検定は精度において十分な実用性を持つと評価できる．

4.2.2 計算速度

表 5 に，Protocol 3 の実行にかかる時間を示す．

結果から，本論文で提案する秘密ログランク検定は入力データ件数の増加に合わせて非線形に計算時間が増加することがわかる．特に，100 万件データまでは 1 ケタ秒の実行時間であるのに対し，1000 万件データではその 10 倍以上の時間がかかってしまっている．しかし 1000 万件のデータでも 1.5 分程度の時間で計算でき，現実的な時間でのレスポンスが可能である．

以上から，本論文の秘密ログランク検定は 100 万件以下のデータセットには十分実用的な時間で実行可能であり，また 1000 万件までのデータセットに対しても現実的な時間での実行が可能である，と評価できる．

5. 秘密オッズ比

本章では，まずオッズ比を計算する秘密計算プロトコルを提案し，次に提案プロトコルの計算精度・計算速度を実機を用いて評価する．

5.1 秘密計算プロトコル

Protocol 4 Get contingency table

Input: 2 値 2 属性データセット $\mathbf{D} \in \mathcal{D}_2^m$ の分散値 $[\![\mathbf{D}]\!]$.
Output: 分割表の分散値
$([\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!], [\![n_{1,*}]\!], [\![n_{0,*}]\!], [\![n_{*,1}]\!], [\![n_{*,0}]\!])$.

procedure COUNT($[\![\mathbf{D}]\!]$)
$[\![n_{1,*}]\!] \leftarrow \sum_{i=0}^{m-1} [\![\mathbf{D}[i]_{(1)}]\!]$
$[\![n_{0,*}]\!] \leftarrow m - [\![n_{1,*}]\!]$
$[\![n_{*,1}]\!] \leftarrow \sum_{i=0}^{m-1} [\![\mathbf{D}[i]_{(2)}]\!]$
$[\![n_{*,0}]\!] \leftarrow m - [\![n_{*,1}]\!]$
$[\![a]\!] \leftarrow \sum_{i=0}^{m-1} \left([\![\mathbf{D}[i]_{(1)}]\!] \times [\![\mathbf{D}[i]_{(2)}]\!]\right)$
$[\![b]\!] \leftarrow [\![n_{1,*}]\!] - [\![a]\!]$
$[\![c]\!] \leftarrow [\![n_{*,1}]\!] - [\![a]\!]$
$[\![d]\!] \leftarrow [\![n_{*,0}]\!] - [\![b]\!]$
return $([\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!], [\![n_{1,*}]\!], [\![n_{0,*}]\!], [\![n_{*,1}]\!], [\![n_{*,0}]\!])$
end procedure

本研究での秘密オッズ比計算の問題設定は以下．

(1) 入力は 2 値 2 属性のデータセット $D \in \mathcal{D}_2^m$(m はレコード数) の分散値とする．

(2) 出力はオッズ比の分散値 $[\![OR]\!]$ とする．

Protocol 5 に，Protocol 4 をサブルーチンとして用いる，秘密オッズ比計算プロトコルの詳細を示した．ただし Protocol 4 は，各データが空間 $\mathcal{D}_2 = \mathbb{F}_2^2$ に属するようなデータセット $\mathbf{D} \in \mathcal{D}_2^m$(これを 2 値 2 属性データセットと呼ぶ) を入力として分割表の分散値を出力するプロトコルであり，通信量 $O(m)$，ラウンド数 $O(1)$ で実行できる．なお，オッズ比の計算では周辺分布 $n_{1,*}, n_{0,*}, n_{*,1}, n_{*,0}$ を用いないため，Protocol 5 では記述を省略する．

[*4] 例えばデータ 1000 万件とし，$p = 0.05$ 付近で同等の誤差 0.0001 が生じたと仮定しても，相対誤差は 0.002 である．

データ件数	オッズ比 (Python)	オッズ比 (秘密計算)
10,000	1.08329**63**	1.08329**59**
100,000	1.04081**216**	1.04081**245**
1,000,000	0.92283**951**	0.92283**842**
10,000,000	0.02395**741**	0.02395**684**

