A Simple Secure M-Commerce Protocol SSMCP*

Elias Haddad and Brian King,

Indiana University Purdue University Indianapolis,
Indianapolis, IN, USA

Summary
The trend in technology is such that small lightweight low-complexity devices, like PDAs, cell-phones, . . .,will become one of the more predominant computing platforms. Consequently, these devices will play an increasing role in e-commerce and m-commerce transactions. In the past, the role of the merchant has been played by workstations/servers, but due to advancements in wireless communications, like Bluetooth, we should expect more peer-to-peer e-commerce/m-commerce transactions and hence we should expect more lightweight devices playing the role of merchants. Here we develop secure efficient m-commerce protocol SSMCP*. This is a low-complexity protocol that is both efficient for mobile lightweight customers as well as mobile lightweight merchants. SSMCP* is both efficient in terms of bandwidth, eliminating any redundant information, as well as efficient in terms of rounds of communication. Further, SSMCP* addresses the important issue of a fair commitment to the transaction between the merchant and the customer.

Key words:
m-commerce, e-commerce, security for mobile devices

Introduction

As computing becomes more pervasive, we will demand to perform many of our customary computing tasks where we are and with what computing devices that are available. One customary task will be e-commerce transactions. A problem that could arise is that the most commonly available computing devices are personal devices like PDAs, cell-phones, and such. Typically, these devices do not have the same resources that are available in the typical PC workstations that today dominate the today's e-commerce transactions. Consequently, as we envision tomorrows e-commerce transactions, we must be cognizant of this. Further, the typical merchant in today's ecommerce transactions is some server (workstation). As wireless mediums like Bluetooth and IEEE802.11 become more popular, peer-to-peer e-commerce/m-commerce will become more common and so the role of merchant could very well be a limited resource device. For example, lightweight merchant KIOSKs may be placed in a "commons area", a well-trafficked area perhaps some space in a mall or community square, where it has no persistent internet connection. In anticipation of this trend we need to develop a lightweight low-complexity ecommerce/m-commerce protocol that is secure for both the merchant (server) and the customer (client). An m-commerce protocol is needed that joins efficiency, fairness and security all at the same time. Efficiency can be defined as low bandwidth for the communication and less computation for the devices that are memory and power limited.

In this paper we will introduce a secure protocol that can be used for any m-commerce transaction. We will use TLS/WTLS [1], [10] in lower layers to reduce the number of required signature generations and within the protocol. Moreover, this protocol has been developed to reduce the amount of communication, in order to comply with wireless bandwidth limitations. One integral tool utilized within this protocol is the use of e-cheque [2] which is a token that commits the signer of the e-cheque to pay a designated amount of money to the recipient. Each time a customer wants to make a payment, they send a request to the bank to credit their account with the amount of money to create the e-cheque to the merchant. The bank will check for fund availability before withdrawing the funds from the customer's account. This e-cheque has a unique serial number or e-cheque identifier to prevent any replay attack and only the merchant (cheque recipient) will be able to cash it.

2 Motivation

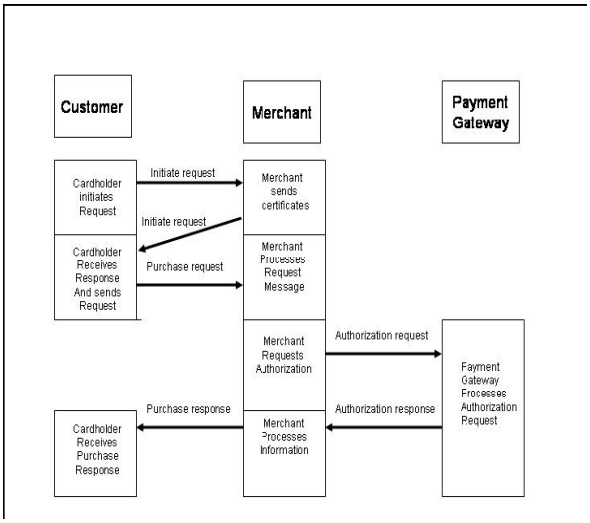
The problem concerning a fair commitment is that a financial transaction will need to be a multi-round communication protocol. What further complicates this is that the transaction is more than a 2-party protocol, because it includes financial institutions. An important aspect of any financial transaction is a fair commitment. What we mean by a fair commitment is that two parties are negotiating an agreement, the exchange starts with an interest from the customer, if a price (cost) is flexible, which is often the case, the customer will make an offer, the merchant could counter. They may say "I will sell it to you for $XXX but I need a response by the end of the day". Once an offer is made, the merchant has made a commitment.

