

A Software Approach to Anonymity In Electronic Transactions

A Report for CS 651: Security in Information Systems

Hichem Boudali, ⁺Sui Kwan Chan, Ganesh Pai
Departments of Electrical and Computer Engineering and ⁺Computer Science
University of Virginia, Charlottesville, 22903

{hb4m, skc2s, gjp5j}@virginia.edu

Abstract

Some of the main goals in security of electronic transactions are guaranteeing confidentiality, authentication, non-repudiation, access control and integrity. However, these security protocols do not always guarantee anonymity of clients. Most businesses use privacy policies to protect their client's privacy. Software programs that prevent *browser chatter*, and cookies from revealing personal information such as credit-card numbers, names and addresses do exist, but there are no protocols (augmented with security protocols) that are in place, to provide client anonymity despite electronic profiling. In this report, we present a protocol, deployed at an application level, to provide anonymity for clients in electronic transactions. We have identified four main entities in an electronic transaction: The client, the vendor, the client's financial organization and the delivering organization. Often, clients do not want the vendors in the transaction to know of the client's identity (name, address, credit card number, etc). The client also does not want the delivering organization in the transaction to know of the product that the client is purchasing from the vendor. We propose to distribute this information between the different entities involved in the transaction, so that the client's privacy is not compromised, despite profiling. We demonstrate, using a simple client-server implementation, that our protocol provides anonymity as part of security. Finally, we also explain how our protocol may also be employed on other platforms, like smart cards.

1 Introduction

Electronic transactions today are subject to *dataveillance* or the surveillance of data [UGW99] where data transmitted from clients during the transactions (names, addresses, credit-card numbers, etc) can be logged and used to profile the client. The concern for anonymity in electronic transactions is a big one, but is often not given the same importance as the security of the transaction is

The main goals of security protocols are confidentiality, authentication, non-repudiation, access control and integrity. However, these security protocols do not always guarantee anonymity of clients. In electronic commerce, privacy of clients usually boils down to a privacy policy that appears on the website of the vendor. The chances that visitors to this site even read the privacy policy are very small, and besides, potential clients really have no reason to believe that the vendor is a trusted individual or organization and that they will not be profiled or that the information gathered from profiling will not be shared with another individual or organization. In fact, once a client reveals certain information, he/she no longer has any control over how this information will be used. This is potentially dangerous, and therefore, the individual must have the ability to either actively participate in how personal information is used or to prevent untrusted parties from even having this information. Therefore, our stand is that anonymity should be inherent in an electronic transaction and that clients should have the right to determine what data should be collected on them and how it is disseminated.

Software programs that prevent *browser chatter*, and cookies from revealing personal information such as credit-card numbers, names and addresses do exist, but there are no protocols (augmented with security protocols) that are in place, to provide client anonymity despite electronic profiling. In this report, we present a protocol, deployed at an application level, to provide anonymity for clients in electronic transactions. We have identified four main entities in an electronic transaction: The client, the vendor, the client's financial organization and the delivering organization. Our protocol distributes information that a client will reveal, between these entities.

The report is organized as follows: Section 2 describes related work. We describe our protocol for anonymity in section 3 along with the format for the different components and we list our assumptions. Section 4 deals with our implementation of the protocol in simple client-server architectures. We talk about some of the issues that our protocol raises in section 5 and then we present a possible implementation scenario of the protocol on a multi-application smart card platform in section 6. We conclude and provide some avenue for future work in section 7.

2 Related Work

Different types of electronic payment systems exist, but not all provide anonymity for the consumers. This section gives a brief review of some schemes of electronic payments that support anonymity.

2.1 CyberCash

CyberCash offers a real-time and secured credit card authentication service over the Internet based on digital signatures. Anonymity is provided, as buyers do not have to set up special accounts with a bank or with CyberCash. All transactions are encrypted. The system operates on top of any general security system such as SSL or S-HTTP.

