International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 12 (December 2014), PP.01-12

Divisible Transferable Anonymous Electronic Cash System for General Use

Israt Jahan[1], Mohammad Zahidur Rahman[2] ,
Liton Jude Rozario[3] and Kaafi Mahmud Sarker[4]
[1,2,3,4]*Dept. of Computer Science and Engineering, Jahangirnagar University, Savar, Dhaka, Bangladesh.*

Abstract:- In this paper we propose an elegant and unique divisible transferable anonymous electronic cash system with observer. The proposed divisible e-cash solves most of the crucial problems with existing paper cash and untraceable e-cash proposals. Electronic cash provides unconditional anonymity to the user. Researchers observed, however, that if anonymity in payment system is unconditional, it might be exploited to facilitate crimes like money laundering. This observation spurred research into the idea of making anonymity in payment systems conditional, and, in particular, revocable by a third party or trustee or observer under bank's order. The idea of having an observer is that it can be incorporated in the wallet in such a way that no user module can do a transaction on its own. For any transaction protocol to be executed by the wallet, it needs help (a secret information) from the observer i.e. the wallet and the observer must confirm mutually when they work together. The advantage of the proposed electronic cash system is that it is able to construct an observer capable of co-operating with divisible and transferable e-cash. Due to the presence of observer, the proposed e-cash has prior resistance of double spending. In each transfer of divisible e-cash, coin authentication and denomination revelation is checked to verify the validity of divisible e-cash. In any stage of coin transfer, the anonymity is guaranteed with protection of double spending.

Keywords:- Binary tree, Blending technique, Smartcard, Java card, Observer.

INTRODUCTION

With the onset of the Information Age, people are becoming increasingly dependent upon net-work communications. Computer-based technology is significantly impacting our ability to access, store, and distribute information. Among the most important uses of this technology is electronic commerce: performing financial transactions via electronic information exchanged over Internet. A key requirement for electronic commerce is the development of secure and efficient electronic payment systems. In light of the explosive increase in electronic services, means for electronic payments become an essential asset. Electronic cash or e-cash refers to cash and associated transactions performed with it on an open communication network. An observer is a tamper resistant device that prevents double-spending physically. The user module is called a wallet since, it actually carries money.

Special protocols are used to manage secure e-cash transactions [1-3]. Ideal e-cash [4] system should be independent, anonymous, unforgeable, divisible, unlinkable, undouble-spending, transferable and of course, offline [5]. Though cryptography solves some of the problems such as anonymity, offline etc., double-spending cannot be prevented only through cryptography. Most of the e-cash authors suggested use of a temper-proof hardware as an observer of the system [5-13]. Due to the presence of observer, the proposed e-cash has prior resistance of double spending. If user spends same coin twice or manipulates information inside observer, the observer drops working [13].

The advantage of the proposed electronic cash system is that it is able to construct an observer capable of co-operating with divisible and transferable e-cash. A user who generates a divisible coin can transfer his any divisible amount of e-cash to another user and to a number of users subsequently without losing anonymity and without contacting the bank between the two transactions. Smartcard is the best candidate to be an observer, but implementation of very efficient and secured e-cash protocols with observer cannot come to reality due to limited resources in smartcards.

Basic Card [12], MUSCLE [15] Card and Java Card [16] are three different types of programmable smartcards available in the market. All these smartcards are resource constrained devices, and we tried to overcome this limitation by applying different techniques and associating with wallet and CORBA-based high-level bank server. Bank server is powered by LiDIA libraries. LiDIA can handle larger integers essential for e-cash security, and can suitably be implemented in bank and wallet components. Length of secret keys generated by smartcard needs to be matched with those generated by the bank and the wallet. However, without compromising security issues, the key published by bank, random numbers, observer's secret key, and different computational parameters exchanged among bank, wallet and smartcard are synchronized in this paper. The processing cost and response time of smartcard is optimized by distributing some processing to card hosts. Parameters are reduced in size using one way hashing and matching when they are being participated in various calculations inside the observer. We refer this technique as *blending technique*. We choose a realistic e-cash protocol and carefully analyze performance of each type of smartcard as e-cash observer.

MODEL OF DIVISIBLE TRANSFERABLE E-CASH WITH OBSERVER

E-cash is similar to paper cash and should guarantee anonymity. David Chaum [17] proposed an anonymous payment protocol in 1983 introducing the concept of e-cash. To prevent illegal copies of coins Chaum also described a method of preserving a list of all coins spent. Bank can verify any coin before it is being reused. But the verification needs to be performed online; otherwise, the payment will not be a valid payment. However, this limitation was corrected by Chaum, Fiat and Naor [18]. They proposed an offline e-cash model where transactions can be done online even the bank remains offline. RSA [19] public key based blind signatures and use of one way hash function are the strength of their protocol. The major drawback of these protocols is bank can detect double-spending only when coins are deposited back to the bank which might incur a huge loss.

Okamoto and Ohta [20] added divisibility property to e-cash based on binary tree, and hardness of the factorization and quadratic residues problems. Though binary tree representation efficiently handles large value of coins, anonymity is compromised in this scheme. Ferguson [7] proposed single-term scheme, where RSA signatures are combined with random blind signatures. He added a secret-sharing concept with which it is possible to know who commits fraud, but unfortunately, the security of this protocol was not guaranteed [21]. S. Brands's scheme superseded others and considered as practical single-term e-cash. Earlier Chaum and Pedersen [22] introduced almost similar protocol. Yet the time between a fraud incident and its detection do not guarantee banks for their financial losses.

