秘密計算の発展

—データを隠しつつ計算する仕組みとその発展—

Progress of Secure Computation:
Basic Constructions and Dedicated Algorithms

菊池 亮 Ryo KIKUCHI
五十嵐 大 Dai IKARASHI

アブストラクト データを隠したまま任意の関数を計算する秘密計算 (secure computation) は、1980 年代から始まった暗号技術の一大研究分野である。秘密計算を用いれば、例えば個人の病歴は秘匿しつつ健康指導を行ったり、通信内容は秘匿しつつ異常検知を行うなど、今まではプライバシーの問題や企業の秘密のために外に出すことが難しかったデータを、安全に流通させることができる。秘密計算は近年に至るまで、理論研究の発展に比べ、その社会実装は限定的であった。その大きな理由の一つはその計算速度の遅さである。「データを隠しながら計算する」という性質上、データを秘匿するオーバーヘッドに加え、通常の計算機で用いるような高速なアルゴリズムを秘密計算ではそのまま用いることができず、結果として秘密計算の計算速度は非常に遅くなってしまっていたのである。しかし近年、計算機やネットワークの性能向上に加え、高速な秘密計算用のアルゴリズムの研究開発が進み、秘密計算の速度は飛躍的に向上し、今正に研究レベルから実用レベルに移ろうとしている。本稿では、この秘密計算について、なぜ秘匿したまま計算ができるのか、その原理を解説するとともに、近年に至るまでの速度向上の取組みについて紹介する。

キーワード 秘密計算, 秘密分散, Garbled circuit, 準同形暗号

Abstract Secure computation has been a major research topic in cryptography since the 1980s. It can accelerate the application of sensitive data since one can analyze data while ensuring their secrecy by using secure computation. Despite this advantage of secure computation, it has not been widely used in practice, one of the main reasons being its inefficiency. Secure computation tends to require heavy computation and communication compared with computation in the clear. In addition, we cannot use an efficient algorithm in secure computation without careful consideration of how not to violate the security. However, the performance of secure computation has been greatly improved recently, and secure computation is expected to be increasingly used in practice. In this paper, we introduce several techniques to carry out secure computation and algorithms to improve the performance of secure computation.

Key words secure computation, multiparty computation, secret sharing, garbled circuit, homomorphic encryption

1. はじめに

皆様は“暗号”というと、秘密にしたいデータを安全に守る技術というイメージを持たれるのではないだろうか。本稿で紹介する「秘密計算」は、そのようなデータを「秘密」にするという機能に加えて、秘密データを「計算」する機能を加えた暗号技術である。この暗号技術を用いることで、データは暗号化されているため誰にも中身は分からないのだが、後述する仕組みにより、暗号化は一度も解くことなく、その計算結果だけを得ることができる。

この技術は、データを安全に秘匿したまま活用する技術とし

て注目を浴びつつある。例えば、個人の病歴と生活習慣を集めてきて、どのような生活習慣と病気に相関があるのか？ ということを調べたいが、個人の病歴と生活習慣は外部に漏えいしないよう、暗号化して保護したいとしよう。普通の暗号化を使う場合、個人の病歴と生活習慣を暗号化してサーバに保存することで、安全にデータを集めることはできるが、相関を分析するためには一度元のデータに戻す必要がある。このとき、サーバの管理者に悪意があったり、またそうでなくてもマルウェアに感染するなどして一時的に悪意ある行動を行う場合、この復号した情報からデータが漏れるリスクがある。一方秘密計算では、暗号化したまま計算ができるため、途中で復号する必要がなく、処理の途中で一度も平文に戻ることはないため、前述した漏えいリスクへの対策となる。

秘密計算は 1980 年代の先進的な成果^{(1)~(4)}から現在に至るまで、暗号における一大分野として積極的に研究されてきており、数多くの実現手段も提案されている。

しかし、秘密計算にはデータの秘匿と活用を両立できるというメリットがあるものの、その社会実装は限定的であった。その大きな理由の一つはその計算速度の遅さである。「データを隠

菊池 亮 日本電信電話株式会社 NTTセキュアプラットフォーム研究所
E-mail kikuchi.ryo@lab.ntt.co.jp
五十嵐 大 日本電信電話株式会社 NTTセキュアプラットフォーム研究所
E-mail ikarashi.dai@lab.ntt.co.jp
Ryo KIKUCHI and Dai IKARASHI, Nonmembers (NTT Secure Platform Laboratories, NIPPON TELEGRAPH AND TELEPHONE CORPORATION, Musashino-shi, 180-8585 Japan).
電子情報通信学会 基礎・境界サイエティ
Fundamentals Review Vol.12 No.1 pp.12-20 2018 年 7 月
©電子情報通信学会 2018

しながら計算する」という性質上、データを秘匿するオーバーヘッドに加え、通常の計算機で用いるような高速なアルゴリズムを秘密計算ではそのまま用いることができず、結果として秘密計算の計算速度は非常に遅くなってしまっていたのである。

しかし近年、計算機やネットワークの性能向上に加え、高速な秘密計算用のアルゴリズムの研究開発が進み、秘密計算の速度は飛躍的に向上し、今正に研究レベルから実用レベルに移ろうとしている。

本稿では、なぜ秘匿したまま計算ができるのか、その原理を様々な実現方法について解説するとともに、各手法の簡単な比較を行う。また、高速な秘密計算を構成するために必要な秘密計算用のアルゴリズムの構成法についても紹介する。

2. 準備

2.1 秘密計算のモデル

秘密計算はデータを秘匿したまま計算を行う技術の総称であり、データの提供者は誰か、秘密計算を実際に行う計算者は誰か、計算結果を得る者は誰か、またそれらは同一なのかそうでないのか、といった観点で幾つものモデルが存在する。本稿ではデータを集めてきて分析する、クライアントサーバモデルを想定する。このモデルでは、クライアント $C_1, \dots, C_m$ がデータを暗号化してサーバ S に提供し、サーバが暗号化したまま計算を行う。サーバは1台であってもよいし複数台であってもよい。計算結果を得る者はサーバ以外の誰かであり、クライアントでもいいし第三者であってもよいが、本稿では簡便のためクライアントの一人である C^* とする。