Manuscript received March 5, 2007
Manuscript revised March 25, 2007

If the customer accepts the offer then they have committed to the purchase. At this point the customer will most likely remove themselves from the "market". Therefore if a merchant was allowed to renege on the "price" then the customer has been penalized. In a multi-round, multi-party (banks as well) protocol it is very likely that the merchant could discover that they can get a better price/offer, this is especially true in mobile-commerce transactions where there maybe some delay to the completion of the transaction. Similarly when the merchant make a price offer to a customer they too may have removed their product off the market for that brief period that the customer gets to consider the price. Thus the merchant does not want a customer to also be able to renege on a consideration-to-purchase offer. Consequently the merchant has an interest as well, for a fair commitment. An important aspect of our m-commerce protocol is that it forces the merchant to commit to their offer. A second motivation factor is our vision of the future of m-commerce transactions. In today's version of e-commerce, the client will contact the merchant which is most likely some server. The e-commerce protocol is often designed with the assumption that the merchant has a persistent internet connectivity, and a large number of resources available. But the future is such that as computing becomes more pervasive the notion of what is a typical merchant will change. It is very likely that organizations place small devices, merchant kiosks into the market place, in some cases remote sites, with no persistent internet connection. The future m-commerce protocol under this vision should not require the merchant to have an established (persistent) internet connection. This should not prohibit the transaction if the protocol is carefully constructed. We recognize that the customer will need some network connectivity (in order for it to connect to its financial institution). Another goal of an m-commerce protocol is to minimize communications between the merchant and the customer, between the customer and their bank, and between the merchant and their bank. The reasoning for this is that we are assuming that the customer (or client) is a lightweight device and transmitting over a wireless medium. Further, we are assuming that the merchant is lightweight and could be transmitting over a wireless channel.

Utilizing WTLS Wireless Transport Layer Security [10] or TLS [1] in lower layers will provide message integrity, authentication and non repudiation. Our protocol takes advantage of the certificates already exchanged by WTLS in lower layers. In the communication between the merchant and the client, we will be using WTLS class 3 that provides both server and client authentication which is the merchant and customer, respectively. WTLS will provide the protocol with integrity authentication and non-repudiation. Therefore, SSMCP* does not deal with message integrity nor authentication since it's provided by WTLS/TLS in lower layers. One popular e-commerce protocol that is used today is the SET protocol [8]. The protocol is designed for credit card services and so an integral aspect is the use of a centralized trusted party, which cannot be used in our model. An important tool used in the SET protocol is the dual signature, which also has no use in our model. There are several examples of e-commerce protocols constructed for the wireless domain, some of them include [4], [5], [9]. In the SSMCP* protocol, the names of stages are similar to those names in the SET protocol. However the information in each stage varies greatly from SET. The content of the data structures that we use is very similar to [5]. One particular security flaw of [5], [9] is that they allow the merchant to generate the transaction ID. Since this is the identification of the transaction this is a serious security problem if the merchant is not trusted (an assumption we should always make).

Figure 1 Set Protocol



Our m-commerce model will make use of a PKI (public-key infrastructure). There are several different assumptions that one can make concerning the use of a trusted party (persistent). We prefer to have an off-line trusted party, some entity that generates the public-keys, private-keys and certificates for each party once. We will not go into any details concerning the PKI generation. Later, we will discuss possible cryptographic primitives that fit well with our protocol. Of course since this is an e-commerce protocol we will be utilizing financial institutions like banks. The customer and the merchant trust the banks because they are federally regulated. Note: The various data structures that will be transmitted, for example certificate, invoice, e-cheque, etc., may have entries that had already been transmitted in a previous communication with the receiving party. Consequently in

order to save bandwidth the sending party could omit a data structure or a field of the data structure if the information was already transmitted. It is important, however, that the field is present whenever a signature is performed on the data structure.

