MODINT: A COMPACT MODULAR ARITHMETIC JAVA CLASS LIBRARY FOR CELLULAR PHONES, AND ITS APPLICATION TO SECURE ELECTRONIC VOTING

Hiroaki Kikuchi and Junji Nakazato

[1] *Dept. of Information Media Technology*
Tokai University
kikn@tokai.ac.jp

[2] *Graduate School of Engineering*
Tokai University
nakazato@ep.u-tokai.ac.jp

Abstract ModInt is a compact Java class designed for the arbitrary-precision modular arithmetic operations used in secure cryptographical applications, including auctions and electronic voting. The library is small enough to port to a Java-enabled cellular phone. In this paper, we present a new electronic voting system using the ModInt library, in which voters have their ballots encrypted with the public key of a trusted agent and cast them to an untrusted server, which tallies them without decrypting the ballot and updates a linear-feedback shift register (LFSR) consisting of log n ciphertexts. The privacy of voters is therefore assured assuming that the security of the public-key crypto system is maintained, and the trusted party keeps their private key secure. This feature reduces the costs of management of the counting server, which is now free from the risks of being compromised.

Keywords: electronic voting, homomorphic encryption, Java applet, modular arithmetic

1. INTRODUCTION

The cellular phone is a widely used communication medium that allows us to communicate more easily and frequently than before. It has been developed to provide a number of attractive services to customers, including web browsing, and digital still camera and GPS (global po-

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

sitioning system) functions. The Java application platform is also one of these new services that expands the range of application of cellular phones. The platform can offer, for example, interactive computer games, instant messaging clients, or interfaces to servers that are more intelligent and user-friendly than simple web browsers.

Electronic voting is also a prime example of a function suitable for a Java-enabled cellular phone. With a mobile voting device, one may easily participate in decision-making processes whenever suitable. According to a Japanese governmental survey [4], more than 90% of Japanese teenagers have their own cellular phone, and they are expected to be potential voters who take part in elections only if they are allowed to cast a vote from the Internet. In the near future, the population of cellular phone users will exceed that of passport holders, and hence it is possible that the cellular phone could replace the passport as a nationwide user identification scheme. Additionally, Japanese regulation has been updated in 2001 making electronic voting legal throughout Japan[13]. In June 2002, the first Japanese electronic election was held in Niimi, Okayama, in the mayoral and local assembly elections[14]. In the elections, more than 19,000 eligible voters cast their ballots from the voting machine by inserting a voting card into a voting machine, and touching the names of their preferred candidates, which appeared on a video screen. Their votes were stored in the memories of the voting machines, and counted electronically after the voting ended. The electronic voting reduces the time and effort spent on tallying vote. The total time is just 25 minutes. According to the questioner, 97 % of voters said that the machine is better than the conventional paper-based election.

The biggest issue to be resolved about real-world Internet voting is the conflict between security and privacy. In order to prevent dishonest voters and potential attackers from manipulating the tally in malicious ways — such as double voting, altering or forging ballots, and unauthorized voting — the security technologies, using public-key infrastructure with public key certificates, provide secure authentication for eligible voters. However, the identification may be used to trace the voters, and the privacy of voters may be violated by the government or the administrators of the counting server.

To avoid these problems, a number of electronic voting protocols have been proposed to date. In 1997, Crameмer, Damgard and Schoenmakers presented an optimally efficient voting protocol using a homomorphic encryption, in which the size of the ballot is a single ciphertext[9] Fujioka et al. proposed an efficient protocol in [1], combining a blind signature and anonymous channels or mix servers, and performed the experimental trial over the Internet. In 2001, Furukawa and Sako proposed a protocol

to prove the correctness of shuffled ballots, and implemented it with several mix servers.

In this paper, we present a new electronic voting protocol using a linear feedback shift register (LFSR), which makes a tally using the internal state of the LFSR. In the proposed protocol, a voter shifts the public register if they wish to vote "Yes", or does nothing if they wish to vote "No". However, using a homomorphic public-key encryption scheme, no one can determine whether the vote is "Yes" or "No" by watching the register except the trusted authority, who keeps the corresponding private key secure and reveals it to the public only when the time comes to open the tally. The communication cost for voters is of $O(\log n)$ where n is a number of voters. The idea of using an LFSR as a tally is not our own. In 2001, Katz et al. proposed cryptographic counters and constructed a concrete protocol using quadratic residue (QR) encryption in [7]. In contrast to this approach, our system adopts El Gamal encryption that allows the decryption process to be distributed across several (incompletely trusted) authorities who hold a share of the corresponding private key in an ordinary manner.

