秘密カウンタによる電子投票システム

中里 純二　　菊池 浩明　　中西 祥八郎

東海大学電子情報学部

{nakazato,kikn}@ep.u-tokai.ac.jp

あらまし: 本論文は，1か0かの秘密を漏らさないまま，公開検証可能な方法で足し算を実行するプロトコルを提案している．提案方式は，従来の膨大な帯域を消費し何回にも及ぶラウンドを必要としたマルチパーティプロトコルとは異なり，非対話的な定数回のラウンドと単純で送信者にも集計者にも検証可能な計算処理を必要とする．提案方式は，[3] に基づいている．提案方式は，投票者が投票用紙を分散された公開鍵について暗号化して投票し，集計者が賛成投票数をわからないまま数えるという秘密投票に応用可能である．今回，この提案方式を Java を用いて実装し，評価を行なった．

Electronic Voting System Using Oblivious Counter

Junji Nakazato　Hiroaki Kikuchi　Shohachiro Nakanishi

Tokai university, School of Infomation Technology and Electronics

Abstract: The paper presents a new protocol for counting 1-bit secrets without revealing if the bit is 1 or 0 in publicly verifiable way. Other than the conventional multi-party protocols that involve enormous number of rounds and huge bandwidth consumption, the proposed protocol, based on the Mix and Match approach [3] in which computations are dealt with ciphertexts, requires a non-interactive constant number of round and simple but verifiable computation for both of sender and counter. The expected application of proposed protocol is an (one-bit) secret voting in which voters cast a ballot encrypted for distributed public key and an oblivious party makes a tally of how many votes are polled. We developed a Java based voting system using the proposed method and esitimate computational costs.

1 はじめに

電子投票には，匿名性や二重投票不能性など，多くの要請を満たす必要がある．従来の研究には大きく2つの流れがある．1つは，ブラインド署名と匿名通進路を応用するものであり，代表例に [6] がある．もうひとつは，[4] に代表されるマルチパーティープロトコルを用いたものである．

MIX-NET を用いた [6] では，ブラインド署名を用いることで，投票者の匿名性を保証し，投票者の認証と投票者以外の水増し投票を防止した．しかし，多段に MIX-NET と呼ばれる暗号サーバを利用することで匿名性を実現しているため，投票者は多重暗号が必要になり，計算量が MIX-NET の数に比例して増大してしまうという欠点がある．また，MIX-NET が中断中に暗号文を捨てたり，無効な偽投票を足してしまったりする利用不能攻撃が本質的に避けられないため，MIX-NET での処理に対するゼロ知識証明を加えたりして，セキュリティ強化が試みられてきた [1], [2]．それ由に，開票に大きな処理コストがかかったり，あるいは，投票者が投票時と開票時に2回処理する必要が生じたりしていた．

一方，マルチパーティープロトコルを用いた [4] では投票内容を1または−1に限定し，準同型暗号を用いる事で，その投票内容を秘密にしながら投票と集

計を可能にした．また，ゼロ知識証明を用いて投票内容を検証することで，投票者の匿名性 (投票者と投票内容の関係) も保証されている．しかし，集計者の分散数に比例して投票者の計算量が増大する問題点がある．

そこで，我々のグループでは，[3] に注目し，投票の内容を秘密にしたまま，正しく集計したことを証明するプロトコルを提案した [7]．このプロトコルには次の特徴がある．

- 投票者の処理量は，鍵管理機関の分散数に依存しない．
- 投票者は，自分の票が正しくカウントされたことを公開された場で即確認できる．(publicly verifiable)
- 集計者は，投票の内容を知らないまま，投票者の認証と，投票値の正当性が公開の場で確認できる．
- しきい値以上の鍵管理機関が協力して，投票結果がわかる．鍵を再発行する必要はなく，何度でも繰り返せる．