CyberCash claims that the customer remains anonymous to the merchant except for the IP address. However, in CyberCash specification, it is said that the merchant is shipping the product to the consumer. The question remains on how the consumer i.e. its name and address remains anonymous to the merchant since the latter ships the product.

2.2 DigiCash

DigiCash uses digital money for security and privacy. With electronic cash based on the “digital signatures” and “blind signatures”, it is impossible to link the payment to the payer, thus

providing anonymity. The electronic cash is digitally signed by the issuer and thus are more resistant to forgery than hand-written ones. DigiCash is supposed to provide authentication and integrity (ECash is signed), privacy and anonymity of the consumer with respect to the bank. However, DigiCash do not guarantee customer privacy because merchants can keep track of where they send their products. In summary, both protocols provide privacy of account information of the customers, but they do not prevent the merchants from knowing the identity of the buyers.

3 A Protocol for Anonymity

In this section we describe our protocol for anonymity. We introduce the concept of an electronically signed check, which is used as legal tender for the electronic transaction. The electronic check may be used as a credit card or the electronic equivalent of an actual check, depending on the service offered to the client, by his or her financial organization.

Briefly, the protocol is end-to-end and mainly provides integrity, anonymity, non-repudiation and authentication. A secure channel can also be used to provide confidentiality. Fig. 1 shows the protocol graphically.

3.1 The Protocol

The steps in the protocol may be explained as:

Step 1

The client browses the vendor's electronic site and may choose a product to purchase. The client may employ a *cookie-filter* so that it can authorize use of cookies, if desired.

Step 2

Client sends a signed electronic-check (described in Section 3.2) to the vendor, for the desired amount required to purchase the product. The signature of the client on the electronic – check guarantees¹ integrity and non-repudiation from the client’s side.

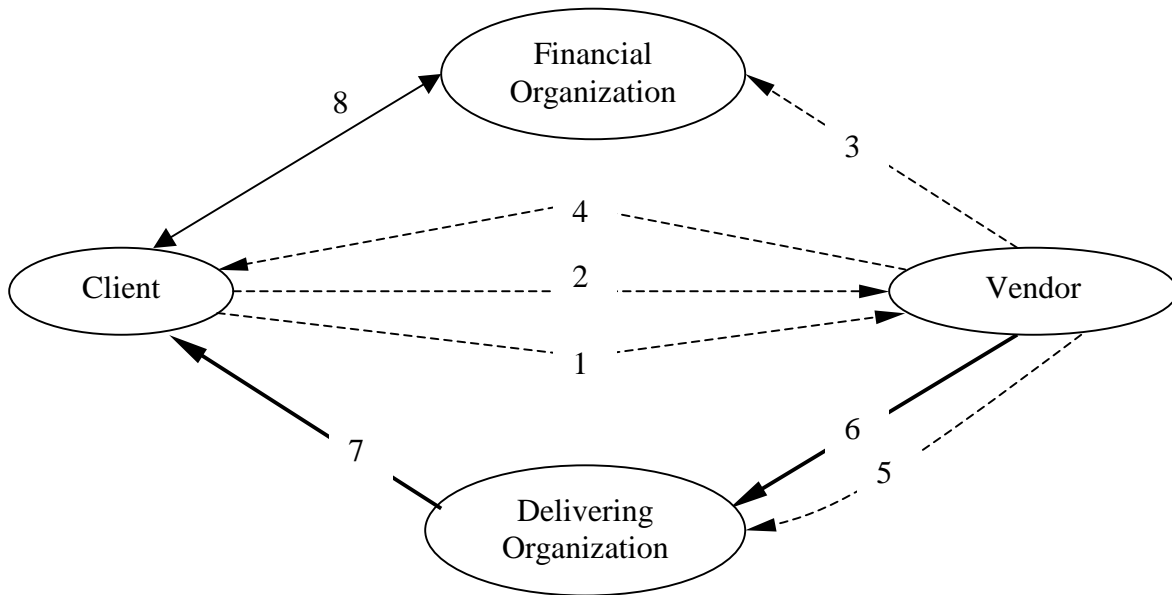


Fig. 1: Protocol for anonymous electronic transaction.