These cryptographic approaches are substantially effective in contemporary personal computers, but the restricted computational power of cellular phones is not sufficient to perform these cryptographical computations fully. Because of restricted storage, the `BigInteger` class library, which is a Java standard class for processing the arbitrary precision arithmetic used in cryptographical protocols, is not supported in any commercial Java-enabled cellular phone device. Moreover, the free memory capacity is 30kB at most in the NTT 504i, or 10kB in the 503i, which is too small to support the comprehensive methods in the `BigInteger` class.

In order to address the issue, we have developed a compact Java class library named `ModInt`, which is designed for the modular arithmetic used in cryptographic protocols. The `ModInt` class is

- specially designed for modular arithmetic, with all operations performed implicitly in modulo, and
- small enough to be ported to restricted environments such as cellular phones.

We are interested in how large overheads are generated in modular arithmetic in the cellular phone environment, and whether such arithmetic is suitable for performing electronic voting protocols.

Thus, in this paper, we demonstrate the performance of the ModInt class library and prove that the cryptographical protocol is feasible, even in an ordinary cellular phone environment. Based on the experimental

results of the library, we clarify the necessary conditions for large-scale electronic voting.

The compact Java class library allows us to hold a variety of web-based electronic voting including large-scale election for governors, election for city mayor, and small-scale election for student president. In this sort of public election, an election committee manages a list of eligible voters and requires user authentication processes using appropriate identification protocols. The simplest is a password-based authentication. If we can assume the national identity numbering system where every citizen has assigned number and the X.509 public-key certificate, PKI can be used to identify the eligible voter. According to the "e-Japan Priority Policy Program"[12], a PKI in Asia region is going to be developed by the 2005. On August 5th 2002, Japanese government has started the Basic Resident Register Network, where the 11-digit resident's certificate codes are assigned to every citizen and a copy of resident's certificate is available at any office around Japan. Though there are some concerns about the protection of the privacy, the option of national-wide PKI is now realistic. Any of these authentication methods can be used in conjunction with the proposed voting system.

August 5th 2002. Basic Resident Register Network Individual info: Address, Names, Birthdays, Sex. 11-digit Regidentfs certificate code. A copy of residentfs certificate is available at any office around Japan. The concern about the protection of privacy by the government. Kokubunji, Hino, Kunitachi: potponement of its operation.

The other direction of the proposed electronic voting system is a questionnaire-based user evaluation. Web-based questionnaire have been used for many applications such as guest questionnaire, commercial products evaluation, or course evaluation. In any of questionnaire, privacy should be preserved though most of them don't take any account of the issue. The proposed Java-based system allows to any party to improve the privacy enhancement just replacing the casting and the tallying processes by the proposed system without developing from the scratch. In comparison with the public election mentioned above, the strong user authentication is not always necessary in this sort of questionnaire. Participation to evaluation is open to any users and checking the source IP address is good enough to prevent from malicious users from double casting.

The structure of this paper is as follows. In Section 2, we describe the protocol for oblivious vote counting with the LFSR. Following this, we demonstrate the system design and the performance of the `ModInt` Java class library in detail. Finally, we evaluate the implemented electronic

voting system in terms of its complexity and communication requirements.

2. PRELIMINARY

2.1 Homomorphic Encryption Scheme

Let M be a set of plaintexts, $\{m_0, m_1\}$, where m_0 and m_1 represent boolean values corresponding to 'false' and 'true', respectively.

Let us suppose a *homomorphic encryption scheme* E that has the following properties.

- E is homomorphic over $GF(2)$, i.e., for elements a and b of $\{m_0, m_1\}$,

$$E[a \oplus b] = E[a] \times E[b] \tag{1}$$

  holds, where $a \oplus b$ is an exclusive-OR, defined as $m_0 \oplus m_1 = m_1 \oplus m_0 = m_1$ and $m_0 \oplus m_0 = m_1 \oplus m_1 = m_0$.

