軽量Nパーティ秘匿関数計算の文字列検索拡張

滝 雄太郎[1]　藤田 茂[2]　宮西 洋太郎[3]　樋地 正浩[4]　白鳥 則郎[5]

概要：本稿では “軽量 N パーティ秘匿関数計算” [1] に対し文字列検索処理の拡張を行う．この文字列検索のために，文字列のハッシュ化を行いその結果を分割・分散する．本稿の結果より従来では整数値型のみであった秘匿関数計算のデータ検索を文字列型をも含めることが可能となる．この結果，システム設計者は文字列を秘密分散された状態で検索可能になるため，利便性が向上する．

1. はじめに

セキュリティ対策やプライバシー保護等，情報の秘密を守ることが，情報システムを構築する上で，重要な課題である．この課題を解決する一つの手法として，情報を複数箇所に分散し，秘密を守る秘密分散が注目・実用化されている．しかし，分散した情報を一か所に集めて処理すると，その一か所にリスクが集中する．このため，情報を分散させるだけでなく，分散したままで処理を実行する秘密計算，秘匿関数計算が提案されている [2].

秘匿関数計算は元の情報を n 個のデータに分割し，サーバなどに代表される n 個の計算主体へと分散する．そして，各処理は k 個の計算主体の協調により実現される．これらを k-out-of-n や N パーティ秘匿関数計算と呼ぶ．

N パーティ秘匿関数計算において，整数値型を対象とした検索手法は提案されている [3]．しかし，文字列型を対象とした秘匿関数計算を用いた検索手法は無い．ここで，データのハッシュ化を効果的に用いることにより文字列型も検索対象として取り扱うことが可能にする．

以下本稿では，2 章で関連研究について述べ，3 章で提案方式について述べ，そして最後に 4 章でまとめについて述べる．

[1] 千葉工業大学大学院情報科学研究科
Graduate School of Computer Science, Chiba Institute Technorogy, JAPAN
[2] 千葉工業大学情報科学部情報工学科
Depatment of Computer Science, Chiba Itstitute Technorogy, JAPAN
[3] 株式会社アイエスイーエム
ISEM, Inc, JAPAN
[4] 東北大学大学院経済学研究科
Graduate School of Economics and Management, Tohoku University, JAPAN
[5] 中央大学研究開発機構
Research and Development Initiative, Chuo University, JAPAN

2. 関連研究

この章では，拡張元となる軽量 N パーティ秘匿関数計算 [1] について述べる．軽量 N パーティ秘匿関数計算では，データを分割，複数の計算主体へと分散を行う．また，その分散したデータを集めることでデータの復元が可能である．また加減算，定数倍，乗算，といった各計算処理はデータを分散したまま復元することなく可能である．これによりデータの安全性・秘匿性を守ったまま各計算結果を得ることが可能となる．

以下に処理の例として次の提案方式で利用するデータの分割・分散処理，加減算処理について述べる．また，処理中の P_i は各計算主体を示し，$[a]_i$ や $[b]_i$ は分割・分散されたデータ a や b のシェアである．

Share： k-out-of-n 秘密分散

入力：$a \in \mathbf{Z}/m\mathbf{Z}$
出力：$P_i([a]_i)$
(1) $a_0, \ldots, a_{n-1} \in \mathbf{Z}/m\mathbf{Z}$ をランダムに選択する．
(2) $a_{n-1} := a - \sum_{i=0}^{n-2} a_i$ を計算する．
(3) $i = 0, \ldots n-1$ について，$[a]_i := (a_i, \ldots, a_{n-k+i})$ として P_i に送信する．

Dec： k-out-of-n 秘密分散のエラー検出を含む復元

入力：$P_i([a]_i)$
出力：a or $\perp$
(1) P_i は $(\alpha_i, \ldots, \alpha_{n-k+i}) := (a_i, \ldots, a_{n-k+i})$ を開示する．
(2) $i = 0, \ldots, n-1$ において各主体 P_i の α_i に対応する a_i について $\alpha_i \neq a_i$ となる α_i, a_i が一つでも存在すれば，異常を示す $\perp$ を返して終了する．
(3) $a = \sum_{i=0}^{n-1} a_i$ を計算する．