投票者は 1bit の投票内容を乗算に関する準同型性を満たす ElGamal 暗号を用いて暗号化する．準同型暗号を用いることで，復号することなく投票された bit をかけ合わせる事 (足し算) が可能になる．

提案プロトコルには，しかしながら，投票者の負荷が投票者数 n に比例するという懸念があった．そこで，本稿では，Java を用いて提案プロトコルを実装し，負荷を明らかにすることを試みる．Java を用いたのは，BigInteger などの多倍長演算 class が，充実していたためである．

2 秘密カウンタ

本章では，[7] を簡単に説明する．

2.1 エンティティ

- $\mathcal{U}_i$：投票者 ($i \in \{1, \cdots, n\}$)
- C：集計サーバ
- S：開票サーバ (分散することも可能である)

2.2 プロトコル

2.2.1 準備

ElGamal 暗号の公開鍵，秘密鍵を用意する．$p = 4q+1$ となるような大きな素数 p, q を生成し，乗法群 Z_p^* での位数 $2q$ となる乗法群 G の原始元を g とする．次に，$x \in_u G$ を定め，$y = g^x \bmod p$ を計算する．ここで，x を秘密鍵 (分散 ElGamal を用いることで，分散することも可能である)，y, g, p を公開鍵とする．また，暗号化を $E[\cdot]$，復号を $D[\cdot]$ とする．また，暗号文 $E[m_a]$，$E[m_b]$ の積を次のように定める．

$$\begin{cases} E_a[m_a] = (m_a y^{r_a}, g^{r_a}) \\ E_b[m_b] = (m_b y^{r_b}, g^{r_b}) \\ E_a[m_a] \times E_b[m_b] = (m_a m_b y^{r_a + r_b}, g^{r_a + r_b}) \end{cases}$$

2.2.2 1-bit カウンタ

Step 1:(投票) 投票者 $\mathcal{U}$ は，票 $v \in \{1, -1\}$ を選び，開票者 S の公開鍵を用いて $E[v]$ を計算し，集計者 C に投票する．ここで，ゼロ知識証明を用いて投票内容が $\{1, -1\}$ のどちらであるということを証明する．

Step 2:(集計) C は現在のカウンタの値 A' と投票された $E[v]$ を掛け合わせ A を計算する．

$$A = A' \times E[v]$$

投票終了後 C は S に A を送る．

Step 3:(開票) S は $D[A]$ を計算する事で，開票結果 $1, -1$ を得る事ができる．

以上が 1bit のカウンタを構成するプロトコルである．次に 2bit を構成するカウンタプロトコルを説明する．

2.2.3 2-bit カウンタ

Step 1:(投票) 投票者 $\mathcal{U}_i$ は，票 $v_i \in \{1, -1\}$ を選び，開票者 S の公開鍵を用いて $E[v_i]$ を計算する．集計者 C に $(E[v_i], E[c_i])$ を次のように選び，投票する．その時，ゼロ知識証明を用いて投票内容が $\{1, -1\}$ のどちらであるということを証明する．

$$E[c_i] = \begin{cases} E[1] & \text{if } v_i = 1, \\ E[A'] & \text{if } v_i = -1. \end{cases}$$

ここで，$i \in \{1,2\}$(投票者数)，A'を1ビット目の現在のカウンタの状態とする．

Step 2:(集計) Cは現在のカウンタの値(A', B')と投票された$(E[v_i], E[c_i])$を掛け合わせA, Bを計算する．

$$A = A' \times E[v_i]$$
$$B = B' \times E[c_i]$$

全ての投票終了後CはSに(A, B)を送る．

Step 3:(開票) Sは$D[A], D[B]$を計算する事で，開票結果を得る事ができる．表1は，AとBの状態とそれに対応する投票数である．

表 1: 2-bit カウンタ

	0	1	2	3
A	1	−1	1	−1
B	1	1	−1	−1

2.2.4 n-bit カウンタ

Step 1:(投票) 投票者U_iは，票$v_i \in \{1, -1\}$を選び，開票者Sの公開鍵を用いて$E[v_i]$を計算する．集計者Cに$(E[v_i], E[c_{k,i}])$を次のように選び，投票する．その時，ゼロ知識証明を用いて投票内容が$\{1, -1\}$のどちらであるということを証明する．

$$E[c_{k,i}] = \begin{cases} E[1] & \text{if } v_i = 1, \\ E[A'_{k-1}] & \text{if } v_i = -1. \end{cases}$$