表 6 R と秘密計算によるオッズ比．

分割表の分散値が与えられたとき，オッズ比の秘密計算は 3.2.2 節に示した秘密乗算・除算アルゴリズムにより実現することができる．ただし，0 除算が起きる場合は未定義のため，選択演算により場合分けを行う必要がある．

Protocol 5 Secure odds-ratio calculation

Input: 2 値 2 属性データセット $\mathbf{D} \in \mathcal{D}_2^m$ の分散値 $[\![\mathbf{D}]\!]$
Output: オッズ比の分散値 $[\![r]\!]$
1: $([\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!]) \leftarrow$ COUNT($[\![\mathbf{D}]\!]$)
2: $[\![cond]\!] = 1 - \big((1 - ([\![b]\!] =_? 0)) \times (1 - ([\![c]\!] =_? 0))\big)$
▷ $(b =_? 0) \vee (c =_? 0)$
3: $[\![odds1]\!] \leftarrow [\![a]\!] \cdot [\![b]\!]$
4: $[\![inv_odds2]\!] \leftarrow [\![d]\!] \cdot [\![c]\!]$
5: $[\![Ratio]\!] \leftarrow [\![odds1]\!]/[\![inv_odds2]\!]$
6: **return** IFELSE($[\![cond]\!], [\![\bot]\!], [\![Ratio]\!]$) ▷ 未定義の場合 $[\![\bot]\!]$

5.2 秘密オッズ比計算の性能評価実験

本節では，Protocol 5 を実機上で実行した際の，精度・速度の評価を行う．実験には，ログランク検定と同様に 1 万件〜1000 万件までのダミーデータを用いた．

5.2.1 計算精度

表 6 に Python でのオッズ比計算結果と Protocol 5 による計算結果を示した．アルゴリズム上，本プロトコルの計算精度は入力レコード数によらず，主に除算に指定する小数精度による．表 6 は小数精度を 20 としたときの結果である．

実験結果では小数第 5 位までに誤差が抑えられており，十分に実用に足る計算精度を持つと評価できる．

5.2.2 実行時間

表 7 に Protocol 5 の実行時間を示す．結果から，入力データ件数の増加に合わせて実行時間が増加し，データ件数 100 万件以内の時 1 秒以内，データ件数 1000 万件のとき 3.5 秒程度の実行時間となることが示された．

以上から，1000 万件の大規模なデータセットに対しても現実的なレスポンス時間を実現可能であると評価できる．

6. 秘密フィッシャー正確検定

本章ではフィッシャー正確検定をシミュレートする秘密計算プロトコルを示し，次に実機による精度・速度の評価

データ件数	実行時間 (秒)
10,000	0.13
100,000	0.18
1,000,000	0.49
10,000,000	3.49

表 7 秘密計算によるオッズ比計算実行時間.

を行う.

6.1 秘密計算プロトコル

本論文における秘密フィッシャー正確検定は次のように問題設定を行った.

(1) 入力は 2 値 2 属性データセット $\mathbf{D} \in \mathcal{D}_2^m$ の分散値と, 有意水準 α とする.

(2) 出力は帰無仮説の棄却判定 $z \in \{0,1\}$ の分散値とする.

Protocol 6 Private array access

Input: 長さ x の配列の分散値 $[\![\mathbf{a}]\!]$, 長さ y の配列の分散値 $[\![\mathbf{i}]\!]$.
Output: 長さ y の配列の分散値 $[\![\mathbf{b}]\!]$; $\mathbf{b}[j] = \mathbf{a}[\mathbf{i}[j]]$

procedure ArrayAccess($[\![\mathbf{a}]\!], [\![\mathbf{i}]\!]$)
  $[\![\mathbf{b}]\!] \leftarrow \textsc{Fill}([\![0]\!], y)$
  for $j = 0$ **to** $y - 1$ **do in parallel**
    $[\![b_j]\!] \leftarrow \sum_{k=0}^{x-1} [\![\mathbf{a}[k]]\!] \times ([\![\mathbf{i}[j]]\!] =_? k)$
    $[\![\mathbf{b}[j]]\!] \leftarrow [\![b_j]\!]$
  end for
  return $[\![\mathbf{b}]\!]$
end procedure