次に不正に秘密データの情報を得ようとする攻撃者を規定する。秘密計算では、その安全性を規定するために攻撃者の存在を仮定し、想定した攻撃者が攻撃したとしても秘密データの情報が漏れないことをもって安全と定義する。攻撃者はサーバ（の一部）を乗っ取り、秘密情報を得ようとする。攻撃者はその挙動により幾つか場合分けが存在するが、本稿では簡単のため、passive と呼ばれる「正常なサーバのように振る舞うが盗み見のみを行う」攻撃者を想定する。Active と呼ばれる「盗み見だけでなくデータの改ざんも行う」攻撃者を想定するモデルは複雑なため本稿では扱わない。興味のある読者は文献(5)～(7)などを参照されたい^(注1)。

2.2 回路計算と算術演算

計算機が処理を行う関数 f は、一般的に AND, NOT, XOR ゲート等の論理ゲートの組合せで書けることが知られている。そのため、論理ゲートを計算することができれば、それらを組み合わせることでどんな関数 f も計算できることになる。

また、論理ゲートの処理は入力の線形演算及び乗算で表すことができる。 $a, b \in \{0, 1\}$ としたとき、例えば a AND b は ab ,

NOT a は $1 - a$, a XOR b は $a + b - 2ab$ と表すことができる。そのため、線形演算及び乗算が計算できれば、任意の論理ゲートを計算でき、ひいては任意の関数 f が計算可能である。

以上の事実から、秘密計算を構成するとき、データを秘匿にしたまま論理ゲートを計算する、若しくは線形演算と乗算を計算することができれば、それらを組み合わせて、データを秘匿にしたまま任意の関数 f を計算することができる。

3. 様々な秘密計算の実現方法

秘密計算には様々な実現方法が知られている。本章では、これらの実現方法の基礎を紹介し、どのように秘密計算が構成できるのかを説明する。

3.1 秘密分散を使う方法

秘密分散とは、秘密情報をシェアと呼ばれる幾つかの断片に変換する方法である。このとき、あらかじめ決められた幾つかのシェアの組合せからは秘密情報が復元できるが、それ以外のシェアの組からは秘密の情報が漏れないようにシェアが生成される。秘密計算では特に (k, n) しきい値法と呼ばれる、 n 個のシェアを生成し、 k 個以上のシェアからは秘密が復元できるが、 k 個未満のシェアからは秘密の情報が全く漏れない方法がよく用いられるため、本稿での秘密分散は (k, n) しきい値法を指すものとする。

秘密分散の具体的構成は Shamir の方式⁽⁸⁾が有名である。これは $k - 1$ 次多項式が、 k 点が決まれば多項式が一意に決定すること、及び $k - 1$ 点では全く定まらないことを利用している。以下では $\mathcal{F}$ を要素数が $\lceil \log n \rceil + 1$ 以上であるような体とし、演算は全て $\mathcal{F}$ 上で行われるものとする（全ての演算で $\text{mod } p$ を行うと考えてもよい。ただし p は素数）。Shamir の方式では下記のような方法で n 個のシェアを生成する。

- (1) データ a を切片とする $k - 1$ 次多項式をランダムに選び、これを $g_a(x)$ とする。すなわち $1 \leq i \leq k - 1$ について r_i を $\mathcal{F}$ からランダムに選び $g_a(x) := a + \sum_{i=1}^{k-1} r_i x^i$ とする。
- (2) $1 \leq i \leq n$ について、 i 番目のシェアを $g_a(i)$ とする。

これが秘密分散になっていることを確認する。 $k - 1$ 個のシェアしか得られない場合、 $k - 1$ 次多項式の $k - 1$ 点しか得られないため、多項式は一意に決まらず、多項式の切片、すなわち元のデータ a の情報は分らない。また、 k 個集まれば多項式が一意に決定するので、切片 a を求めることができ、元のデータ a を復元することができる。特にシェア列 $(g_a(1), \dots, g_a(n))$ を $[a]$ と書き、各シェア $g_a(j)$ を $[a]_j$ と書く。

上記の秘密分散を用いた秘密計算では、サーバは $S_1, \dots, S_n$ の n 台存在し、以下のように行われる。

- (1) 各クライアント C_i はデータ a_i からシェア $[a_i]$ を

(注1) : Passive は semi-honest 若しくは honest-but-curious, active は malicious と呼ばれることに注意。

生成し、各サーバ S_j に $[a_i]_j$ を送る。結果、各 S_j は $([a_1]_j, \dots, [a_m]_j)$ を持つ。

- (2) C^* が計算したい関数 f を全てのサーバに送る。
- (3) 全てのサーバ S_j は $([a_1]_j, \dots, [a_m]_j)$ を入力として秘密計算を行い、 $[f(a_1, \dots, a_m)]_j$ を得る。
- (4) 全てのサーバは計算結果のシェア $[f(a_1, \dots, a_m)]_j$ を C^* に送る。
- (5) C^* は受け取ったシェアから $f(a_1, \dots, a_m)$ を復元する。

サーバは秘密分散されたデータしか得られないため、 n 台のうち $k-1$ 台が攻撃者によって乗っ取られたとしても元データの情報は漏れない。

次に上記 (3) の具体的な秘密計算の処理を説明する。前節でも述べたとおり、線形演算と乗算が計算できれば任意の回路が計算できるため、ここではそれら二つの演算のみを扱う。

秘密分散されたデータの線形演算は簡単に計算できる。各 S_j に $[a]_j$ と $[b]_j$ 及び定数 c, d が与えられたとき、 $c[a]_j + d[b]_j = [ca + db]_j$ が成り立つため、各 S_j は自身の持つシェアの線形演算を行うだけで、元データの線形演算を行うことができる。

一方で乗算は簡単ではない。単純にシェア同士の乗算 $[a]_i[b]_i$ を計算した場合、各シェアは $k-1$ 次多項式の点であるため、 $[a]_i[b]_i$ は $2k-2$ 次多項式の点となり、復元のための多項式の決定には $2k-1$ 個のシェアが必要となる。また、一度乗算を行ったシェア同士で更に乗算を行うと、次は $4k-4$ 次多項式の点となる。このように多項式の次数が乗算により大きくなってしまいう一方、シェア（多項式の点）は n 個しか作られていないため、どこかで復元が不可能になってしまう。

これを回避する方法には多項式の次数を下げる必要がある。以下では $2k-2 < n$ を仮定し、また簡単のため $n = 2k-1$ とする。また、多項式の点から他の点を効率的に計算する方法としてラグランジュ補間が知られており、 $1 \leq i \leq 2k-1$ について定数 λ_i を用いて $ab = \sum_{i=1}^{2k-1} \lambda_i [a]_i [b]_i$ と書けることを利用する。