ここで，$i \in \{2, \cdots, n\}$(投票者数)，$k \in \{1, \cdots, 2^{n-1}\}$，$A'_k$を$k$ビット目の現在のカウンタの状態とする．

Step 2:(集計) Cは現在のカウンタの値A'_kと投票された$(E[v_i], E[c_{k,i}])$を掛け合わせA_kを計算する．

$$A_1 = A'_1 \times E[v_i] \quad (k = 1)$$
$$A_k = A'_k \times E[c_{k,i}] \quad (k \geq 2)$$

全ての投票終了後CはSにA_kを送る．

Step 3:(開票) Sは次のように開票する事によって，効率良く開票結果を得る事ができる．

$$D[A_k] = A_k \quad \text{if } k = 2^{l-1}$$

ここで，$l = \{1, \cdots, n\}$とする．表2は，A_kの状態とそれに対応する投票数である．

表 2: n-bit カウンタ

	0	1	2	3	4	$\cdots$	i
A_1	1	−1	1	−1	1	$\cdots$	−1
A_2	1	1	−1	−1	1	$\cdots$	−1
A_3	1	1	1	−1	1	$\cdots$	−1
A_4	1	1	1	1	−1	$\cdots$	−1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
A_k	1	1	1	1	1	$\cdots$	−1

2.2.5 性質

ここでは，本提案プロトコルの性質について述べる．

- 投票者の処理が鍵の分散数に依存しない
  - 集計者は一人のため，投票内容を分散する必要が無い．
- 投票の検証可能性
  - 集計者が投票や，途中経過を全て公開しても，投票内容の情報は洩れない．
- 投票の正当性の確認
  - 集計者は全ての情報を公開可能なために実現．
- 集計者の不正の不可能性
  - 集計者の持つ情報を全て公開し，検証可能なため実現．
- しきい値法の適応
  - 分散 ElGamal を用いることで実現可能．

3 試験実装

3.1 構成

今回本提案プロトコルを Java を用いて実装した．プログラムは`MkKey`, `Vote`, `ElGamal`, `Counter`, `Open`の5つのクラスから構成されている．

- `MkKey` クラス
  - ElGamal 暗号に基づいた 1024bit の鍵を生成．秘密鍵ファイル`FileName.sk`，公開鍵ファイル`FileName.pk`が出力される

- 実行例：$java MkKey FileName

- Vote クラス
  - 指定した公開鍵ファイルを用いて指定 bit 数の投票を行なう
  - 投票内容 (vote) は公開鍵 (key.pk) によって暗号化される
    - * ElGamal 暗号，復号，乗算を行なう (ElGamal クラス)
  - 投票ファイルVote1, Vote2 が出力される
  - 実行例：$java Vote bit vote key.pk

- Counter クラス
  - 現在のカウンターの状態 (Carry1, 2) に投票ビット (Vote1, 2) を加算する．
  - 加算結果 (Carry1, 2) を更新する
  - 実行例：$java Counter bit key.pk

- Open クラス
  - カウンタの状態 (Carry1, 2) を復号し，投票結果を集計する
  - 集計結果ファイルResult が出力される
  - 実行例:$java Open bit key.pk key.sk