Protocol 7 Private sequential array access

Input: 長さ x の配列の分散値 $[\![\mathbf{a}]\!]$, $-x \le i < x$ である分散値 $[\![i]\!]$, 自然数 S.
Output: 長さ S の配列の分散値 $[\![\mathbf{b}]\!]$; $\mathbf{b}[j] = \mathbf{a}[i + j \mod x]$

procedure SeqArrayAccess($[\![\mathbf{a}]\!], [\![i]\!], S$)
  $[\![\mathbf{b}]\!] \leftarrow \textsc{Fill}([\![0]\!], S)$
  $[\![i']\!] \leftarrow \textsc{LMod}([\![i]\!], x)$
  $[\![\mathbf{a}']\!] \leftarrow \textsc{Rotate}([\![\mathbf{a}]\!], [\![i]\!])$
  for $j = 0$ **to** $S - 1$ **do in parallel**
    $[\![\mathbf{b}[j]]\!] \leftarrow [\![\mathbf{a}'[j \mod x]]\!]$
  end for
  return $[\![\mathbf{b}]\!]$
end procedure

秘密計算を用いたフィッシャー正確検定は [11], [13], [14], [15] で議論されているが, 本論文では特に周辺分布を秘匿したまま計算可能という点で秘匿性に優れる [14] の方式に着目する.

[14] に提案される手法では, 分割表 (a, b, c, d) の分散値を入力としており, 周辺分布を全て秘匿する代わりに標本数の上界 (平文値) をパラメータに用いている. 一方, 本論文ではデータセットを入力として想定するため, 標本の総数 m は公開値と見なす.

また [14] では, 浮動小数点数の秘密計算 [8] に基づく加算・乗算・指数関数を用いているが, 本論文では各演算を有限体 $\mathbb{F}_p$ 上に定義しているため, 浮動小数点数の代わりに小数部 u ビットの固定小数点数を用いる.

以上を踏まえ, [14] のアルゴリズムを踏襲しつつ, 上記の設定においてフィッシャー正確検定を行う秘密計算プロトコルを示す.

Protocol 6 は [14] の配列への単一要素アクセスを, 複数の入力について並列化した演算である. 参照される配列を $\mathbf{a}$, 参照位置を示す値の配列を $\mathbf{i}$ として, 全ての位置 $\mathbf{i}[j]$ の要素 $\mathbf{a}[\mathbf{i}[j]]$ を線形探索により取得する.

Protocol 7 は [14] の配列への一括アクセスである. Protocol 6 と同様, この処理も $[\![i]\!]$ を配列とすることで並列化することができるが, 本論文では表記の簡潔化のため単独の $[\![i]\!]$ を入力とするよう記述する.

Protocol 8 Secure Fisher's exact test.

Input: 2 値 2 属性データセット $\mathbf{D} \in \mathcal{D}_2^m$ の分散値 $[\![\mathbf{D}]\!]$, 有意水準 α.
Output: 帰無仮説の棄却判定 $z \in \{0,1\}$ の分散値 $[\![z]\!]$.