Recently, Liu, Luo, Si Ya-li, Wang and Li Feng worked with *N*, *K* based payment protocol with observer, where *N* denotes total coins value and *K* denotes payment times. It solves double-spending effectively with the prior restraint smartcard. But it demands huge storage when large value of e-cash needs to be withdrawn. However, Israt Jahan [13] proposed an elegant method with binary tree representation of coins, where large valued e-cash can be efficiently handled. Use of a prior-restraint smartcard not only protects reuse at the time a coin is being spent but also lightens the burden of the bank. Fig. 1 shows the basic model.

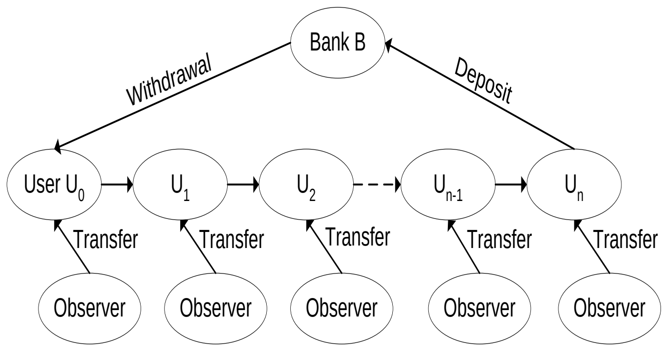


Figure 1. Model of offline transferable e-cash with observer.

PROTOCOL DESCRIPTION

In the protocol described in [11]. Here p, q, g, g_1, g_2 and H are the system parameters published by the bank where the orders of g, g_1, g_2 are x and x_t *are q*. x and x_t are the secret key of bank. x will be used for issuing digital cash and x_t will be used for issuing coin extension. $h=g^x$ and $ht=g^{xt}$ are the public keys of the bank.

A. Bank's Setup, Account Opening and Withdrawal of E-cash by User U_0 from Bank

Bank setup, account opening and withdrawal of e-cash by user U_0 appears in Fig. 2. While opening an account and issuing the observer to the user, in *Setup1()*, bank chooses $oa_0 \in Z_q^*$ as the secret key of the observer. In *Calculate1()* the observer computes $AO_0 = g_1^{oa_0}$ and sends it to the host. Let u_0 is the secret key of the user which may be stored in user's wallet. The wallet calculates $I_0 = OAg_1^{u_0}$ in *Setup2()* and sends this value to the bank. The user sends the withdrawal amount, ω_l to the bank. The wallet performs computation and forms a binary tree rooted with the amount ω_l. In *Calculate2()* the observer performs calculation and forms the tree by itself. It sends the encrypted value of nodes and the root to the wallet. The bank deducts ω_i amount from user's account balance and calculates z_0, a_0, b_0 in *Calculate3()* and sends to the wallet. The wallet generates c_0 and sends it to the bank. Bank calculates r_0 and sends the value to the wallet. At last, the wallet obtains, verifies and stores the signature in it.