- E is semantically secure, that is, no one is able to distinguish ciphertexts of m_0 and m_1 with a probability significantly greater than a random guess.

- The key generation can be distributed among a certain number of players. The size of the public key should not depend on the number of shares.

- The decryption process can be distributed among t-out-of-n players who share the corresponding private key. The computational and communication (bandwidth and rounds) costs should be as small as possible.

El Gamal encryption satisfies all requirements under the Decision Diffie–Hellman (DDH) assumption, if we let $m_0 = 1$ and $m_1 = -1 \pmod p$. Let p and q be large primes such that $p = 2q + 1$ and let $\mathcal{G}$ be the set of multiplicative groups of order q in Z_p^*. Let g be a primitive element of $\mathcal{G}$. If we have $p = 2q + 1$, which is commonly used for assignment, then message -1 does not belong to $\mathcal{G}$ and thereby testing quadratic residue of ciphertexts will reveal the correspondence of $E[m_0]$ and $E[m_1]$.

An El Gamal encryption of the message m with the public key $y = g^x$ is of the form $E_a[m] = (M, G) = (my^a, g^a)$, where a is a random number chosen from Z_q. To decrypt the ciphertext (M, G), we use the corresponding private key x to compute $M/G^x = mg^{xa-ax} = m$. By element-wise multiplication, we define $E[a] \times E[b] = (M_a M_b, G_a G_b)$,

which yields a new ciphertext $E[a \oplus b]$, and it can be seen that the El Gamal encryption is homomorphic over $GF(2)$.

A distributed decryption is also feasible. A private key is jointly generated by the collaboration of t honest parties (key holders) out of n, and distributed among them using $(t-1)$-degree random polynomials $f(x)$ as $f(1), f(2), \ldots, f(n)$. To decrypt a provided ciphertext with the public key $y = g^{f(0)}$, the i-th party publishes $G^{f(i)}$ for $i = 1, \ldots, t$, and then computes $G^{f(1)\gamma_1} \cdots G^{f(t)\gamma_t} = G^{f(0)}$, where γ_i is the Lagrange coefficient for i. For verifiability, the players use Verifiable Secret Sharing (VSS) as the proof of possession of $f(i)$ such that $G^{f(i)}$.

Another instance of a homomorphic encryption scheme is QR encryption, used in [7].

2.2 Proof of Knowledge

We will use the proof of knowledge of the private input to the computer, which is based on the disjunctive and conjunctive proofs of knowledge in [10].

Conjunctive Proof of Knowledge By $PK\{(\alpha) : y_1 = g_1^{\alpha} \wedge y_2 = g_2^{\alpha}\}$, we denote a proof of knowledge of discrete logarithms of elements y_1 and y_2 with bases g_1 and g_2. Selecting random numbers r_1 and $r_2 \in Z_q$, a prover sends $t_1 = g_1^{r_1}$ and $t_2 = g_2^{r_2}$ to a verifier, who then sends back a random challenge $c \in \{0,1\}^k$. The prover shows $s = r - cx \pmod{q}$, which should satisfy both $g_1^s y_1^c = t_1$ and $g_2^s y_2^c = t_2$.

Disjunctive Proof of Knowledge We use $PK\{(\alpha, \beta) : y_1 = g^{\alpha} \vee y_2 = g^{\beta}\}$ to denote a proof of knowledge of one out of the two discrete logarithms of y_1 and y_2, to the base g. The prover can prove that they know a secret value under which either $y = y_1$ or $y = y_2$ must hold, without revealing which identity was used. Without loss of generality, we assume that the prover knows α for which $y = g^{\alpha}$ holds. The prover uniformly selects $r_1, s_2 \in Z_q$ and $c_2 \in \{0,1\}^k$ and sends $t_1 = g^{r_1}$ and $t_2 = g^{s_2} y_2^{c_2}$ to the verifier, who then provides a random challenge $c \in \{0,1\}^k$, where k is a security parameter. On receiving the challenge, the prover sends $s_1 = r_1 - c_1\alpha \pmod{q}$, s_2, c_1 and c_2, where $c = c_1 \oplus c_2$. The verifier can see if the prover is likely to have the knowledge by testing both $t_1 = g^{s_1} y_1^{c_1}$ and $t_2 = g^{s_2} y_2^{c_2}$ with provability $1 - 2^{-k}$. Note that the same test can be used when t_1 and t_2 are prepared for the other knowledge β.