$h \leftarrow \lfloor m/2 \rfloor$
$[\![\mathbf{t}]\!] := ([\![a]\!], [\![b]\!], [\![c]\!], [\![d]\!], [\![n_{1,*}]\!], [\![n_{0,*}]\!], [\![n_{*,1}]\!], [\![n_{*,0}]\!]) \leftarrow \textsc{Count}([\![\mathbf{D}]\!])$
$f(x) := \log(x!)$
$[\![\mathbf{f_x}]\!] \leftarrow ([\![f(0)]\!], \ldots, [\![f(m)]\!]) \parallel \textsc{Fill}([\![-\perp]\!], h)$
$([\![f_a]\!], [\![f_b]\!], [\![f_c]\!], [\![f_d]\!], [\![f_{n_{1,*}}]\!], [\![f_{n_{0,*}}]\!], [\![f_{n_{*,1}}]\!], [\![f_{n_{*,0}}]\!]) \leftarrow \textsc{ArrayAccess}([\![\mathbf{f_x}]\!], [\![\mathbf{t}]\!])$
$[\![\log p_*]\!] \leftarrow [\![f_{n_{1,*}}]\!] + [\![f_{n_{0,*}}]\!] + [\![f_{n_{*,1}}]\!] + [\![f_{n_{*,0}}]\!] - \big([\![f_a]\!] + [\![f_b]\!] + [\![f_c]\!] + [\![f_d]\!] + f(m)\big)$
$[\![p_*]\!] \leftarrow \textsc{Exp}([\![\log p_*]\!])$
$[\![i]\!] \leftarrow \textsc{IfElse}([\![a]\!] \le_? [\![d]\!], [\![a]\!], [\![d]\!])$ ▷ $i = \min(a, d)$
$[\![\mathbf{f_{a'}}]\!] \leftarrow \textsc{SeqArrayAccess}([\![\mathbf{f_x}]\!], [\![a]\!] - [\![i]\!], h + 1)$
$[\![\mathbf{f_{b'}}]\!] \leftarrow \textsc{SeqArrayAccess}([\![\mathbf{f_x}]\!], [\![b]\!] + [\![i]\!] - h, h + 1)$
$[\![\mathbf{f_{c'}}]\!] \leftarrow \textsc{SeqArrayAccess}([\![\mathbf{f_x}]\!], [\![c]\!] - [\![i]\!] - h, h + 1)$
$[\![\mathbf{f_{d'}}]\!] \leftarrow \textsc{SeqArrayAccess}([\![\mathbf{f_x}]\!], [\![d]\!] - [\![i]\!], h + 1)$
for $j = 0$ **to** h **do in parallel**
  $[\![\log p_j]\!] \leftarrow [\![f_{n_{1,*}}]\!] + [\![f_{n_{0,*}}]\!] + [\![f_{n_{*,1}}]\!] + [\![f_{n_{*,0}}]\!] - \big([\![\mathbf{f_{a'}}[j]]\!] + [\![\mathbf{f_{b'}}[h-j]]\!] + [\![\mathbf{f_{c'}}[h-j]]\!] + [\![\mathbf{f_{d'}}[j]]\!] + f(m)\big)$
  $[\![p_j]\!] \leftarrow \textsc{Exp}([\![\log p_j]\!])$
  $[\![p_j]\!] \leftarrow \textsc{IfElse}([\![p_j]\!] \le_? [\![p_*]\!], [\![p_j]\!], [\![0]\!])$
end for
$[\![p]\!] \leftarrow \sum_{j=0}^{h} [\![p_j]\!]$ ▷ $p_j \le p_*$ となる p_j のみ p 値に加算
return $[\![z]\!] \leftarrow [\![p]\!] \le_? \alpha$

Protocol 8 は Protocol 4,6,7 をサブルーチンとして, フィッシャー正確検定を行うプロトコルである. 入力は 2 値 2 属性データセットの分散値と有意水準 α であり, 出力は p 値と α の比較による検定結果の分散値である.

6.2 秘密フィッシャー正確検定の性能評価実験

本節では, Protocol 8 を実機上で実行した際の, 精度・速度の評価を行う. ただしここでは, 1 万件〜100 万件までのダミーデータを用いて実験した.

データ件数	p 値 (R)	p 値 (秘密計算)
10,000	0.1498637	0.1498634
100,000	0.3492583	0.3492578
1,000,000	0.4727567	0.4727554

表 8 R と秘密計算によるフィッシャー正確検定 p 値.

データ件数	実行時間 (秒)
10,000	0.76
100,000	5.44
1,000,000	266

表 9 秘密計算によるフィッシャー正確検定実行時間.

6.2.1 計算精度

表 8 に，R のライブラリに実装されているフィッシャー正確検定を実行した結果 (p 値) と，Protocol 8 の実行により与えられる p 値との誤差を比較する．この実験では，秘密計算における小数精度 $u = 26$ ビットとした．

結果から，本論文の秘密フィッシャー正確検定はデータ件数の増加に合わせて精度が落ちることがわかる．しかしながら，100 万件のデータに対しても小数点以下第 6 位までに誤差を抑えており，検定結果の有意性は十分保証できる．

以上により，本論文の秘密フィッシャー正確検定は制度において十分な実用性を持つと評価できる．

6.2.2 計算速度

表 9 に，Protocol 8 の実行にかかる時間を示す．

結果から，本論文の秘密フィッシャー正確検定はデータ件数の増加に合わせて非線形に計算時間が増加することがわかる．特に．100 万件データで大きく実行時間が伸びてしまっている．