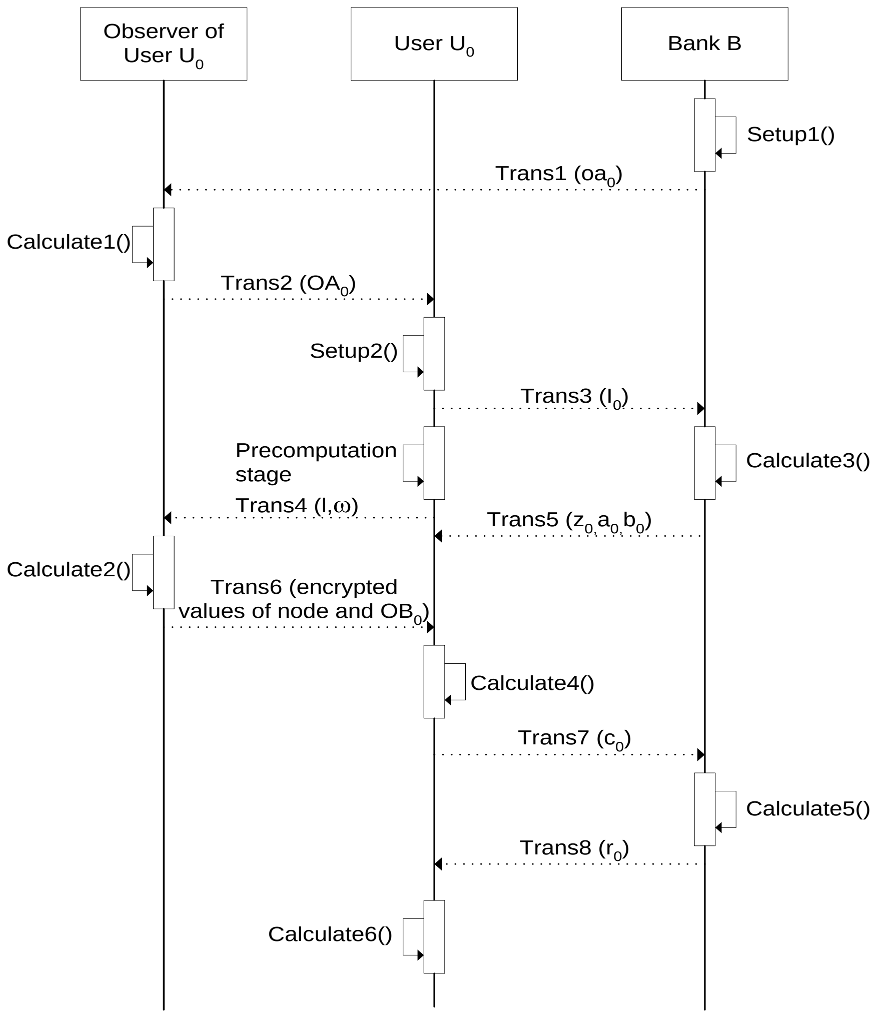


Figure 2. Bank's setup, account opening and withdrawal of e-cash by user U$_0$.

B. *Withdrawal of Coin Extension by User U_1 to Receive Transferred E-cash*

Withdrawal of coin extension is almost similar to withdrawal protocol described in the earlier section. Only difference is here user pre-withdraws zero-valued coins from bank. Fig. 3 elaborates the protocol. In *Calculate2()* the smartcard chooses $O \in_R Z_q^*$ and calculates $OB_1 = g_1^{O_1}$.

C. *Coin Transfer from User U_0 to U_1*

Fig. 4 shows coin transfer protocol between users. In *Calculate1()* the user's wallet U_0 reveals $n_{0j_1j_2-j_k}$'s contribution of its corresponding ancestor nodes as $\beta_{0j_1j_2-j_k} = g_1^{y_{0j_1j_2-j_k,1}} g_2^{y_{0j_1j_2-j_k,2}} \bmod p$. In the next step the user U_0 reveals other related nodes. User U_0 forms the transcript $Tr_0 = \{m_0', T, A_0, Sign(m_0', T), a_k, a_{k-1}, \ldots, a_2, a_1, \beta_{0j_1j_2-j_k}\}$

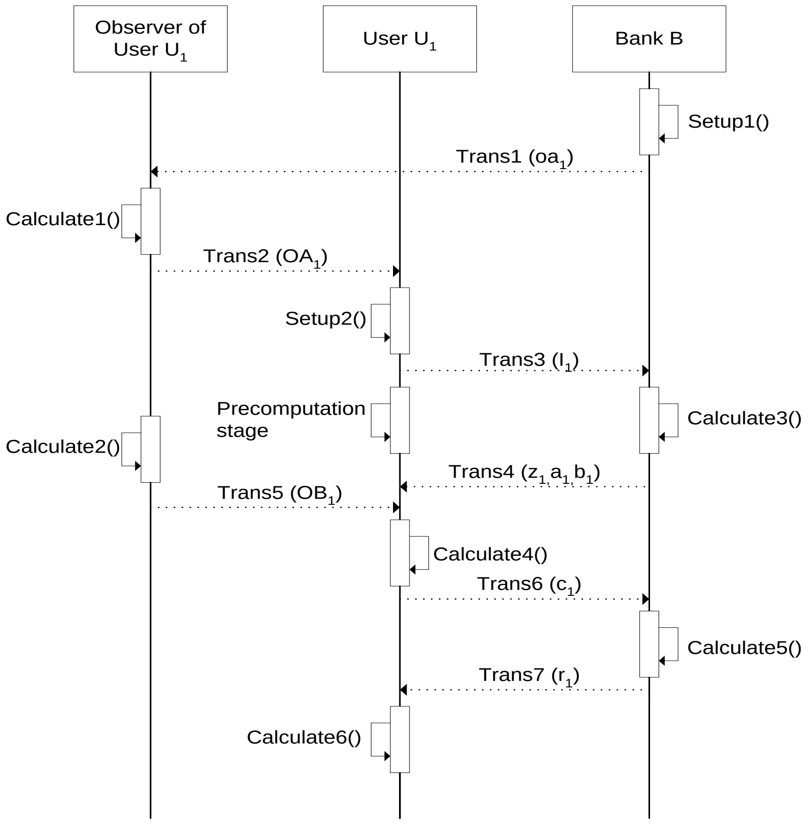


Figure 3. Withdrawal of coin extension by user U_1.

and sends it to user U_1. In *Calculate2()* user calculates ϕ_0 and sends it to U_0. In *Calculate3()* U_0 calculates $\phi_0^{'}$ and sends $\phi_0^{'}$ and corresponding node of the coin to the observer of U_0. If the node is found then it is erased. The observer calculates $\rho_0^{'} = \phi_0^{'} oa_0 + O_0 \bmod q$. In the last step it checks if any node exists; if not, then erases O_0.