各 S_j が $[a]_j$ と $[b]_j$ を持っているとき、乗算結果 $[ab]_j$ は以下のように計算される。

- (1) 各 S_j は $\lambda_j [a]_j [b]_j$ を入力としてシェア $[\lambda_j [a]_j [b]_j]$ を生成し、シェアを各サーバに送る。
- (2) 各 S_j は受け取った $[\lambda_1 [a]_1 [b]_1]_j, \dots, [\lambda_n [a]_n [b]_n]_j$ を足し合わせ、 $[\sum_{i=1}^{2k-1} \lambda_i [a]_i [b]_i]_j = [ab]_j$ を得る。

シェアの加算はそのまま元データの加算となっていることから、最後の等式が成り立つ。また各サーバはシェアしか受け取っていないことから、やはり攻撃者によって $k-1$ 台が乗っ取られたとしても元データの情報は漏れていない。

以上から、元データを隠したまま線形演算及び乗算が可能であるため、任意の f が計算できることが分かる。

興味ある読者のために発展的な内容として、例に挙げた乗算は文献(9)であるが、サーバ数 n が大きいときに通信量が少ない乗算⁽¹⁰⁾も知られている。また高速な秘密計算では Shamir 方

式以外にも複製秘密分散 (replicated secret sharing)^{(11), (12)} が用いられることも多い。複製秘密分散は基本的にサーバ数が少ないときにしか用いることはできないが、効率的な乗算が知られている^{(13), (14)}。

3.2 Garbled circuit を使う方法

Garbled circuit⁽⁴⁾とは、通常の回路と異なり、入力は 0, 1 ではなく、0, 1 に対応したラベル K_0, K_1 を用いて計算を行うような回路である。

例として、AND ゲート一つの garbled circuit の作り方を紹介する。この手法ではハッシュ関数 $H: \{0, 1\}^e \times \{0, 1\}^e \rightarrow \{0, 1\}^e$ を構成要素として用い、ハッシュ関数は入力を変えたときに出力が十分ランダムに振る舞うものとする^(注2)。

- (1) ラベルの生成: AND ゲートにつながる左入力ワイヤを A , 右入力ワイヤを B , 出力ワイヤを C としたとき、全てのワイヤに 0 と 1 に対応する e ビットのラベルを、最下位ビットが異なるように生成する。例えば A のラベルを (K_0^A, K_1^A) としたとき、 $e-1$ ビット列を二つ、1 ビット b_A を一つランダムに選び、片方の $e-1$ ビット列に b_A を結合したものを K_0^A , もう片方の $e-1$ ビット列に $(1-b_A)$ を結合したものを K_1^A とする。 B, C も同様である。
- (2) Garbled AND ゲートの生成: Garbler は、四つのゲート暗号文を生成する。ただし $\oplus$ はビットごとの XOR 演算を意味する。
 - $H(K_0^A, K_0^B) \oplus K_0^C$
 - $H(K_0^A, K_1^B) \oplus K_0^C$
 - $H(K_1^A, K_0^B) \oplus K_0^C$
 - $H(K_1^A, K_1^B) \oplus K_1^C$
- (3) 暗号文のシャッフル: $\text{LSB}(K_0^A) = 1$ ならば四つの暗号文の上二つと下二つを入れ替える。更に $\text{LSB}(K_0^B) = 1$ ならば、一つ目と二つ目、及び三つ目と四つ目の暗号文を入れ替える。ここで LSB はビット列の最下位ビットを取り出す関数とする。

上記の garbled circuit ではどのように計算が行われるか、 $a \text{ AND } b = c$ を計算する例を用いて説明する。入力は (K_a^A, K_b^B) 及びゲート暗号文四つであり、四つのゲート暗号文のうち上から $2\text{LSB}(K_a^A) + \text{LSB}(K_b^B)$ 番目の暗号文を D としたとき、 $H(K_a^A, K_b^B) \oplus D$ を計算する。このとき、 $K_c^C = H(K_a^A, K_b^B) \oplus D$ であり、 $a \text{ AND } b = c$ が成り立っている。四つの暗号文の K_0^C の並び方がそのままゲートの真理値表になっているので、並び順を入れ替えれば任意のゲートを対象とできる。例えば上から順に $K_0^C, K_1^C, K_1^C, K_0^C$ とすれば XOR ゲートの garbled circuit となる。回路がより深い場合は (K_0^C, K_1^C) を入力ワイヤのラベルと見て、順にゲートごとにゲート暗号文を作ればよい。

(注2): 正確には擬似ランダム関数で十分である。

Garbled circuit を用いた秘密計算は、以下のように構成できる。

- (1) C^* が計算したい関数 f の garbled circuit を作り、各ゲート暗号文をサーバに送る。
- (2) C^* は f の全ての入力ワイヤについて、そのワイヤの入力を持つクライアントに、当該のワイヤの 0/1 のラベルを送る。
- (3) 各クライアント C_i は、自身の入力 a_i の各ビットに対応する、0/1 どちらか片方のラベルをサーバに送る。
- (4) サーバは得たラベルから garbled circuit を計算し、出力ワイヤのラベルを得て、 C^* に送る。
- (5) C^* はサーバから得たラベルと自分が生成した出力ワイヤの 0/1 のラベルを比較し、 $f(a_1, \dots, a_m)$ を得る。

ラベル及び各ゲート暗号文は、各クライアントのデータに依存せずに生成されているため、サーバはデータの情報を得ることはできない。そのため、上記の方法はデータを隠しながら任意の論理ゲートを計算できるため、任意の f を計算できる。

より実用的には、garbled circuit の生成処理に時間が掛かるため、別のサーバを用意し、代理で garbled circuit を生成すること⁽¹⁵⁾や、ゲート暗号文を削減する研究^{(16)~(18)}などが行われている。また、サーバ側にも入力がある場合には紛失通信⁽¹⁹⁾、⁽²⁰⁾を用いることで計算が可能である。

3.3 準同形暗号を使う方法

準同形暗号とは、暗号文のままで平文の加算若しくは乗算ができる暗号である。ここでは便宜的に加算 (+) が計算できるとしよう。すなわち、 a, b を平文、 pk を公開鍵、 sk を秘密鍵、 Enc を暗号化アルゴリズム、 Dec を復号アルゴリズムとしたとき、準同形演算 田 が存在し、 $\text{Enc}_{pk}(a)$ と $\text{Enc}_{pk}(b)$ から新しい暗号文 $\text{Enc}_{pk}(a) \text{ 田 } \text{Enc}_{pk}(b)$ を作成することができ、これは $\text{Dec}_{sk}(\text{Enc}_{pk}(a) \text{ 田 } \text{Enc}_{pk}(b)) = a + b$ を満たす。また、加算を繰り返せば、 $\text{Enc}_{pk}(a)$ と定数 c が与えられたとき ca の暗号文（以下では簡単のためこれを $\text{Enc}_{pk}(ca)$ と書く）も作成することができる^(注3)。具体的な構成としては ElGamal 暗号⁽²¹⁾や Paillier 暗号⁽²²⁾が知られている。