2.3 Oblivious LFSR Protocol

An oblivious LFSR is an array of n ciphertexts that are updated according to a linear irreducible polynomial.

An LFSR consists of n flip flops, $A_1, \ldots, A_n$. A shift of LFSR is defined for $i = 1, \ldots, n$, by

$$A'_i = A_{i-1} \oplus a_{i-1} A_n \tag{2}$$

where a_i is of characteristic polynomial of t-degree

$$f(x) = 1 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

and $A_0 = 0$, $A'_1 = A_n$, and $a_j \in \{0, 1\}$. A (primitive irreducible) polynomial f has at most $2^n - 1$ states.

An oblivious LFSR is a protocol in which a counter $\mathcal{C}$ communicates with 2^n voters $\mathcal{V}$ once each, and sends the tally (register) to an authority $\mathcal{S}$, who decrypts it and publishes the result. The following are the steps required for oblivious LFSR.

Step 1: (initialization) Counter $\mathcal{C}$ initializes a register $A_1 = E[-1]$, $A_2 = E[1], \ldots, A_n = E[1]$.

Step 2: (voting) Let $b \in \{-1, 1\}$ denote a "Yes" and a "No" vote, respectively. Voter $\mathcal{V}$ sends the value to be updated to counter $\mathcal{C}$, $C_1, \ldots, C_n$, defined by

$$C_i = \begin{cases} E_{r_i}[1] \times A_{i-1} \times A_n^{a_{i-1}} & \text{if } b = -1, \\ E_{r_i}[1] A_i & \text{if } b = 1. \end{cases} \tag{3}$$

Note that when $b = 1$ ("No"), the voter does not change the state of the LFSR, but multiplying with the ciphertext of 1 makes the vote indistinguishable by a third party.

Step 3: (proof of availability) Voter $\mathcal{V}$ proves that the values to be updated are properly computed by Equation 3 in the following zero-knowledge proof,

$$PK \left\{ (r_1, \ldots, r_n) : \begin{array}{c} C_1 = E_{r_1}[A_1] \wedge \cdots \wedge C_n = E_{r_n}[A_n] \\ \vee \\ C_1 = E_{r_1}[A'_1] \wedge \cdots \wedge C_n = E_{r_n}[A'_n] \end{array} \right\}$$

With the proof of the votes, counter $\mathcal{C}$ verifies that the new $C_1, \ldots, C_n$ sent from $\mathcal{V}$ are consistent with the internal states $A_1, \ldots, A_n$ and $f(x)$. If they are incorrect, $\mathcal{C}$ discards $C_1, \ldots, C_n$, otherwise, it replaces the register with the new $C_1, \ldots, C_n$.

Step 4: (opening) When the opening time comes, authority $\mathcal{S}$ decrypts the register as $B_1 = D[A_1], B_2 = D[A_2], \ldots, B_n = D[A_n]$ and publishes $B_1, \ldots, B_n$, which represent a k-th shifted pattern when k voters cast "Yes" votes.

3. IMPLEMENTATION ON CELLULAR PHONES

We implement the oblivious LFSR protocol with the NTT Docomo cellular phone series that supports the Java applications called i-appli [5].

3.1 System Design

Table 1 shows the system specification of the Java software development toolkit [6].

Table 1. Specification for the system

Language	JDK1.3,J2ME Wireless SDK for the DoJa release 2.2
Public-key encryption	ElGamal with 512 bit
Protocol	HTTP/CGI

In our implementation, a client (voter) is a simple Java applet and a server is a Java application equipped with an HTTP server. Both the Java applet and the application consist of seven Java classes, e.g., ModInt class for modular arithmetic integers, and ElGmal class.