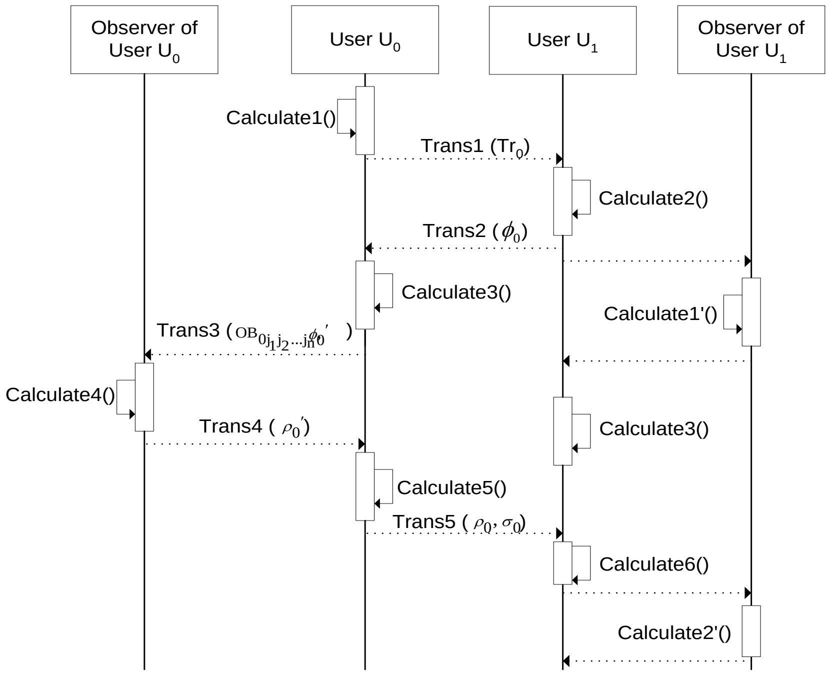


Figure 4. User U_0 spends his coin to user U_1.

D. Deposit

User deposits his coin to the bank. If the coin is not deposited previously it is deposited to the account.

IV. IMPLEMENTATION OF OBSERVER IN SMARTCARD

The objective is to use smartcard as an observer of an e-cash model. To be an observer, smartcard needs to ensure enough security by itself, and of course, it must be synchronized with the user wallet application. Moreover, smartcard processing cost must be within the tolerable limit. We designed and built an observer which is fully compatible to the divisible, transferable, e-cash protocol proposed in [13] and completed the proposal. The user (the wallet application running in customer's PC) and the bank are the major actors in this protocol. Observer is another piece of program which is associated with the user and loaded into a smartcard. The bank issues it to its customer to operate along with his/her user application. The customer possesses the smartcard and uses it in every transaction he/she makes through his/her user application. The user and the bank communicate between themselves using a network in a distributed environment. The interface between the bank and the user is designed as CORBA Interface Definition Language (IDL). The bank and the user program are developed in C++ language under Linux platform.

Several experiments are carried out to find out the suitability to be an observer without compromising major features of an efficient e-cash scheme. Suitable environments are setup both for development and testing to carry out each experiment. At first we set up environment using Basic Card and analyze its suitability. Then we choose MUSCLE card with PC/SC lite driver installed in Linux environment. Our last experiment is carried out with Java card. Following sections describe research approaches to find a realistic observer solution for e-cash operation.

A. Research Approach with Basic Card

Basic Card [12] is a programmable smartcard. Programs executed in Basic Card are written in ZC-Basic language. Size of RAM ranges from 265 to 1768 bytes and that of user programmable EEPROM is from 1 to 31 kilobytes. The EEPROM contains the user's application code which is compiled and converted into virtual machine language called P-Code. User's permanent data is stored in EEPROM, and the RAM contains run-time data including the P-Code stack.

The observer is implemented in card-side, and the request towards the card comes through the card terminal. The whole setting works in the basis of client-server model. We develop a set of programs for both the card and the terminal. The card-side code receives parameters and instructions from the outside world (usually the user wallet), generates secret key with the help of cryptographic libraries which could be used as digital coins, and stores them safely inside the card. The card processor sends a successful status to the outside world when the generation of all coins of demand is completed.

Another piece of code is written for the card processor to erase the spent coin from the stored digital coins' list. If the coin is not spent i.e., not erased earlier, the processor erases the same coin, and sends a successful status to the outside world; otherwise, it sends a “double spent” status. Sample code is given in Fig. 5.

```
...
Call Compare Key (Key$)
If Key$ = "" Then
Print "Double spent"
Exit
Else
Call Remove Key (Key$)
...
...
```

Figure 5. Sample code for Basic Card.

Source code is compiled into .img file and the image file is downloaded into the Basic Card using a card loader tool. Another way is to compile the source file into P-Code, an intermediate language that can be thought of as the machine code for the card's virtual machine. P-Code is downloaded to the card using the card loader tool. The virtual machine in the Basic Card executes the P-Code instructions at run-time when instructions are passed to the card.

A terminal-side code initializes and calls the functions and procedures stored inside the card. This part of the source code is compiled into standard executable files with .exe extension.

Limitations of Basic Card as Observer: Though Basic Card has the capability of handling standard cryptographic functions and it is much simpler and less costly to develop, it loses suitability to be the candidate observer because of the following reasons:

1. The Observer operating environment must support, or, at least, must be capable of interacting to CORBA interfaces. ZC-Basic does not have the capability to interface with CORBA.