3.2 ファイル形式

本実装は試験実装であるため，全てのファイル形式はテキスト形式で出力される．

3.3 パフォーマンス

以下に，投票者，集計者，開票者のパフォーマンスのグラフを示す．

図 1，2 より，開票者 $\mathcal{S}$ 以外の計算量は，投票ビット数に対応して指数関数的に増加してしまうことが分る．これは，開票時には必要なビットのみを開票するために計算量が少なくて済む[1]が，他の投票者 $\mathcal{V}$，集計者 $\mathcal{C}$は，カウンタの内部状態を保存するためのビットの計算も必要なため，計算量[2]が増えてしまうことが予測される．

[1]計算量は，$O(\log n)$ になる．
[2]計算量は，$O(n)$ にまでなってしまう．

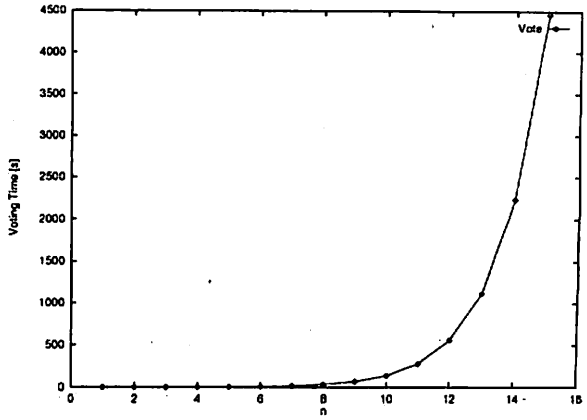

図 1: 投票者の計算量

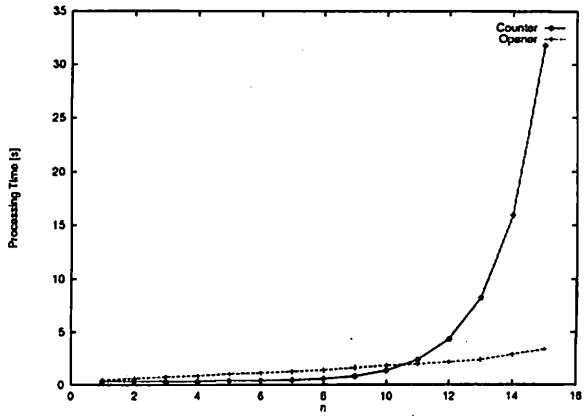

図 2: 集計者と開票者の計算量

4 おわりに

本論文では Java を用いて秘密カウンタを用いた電子投票システムの実装及び評価を行なった．評価の結果，投票者 $\mathcal{V}$や，集計者 $\mathcal{C}$の計算量が非常に多くなってしまった．これは，カウンタを構成するに当たって，必要なビットが多くあるためである．今後，これらの問題点を解決する必要がある．

参考文献

[1] M. Abe, "Universally Verifiable Mix-Net with Verification Work Independent of the Number of Mix-Servers," IEICE Trans. Fundamentals, Vol. E83-A, No.7, July 2000.

[2] M. Ohkubo, "暗号文の長さが不変な Hybrid Mix", 暗号と情報セキュリティシンポジウム-SCIS2000, SCIS2000, 2000.

[3] M. Jakobsson and A. Juels, "Mix and Match: Secure Function Evaluation via Ciphertexts," in ASIACRYPTO '00, pp.162-177, 2000.

[4] K.Sako and J.Kilian, "Secure Voting Using Partially Compatible Homomorphisms," in CRYPTO '94, pp.411-424, 1994.

[5] A. Fujioka, T. Okamoto, and K. Ohta, "A Practical Secret Voting Scheme for Large Scale Elections," in AUSCRYPTO '92, pp.244-251, 1993.

[6] 藤岡 他, "実用的な電子投票方式の実装および実験", 暗号と情報セキュリティシンポジウム-SCIS2000, SCIS2000-48, 2000

[7] H.Kikuchi, J.Nakazato, S.Nakanishi , "秘密カウンタ", 信学技法 ISEC, 2001.