3.2 ModInt class

ModInt is a compact Java class designed for arbitrary precision modular arithmetic operations. In order to reduce the size of the library, it omits the following functions supported in the BigInteger class.

- bit manipulation (AND, OR, NOT),
- a sign digit,
- a hexadecimal representation for input and output,
- prime number generation (which may be performed off-line by a more powerful computer).

The ModInt class provides the fundamental 15 methods in 6 kB of bytecode.

ModInt	BigInteger
`p = new ModInt("71a04e8b02");`	`p = new BigInteger("71a04e8b02", 16);`
`q = new ModInt("e0ae02e788");`	`q = new BigInteger("e0ae02e788", 16);`
`m = p.getInstance("dedf322");`	`m = new BigInteger("dedf322", 16);`
`y = p.getInstance("12af7cc");`	`y = new BigInteger("12af7cc", 16);`
`r = q.getInstance("98af3df");`	`r = new BigInteger("98af3df", 16);`
`c = m.multiply(y.power(r));`	`c = m.multiply(y.modPow(r, p)).mod(p);`

Figure 1. Examples of El Gamal encryption using the ModInt and BigInteger libraries

Table 2. Difference between BigInteger and ModInt

	`BigInteger`	`ModInt`
Size of bytecode	40kB	6kB
# of public methods	43	15
Integers	signed, variable length	unsigned, fix length (specified by p)
Input format	decimal, string, byte array	hex,int array
Internal object	big endian, int array $\times$ 1, sign, bit length	big endian, int array $\times$ 2

In the ModInt class, a value is an object implicitly bound to a modulo p and is computed in (mod n). In order to distinguish ModInt from BitInteger, let us consider an example of part of the ElGamal encryption algorithm, defined by

$$c = my^r \bmod p.$$

If this is executed in BigInteger, after instantiating four objects, m, y, r and p, we must write explicitly

```
c=m.multiply(y.modPow(r, p)).mod(p),
```

where two modular commutations are performed, incurring an overhead when c is extended to size $2p$, and then is reduced to size p.

In ModInt, we simply write

```
c=m.multiply(y.Pow(r))
```

where modulo p is implicitly computed. A more complete source is shown in Figure 1.

We illustrate the difference between `BigInteger` and `ModInt` in Table 2, and show some fundamental Java constructors and methods in Table 3 and 4, respectively.

Table 3. List of Constructors

Constructor	Description
`ModInt(String p)`	Set modulus p in hexadecimal format
`ModInt(int[] a, int[] p)`	Generates an instance a mod p

Table 4. List of instance methods

Return value	Method	Description
`ModInt`	`getInstance(String s)`	Generates an instance of s in hexadecimal format. The modulo p of s is identical to `this`.
`String`	`toString()`	Returns a string of `this` value represented in hexadecimal format.
`ModInt`	`subtract(ModInt b)`	Returns an instance of $this - b \pmod p$
`ModInt`	`add(ModInt b)`	Returns an instance of $this + b \pmod p$
`ModInt`	`multiply(ModInt b)`	Returns an instance of $this \times b \pmod p$
`ModInt`	`shiftLeft()`	Returns an instance of $this \ll 32 \pmod p$
`ModInt`	`power(ModInt e)`	Returns an instance of $this^e \pmod p$
`long`	`compareTo(ModInt b())`	Compares $this$ with b in number, and returns 0 if $this = b$

Table 5. Execution environment of the emulator

CPU	Pentium III 1.13G, 512 MB, 40GB
OS	Windows 2000 Server

4. EVALUATION

We evaluated our system on two platforms: an i-appli emulator and a commercially-available cellular phone (the NTT Docomo D503is).

4.1 Performance in the J2ME Emulator

Table 5 shows an environment running the i-appli emulator.

In Table 6 and 7, we show the performance of our system in the i-appli emulator for J2ME, where Key, Carry, Enc and Shift are the processing times for reading the public key, reading the register, computing re-encryption, shifting the LFSR, and sending votes, respectively. The $|p|$ represents the size of p in bits.