準同形暗号を用いて秘密計算を行うとき、線形演算は暗号化したままでも計算することができるが、乗算は暗号化したままでは計算できないため、特別な工夫が必要となる。ここでは、暗号化したまま乗算を計算するため、準同形暗号を変形しきい値準同形暗号に置き換える。変形しきい値準同形暗号では暗号化したまま加算が計算できるという性質はそのままに、秘密鍵 sk が $sk_1, \dots, sk_n$ に分割されてそれぞれサーバ $S_1, \dots, S_n$ に保存され、復号はサーバ全てが協力することで、クライアント若しくはサーバのうち指定した一人のみが復号結果を得られる分散復号になる。

変形しきい値準同形暗号を用いた秘密計算は以下となる。

- (1) 各クライアント C_i は入力 a_i を暗号化し暗号文 $\text{Enc}_{pk}(a_i)$ を全サーバに送る。
- (2) C^* が関数 f のリクエストをサーバに送る。
- (3) サーバは暗号化したまま f を計算し $f(a_1, \dots, a_m)$ の暗号文を得る。
- (4) サーバは前述の分散復号を用いて C^* に $f(a_1, \dots, a_m)$ を渡す。

具体的な f の計算方法について説明する。線形演算は前述したとおり計算できる。乗算は、各サーバが $\text{Enc}_{pk}(a)$, $\text{Enc}_{pk}(b)$ を持っているとき、下記のように計算できる。

- (1) 各サーバ S_i は乱数の暗号文 $\text{Enc}_{pk}(d_i)$ を計算し全サーバへ送る。
- (2) 各サーバ S_i は乱数の暗号文 $\text{Enc}_{pk}(a)$ 田 $\text{Enc}_{pk}(d_1)$ 田 $\dots$ 田 $\text{Enc}_{pk}(d_n)$ を計算する。
- (3) サーバは分散復号を行い、 S_1 に $a + \sum_{i=1}^n d_i$ を渡す。
- (4) S_1 は $a_1 := (a + \sum_{i=1}^n d_i) - d_1$ 、それ以外の S_i は $a_i := -d_i$ とする。
- (5) 各サーバ S_i は $\text{Enc}_{pk}(b)$ と a_i から $\text{Enc}_{pk}(a_i b)$ の暗号文を作成し全サーバへ送る。
- (6) 各サーバは $\text{Enc}_{pk}(ab) := \text{Enc}_{pk}(a_1 b) \text{ 田 } \dots \text{ 田 } \text{Enc}_{pk}(a_n b)$ を得る。

以上から、線形演算と乗算が計算できるため、準同形暗号を用いて任意の f を秘密に計算することができる。

3.4 完全準同形暗号を使う方法

完全準同形暗号とは、暗号文のままで平文の線形演算と乗算の両方が計算可能な暗号である。完全準同形暗号は 2009 年に Gentry⁽²³⁾ によって提案されて以降、活発に研究がなされてきている。正式な定義や具体的な構成については本誌小特集⁽²⁴⁾を御覧頂きたい。

完全準同形暗号では暗号化したまま任意の関数 f を計算するため、シンプルに秘密計算を実現できる。まずクライアントは自身の入力 a_i を暗号化して 1 台のサーバに送信する。サーバはクライアントから得た暗号文を使って、暗号化したまま f を計算し $f(a_1, \dots, a_m)$ の暗号文を得て、クライアントに送信する。最後にクライアントは受信した暗号文を復号することで、所望の結果 $f(a_1, \dots, a_m)$ を得ることができる。

3.5 カードを用いた秘密計算について

カードを用いた秘密計算という手法も知られている。これはトランプゲームで使うような物理的なカード組を利用して、秘密計算を実現するものであり、本稿で説明してきたような多量のデータを分析することに使う秘密計算とは目的が異なる。実際にカードを入れ替えたりめくったりするため処理速度は非常に遅いが、計算機ネットワークを必要とせず手軽に実行がで

(注3)：実際には 田 を用いずとも簡単に定数倍を計算できることが多い。

き、その正当性や安全性が高校生等にも理解しやすいという利点がある。本稿ではこれ以上言及しないが、興味のある読者は文献(25)を参照されたい。

4. それぞれの方法の比較

これまで幾つかの秘密計算の実現方法を紹介した。すると「どの方法を用いればいいだろうか？」という疑問が湧くのは自然なことであろう。しかし一般に、秘密計算には2.で述べたように様々なモデルがあり、またどのような関数 f を計算するのか、どのような計算機環境なのかによっても適切な実現手段は変わってくるため、一概に異なる方式同士を比較することは簡単ではない。

そこで本稿では、「多数の情報を集めてきて分析する」というクライアントサーバモデルに限定し、passive と呼ばれる最も性能が出やすい攻撃者を想定した場合の性能を比較した。計算機環境はサーバは通常のデスクトップ PC 程度で、サーバが複数存在する場合の通信路は 10 Gbit/s の完全ネットワークとした。関数 f として、基礎的な演算として論理回路、より高級な演算としてソートを挙げ、それぞれの場合について比較した。ソートは中央値や四分位数のような統計値を計算するときには有用だけでなく、集計表の作成やテーブル参照などの重要な処理の構成要素に用いられており、秘密計算で高速に実現する価値が高い演算である。更に、サーバで保存されるデータが元データの何倍になるのかという総データ量でも比較した。