しかしながら実験結果は，100 万件のデータであっても 5 分以内という現実的な時間で計算可能であることも示唆しており，本論文の秘密フィッシャー正確検定が十分実用に足ることを示している．

7. 結論

本論文ではログランク検定，オッズ比，フィッシャー正確検定という 3 つの統計分析手法について，秘密計算によりデータを秘匿したまま分析を行う方法を示した．

また実験により，各演算の実際の計算精度・計算速度を測定し，秘密計算が大規模データに対して十分実用的な精度と速度を実現できることを示した．

今後の展望は医療統計などで多く使われる他の分析手法について，同様に秘密計算を用いた実用的な演算方法を考案・評価することが考えられる．具体例としては，Cox 比例ハザードモデルなどが挙げられる．

参考文献

[1] R.A. Fisher: On the interpretation of χ^2 from contingency tables, and the calculation of P. Journal of the Royal Statistical Society 85(1), 87–94 (1922).

[2] E. L. Kaplan, P. Meier: Nonparametric estimation from incomplete observations. Journal of the American Statistical Association 53, 457–481

[3] N. Mantel: Evaluation of survival data and two new rank order statistics arising in its consideration. Cancer Chemotherapy Reports 50, 163–170.

[4] R. Peto, J. Peto: Asymptotically efficient rank invariant test procedure. Journal of the Royal Statistical Society. Series A135, 185–207.

[5] A. Shamir: How to share a secret. Commun. ACM, Vol. 22, No. 11, pp. 612–613, 1979.

[6] A. C. Yao. How to Generate and Exchange Secrets. In Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 162–167, 1986.

[7] M. Ito, A. Saito, and T. Nishizeki: Secret sharing schemes realizing general access structures. Proceedings of the IEEE Global Telecommunication Conference, Globecom 87, pp. 99–102, 1987.

[8] M. Aliasgari, M. Blanton, Y. Zhang and A. Steele: Secure Computation on Floating Point Numbers, 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24–27, 2013, The Internet Society (2013).

[9] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo: A new way to protect privacy in large-scale genome-wide association studies, Bioinformatics, Vol. 29, No. 7, pp. 886–893 (online), DOI:10.1093/bioinformatics/btt066 (2013).

[10] W.-J. Lu, Y. Yamada, and J. Sakuma: Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption, BMC medical informatics and decision making, Vol. 15, No. Suppl 5, p. S1 (2015).

[11] K. Hamada, S. Hasegawa, K. Misawa, K. Chida, S. Ogishima, and M. Nagasaki: Privacy-preserving fisher's exact test for genome-wide association study. International Workshop on Genome Privacy and Security (GenoPri), 2017.

[12] 濱田浩気，桐淵直人，五十嵐大：ラウンド効率のよい秘密計算パターンマッチング，コンピュータセキュリティシンポジウム 2014 論文集， Vol. 2014, No. 2, pp. 674–681 (2014).

[13] 千田浩司，長谷川聡，濱田浩気，荻島創一，三澤計治，長崎正朗：秘密計算フィッシャー正確検定 (1) ～標本数が少ない場合，第 74 回コンピュータセキュリティ研究会 (CSEC) 予稿集 (2016)

[14] 濱田浩気，長谷川聡，千田浩司，荻島創一，三澤計治，長崎正朗：秘密計算フィッシャー正確検定 (2) ～標本数が多い場合，第 74 回コンピュータセキュリティ研究会 (CSEC) 予稿集 (2016)

[15] 長谷川聡，濱田浩気，千田浩司，荻島創一，三澤計治，長崎正朗：プライバシ保護ゲノム解析のための秘密計算フィッシャー正確検定，第 74 回コンピュータセキュリティ研究会 (CSEC) 予稿集 (2016)

[16] 桐淵直人，五十嵐大，濱田浩気，菊池亮：プログラマブルな秘密計算ライブラリ MEVAL3，暗号と情報セキュリティシンポジウム (SCIS)2018 予稿集 (2018).

[17] 三品気吹，濱田浩気，菊池亮，五十嵐大：秘密計算によるロジスティック回帰は本当に使えるか？，暗号と情報セキュリティシンポジウム (SCIS)2019 予稿集 (2019).

[18] 五十嵐大：逆数の秘密計算，コンピュータセキュリティシンポジウム (CSS)2019 予稿集 (2019).