2. Though Basic Card provides support for almost all security algorithms such as RSA with AES and DES encryption, EC cryptography, SHA-1 etc, it is not dependable when it would be used in off-line e-cash domain or even as e-cash processing unit. At least, Basic Card has to support the calculations used in the e-cash protocol.
3. A general purpose e-cash needs to interact to various types of applications such as travel tickets, gasoline, parking, shopper, etc in addition to the wallet, purse, loyalty etc written in different programming languages and deployed for different organizations. Integration among such applications, specially in the global aspect, would be almost impossible unless there is a common interface platform. ZC-Basic language is designed with the Basic Card in mind. Here, all programs must be written in ZC-Basic language and the use of Basic Card hardware must be ensured.
4. Basic Card does not provide open platform. Hence, deployment and use of new encryption algorithms are not possible. Someone has to wait until it is enabled by the manufacturer itself.

B. Research Approach with MUSCLE Card

MUSCLE [15] has been passed through a number of changes, Hence, installation and usage of smartcard software under MUSCLE is a complex task. The experiment was carried out on Linux (Fedora 8) environment and the smartcard devices were collected from ACS [21]. We used the ACR38DT smartcard reader/writer and the Gem Combi smartcard from Gem Plus Inc [24].

We installed and configured PC/SC Lite driver (pcsc-lite-1.3.3.tar.gz), ACR38DT driver (ACR38UDriver-1.8.0-1.i386.rpm), libmusclecard-1.3.3, MuscleCard BundleTool, MuscleCard Framework 1.1.6 and MuscleCard Plugin. Then we inserted the GemCombi smartcard into the card reader, selected appropriate communicator from the BundleTool and ran the test program. Fig. 6 shows a code snippet to connect the card reader.

```
// connect to card
Sz Readers = msz Readers;
rv = SCard Connect ( hContext,
              sz Readers,
              SCARD_SHARE_SHARED,
              SCARD_PROTOCOL_T0,
              &hCard,
              & dw Active Protocol);
...
...
```

Figure 6. MUSCLE card code snippet.

Findings from MUSCLE Card Experiment: MUSCLE Card could be the best e-cash observer candidate. Recent MUSCLE researches approach towards distributed processing where there is separation between application code inside the card and the card operating system (COS) itself. Java has the same capability and hence, MUSCLE extends the research towards Java enabled smartcards called Java Card.

C. Research Approach with Java Card

In Java card, the Java Card Runtime Environment (JCRE) requires a fair amount of computational power in order to work properly. Experiments were carried out in Java card simulator using 16K of ROM, 8K of EEPROM, and 256 bytes of RAM. We used Java Card API specification provided by Sun Microsystems Inc. It requires Java card Workstation Development Environment (WDE). We developed the card services using new classes to run on Java card platform RMI API, and the card services ran on simulated environment using T=1 protocol. The card services used port number 9011 and 9022 with interfaces to bank and wallets. Bank and wallet components were developed in CORBA architecture using LiDIA library sets in C++ language, and the observer is was implemented in Java language. The bank server and wallet ran on two separate PCs with 1.7 GHz processor each. The experiment was carried out in Linux environment. Fig. 7 shows components' architecture and their interfaces. Fig. 8 shows an interfacing code segment.

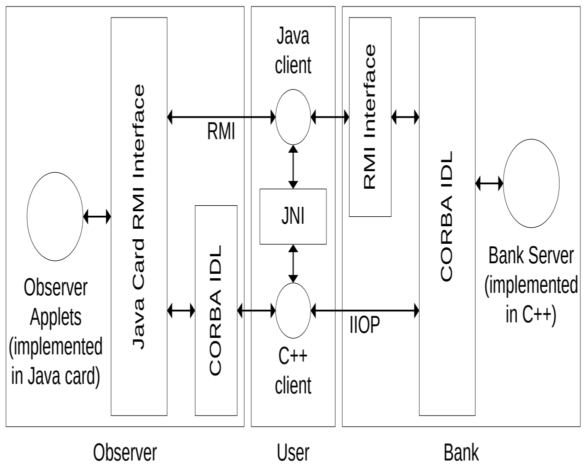


Figure 7. Wallet with Java card observer and bank server component architecture.

```
...
...
Account_ptr
Bank_impl::create()
Account_impl * ai = new Account_impl;
Account_ptr aref = ai->this();
assert (!CORBA::in_nil (aref));
...
...
```

Figure 8. Code segment shows pointer for bank interface.

V. DESIGN ISSUES

Bank issues Java based smartcards to users. The smartcard contains the observer for e-cash transactions. User performs e-cash transactions through ATM or his/her PC. If ATM is used to perform e-cash transactions, ATM carries out the tasks those the user's wallet does. If e-cash transactions are not performed through ATMs, the user's PC runs the wallet. In this case, a smartcard reader device is attached to the user's PC. Both the ATM and the wallet, running in user's PC, are connected to the bank component. The wallet, running in ATM or user's PC, performs e-cash loading, spending and depositing to bank in presence of respective observers.

In this design, we use *ATM* as both the smartcard reader and the execution platform of the wallet application. The term *User* is used to represent the wallet. *Observer* of the e-cash system is the smartcard system.

A. Use Cases

Following are the use cases of the system.

Use Case 1: Bank setup and account opening

Precondition	Customer inserts Observer (smartcard) into the ATM.
Successful outcome	*h* and h^t are stored in Observer and ejects the Observer to customer.
Primary actor	ATM
Secondary actor	User, Observer and Account
Main scenario	1. User inputs *x* and x^t into the Bank system. 2. Bank generates *h* and h^t, and stores them into the Observer.
Post scenario	*h* and h^t are stored in the Observer and ejects the Observer to customer.