Table 6. Processing time for key size $|p|$ [s] ($n = 3$, Emulator)

$\|p\|$ [bit]	Key	Carry	Enc	Shift	Write	Total
128	0.6	0.3	0.7	0.00	0.03	6.4
256	0.5	0.4	2.7	0.01	0.05	8.6
512	0.7	0.6	18.9	0.10	0.10	27.4
1024	0.8	1.1	40.0	0.17	0.16	47.1

Table 7. Processing time for bit length of register n [s] ($|p| = 512$,Emulator)

n	Key	Carry	Enc	Shift	Write	Total
3	0.5	0.4	2.7	0.10	0.05	8.6
5	0.6	0.8	23.2	0.04	0.14	30.9
8	0.6	1.2	40.1	0.10	0.22	48.9

Table 8. Processing time for key sizes $|p|$ [s] ($n = 3$, D503is)

$\|p\|$ [bit]	Key	Vote	Total
128	2.8	25.4	41.7
256	5.4	92.7	145.8
512	3.2	426.7	443.6

We show the processing time varying with the size of p for $n = 3$ in Table 6, and n in $|p| = 512$ in Table 7.

From the tables, we notice that the performance bottleneck is within the cryptographical processes rather than the file input-output processes. At most, the encryption takes more than 80% of the total time.

4.2 Performance in the Docomo D503is

We demonstrated the implemented system on the NTT Docomo D503is (see Figure 2). The performance of the D503is is reported in Table 8 and 9 and in Figure 3 and 4. The value $|p| = 512$ is used for this system.

Based on the experimental performance, we observe that the time for processing depends on n and $|p|$. It should be noted that the D503is takes about 23 longer than the PC emulator does, as power consumption concerns limit the computational power of cellular phones in mobile environments.

4.3 Pre-computation to Reduce Processing Time

According to the experimental results, we can estimate an average processing time for any given parameter. For example, if $|p| = 512$ and

Figure 2. A screenshot of the implemented system

Table 9. Processing time for bit lengths of register n [s] ($|p| = 512$, D503is)

n	Key	Vote	Total
3	3.2	426.7	443.6
5	7.3	664.6	686.2
8	5.2	1055.4	1084.7

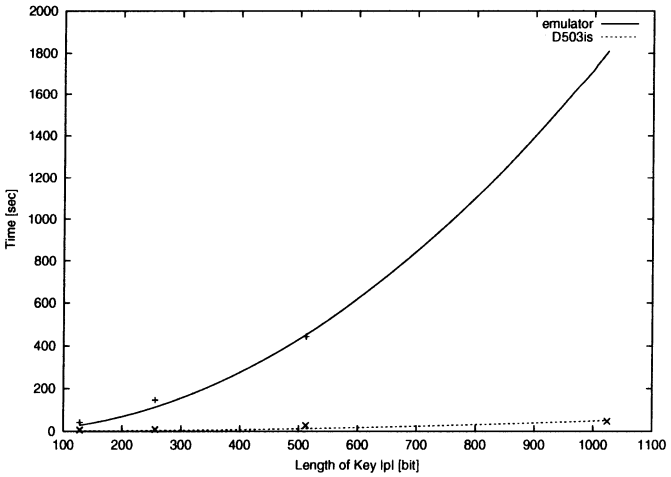

Figure 3. Processing time for key size $|p|$ [bit]

there are 0.1 million voters, then by calculating $n = \log_2 100,000 = 17$, we expect about 37 minutes per voter, which is not realistic.

Hence, we try to reduce the number of cryptographical processes. Recalling the updating formula

$$C_i = E[1] \times A_{i-1} \times A_n^{a_i - 1},$$

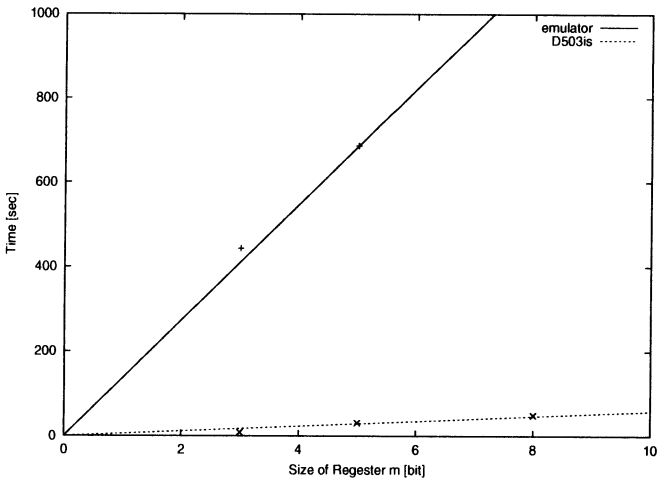

Figure 4. Processing time for length of register n

Table 10. The estimated reduction in voting time [s] ($|p| = 512$). Original: the performance in N503is, Enc: the estimated encryption time based on Table 7, and Reduced: the expected reduction by pre-computation