分散 **Share** 処理を行うことにより，分散したいデータを各計算主体へと分散することが可能となる．また，復元 **Dec** 処理を行うことにより，分散されたデータを復元することが可能となる．また，加減算処理の手順を以下に示す．

Add/Sub： a, b のシェアから $a \pm b$ のシェアを作成

入力：$P_i([a]_i, [b]_i)$
出力：$P_i([a \pm b]_i)$
(1) P_i は $[a \pm b]_i = (a_i \pm b_i, \ldots, a_{n-k+i} \pm b_{n-k+i})$ を計算する．

加減算 **Add/Sub** 処理を行うことにより，各計算主体には $a \pm b$ の結果のシェア $[a \pm b]_i$ が作成される．そして，この計算結果が必要な場合は，先に説明した復元 **Dec** 処理をシェア $[a \pm b]_i$ に行うことで得ることができる．提案方式では，文字列検索の際にこの加減算処理を利用して検索を行う．

3. 提案方式

この章では 2 章で説明した軽量 N パーティ秘匿関数計算に文字列検索の処理を導入することで拡張を行う．手順としては文字列の一致を確認するために前もって，各文字列のハッシュ化された値を各計算主体へと配置しておく．これを行うことにより，文字列の検索をハッシュ値の減算処理結果から求めることを可能とする．

また，秘匿関数計算を用いるデータは安全せや秘匿性が重視されるデータである．そのようなデータに対して，検索処理を行ったとしてもデータの安全性や秘匿性に対して影響が無いことについても述べる．

3.1 k-out-of-n 文字列検索

文字列検索を行う前提として，各計算主体にはいくつかの文字列 $StrA, StrB, \ldots$ のハッシュ化された値 $HashA, HashB, \ldots$ が分散 **Share** 処理により送信されているものとする．また，各文字列とそのハッシュ化された値は互いに紐付いているものとする．

このとき，文字列 $StrA$ を検索 **Search** 処理により検索する手順を以下に示す．なお，処理中では先に送信されていた文字列 $StrA, StrB$ やそのハッシュ化された値 $HashA, HashB$ と区別するために，検索対象となる文字列を $StrA'$，ハッシュ化された値を $HashA'$ と表記する．

Search：k-out-of-n 文字列検索

(1) $StrA'$ をハッシュ化し $HashA'$ を求める
(2) $HashA'$ を分散 **Share**
(3) $HashA'$ と $HashA, HashB$ を用いて減算 **Sub**
(4) $HashA' - HashA, HashA' - HashB$ を復元 **Dec**
(5) 復元された値が 0 ならば検索文字列である

検索 **Search** 処理を行うことにより，計算主体にある文字列に対し検索を行うことが可能となる．次に，この検索処理を行ったとしても安全性や秘匿性に対して影響が無いことについて述べる．

3.2 k-out-of-n 文字列検索の安全性

この処理の安全性や秘匿性に対して影響が無いか確認することが必要である．ここで，検索 **Search** 処理で使用される各処理を列挙する．

Search：k-out-of-n 文字列検索で使用される処理

(1) 分散 **Share**
(2) 減算 **Sub**
(3) 復元 **Dec**

この 3 つの処理は軽量 N パーティ秘匿関数計算 [1] 内にて安全性についての確認が済んでいる．また，各計算主体へと分散 **Share** 処理を用いて値を送信する際にはたとえ同じ文字列・ハッシュ値であったとしても異なる値が送信される．理由は，分散 **Share** 処理でデータ a を分割してシェア $[a]_i$ を作成する際には，毎回ランダムな値が使用されるからである．

Share： k-out-of-n 秘密分散

入力：$a \in \mathbf{Z}/m\mathbf{Z}$
出力：$P_i([a]_i)$
(1) $a_0, \ldots, a_{n-1} \in \mathbf{Z}/m\mathbf{Z}$ をランダムに選択する．
(2) $a_{n-1} := a - \sum_{i=0}^{n-2} a_i$ を計算する．
(3) $i = 0, \ldots n-1$ について，$[a]_i := (a_i, \ldots, a_{n-k+i})$ として P_i に送信する．

この処理中の (1) に毎回ランダムな値が選択される事により計算主体からは元のハッシュ化された値の類推ができない．これにより，各計算主体で減算 **Sub** 処理により計算される値も毎回異なる値になる．そして，その結果復元 **Dec** 処理により復元される際に送信される値もランダムな値となる．