Step 3

The vendor sends the client’s check to the client’s financial organization / bank, along with the amount of the transaction (that the vendor has to get). If the bank certifies the client’s check, then the transaction proceeds, or the client is informed that the transaction has been denied.

Step 4

The vendor sends, to the client, a signed electronic receipt comprising the vendor’s ID, the product ID, the product price and the transaction number. The signature of the vendor ensures integrity of the electronic receipt and non-repudiation from the vendor’s side. If

¹ If we rely on the strength of public-key algorithms and one-way hash functions, then a one way hash of the message, encrypted with the client’s private key will ensure that the message originated only from that client and no one else i.e. non-repudiation. The one-way message-digest or hash guarantees that the message has not been tampered with.

this step does not take place, the client can block the money transfer from his account to the vendor's account. The transaction number identifies a unique transaction. The client can also block the money transfer and notify the vendor if the receipt does not conform to the client's choice.

Step 5

The vendor electronically transmits to the delivering organization (selected by the client) an electronic delivery order containing the receiver's encrypted ID and the transaction number.

Step 6

Physical shipping of the product, with the transaction number only, on the package, to the delivering organization takes place.

Step 7

The delivering organization matches the transaction number it received electronically with the transaction number received on the package and delivers it to the receiver.

Step 8

The client communicates with the bank, for instance once a month, to receive an *electronic checkbook* consisting of electronic check numbers (and perhaps for other bank transactions).

Hashing the data contained in the document and then encrypting the hash code with the sender's private key achieves a digital signature of the document. This provides integrity and non-repudiation from the sender side. Once the signed document is received, the data is hashed again and the hash code is compared with the decrypted (with the sender's public key) hash code sent along with the document (done by the bank). Therefore, the client signs the E-Check when it is sent to the vendor, and the vendor signs the E-Receipt when it is sent to the client.

This protocol could be used in conjunction with other security measures. For example, *cookie filters* can be used for managing cookies and allowing only certain cookies to be written or retrieved from a machine. Another commercially available software is *the Anonymizer* for providing anonymous services including anonymous email, and web browsing. Our protocol may also be used over the secure sockets layer (SSL) for establishing a secure channel *and*

having anonymity at the same time. In the ensuing paragraphs, we describe the details of this framework to obtain an application level implementation of the protocol. Before we describe the components used in the protocol, we briefly digress from our discussion to introduce the notation that we will use in the rest of the report.

$K_{PU-A}(X)$ implies that message X is encrypted with A 's public key.

$K_{PR-A}(X)$ implies that message X is encrypted with A 's private key.

$H(X)$ is a one-way hash of message X yielding its fixed length message digest.

C is the Client.

V is the Vendor

FO is the financial organization.

DO is the delivering organization.

ID_C is the identity of the client.

ID_R is the identity (name and address) of the receiving entity. (This may or may not be the same as the client's name and address)

3.1 Format of the Electronic Check

The client issues a signed electronic check (Fig.2), to the vendor. The check contains the client's identity encrypted with both the financial organization's public key [$K_{PU-FO}(ID_C)$] and the delivering organization's public key [$K_{PU-DO}(ID_R)$].

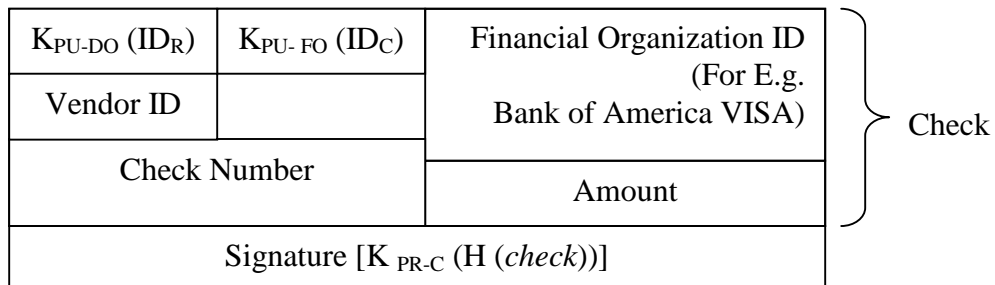


Fig. 2: Signed Electronic-Check.