Use Case 2: Withdrawal of e-cash by user U_0

Precondition	Customer has balance in his account maintained in the bank and has a valid Observer (smartcard) inserted into the ATM.
Successful outcome	Coins are generated into the wallet and transferred to the Observer.
Primary actor	ATM, User and Observer
Secondary actor	Account
Main scenario	1. Customer inputs withdrawal amount. 2. Bank creates coins and stores them into the Observer.

Use Case 3: Withdrawal of coin extension by user U_1

Precondition	Customer has an account in the bank.
Successful outcome	Blank coins are generated into the Observer.
Primary actor	ATM, User and Observer
Secondary actor	Account
Main scenario	Bank creates blank coins and stores them into the Observer.

Use Case 4: Coin transfer from user U_0 to user U_1

Precondition	1. Two Observers inserted into ATM. 2. Two Users communicate to each other.
Successful outcome	Coins are transferred from one User to other User.
Primary actor	1. Spending User 2. Observer of spending User 3. Receiving User 4. Observer of receiving User
Main scenario	1. Spending User inputs amount to be transferred. 2. Spender’s Observer validates input and erases required number of coins. 3. Spending User creates a transcript and transfers it to receiving User. 4. Receiving User stores the received transcript into receiver’s Observer.

Use Case 5: Deposit

Precondition	Customer inserts Observer (smart-card) into the ATM.
Successful outcome	Coins are deposited to customer's account in his bank.
Primary actor	ATM
Secondary actor	User, Observer and Account
Main scenario	1. Customer inputs amount to be deposited. 2. User generates a transcript and sends it to the ATM. 3. ATM validates the transcript. 4. Observer erases required coins if the Bank sends acknowledgement.
Post scenario	h and h^t are stored in the Observer and ejects the Observer to customer.

B. Parameters

p, q, g, g_1, g_2, H : System parameters published by the bank

x, x^t : Secret keys of bank

h, h^t : Public keys of bank

u_0, u_1 : Users' secret keys

oa_0, oa_1 : Observers' secret keys

I_0, I_1 : Identity of users

$\{m, T, z, a, b\}$: Generated coin using binary tree based divisibility

$\{A, B, z, a, b\}$: Coin extension

$\{m, T, I, Date, Time\}$: Transferred coin to other user

C. Implementation Classes

Since APIs, supporting blending technique, are not specified in Java card specification, we designed blending APIs to perform various actions inside Java card. Classes and interfaces related to blending keys and parameters those used in Java card are as follows:

- javacard.security.BlendRandom - This class helps generating random values.
- javacard.security.BlendBigInt - This class helps blending large integers of bank server and unmatched integers inside Java card observer.
- javacard.security.BlendBigMod - This class helps calculating blended modulus for large parameters of bank server and unmatched parameters inside Java card.
- javacard.security.BlendPower - This class helps calculating blended power for large integers of bank server and unmatched integers inside Java card.
- javacard.security.BlendGenerateTree - This class helps constructing binary tree for blended values of coin inside Java card.

Fig. 9 shows sample code for using APIs related to blending technique.

```
...
...
Blend Big Int coin Count = (Blend Bid Int) (wl/l);
Byte [] buffer = new byte[coin Count];
Blend Random random = Blend Random. Get Instance (Blend Random. RAND);
...
Blend Generate Tree tree = Blend Generate Tree. set Coin (total Value, tree Depth);
...
...
```

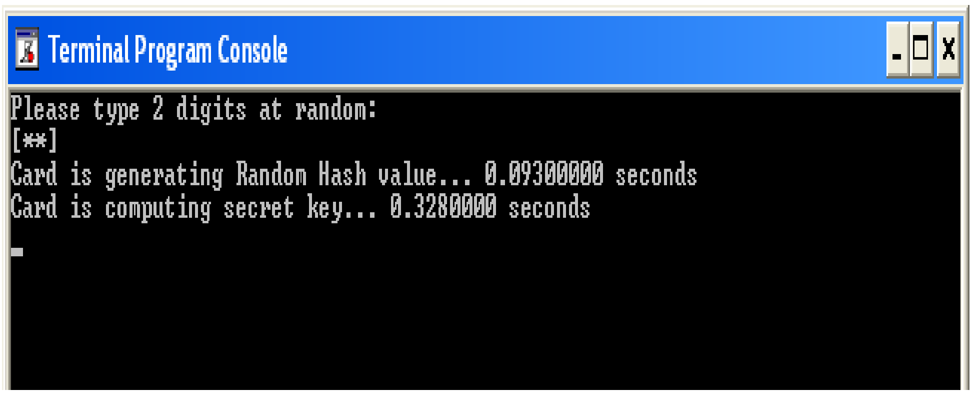
Figure 9. Sample code for using APIs related to blending technique.

VI. RESULTS

We used various sizes of keys in the observer with fixed key size of bank and wallet. Fig. 10 and Fig. 11 are two screenshots while the experiment was carried out using Basic Card.

```
..
Generating Secret key for card
EEPROM Max size: 8096 byte
Card side Code size: 7258 byte
EEPROM Used: 7258 byte
Inititlization code size: 3552 byte
Terminal code size: 3674 byte
..
```

Figure 10. Basic card runtime environment.



```
Please type 2 digits at random:
[**]
Card is generating Random Hash value... 0.09300000 seconds
Card is computing secret key... 0.3280000 seconds
```

Figure 11. Sample output of Basic Card program.

