準同型暗号における大小比較とその評価
Homomorphic Encryption for Greater-Than Relation and Its Application

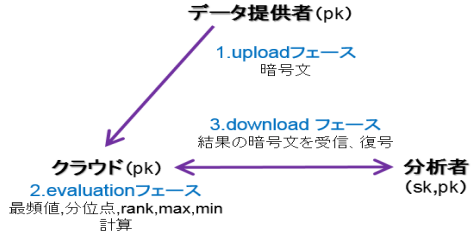
李泰潤・システム分科会・情報セキュリティ大学院大学

We propose a Greater-Than method that can perform computation processes with encrypted state for relation operations considered to be important in data mining. For this purpose, FV scheme based on HElib in the homomorphic encryption library is used for homomorphic encryption method. Finally, we compare the results with the previous studies and perform an assessment for this study.

研究の目的

FV暗号を利用してデータマイニングにおいて重要な演算の1つである大小比較を、プロトコルなしで準同型演算で提案する。また、実験と結果に関する評価を行う。

先行研究[Sak15]



最頻値などを求める際に大小比較を行う。しかし、この大小比較では原理上、大量の暗号文のやりとりが必要
→効率が気になる

実験と評価

- パラメータ

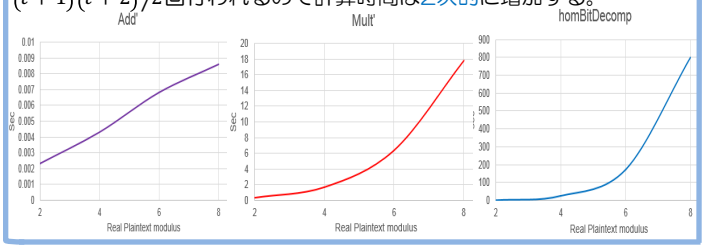
Real Plaintext modulus	Virtual Plaintext modulus	Ciphertext modulus	m-th Cyclotomic Ring	Security parameters
2^2	2^4	2^{55}	1399	80
2^4	2^6	2^{100}	2579	80
2^6	2^8	2^{180}	4177	80
2^8	2^{10}	2^{210}	5483	80

- 実験結果

Real Plaintext modulus	Enc′	Dec′	Add′	Mult′	homBit Decomp	Greater -Than
2^2	0.0033433	0.0053731	0.0023105	0.318956	1.87852	1.8611
2^4	0.0096565	0.0136643	0.0043047	1.70277	23.6435	23.2873
2^6	0.0216651	0.027032	0.0067638	6.41201	174.057	173.953
2^8	0.0460832	0.051633	0.0086462	17.7869	798.952	799.617

- 評価

Addは暗号文同士に加算するだけなので、平文モジュラスが増えるごとに計算時間が1次的に増加する傾向がみられる。
Multの場合は、暗号文の要素の掛け算があるので2次的に増加することがわかる。
homBitDecompでの準同型乗算Multの回数は、平文がtビットのとき$(t+1)(t+2)/2$回行われるので計算時間は2次的に増加する。



提案モデル

HEを利用してGreater-Thanに対応できる準同型暗号スキームHE'を提案する。

t：小数点　q：暗号文modulus 2^r：仮想平文modulus 2^{t+1}：仮想平文modulus

Keygen′():
　return $(sk, pk) \leftarrow$ HE.Keygen()

Add′$((c,k),(c',k'))$:
　if $k' \geq k$:
　　return(HE.Add$(2^{k'-k}c, c'), k')$
　else :
　　return(HE.Add$(c, 2^{k-k'}c'), k)$

Mult′$((c,k),(c',k'))$:
　if $k' \geq k$:
　　return(HE.Mult$((2^{k'-k}c)/2^{k'}, c'/2^{k'}) \cdot 2^{k'}, k')$
　else :
　　return(HE.Mult$(c/2^k, (2^{k-k'}c')/2^k) \cdot 2^k, k)$

Enc′$(pk, x \in R_{2^t})$:
　$k \leftarrow 0$
　$c \leftarrow$ HE.Enc(pk, x)
　return (c, k)

Dec′$(sk, (s,k))$:
　return $[$HE.Dec$(sk, c)/2^k]_{2^t}$

Algorithm 2 homBitDecomp②

INPUT: $(c,k), r$
OUTPUT: $(w_{0,r}, k_{0,r}), (w_{1,r}, k_{1,r}), \cdots, (w_{r,r}, k_{r,r})$

1: $(w_{0,0}, k_{0,0}) \leftarrow (c,k)$
2: **for** $i = 0$ to $r-1$ **do**
3:　$(y, k_y) \leftarrow (c,k)$
4:　**for** $j = 0$ to $i-1$ **do**
5:　　$(w_{j,i+1}, k_{j,i+1}) \leftarrow$ Mult′$((w_{j,i}, k), (w_{j,i}, k))$
6:　　$(y, k_y) \leftarrow$ Add′$((y, k_y), (-w_{j,i+1}, k_{j,i+1}))$
7:　　$k_y \leftarrow k_y + 1$
8:　$(w_{j+1,i+1}, k_{j+1,i+1}) \leftarrow (y, k_y)$
9:　**end for**
10: **end for**
11: **return** $((w_{0,r}, k_{0,r}), (w_{1,r}, k_{1,r}), \cdots, (w_{r,r}, k_{r,r}))$

$x = 1111b$　$x^2 = 1110b$　$x^4 = 1000b$　$x^8 = 0000b$

1111b
-0000b
1111　÷2

111b
-000b
111　÷2

11b
-00b
11　÷2

1b
-0b
1　÷2

$b = 0 \text{ or } 1$
$x \equiv b \pmod{2^k}$
$\Rightarrow x^2 \equiv b \pmod{2^{k+1}}$
この方法を応用する

大小比較したい暗号文cとc'に対して、cが大きいとき$c' - c$の最上位ビットに1が立ち、c'が大きいとき$c' - c$の最上位ビットは0である。このような最上位ビットの暗号文γ_tとk_tを出力する

Algorithm 3 GreaterThan

INPUT: $(c,k), (c',k')$
OUTPUT: (γ_t, k_t)

1: $((\gamma_0, k_0), \cdots, (\gamma_t, k_t)) \leftarrow$ homBitDecomp(Add′$((c'_{tmp}, k'_{tmp}), (-c_{tmp}, k_{tmp})), t)$
2: **return** (γ_t, k_t)