The E-Check contains the financial organization's identity in clear, which enables the vendor to contact them to verify that the client can really pay for the product. It also contains the vendor's identity, which enables the FO to check if the vendor ID mentioned on the E-Check is the same as the ID of the vendor actually sending the E-Check to the FO (we assume that the FO can in some ways authenticate the vendor communicating with it).

The amount of the transaction is sent in plain text, although this can be encrypted with financial organization's public key [K_{PU-FO} (Amount)] as well.

The bank compares the amount authorized by the client, with the amount demanded by the vendor. The electronic check also has a unique check number (provided by the financial organization to the client). The check number issued by the financial organization avoids the re-use of the check by the vendor or the client. The client signs the electronic check, and the signature is appended to the electronic check. As mentioned before, this ensures integrity of the check and provides non-repudiation from the client side.

We see here that since all the fields that could possibly identify the client are encrypted, the vendor may not know who the client is. At the same time, if we assume that the delivering organization is ethical, it may not know what the client's purchase comprises and it only knows who the client is and where the client lives. Only the financial organization has knowledge of the client's purchase but it too has no knowledge of what the client purchased. It simply authorizes the transaction based on the electronic check it receives. Clearly, the client's anonymity is preserved if the information that the client supplies is distributed among the entities involved in the transaction. Of course, if all this information were to be somehow collated, then the client can be identified.

One of the advantages of the E-Check is that the client fixes the amount of the transaction, unlike in a credit-card transaction, where an un-trusted vendor could easily charge more on the card than the actual transaction amount.

3.2 The Electronic Receipt and the Electronic Delivery Order

The vendor sends a signed electronic receipt (E-Receipt) back to the client, when the E-Check is approved by the client’s financial organization. The E-Receipt (Fig. 3) contains a transaction number that uniquely identifies the electronic purchase. The vendor decides what format should be used for the transaction number. It also contains the vendor ID to identify the vendor, and a DO ID to identify the delivering organization. Some additional fields are a date to identify when the transaction took place and the amount of the transaction. The E-Receipt is signed by the vendor i.e. the E-Receipt is hashed and the hash is encrypted with the vendor’s private key. This provides authenticity and non-repudiation from the vendor side.

After sending the E-Receipt to the client, the vendor also sends an electronic delivery order to the delivering organization. This contains the identity (often the name and address) of the receiver encrypted with the delivering organization’s public-key and the transaction number that was assigned to that transaction. These are steps 4 and 5 in the protocol.

$K_{PU-DO} (ID_R)$	DO - ID	Financial Organization ID (For E.g. Bank of America VISA)	} Receipt
Vendor ID	Date		
Transaction Number		Amount of Transaction	
Signature [$K_{PR-V} (H (receipt))$]			

Fig. 3: Signed Electronic Receipt

The vendor also physically gives the package to the delivering organization, along with the transaction number on top of the package. The delivering organization must compare the transaction number on the package with the transaction number that it receives on the E-Delivery Order (Fig. 4).

$K_{PU-DO} (ID_R)$	DO - ID
Vendor ID	Date
Transaction Number	

Fig. 4: Electronic Delivery Order

Since the identity of the receiver is also sent in it, the delivering organization can decrypt the contents of this field with its private key and obtain a shipping address for the package. The E-Delivery Order can also contain fields indicating the date of the transaction and the date that it was given to the delivering organization. Presumably, the vendor would use this information to audit its transactions.

3.4 Some Assumptions

An important point to note about this protocol is its implicit assumption that the vendor and the delivering organization are two separate entities. We also assume that the delivering organization follows the privacy policies it sets down and does not divulge information about its clientele to the vendor.