各実現方法を用いた場合の性能は、前章で説明した基礎的な手法を発展させた、より効率的な手法を用いて見積もった。秘密分散を用いた方法は、各々のデータを3台のサーバに分散し、1台のサーバが攻撃者に乗っ取られたとしても安全であるような設定で、筆者らの実装⁽²⁶⁾の実測値を用いて比較した。Garbled circuit を用いた方法では、近年のゲート暗号文が少ない方法⁽¹⁸⁾を用い、一つ一つのラベル及び暗号文のサイズは 160 bit とし、garbled circuit の生成をサーバが行いつつ紛失通信を使わない手法⁽²⁷⁾で通信量を見積もり、性能は実測値ではなく通信量から計算した理論値とした。準同形暗号を用いた方法はだ円 ElGamal 暗号を用いる手法⁽²⁸⁾で、群要素の表現に必要なビット数を 256 bit とし通信量を見積もり、こちらも性能は通信量から計算した理論値とした。完全準同形暗号は通信量では比較しにくいので、ゲート性能を実測値(マシン1台のフルパワー例として文献(29)記載の level 1 方式のシングルスレッド実測値の8倍速とした)とした。

まず論理回路の計算能力を表1にまとめた。スループットは1秒間に処理できるゲート数であり、大きいほど性能が高い。秘密分散を用いた場合、論理回路の計算では、AND ゲート1個の計算に1パーティ当たり1bitの送受信で済み^{(13), (14)}、双方向で送り合うことで二つ同時に処理ができる。Garbled circuit を用いた場合、AND ゲート一つの処理に160 bit のハッシュ値を2個送信するため、計320 bit と見積もった。準同形暗号を用いた場合、1パーティ当たり256 bit の暗号文を6個送信として、計1,536 bit である。ただし秘密分散と garbled circuit では XOR

表 1 論理回路の処理能力

用いる実現方法	通信量 (bit)	スループット (gate/s)
秘密分散	AND 3 (XOR 0)	18 G
Garbled circuit	AND 320 (XOR 0)	31.3 M
準同形暗号	AND 1,536	6.5 M
完全準同形暗号	なし	630 k

表 2 20 bit 100 万件データのソート

用いる実現方法	通信量 (bit)	処理時間 (s)
秘密分散	(複雑なので割愛)	1.1
Garbled circuit	2.43 T	243
準同形暗号	35.64 T	3,564
完全準同形暗号	0	48,877

表 3 サーバで保存される総データ量

用いる実現方法	平文と比較したデータサイズ
秘密分散	3 倍
Garbled circuit	2 倍
準同形暗号	1,024 倍
完全準同形暗号	64 倍

は通信不要である。完全準同形暗号では1,024個の並列論理演算を13 msで行えることから見積もった。

次に複雑な処理としてソートを例に挙げ、20 bit, 100 万件のデータ(キー)をソートする際の通信量と処理時間の比較を表2にまとめた。秘密分散では五十嵐らの基数ソート⁽³⁰⁾の簡易版、他の方式ではソーティングネットワークで評価した。ソーティングネットワークではソート対象のデータ数 m に対し $m \log^2 m$ 回の比較回路を実行するものであり、データ数100万としたとき、比較回路数は4億である。実データ長20 bit のとき、通信量と処理時間は表2のようになる。

最後に、サーバが平文と比較して何倍のデータを保持する必要があるのかを表3にまとめた。秘密分散を用いた方式では Shamir の秘密分散によりデータを保存し、3台の各サーバが持つデータは平文と同じサイズであるため、総データ量は3倍となる。Garbled circuit を用いた方式では1 bit の平文を1 bit ずつ2サーバに分散保存するため、2倍となる。準同形暗号では512 bit の暗号文を2サーバが保存し、1,024倍となる。完全準同形暗号では文献(29)のパラメータでは64倍となる。

これらの性能の比較をまとめると、演算の処理速度は

秘密分散 > Garbled circuit > 準同形暗号 > 完全準同形暗号

の順であり、それぞれの比も近いものではなく、10~1,000倍程度の大きな差であり、秘密分散と完全準同形暗号では4,5桁ほどの差である。ただし、完全準同形暗号は並列処理で性能が向上する。例えば8スレッドではなく128スレッドマシンを100台用意すると、性能は1,600倍に向上するが、それでもまだ1桁ほど秘密分散ベースが高速である。データ量に関しては秘密分散と garbled circuit が良好である。

5. 高速な秘密計算のためのアルゴリズム

前章の設定において、現在最も高速な秘密分散を用いた秘密

計算の論理回路のスループットは 18 G gate/s であり、一定のレベルに達していると言える。しかし、論理回路は確かに任意の関数 f を計算可能であるが、その効率までは言及していないため、 f を論理回路で計算すると回路が非常に大きくなってしまい、長大な計算時間が掛かってしまう場合がある。このような場合は f を計算する秘密計算用のアルゴリズムを専用に設計する必要がある。

高速な秘密計算のアルゴリズムでは、通常の計算機でのアルゴリズムとは少々異なる観点が必要となる。単純な例として、秘密分散を用いた場合、線形演算は各サーバが自身のシェアを用いて演算するだけでよいが、乗算には通信が必要となるため線形演算との速度差の開きが大きい。そのため、できる限り乗算を線形演算で置き換えたり、通信の遅延の影響を少なくするために、乗算を並列化し同時に通信する、といった工夫が必要となる。このような単純な例以外にも、秘密計算では幾つか考慮すべき観点が存在する。本章では秘密分散を用いた秘密計算に限定して、そのような高速化のための秘密計算のアルゴリズムの設計方法を紹介する。

5.1 秘密計算だと非自明になる処理：ビット分解

秘密分散を用いて整数のデータを分散するとき、整数として分散する（整数表現）か、それともビット列として分散する（ビット表現）かという二つの選択肢がある。例えば 13 の整数表現のシェアは [13]、ビット表現のそれは [1], [1], [0], [1] である。整数表現であれば、整数上の加算は通信なしで処理することができるが、整数の値のまま他の値との大小比較や（ビット列の）ハミング距離を計算する方法は非自明である。一方でビット表現の場合、大小比較やハミング距離はそれらを計算する回路を用いて秘密計算を行えばよいが、加算には通信を伴う加算回路を計算する必要がある、整数として分散した場合に比べ性能が落ちてしまう。

以上のように処理によって整数表現とビット表現のどちらがよいかは異なるため、処理によって両者を変換できるとよい。このような二つの表現の相互変換は平文では自明であるが、秘密計算ではどうだろうか。

秘密分散に用いる体がどちらの表現でも同一だと仮定すると、ビット表現から整数表現は下記のように簡単に変換できる。

$$[13] = 2^3[1] + 2^2[1] + 2[0] + [1].$$

しかし、整数表現からビット表現は自明ではない。秘密計算にはこのように平文では自明な処理が非自明になることがある。秘密計算においてこのような整数表現からビット表現への変換方法はビット分解と呼ばれ、一つの研究テーマとなっている。