n	Original	Enc	Reduced
3	443.6	348.8	94.8
5	686.2	539.5	146.7
8	1084.7	852.8	231.9

where $E[1]$ is independent of both the register A_i and the vote b, and thus can be recomputed before casting. Unless the internal memory is compromised, we can store as many $E[1]$ values as possible to skip the encryption steps. Based on the experimental data in Table 7, we show the expected reduction in time in Table 10, which demonstrates that the highest reduction is about 22 %.

4.4 Scalability of the Proposed System

We estimate the scalability of the proposed system based on the experimental processing time. Let us assume that the maximum waiting time is 30 seconds. We may then estimate how many powerful computers would be required when elections are conducted at several scales, ranging from 32 voters for a tiny class president election to a 300 million-scale election for governors. Table 11 illustrates the estimation. It shows that computation time can be reduced by about 22%. If we hold a 1000-scale election, we need a Java processor in the cellular phone that is 10.1 times faster.

Table 11. Estimated improvement of CPU power requirement for elections at several scales

Election scale	# of users m	# of registers n	orig.		w. precomp.	
			Emu.	D503	Emu.	D503
Class president	32	5	1	23.5	1	5.17
Student body president	1000	10	1	45.7	1	10.1
City mayor	10000	14	2.6	60.7	1	13.4
State governor	3 million	22	4.2	98.4	1	21.6

Table 12. Comparison between [1] and the proposed system

System	System [1]	Proposed system
Voting	$O(1)$	$O(n)$
	0.5 s	23.3 s
Mixing	$O(2^n)$	
	2.52 s	N.A.
Opening	$O(1)$	$O(n)$
	3.05 s	1 s

4.5 Comparison to the Conventional System [1]

In [1], an electronic voting system is reported. The system uses a blind signature with mix servers for preserving the privacy of voters, the Schnorr signature for the administrator, threshold El Gamal encryption for the public key cryptosystem, and hybrid mix networks combining the Diffie-Hellman and the DES algorithms for shuffling the votes.

The largest difference to our system is the existence of an anonymous channel, such as mix and shuffle. In our system, the anonymous channel is not required, but a greater computational cost is needed for updating registers.

We summarize the comparison in Table 12, where the performance with 10 users is evaluated on a Pentium II 400 MHz and the digital signature processes are omitted in [1]. The performance in our system is estimated from Table 4.

In conclusion, by removing the time-consuming process of mixing, which takes a time proportional to the number of voters, we reduce the processing time at the server. The cost is dependent on the number of registers, which is the log of the number of voters.

In comparison to the electronic voting in Niimi city, our system does not require any special devices such as the voting machine and the voting card. The cellular phone is a versatile tool and hence it is better choice as the voting device. Moreover, the black-box machine cannot convince