We also assume that the delivering organization itself does not tamper with the products it delivers to its clients, in that it does not open the packages to monitor the client's purchases and then build a profile of the client. Some other assumptions in this protocol are:

- The client has an account with the financial organization and may have an account with the delivering organization as well. The financial organization has a database of the client's identity corresponding to his/her account number and also has a record of the client's public key.
- The financial organization of the client and the vendor don't have to be the same. However, for the sake of simplicity, we only consider one financial organization (FO). We further assume a unique delivering organization (DO). For future work, one might consider the existence of multiple financial and delivering organizations.
- Communications between the vendor, the FO and the DO can all be authenticated, i.e. they can use passwords or shared secrets or a PKI.

- We assume the existence of a Public Key Infrastructure (PKI), an example would be a PKI provided by VeriSign.

We have now described a framework in which the client's anonymity is preserved, by distributing the sensitive information that the client reveals in the course of the transaction. The protocol is completely transparent to the user. Later in this report, we show that the client may still be profiled; it is not of much consequence when the profile cannot be linked back to the client's actual identity. In the next section, we describe our implementation of the protocol.

4 Implementation Details

Our current implementation is in Java, and is based on simple *client-server* architecture, using *socket* communications. The implementation follows the protocol exactly, but few simplifications related to the cryptographic system have been introduced. There are four main classes *Client*, *Vendor*, *FO* and *DO* each corresponding to the four entities identified in an electronic transaction (Fig. 1)

The protocol implementation is abstracted into a *Handler* class. Each entity has its own handler, and thus we have *ClientHandler*, *VendorHandler*, *FOHandler*, and *DOHandler* as the four protocol handlers. We have created classes for the E-Check, the E-Receipt and the E-Delivery Order, namely *ECheck*, *EReceipt*, and *EdeliveryOrder*. Instances of these classes are created as serializable objects and then sent over the network. The simple client-server implementation is realized using *tcpServer* and *tcpClient* classes, that abstract the type of communication used. These classes provide generic functions such as connecting to a server, sending an object, receiving an object, opening and closing socket connections. Finally we also use a class called *Cipher* that provides basic cryptographic functions.

Originally, we were to use public key cryptography (the popular RSA algorithm for encryption) to encrypt the client and receiver ID and to sign checks, receipts and delivery orders. However, the *Java Cryptography Extension* (JCE) bundled with Sun's Java2 SDK does not provide any

asymmetric key encryption methods (Other than a method for generating digital signatures with DSA and SHA1). Therefore we restored to using a symmetric key cryptosystem. We demonstrate that so long as the shared secret remains secret and shared only between the two parties, the protocol still succeeds in providing anonymity. There are other commercially available security extensions from security providers such as Cryptix. (<http://www.cryptix.org>) but we did not use these extensions due to time constraints. We have used a simple substitution algorithm for encryption and decryption, and the *Cipher* class provides this functionality. In fact, the two first fields of the check are encrypted with two different secret keys. One shared between the receiver (ID_R) of the product and the delivering organization and the second one is shared between the client (ID_C) and the financial organization.

The program basically simulates a typical transaction. We focus on showing how a client can order a product without disclosing its identity to the vendor. A client connects to a vendor's server and orders a product and by the end of the transaction, the vendor ends up having a check issued by the client, but has no knowledge of the identity of either the client or the receiver. The delivering organization ends up with the decrypted receiver ID. For ease in implementation, we assume that the DO maintains a database from where an address for the receiver can be obtained, when the receiver ID is known. The DO also ends up with a product ID that does not tell what the product is. We have considered only one client and therefore the delivering organization knows which key to use to decrypt the receiver's ID.

The client also functions as the receiver of the product. This will not work if we have multiple receivers, because the delivering organization will not know which key to use to decrypt the receiver ID. However, this is the case only because we use conventional encryption instead of public key encryption. In fact, by using public key encryption, the receiver's ID would be encrypted with the delivering organization's public key (and only the delivering organization would be able to decrypt it). The server handles one client at a time. Handling multiple clients can be easily extended by the use of threads, inherently supported in Java.