では実際にどのようにビット分解が行われるのか、文献(31)の簡易版を用いて説明する。基本的な発想は、平文であればビット分解が自明に可能であることを利用して、一度平文に戻し、ビット分解し、分散値に戻すという処理を行う。このとき、単純に平文に戻すと秘密のデータが漏れてしまうため、乱数でマスクしてから平文に戻し、ビット表現に直した後、乱数のマス

クを外す、という処理が行われる。以下では全ての要素は ℓ ビットの素数 p について $\mathbb{Z}/p\mathbb{Z}$ に属する。

- (1) 乱数のビット表現と整数表現のシェアを得る。すなわち、ある乱数 r とそのビット表現 $r_\ell, \dots, r_1$ について、 $[r]$ と $([r_\ell], \dots, [r_1])$ を生成する。
- (2) サーバは $[a] - [r]$ を計算し、復元して $a' := a - r$ を得る。 a' のビット表現を $(a'_\ell, \dots, a'_1)$ とする。
- (3) $(a'_\ell, \dots, a'_1)$ と $([r_\ell], \dots, [r_1])$ を入力として加算回路を計算し、その結果を $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ とする。
- (4) $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ と p で比較回路を計算し、 $p \leq \sum_{i=1}^{\ell} 2^{i-1} \tilde{a}_i$ ならば [1]、そうでなければ [0] となるような $[c]$ を得る。
- (5) $[c]$ と p のビット表現 $([p_\ell], \dots, [p_1])$ を掛け合わせて $([p'_\ell], \dots, [p'_1])$ を得る。
- (6) $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ から $([p'_\ell], \dots, [p'_1])$ を、減算回路を使って引き、その結果を $([a_\ell], \dots, [a_1])$ とする。

ステップ (1) の乱数生成の方法は文献(31)を参照されたい。ステップ (3) 時点で乱数を引いて足しているの、 $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ は a のビット表現になっているように見えるが、加算回路での加算は $\text{mod } 2^\ell$ 上での加算になっているため、 $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ は $a + p$ のビット表現となっている場合がある。例えば具体的に $p = 5$, $\ell = 3$, $a = 1$, $r = 3$ のとき、 $([\tilde{a}_\ell], \dots, [\tilde{a}_1])$ は $[1], [1], [0]$ になっており、これは $a + p$ のビット表現である。このような場合、ステップ (4) 以降で p が減算され、正しいビット分解の結果が得られている。より高速なビット分解方法は文献(32)及び(33)を参照されたい。

5.2 データに依存した処理：ソート

秘密計算ではほかにも、通常の計算機で用いるアルゴリズムがそのまま使えない場合がある。例として、三つの GF(5) の要素列 (a, b, c) から、値が 0 であるようなセルの数を数えるという“0 カウント”を考える。例えば $(a, b, c) = (1, 0, 4)$ であれば 0 カウントの出力は 1、 $(3, 0, 0)$ であれば出力は 2 である。平文であれば例えば以下のアルゴリズムで 0 カウントが計算できる。

- (1) 0 のセルの数 num を 0 で初期化する。
- (2) a が 0 ならば num に 1 を加える。そうでなければ何もしない。
- (3) 同様の処理を b, c にも行う。
- (4) 0 のセルの数として num を出力する。

では次に、秘密計算を用いる場合、すなわち $([a], [b], [c])$ から 0 カウントを計算するために、まず上記の方法でカウントを行うことを考えてみる。原理的には入力を隠しながらどんな処理でも行うことができるため、 a を隠したままでも「 a が 0 ならば num に 1 を加える。そうでなければ何もしない」処理を行うことは可能である。しかし、ここで一つ問題が生じる。 a が 0 なら

ば「num に 1 を加える」という処理が発生するが、 a が 0 でないならば何も処理が発生しない。そのため、もし「num に 1 を加える」という処理が発生したら、 a は 0 であるということが分かってしまうし、何も処理が行われなければ、 a は 0 以外であるということが分かってしまう。すなわち、処理のステップから a の部分情報が漏れてしまっている。そのため秘密計算では最初に示した 0 カウントのような、処理ステップが秘匿されたデータに依存するアルゴリズムはそのまま用いることができない。

このような問題を回避するために、秘密計算のアルゴリズムでは、その処理ステップがデータに依存しないようにする必要がある。例として、先ほどの 0 カウントをデータに依存しないアルゴリズムに書き換える。具体的なアルゴリズムの説明の前にフェルマーの小定理を確認しておく。

$$x^4 \bmod 5 = \begin{cases} 1 & x \text{ が } 0 \text{ 以外のとき} \\ 0 & x \text{ が } 0 \text{ のとき} \end{cases}.$$

この性質を利用することで、0 カウントは以下のアルゴリズムでも計算できる。

- (1) 0 の数 num を 0 で初期化する。
- (2) num に $1 - a^4 \bmod 5$ を加える。
- (3) 同様の処理を b, c にも行う。
- (4) 0 の数として num を出力する。

ステップ (2) において、もし a が 0 であれば ($a^4 \bmod 5$ は 0 になるので、 $1 - a^4 \bmod 5$ は 1 になり) num が 1 が加えられる、また a が 0 でなければ num に 0 が加えられる。ゆえに、このアルゴリズムは 0 カウントを計算できている。しかも、処理ステップは a や b, c の値によらず一定なので、処理ステップから入力値が漏れるようなこともない。秘密計算のアルゴリズムの構成では、このようにデータに依存した処理を依存しないように書き換えることが必要となる。

実際にこのような処理ステップの工夫が大きな効果を発揮する処理の例としては、ソートが挙げられる。これは例えば (2, 1, 3, 0, 4) のような列から (0, 1, 2, 3, 4) を得る処理である。ソートを秘密計算で実現する、すなわち ([2], [1], [3], [0], [4]) を入力として、各秘密の情報は全く漏らさず、([0], [1], [2], [3], [4]) を得る方法を考える^(注4)。前述のとおり秘密計算ではデータに依存しないアルゴリズムが必要である。

データに依存しないソートのアルゴリズムとしてソーティングネットワークが知られており、これを秘密計算で使えばソートを実現することができる。しかしソーティングネットワークのような元々データに依存しないソートは、速度が遅いソートであり、もっと高速なソート、例えばクイックソートを秘密計算で実現できたほうが好ましい。

しかしクイックソートは各値の比較結果を利用して処理が分岐するため、分岐によって値の部分情報が漏れてしまい、そのままでは秘密計算で用いることができない。このような問題を