Table I shows Basic Card performance with various sizes of integers used to generate secret keys.

Table I. Performance analysis of basic card with various sizes of secret keys

Integer Size	Time for Random Hash (sec)	Time for secret key (sec)	Total time taken (sec)
1	0.092	0.0291	0.1211
2	0.094	0.0328	0.1268
3	0.095	0.0355	0.1305

In the experiment using Java card we used various sizes of keys in the observer with fixed key size of bank and wallet. Sample output of compiling, loading and execution of applets is given in Fig. 12.

```
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
...
...
CLA: 80, INS: 30, P1: 00, P2: 00, Lc: 01, 64, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: 50, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 64, SW1: 90, SW2: 00
CLA: 80, INS: 40, P1: 00, P2: 00, Lc: 01, 32, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: 50, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 32, SW1: 90, SW2: 00
CLA: 80, INS: 30, P1: 00, P2: 00, Lc: 01, 80, Le: 00, SW1: 6a, SW2: 83
CLA: 80, INS: 50, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 32, SW1: 90, SW2: 00
CLA: 80, INS: 40, P1: 00, P2: 00, Lc: 01, 33, Le: 00, SW1: 6a, SW2: 85
CLA: 80, INS: 50, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 32, SW1: 90, SW2: 00
CLA: 80, INS: 40, P1: 00, P2: 00, Lc: 01, 80, Le: 00, SW1: 6a, SW2: 83
CLA: 80, INS: 50, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 32, SW1: 90, SW2: 00
...
...
```

Figure 12. Snapshot of the output of Applet Compilation and Loading.

Table II shows processing time for various sizes of keys generated by the observer while the number of tree nodes were 63.

Table I. Processing Time for The Combination Of Various Lengths Of Keys

Length of Keys generated by bank and wallet	Length of keys generated by Observer	Processing Time (msec)
310	8	165.882
310	10	254.156
310	22	478.521

VII. SECURITY

Inherent security mechanism of smartcard prevents coin tempering inside it. Specially, Java has a clear separation between the card operating system (COS) and applications running on it to ensure distributed and secured execution environment [25]. Moreover, adoption of Schnorr identification scheme [26] guaranties efficiency of the secret keys.

Smartcard and the user's personal computer compose the user's electronic wallet, and they participate in transactions together. Only keys and calculation parameters are exchanges via open network; no coins are transmitted out of the card or even allowed to get in from the outer world. When user spends any coins, he needs to activate smartcard by any means. Smartcard erases the coin trace from its coins' tree by itself. Any illegal attempts can be traced instantly by the observer during spending coins. On the other hand, smartcard cannot be activated without active command issued from the wallet. Thus, this protocol not only prevents double-spending but also ensures that the card is useless if stolen or theft.

VIII. CONCLUSION

Most electronic cash (e-cash) based payment systems that have been proposed do not possess the property of both divisibility and transferability. Transferability in an e-cash based system means that when a payee receives an electronic coin in a transaction, he/she may spend it without depositing the coin and getting a new coin issued from a bank. In this paper, a single-term e-cash system where coins can be divided into smaller tokens and can be transferred over multiple hands, spread over various transactions similar to physical cash has been presented. It has also the prior resistance of double spending. At withdrawal time, the user must construct the tree 'bottom-up'; hence, the divisibility precision is determined and set at withdrawal time and, most importantly, the user's computation is on the order of the divisibility precision, O(N). Similarly, at payment time the user must either reconstruct the nodes to be spent from the scratch or store them in memory; hence, either the computation or the storage requirements are in O(N). Additionally, the user must 'link' each node to the root of the tree at payment. Hence, the communication per spent node is O(log(N)). This paper introduces a divisible transferable e-cash scheme which can be practically implementable. A prototype of the divisible e-cash protocol is implemented and tested in small scale with CORBA so that it can be easily transferred to mobile devices. The security of the coin transfer is double checked i) by the observer (which indicates safe off-line transaction) and ii) by checking double spender at any stage of transaction(which is performed by the bank while the coin is deposited). The design of the divisible e-cash is basically done for mobile devices (Jahan, Sarker and Rahman, 2009) which has more computational capabilities and able to form a small scale nework with bluetooth technology. The proposed e-cash protocol successfully handles both on-line and off-line transaction. The security of all transactions are defended which indicates the use of the e-cash protocol is ready for mass scale. Moreover, the proposed scheme can attach expiration date to coins so that the banking system can manage its databases more efficiently. We can extend our idea with currency conversion technique to cope with the different currencies of the world and the use of multiple banks. To transfer an amount that is specialized in a different currency than the currency maintained by the tamper-resistance device, the following technique can be used. When the tamper resistant device receives e-cash it multiplies the amount by an appropriate conversion rate-a field, specification, can be reserved in the tamper-resistant device that indicates the two currencies involved in the payment, and the conversion rate used for these two currencies. This field can be filled in at payment time by the parties involved in the payment. This specification will also contain the date and time. So, the bank can verify at deposit whether a correct conversion rate has been applied to the two currencies specified in specification. If not, it rejects the payment transcript. Different Internet banks can co-exist. Each bank can issue its own electronic cash tokens, by using its own signature scheme. One way to embody this is to let each bank use its own secret key x. When only a limited number of banks is participating, the public keys of each bank can be stored locally at the service providers. If there are a great many banks, the public-key certificate technique can be used to enable service providers to verify the validity of electronic cash tokens of banks that are not known to them. Hereto, a master-organization must issue a public-key certificate on the public key of each Internet bank, which can then be transferred along with the other payment data to the service provider. To settle between multiple banks, a clearing center must be used.