A note about asymmetric encryption, is that it is generally a slow and resource consuming process. However in our case, a small amount of data is encrypted and therefore asymmetric

encryption would not produce much overhead. In this proof-of-concept implementation, minor exception handling is performed. Future work could easily include robust servers and clients that support rigorous exception handling.

5 Issues with the Protocol

Although our protocol provides anonymity, the client can still be profiled based on the encrypted client ID (ID_C). However, this kind of profiling is of little consequence because the real identity of the client cannot be known. We can eliminate this kind of profiling if ID_C is encrypted with a frequently changing key. But this adds more overheads for FO since it has to change its public-private key pair regularly.

By having the E-Receipt, the client can always demonstrate the non-delivery of his products (but the payment for the same to the vendor), if a situation where the product has not been shipped arises. But once the vendor has sent the product to the DO i.e. step 6, it is the DO's responsibility to deliver the product. Even though a client has received the E-Receipt and the product, he can still deny having received the product. In this case, the DO can easily interfere and assert that a package with a transaction number X has been sent to person Y at address W.

The whole protocol involves some changes on the financial organization and delivering organization sides. Both organizations have to include a new service for handling this kind of a transaction. The financial organization has to issue E-Check numbers to its customers on a regular basis, it also has to be able to communicate with different vendors from whom it receives the E-checks and include a function that verifies the authenticity of those E-checks. The delivering organization has to do one more operation in its delivering process. Indeed, since only a transaction number comes along with the product i.e. no receiver name and address, it has to match the transaction number of the physically received product with the transaction number appearing on the E-Delivery Order.

Another issue is scalability. The protocol relies on a PKI; such an infrastructure exists and has proved satisfactory in its performance. However, in this protocol, a financial organization has to deal with a large number of customers and vendors in order to authenticate E-checks. The authentication has two parts: first the FO has to authenticate the identity of the client and the signature provided in the E-check and second, it has to verify that the client's account contains the amount of money that appears on the check. This implies a burden on the financial organization's server and one has to evaluate if this is reasonable to be implemented in the existing financial organizations' servers.

6 Practical Implementation on A Smart Card

This section investigates the use of our protocol on a different and an increasingly popular smart card platform. Multi-application smart cards such as the JavaCard [UH00] permit multiple applications to reside on the same card. Before we discuss implementation of the protocol on a multi-application smart card, we briefly introduce these smart cards and specifically talk about Java Card.

6.1 Smart Cards and the Java Card

Smart cards are credit-card sized cards that usually have either a memory chip or a microprocessor chip embedded within them. The cards have a metallic contact on the surface that enables data to be either stored into or retrieved from the card. More sophisticated cards also offer contact-less data retrieval and storage capabilities. Current smart cards generally have an embedded 16-bit or 32-bit microprocessor, along with some mutable and non-mutable storage. One may think of it as an embedded computing device with IO, controlled by an operating system. Smart cards may also have an optional cryptographic co-processor.

Smart cards and the Internet have co-operated to facilitate electronic banking and payment, and secure business-to-business (B2B) and business-to-consumer (B2C) e-commerce. Smart cards in conjunction with the GSM standard for mobile phones enable secure subscriber authentication, roaming facilities for mobile phones and secure value added mobile services. Frequently,

multiple applications can be stored on the card, enabling partnering of on card programs and providing added convenience to the cardholder. In 1997, the *Open Card Framework* (OCF) was introduced as a means to provide stable interfaces and function separation between the card terminal vendor dependent parts, the card operating system dependent parts and the card issuer dependent parts. The OCF provides a means by which the application developers can deploy their applications and services without worrying about the underlying dependent components of the card.