回避して秘密計算でクイックソートを実現する方法として、文献 (34) の方法がある。

基本的な発想は、クイックソートは並べられた二つの値のうち左右どちらが大きいのか？ という結果にのみに基づいてソートされるものであるため、その比較結果をデータに依存しないようにすればよく、そのためには最初に順序をランダムにシャッフルすればよい。順序がランダムにシャッフルされているため、「並べられた二つの値のうち左右どちらが大きいのか？」という情報は、元のデータに依存しないため、データに依存しないソートとなる。このような工夫で、秘密計算でそのままでは用いることができないようなアルゴリズムを、秘密計算でも使うことができる。詳細は文献 (34) を参照されたい。更に高速な秘密計算上のソートは文献 (30), (35) を参照されたい。

6. 秘密計算の実社会への適用

秘密計算は近年の計算機の発展に加え、前述のような専用のアルゴリズムの発展も進み、実用化へ大きく前進している。本稿で紹介したビット分解やソート以外にも多くのアルゴリズムが提案されており、例えば複数の表を ID などのキーを用いて名寄せを行う等結合⁽³⁶⁾、⁽³⁷⁾や、浮動小数点演算⁽³⁸⁾、回帰分析^{(39)~(41)}などが提案されている。

これらを用いて、幾つか実際のデータを用いた実験も行われている。数の少ない白血病患者の医療分析を行うために、医療データを隠しつつ診療の効果を測定した例や⁽⁴²⁾、個人の遺伝情報は隠しつつ、疾病と遺伝情報の相関を分析した例⁽⁴³⁾、⁽⁴⁴⁾、企業の取引データは隠しつつ、脱税の有無の調査を行った例⁽⁴⁵⁾、衛星の軌道は軍事機密であるため隠しつつ、複数の衛星が衝突する危険性を導出した例⁽⁴⁶⁾、更には暗号の鍵の管理を安全にするために、鍵を単一のサーバに置くのではなく、鍵を秘密にしたまま認証などを行う例⁽¹³⁾など、様々な応用例が出てきており、今正に実用に近づいてきていると言ってよいであろう。

7. おわりに

本稿では秘密計算の様々な実現方法に触れ、それらの性能の比較を行った。また、高速な秘密計算を実現するために必要なアルゴリズムの構成方法も紹介した。秘密計算は現在多くの実験も行われ社会実装に進みつつあるが、社会に普及するためには暗号学者以外からの理解も不可欠であり、本稿がそれに寄与できれば幸いである。

文 献

- (1) M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract),” *Proc. 20th Annual ACM Symposium on Theory of Computing*, J. Simon, ed., pp. 1–10, Chicago, USA, May 1988.
- (2) D. Chaum, C. Crépeau, and I. Damgård, “Multiparty unconditionally secure protocols (extended abstract),” *Proc. 20th Annual ACM Symposium on Theory of*