Dec： k-out-of-n 秘密分散のエラー検出を含む復元

入力：$P_i([a]_i)$
出力：a or $\perp$
(1) P_i は $(\alpha_i, \ldots, \alpha_{n-k+i}) := (a_i, \ldots, a_{n-k+i})$ を開示する．
(2) $i = 0, \ldots, n-1$ において各主体 P_i の α_i に対応する a_i について $\alpha_i \neq a_i$ となる α_i, a_i が一つでも存在すれば，異常を示す $\perp$ を返して終了する．
(3) $a = \sum_{i=0}^{n-1} a_i$ を計算する．

このことより検索 **Search** 処理中で計算される値は毎回ランダムであり，情報の類推を行うことはできない．つまり，安全性や秘匿性を保ったまま検索処理が可能であると言える．

4. まとめ

本稿では “軽量 N パーティ秘匿関数計算” [1] に対し文字列検索処理の拡張を行った．その結果，ハッシュ化を用いることにより情報の安全性や秘匿性を保ったまま文字列の一致を検索することが可能となった．また，今後の課題は 3 点ある．

課題の 1 つ目に，文字列をハッシュ化するためのハッシュアルゴリズムについてである．現在ハッシュアルゴリズムには様々な種類がある．例としては，ファイルのチェックサムとして多く利用されている MD5 や HTTPS 通信中でサーバの証明書をチェックする際に使用される SHA-256，誤り検出のために使用される CRC-32 である．このようにあるアルゴリズムから現在考慮しなければならないと考えている点は以下の 2 点である．

ハッシュアルゴリズムの考慮点

(1) 安全性・衝突耐性
(2) ビット長・桁数

MD5 などには衝突耐性が無い．もし仮に衝突耐性が無いようなハッシュアルゴリズムを使用した場合に秘匿関数計算の安全性が低下してしまうのか，そうではないのかについて確認が必要である．また，ハッシュアルゴリズムにより出力される値の桁数が大きい場合には秘匿関数計算を用いたシステム全体で取りうる値を余分に取らなくてはならなくなり，この点についても考慮が必要となる．

課題の 2 つ目に，現状の方式では完全一致検索しか可能でない点である．例えば，住所データを検索する場合を考える．“都道府県 { 区, 市町村 } 番地” のような文字列がある場合に最初の都道府県を対象に前方一致検索を行いたい場合はあるといえる．しかし，現状の方式では文字列全体のハッシュ値のみ検索対象になる．これを解決する方法としては 2 点考えられる．

前方一致検索を達成する方法

(1) 前方一致情報を別に保存
(2) 前方一致情報を分散情報中に埋め込む

前方一致情報を別に保存する方法は単純である．住所データの例で挙げるならば，“都道府県 { 区, 市町村 } 番地” の文字列のハッシュ値の他に “都道府県” の文字列のハッシュ値を保存しておけば良い．しかしながらこの方法では前方一致したい情報分だけデータ量が増えていくと言える．前方一致情報を分散情報中に埋め込む方法ではデータ量の増加は無い．しかし，データの分散の際に工夫が必要であり，かつ安全性や秘匿性にどのような影響が出るかについての考慮が必要となる．

課題の 3 つ目に，実際に本方式を実装しデータ量に応じた検索速度や先に上げたハッシュアルゴリズムごとのビット長・桁数の確認，計算主体の個数によって結果がどのように変わっていくかなど性能評価を行っていく必要がある．

参考文献

[1] 滝雄太郎，藤田　茂，宮西洋太郎，白鳥則郎：軽量 N パーティ秘匿関数計算の一般化，情報処理学会論文誌，Vol. 59, No. 10, pp. 1895–1902 (2018).
[2] 千田浩司，五十嵐大，濱田浩気，高橋克巳：エラー検出可能な軽量 3 パーティ秘匿関数計算の提案と実装評価，情報処理学会論文誌，Vol. 52, No. 9, pp. 2674–2685 (2011).
[3] 志村正法，宮崎邦彦，西出隆志，吉浦　裕：秘密分散データベースの構造演算を可能にするマルチパーティプロトコルを用いた関係代数演算，情報処理学会論文誌，Vol. 51, No. 9, pp. 1563–1578 (2010).