JavaCard is a stripped down version of the Java platform that is used for deploying architecture independent applications on smart cards. Applications on the JavaCard have application IDs, and therefore, can authenticate themselves to other applications residing on the same card. The Java Card Runtime Environment (JCRE) permits multiple applications to reside on the same card. The JCRE implements an *applet firewall* to enforce security between the applications so that another application may not access sensitive data from one application, unless it has the required permissions. A more detailed description of the enforcement of security between the applications is provided in [JAV00, KS98, PG00, ZC00 and MB99]

6.2 Implementation of the Anonymity Protocol with JavaCard

One of the main differences with the JavaCard implementation of the anonymity protocol is the use of a crypto-coprocessor, to implement the security algorithms. The E-Check, E-Receipt and E-Delivery Order are generated as before, but the encryption of the fields would be with the use of a hardware encryption scheme instead of implementing the encryption in software. The JavaCard security enforcement ensures that data object belonging to one application may not be accessed by another application. The JavaCard security package provides key generation capabilities and public-private key pairs can be stored in its electrically mutable memories. The FO would have to provide its public key on the card that it issues to the client. This key is used to encrypt the client ID when generating the E-Check. The applications on the card employ the anonymity protocol and use the E-Check objects to perform a transaction on the card.

For example, let's assume that the vendor is an airline application. The FO is the client's bank, which has issued the card to the client. Therefore, the card has the FO's public-key on the card. The delivering organization application also resides on the card. The main functionality of a multi-application smart card is that it permits the client to have associations with multiple application service providers, and have all these applications reside on the same card. We also assume that the client has an account with delivering organization, and therefore, the DO application has knowledge of the DO's public key.

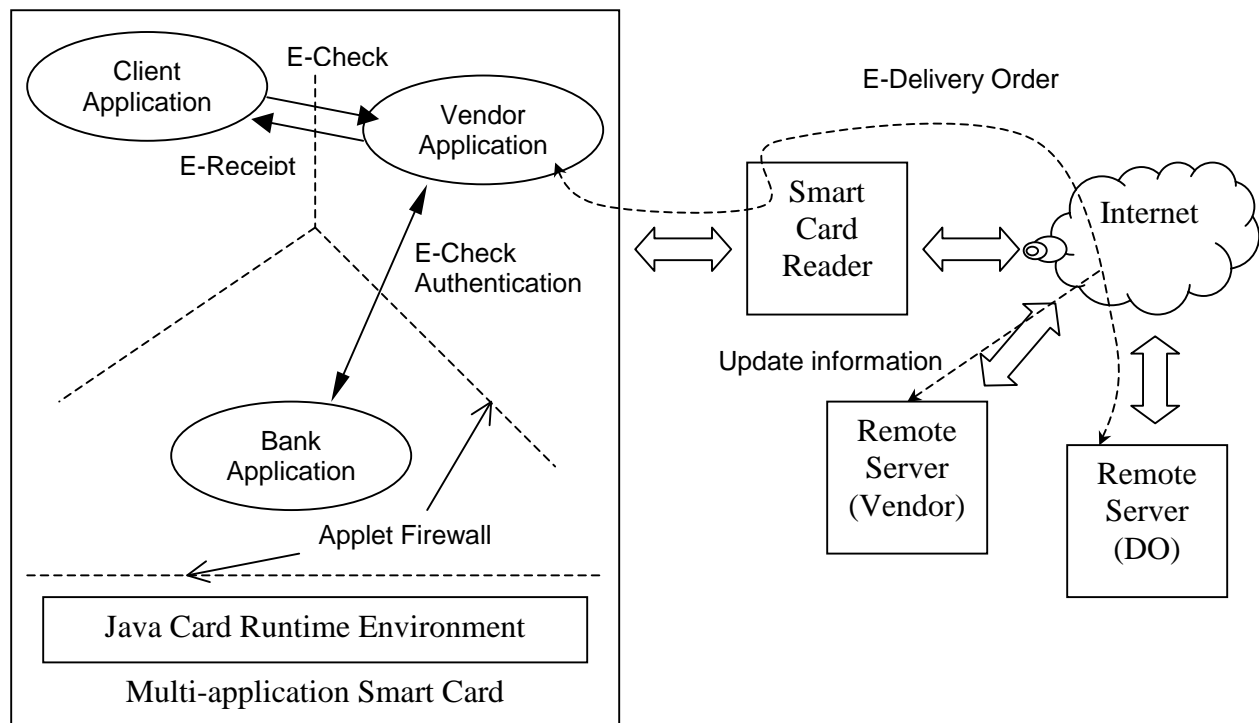


Fig. 5: A Possible Implementation Scenario on a Smart Card platform

When the client wishes to purchase an airline ticket, the client inserts the card into the smart card reader and either is physically purchasing the ticket, or is purchasing the ticket online. In either case, instead of the vendor contacting the bank separately, the applications on the card perform these steps of the protocol (Fig. 5).