(注4)：ここでは簡単のため値を入れ替える処理をソートとしたが、並び変え先を示す置換を出力する定義の方が一般的と思われる。

- Computing*, J. Simon, ed., pp. 11–19, Chicago, USA, May 1988.
- (3) O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game or A completeness theorem for protocols with honest majority,” *Proc. 19th Annual ACM Symposium on Theory of Computing, 1987*, A.V. Aho, ed., pp. 218–229, New York, USA, 1987.
 - (4) A.C. Yao, “How to generate and exchange secrets (extended abstract),” *27th Annual Symposium on Foundations of Computer Science*, pp. 162–167, Toronto, Canada, Oct. 1986.
 - (5) O. Goldreich, *The Foundations of Cryptography - Volume 1, Basic Techniques*, Cambridge University Press, 2001.
 - (6) M. Hirt, “Multi-party computation: Efficient protocols, general adversaries, and voting,” PhD thesis, ETH Zurich, 2001.
 - (7) Y. Lindell and A. Nof, “A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority,” *ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, B.M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, eds., pp. 259–276, 2017.
 - (8) A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
 - (9) R. Gennaro, M.O. Rabin, and T. Rabin, “Simplified VSS and fact-track multiparty computations with applications to threshold cryptography,” *PODC*, B.A. Coan and Y. Afek, eds., pp. 101–111, 1998.
 - (10) I. Damgård and J.B. Nielsen, “Scalable and unconditionally secure multiparty computation,” *CRYPTO*, A. Menezes, ed., *Lect. Notes Comput. Sci.*, vol. 4622, pp. 572–590, Springer, 2007.
 - (11) R. Cramer, I. Damgård, and Y. Ishai, “Share conversion, pseudorandom secret-sharing and applications to secure computation,” *TCC*, J. Kilian, ed., *Lect. Notes Comput. Sci.*, vol. 3378, pp. 342–362, 2005.
 - (12) M. Ito, A. Saito, and T. Nishizeki, “Secret sharing scheme realizing general access structure,” *IEICE Transactions*, 72:56–64, 1989.
 - (13) T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, “High-throughput semi-honest secure three-party computation with an honest majority,” *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security*, E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, and S. Halevi, eds., pp. 805–817, Vienna, Austria, Oct. 2016.
 - (14) D. Ikarashi, R. Kikuchi, K. Hamada, and K. Chida, “Actively private and correct MPC scheme in $t < n/2$ from passively secure schemes with small overhead,” *IACR Cryptology ePrint Archive*, 2014:304, 2014.
 - (15) M. Naor, B. Pinkas, and R. Sumner, “Privacy preserving auctions and mechanism design,” *EC*, pp. 129–139, 1999.
 - (16) C. Kempka, R. Kikuchi, and K. Suzuki, “How to circumvent the two-ciphertext lower bound for linear garbling schemes,” *ASIACRYPT 2016*, J.H. Cheon and T. Takagi, eds., *Lect. Notes Comput. Sci.*, vol. 10032, pp. 967–997, 2016.
 - (17) B. Pinkas, T. Schneider, N.P. Smart, and S.C. Williams, “Secure two-party computation is practical,” *Advances in Cryptology - ASIACRYPT 2009*, M. Matsui, ed., *Lect. Notes Comput. Sci.*, vol. 5912, pp. 250–267, Springer, 2009.
 - (18) S. Zahur, M. Rosulek, and D. Evans, “Two halves make a whole - reducing data transfer in garbled circuits using half gates,” *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, eds., *Lect. Notes Comput. Sci.*, vol. 9057, pp. 220–250, 2015.
 - (19) S. Even, O. Goldreich, and A. Lempel, “A randomized protocol for signing contracts,” *Commun. ACM*, vol. 28, no. 6, pp. 637–647, June 1985.
 - (20) M.O. Rabin, “How to exchange secrets by oblivious transfer,” *Technical Report TR-81*, 1981.
 - (21) T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985.
 - (22) P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” *EUROCRYPT*, J. Stern, ed., *Lect. Notes Comput. Sci.*, vol. 1592, pp. 223–238, 1999.
 - (23) C. Gentry, “Fully homomorphic encryption using ideal lattices,” *Proc. 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, M. Mitzenmacher, ed., pp. 169–178, 2009.
 - (24) “小特集 完全準同形暗号の研究動向,” *信学誌*, vol. 99, no. 12, pp. 1150–1183, Dec. 2016.
 - (25) 永木敬明, “カード組を用いた秘密計算,” *IEICE Fundamentals Review*, vol. 9 no. 3, pp. 179–187, 2015.
 - (26) 桐淵, 五十嵐, 濱田, 菊池, “プログラマブルな秘密計算ライブラリ MEVAL3,” *SCIS*, 2018.
 - (27) K. Chida and K. Takahashi, “Privacy preserving computations without public key cryptographic operation,” *Advances in Information and Computer Security, Third International Workshop on Security, IWSEC 2008*, pp. 184–200, 2008.
 - (28) K. Chida, G. Yamamoto, K. Suzuki, S. Uchiyama, N. Taniguchi, O. Shionoiri, and A. Kanai, “Non-optimistic secure circuit evaluation based on elgamal encryption and its applications,” *IEICE Trans. Fundamentals*, vol. E90-A, no. 1, pp. 128–138, Jan. 2007.
 - (29) I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE,” *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, T. Takagi and T. Peyrin, eds., *Proc., Part I, Lect. Notes Comput. Sci.*, vol. 10624, pp. 377–408, Hong Kong, China, Dec. 2017.
 - (30) 五十嵐, 濱田, 菊池, 千田, “超高速秘密計算ソートの設計と実装: 秘密計算がスクリプト言語に並ぶ日,” *CSS*, 2017.
 - (31) I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft, “Unconditionally secure constant-rounds multiparty computation for equality, comparison, bits and exponentiation,” *TCC*, S. Halevi and T. Rabin, eds., *Lect. Notes Comput. Sci.*, vol. 3876, pp. 285–304, 2006.
 - (32) 五十嵐, 濱田, 菊池, 千田, “少パーティの秘密分散ベース秘密計算のための $o(\ell)$ ビット通信ビット分解および $o(|p'|)$ ビット通信 modulus 変換法,” *CSS*, 2013.
 - (33) 大原, 荒木, 土田, 古川, “異なるサイズの環が混在する不正検知可能なマルチパーティ計算,” *SCIS*, 2018.
 - (34) K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi, “Practically efficient multi-party sorting protocols from comparison sort algorithms,” *ICISC*, T. Kwon, M.-K. Lee, and D. Kwon, eds., *Lect. Notes Comput. Sci.*, vol. 7839, pp. 202–216, 2012.
 - (35) K. Hamada, D. Ikarashi, K. Chida, and K. Takahashi, “Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation,” *IACR Cryptology ePrint Archive*, 2014:121, 2014.
 - (36) S. Laur, R. Talviste, and J. Willemson, “From oblivious AES to efficient and secure database join in the multiparty setting,” *ACNS*, M.J.J. Jr., M.E. Locasto, P. Mohassel, and R. Safavi-Naini, eds., *Lect. Notes Comput. Sci.*, vol. 7954, pp. 84–101, 2013.
 - (37) 桐淵, 五十嵐, 諸橋, 濱田, “秘密計算における表の等結合アルゴリズムの提案とその実装,” *CSS*, 2016.
 - (38) V. Dimitrov, L. Kerik, T. Krips, J. Randmets, and J. Willemson, “Alternative implementations of secure real numbers,” *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security*, E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, and S. Halevi, eds., pp. 553–564, Vienna, Austria, Oct.

2016.

- (39) D. Bogdanov, L. Kamm, S. Laur, and V. Sokk, "Rmind: a tool for cryptographically secure statistical analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, 2017.
- (40) V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," *2013 IEEE Symposium on Security and Privacy, SP 2013*, pp. 334–348, 2013.
- (41) S. Tanaka, Y. Abe, S. Takahashi, R. Kikuchi, A. Doi, K. Chida, and K. Shirakawa, "Secure statistical computation system on encrypted data: An empirical study of secure regression analysis for official statistics," *UNECE Work session on Statistical Data Confidentiality*, 2017.
- (42) K. Chida, G. Morohashi, H. Fuji, F. Magata, A. Fujimura, K. Hamada, D. Ikarashi, and R. Yamamoto, "Implementation and evaluation of an efficient secure computation system using 'r' for healthcare statistics," *J. Am. Med. Inform. Assoc.*, (21):326–331, 2014.
- (43) 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.
- (44) 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, 2013.
- (45) D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "How the Estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation," *FC 2015*, R. Böhme and T. Okamoto, eds., *Lect. Notes Comput. Sci.*, vol. 8975, pp. 227–234, 2015.
- (46) L. Kamm and J. Willemson, "Secure floating point arithmetic and private satellite collision analysis," *Int. J. Inf. Secur.*, vol. 14, no. 6, pp. 531–548, 2015.

(ISEC 研究会提案, 平成 30 年 3 月 20 日受付,
平成 30 年 4 月 23 日再受付)

菊池 亮

2010 東工大大学院理工学研究科博士課程前期了, 同年 NTT 入社, 2015 東工大大学院博士課程後期了. 博士 (工学). 暗号理論及び秘密計算, 匿名化, 統計的開示制御などデータの二次利用に資する技術の研究開発に従事. ISO/IEC JTC 1/SC 27/WG 2 エキスパート, 独立行政法人統計センター非常勤研究員. CSS2012/CSS2013/CSS2017 論文賞, SCIS2017 イノベーション論文賞各受賞.

五十嵐 大

2008 東工大大学院情報理工学系研究科博士課程前期了, 同年 NTT 入社. 秘密計算, 匿名化などプライバシー保護データ活用技術の研究開発に従事. PPL2008 論文奨励賞, CSS2009/SCIS2011/CSS2012/CSS2013/CSS2017 論文賞, 2011 年度情報処理学会論文賞, 2012 年度山下記念研究賞, SCIS2017 イノベーション論文賞各受賞.