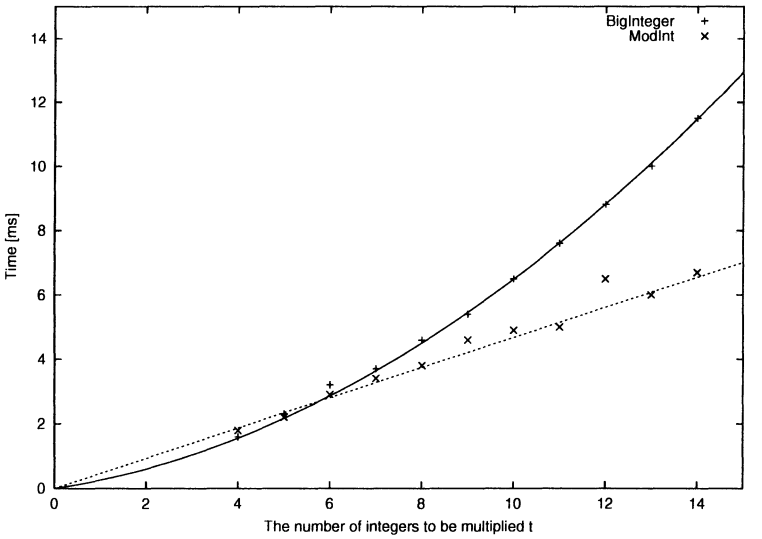

Figure 5. Processing time for `BigInteger` and `ModInt` [ms]

people that the tallying has been performed correctly and the privacy of voting is preserved. The every step in our system is publicly verifiable to any party who wish to verify the accountability.

4.6 Performance of the ModInt Library

The performance advantage of the ModInt library, compared to the BigInteger library, is due to improvements to the multiplication of a few integers at once. For instance, letting a, b and c be 1024-bit BigIntegers, we write

```
a.multiply(b).multiply(c).mod(p),
```

whose size is expanded up to 3072 bits once, before being reduced back to 1024 bits. The ModInt, in contrast, constantly maintains a size of 1024 bits. In general, the processing time with respect to the number of multiplications t is shown in Figure 5.

The result shows that the ModInt achieves better performance than BigInteger when t is greater than 6, although t is limited to less than 3 in the El Gamal encryption in our system.

5. CONCLUSION

In this paper, we have proposed an efficient cryptographic protocol for a secure electronic voting system and have shown that the proposed protocol is feasible for a Java-enabled cellular phone platform with limited

computing power. From the experimental results, we determined that a cellular phone processor 22 times faster than those currently available is required to hold a 3 million voter election, and that pre-computation helps by saving 22% of the processing time. Our implemented ModInt Java class library is specially designed for modular arithmetic and provides better performance than BigInteger.

References

[1] A. Fujioka, M. Abe, M. Ohkubo and F. Hoshino, "Implementation and Experimentation of Practical Electronic Voting", Proc. of Symposium on Cryptography and Information Security, Vol 48, 2000 (in Japanese).

[2] M. Abe, "Universally Verifiable Mix-Net with Verification Work Independent of the Number of Mix-Servers," IEICE Trans. Fundamentals, Vol. E83-A, 2000.

[3] J. Furukawa and K. Sako, "An Efficient Scheme for Proving a Shuffle," Proc. of CRYPTO'01, LNCS 2139, pp. 368–387, 2001.

[4] Japanese Governmental Survey in Influence of Information Technologies at Home in 2002, (http://www5.cao.go.jp/seikatsu/2002/0405it-chousa/index.html).

[5] NTT Docomo Official Page, (http://www.nttdocomo.co.jp/ September 2002).

[6] i-mode Official Page, (http://www.nttdocomo.co.jp/p_s/imode/ September 2002).

[7] J. Katz, S. Myers and R. Ostrovsky, "Cryptographic Counters and Applications to Electronic Voting," in proc. of EUROCRYPT'02, pp. 78-92, 2001.

[8] R. Crammer, I. Damgard, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," Proc. of CRYPTO '94, pp. 174–187, 1994.

[9] R. Cramer, R. Gennaro and B. Schoenmakers, "A Secure and Optimally Efficient Multi-Authority Election Scheme," Proc. of *EUROCRYPT 1997*, 2001.

[10] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," Proc. of *CRYPTO '94*, pp.174–187, 1994.

[11] M. Jakobsson and A. Juels, "Mix and Match: Secure Function Evaluation via Ciphertexts," Proc. of *ASIACRYPTO 2000*, LNCS 1967, pp. 162–177, 2000.

[12] "E-Japan Priority Policy Program 2003", IT Strategic Headquarter, (http://www.kantei.go.jp/foreign/policy/it/0808summary/030808gaiyo_e.pdf), August 2003.

[13] "'The electromagnetic recording voting device" (the special law of electronic voting), February, 2002.

[14] "Regulation regarding Voting by Electromagnetic Recording Voting Device in the Election of Niimi City Assembly and Mayor", Niimi City, 2002.