The applications update the accounts of the client so that it reflects that the client purchased a ticket, and the bank application on the card reflects a corresponding deduction for the fare, in the

client's account. Note that here; the airline application need not have any knowledge of who the client is or where the client lives. Such a card could have been issued directly by the bank with both the airline and bank application residing on the card at the time of issue. Once the airline application has determined that the client can purchase the ticket, it simply uses the bank application to obtain the desired amount and communicates this amount with the remote application online. At the same time, the airline application also generates an E-Delivery Order and communicates this to the remote machine where the DO application resides. The DO has now to ensure that the ticket is delivered to the client.

In all of this, the airline application only has knowledge of a purchase of a ticket. It has no knowledge of the identity of the purchaser. It has also sent to the DO, a transaction number electronically and physically given the ticket to the DO. The DO now, only has knowledge of who the client is and not of the contents of the package. The bank application only has knowledge of the fact that a certain amount of money was deducted from the client's account by an application that resided on the same card, and authenticated itself to the bank application. We see that the information of the client has been distributed among the entities involved in the transaction, so that the client's anonymity has been preserved.

7 Conclusions and Future Work

In this report, we have presented a generic protocol and a framework in which client anonymity is preserved in an electronic transaction. We have demonstrated how the protocol operates on simple client-server architecture and we have also described a potential scenario for using this protocol on the smart card platform.

In our implementation, we have used a symmetric key cryptosystem. An asymmetric-key cryptosystem may be used in future implementations, since the protocol has been designed with a public-key infrastructure in mind. A future development that is envisioned is the integration of the protocol into web-browsers so that a protocol for anonymity is inherently supported. Future work would also involve implementation of robust clients and servers so that exception handling

is strictly supported. Implementation of the protocol on a wireless application protocol platform can also be investigated. We presented an informal proof-of-concept in this report, and therefore, a more formal proof that the protocol is valid is another avenue of future work.

The multi-application smart card is constrained by space and power and therefore, current smart cards may not be able to hold more than one or two vendor applications besides the bank and client applications. We propose that a generic vendor application interface can be created as a standard that all vendors implementing our protocol can use. The idea behind this is to have one trusted generic vendor application that resides on the smart card. All vendors dealing with the client and his/her financial organization will have to do so, through this generic vendor application interface. This could be yet another area for future research.

References

- [KS98] K. Schier, "Multifunctional Smartcards for Electronic Commerce - Application of the Role and Task Based Security model", *In Proc. of 14th Annual Computer Security Applications Conference*, Dec. 1998
- [MB99] M. Baentsch, et al., "JavaCard - From Hype to Reality", pp. 36 - 42 *IEEE Concurrency Vol. 7, No. 4*, October-December 1999.
- [PG00] P. Karger, G. Schellhorn, et al., "Verification of a Formal Security Model for Multiapplicative Smart Cards", *Research Report RC 21809*, IBM Zurich Research Laboratory, July 2000
- [UGW99] Uwe G. Wilhelm, "A Technical Approach to Privacy based on Mobile Agents, protected by Tamper Resistant Hardware", PhD Thesis, École Polytechnique Fédérale de Lausanne, 1999.
- [UH00] Uwe Hansmann et al., "Smart Card Application Development Using Java", Springer-Verlag, 2000.
- [ZC00] Zhiqun Chen, "Java Card TM Technology for Smart Cards: Architecture and Programmer's Guide", *Addison-Wesley*, 1st Ed. June 2000.