REFERENCES

[1] Jiangxiao Zhang, Zhoujun Li, Hua Guo, Chang Xu, “Efficient Divisible E-Cash in the Standard Model”, Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, pp. 2123-2128, August 2013.

[2] Cong Wang, Hongxiang Sun, Hua Zhang, Zhengping Jin, “ An Improved Off-Line Electronic Cash Scheme”, Computational and Information Sciences (ICCIS), 2013 Fifth International Conference, pp. 438-441, June 2013.

[3] Bin Lian, “A provably secure and practical fair E-cash scheme”, IEEE International Conference on Information Theory and Information Security (ICITIS), pp. 251-255, December 2010.

[4] Kamlesh Tiwari, “Transferable E-Cash Without Observer”, A Thesis for Master of Technology, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, July 2009.

[5] L. Wen-yuan, L. Yong-an, S. Ya-li, W. Bao-wen and L. Feng, “Offline divisible e-cash scheme based on smart card”, in Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 799-804, IEEE, 2007.

[6] I. Jahan and M. Z. Rahman, “A realistic divisible transferable electronic cash for general use”, Journal of Discrete Mathematical Science and Cryptography, vol. 10 (2007), No. 1, pp. 125-150, 2007.

[7] C. Lam and J. Liu, “Mobile agent clone detection system using general transferable e-cash”, in International Conference on Information Security, 2002.

[8] D. Chaum and T. Pederson, “Wallet database with observer”, pp. 89-105, Spriger-Verlag, Berlin Heidel-berg, 1993.

[9] N. Ferguson, “Single term offline coins”, in Advances in Cryptology: Eurocrypt '93, Proceedings, Lecture Notes in Computer Science no. 765, pp. 318-328, Springer-Verlag, 1993.

[10] R. Cramer and T. Pedersen, “Improved privacy in wallets with observers”, in Advances in Cryptology: Eurocrypt '93, Proceedings, Lecture Notes in Computer Science no. 765, pp. 329-343, Springer-Verlag, 1993.

[11] S. Brands, “Untraceable off-line cash in wallets with observers”, in Advances in Cryptology: Pre-Proceedings of Crypto '93, 1993.

[12] X. Hou and C. Tan, “Fair traceable off-line electronic cash in wallets with observers”, in Proceedings of the 6th International Conference on Advanced Communication Technology, pp. 595-599, 2004.

[13] I. Jahan, Efficient Electronic Cash for General Use, PhD Thesis, Jahangirnagar University, Savar, Dhaka, Bangladesh, 2009.

[14] Basic Card, “Basic Card Products”, available at http://www.ZeitControl.de, last visited on September 2008.

[15] MUSCLE, Movement for the use of smart card in Linux environment, “PC/SC implementation for Linux”, available at http://www.linuxnet.com, last visited on May 2008.

[16] Sun Microsystems Inc., Java Card 2.2.2, Application Programming Interface specification, 2005.

[17] D. Chaum, “Blind signature for untraceable payments”, in Proceedings of Advances in Cryptology - Crypto '83, pp. 199-203, 1983.

[18] D. Chaum, A. Fiat, and M. Naor, “Untraceable electronic cash”, in Proceedings. Crypto '88 - Advances in Cryptology, Santa Barbara, California, Lecture Notes in Computer Science, vol. 403, pp. 319-327, Springer, Berlin, 1990.

[19] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public key cryptosystems”, ACM, vol. 21, pp. 120-126, 1978.

[20] T. Okamoto and K. Ohta, “Universal Electronic Cash”, in Advances in Cryptology- Crypto ‘91, pp. 321-337, 1991.

[21] Y. Tiannis, Efficient Electronic Cash: New Notations and Techniques. PhD Thesis, North-eastern University Boston, Massachusetts, 1997.

[22] D. Chaum and T. Pederson, “Wallet databases with observers”, in E. Brickell editor, Advances in Cryp-tology - Crypto '92, Proceedings, Lecture Notes in Computer Science, pp. 89-105, Springer-Verlag, New York, 1993.

[23] ACS, Advanced Card Systems Ltd, available at http://www.acs.com.hk, last visited on May 2008.

[24] Gemalto, Home Page, available at http://gemalto.com, last visited on April 2009.

[25] Z. Chen, Java Card Technology for Smart Cards, Addison-wesley, 2000.

[26] C. P. Schnorr, “Efficient signature generation by smart cards”, Journal of Cryptography, vol. 4, pp. 161-174, 1991.

[27] Gemalto NV, .NET Card, available at http://www.gemalto.com/products/dotnet_card/, last visited on June 2009.

[28] I.Jahan, K.M. Sarker and M.Z. Rahman “ Ecash observer implementation in smartcards”. 12th International Conference on Computers and Information Technology, pp. 626-631. 2009.