2.1 SET

Secure Electronic Transaction (SET) [8] a standard security specification designed to protect credit card transactions on the World Wide Web and designed by Visa and MasterCard. SET provides a number of security standards for protecting all payments made over the internet and consists of three essential services: 1-Ensuring a secure communication channel for all parties involved in the transaction. 2-Providing authentication by the use of certificates. 3-Providing privacy by making information available to the right parties when and where required.

Transaction Processing: SET protocol consists of three major phases: purchase request, authorization request, and the payment authorization. We will describe each of the phases in details in the following sections (for more *complete* description of SET, see [8]). The protocol is illustrated in Figure 1.

Purchase Request Phase This phase consists of the following messages exchanged between the merchant and the cardholder: initiate request, initiate response, purchase request, and purchase response.

Initiate Request The process starts with the customer shopping and selecting the item(s) from the merchant website. After selecting a particular payment card, they send the initiate request message requesting the merchant and the payment Gateway's certificate.

Cardholder -→ Merchant:

the brand of the credit card, his ID and a nonce (N_C).

Initiate Response After receiving the initiate request, the merchant generates a unique transaction identifier ID_{trans} and a nonce N_M. The merchant then signs the concatenation of the ID_{trans}, the credit card brand CCBrand and the nonces N_C. and N_M.. They proceed by sending the signature along with the merchant and Gateway's certificates $CERT_M$ and $CERT_G$.

Merchant → Cardholder:

$SIGN(ID_{trans}, CCBrand, N_C, N_M) + CERT_M + CERT_G$

Purchase Request The cardholder verifies the merchant and Gateway's certificates by means of the certificate authority signatures. They then generate the order information OI and the payment information PI. The OI consists of the concatenation of ID_{trans}, ID_{brand}, date, N_M and N_C where ID_{brand} is the unique credit card brand identifier. The PI consists of the concatenation of ID_{trans}, Amount and ExtraStrongEncryption (CardData) where the CardData is the credit card number, Expiry and the PIN.

The cardholder computes the dual signature DS as follows:

$$DS = SIGN(hash(hash(PI)||hash(OI)))Priv_C$$

$Priv_C$ denotes the cardholder's private key. The dual signature is an integral tool in SET protocol, the reason being that the Gateway will possess the digest of OI[1] and not the OI itself. Similarly the Gateway will have the message digest of PI but not PI itself. Consequently, the strength of the DS is that the Gateway and the merchant will be able to verify the validity of the dual signature (which binds the OI to PI) without being able to determine OI and PI respectively. The cardholder generates a symmetric Key K1, computes *hash(OI),* denoted by OIMD and the *hash(PI),* denoted by PIMD. They then proceed by sending the following information:

Cardholder → Merchant:

$ENC(PI||DS||OIMD)K1 + ENC(K1||ACC_{inf})Pub_G +$
$OI + PIMD + DS + CERT_C$

Pub_G and ACC_{inf} denotes the Gateway's public key and the customer bank account information, respectively.

Purchase Response Once the merchant receives the order from the cardholder, they proceed to the authorization phase. When the merchant receives the authorization response from the Gateway, they verify the Gateway's certificate and decrypt the message to obtain the symmetric key. This key is then used to decrypt the response message. Next, they verify the digital signature of the Gateway on the response. Knowing that only the Gateway is capable of decrypting the capture token, the merchant saves the capture in a safe location. The capture token, denoted CapToken, allows the merchant to have a guarantee of payment from the Gateway.

Payment Authorization Phase This phase is initiated by the merchant after receiving the purchase request message and is followed by sending the purchase response from the merchant to the cardholder. The purchase response includes the CapToken.

Authorization Request After receiving the purchase request from the cardholder, the merchant verifies the client's certificate and the dual signature DS by computing and decrypts the DS using the cardholder public key. If DS is the signature of *hash(PIMD||hash(OI)),* then the DS is validated. Provided that all information is validated, the merchant generates the authorization request AuthReq that consists of the date, OIMD, merchant details, the cardholder billing address and $\{ ENC(PI||DS||OIMD)_{K1}+ENC(K1||ACC_{inf})_{PubG} \}_{sent\ by\ cardholder}$. The merchant then generates a new secret key K2 before sending the following message:

Merchant →Gateway:

$\{ENC(PI||DS||OIMD)_{K1}+ ENC(K1||ACC_{inf})_{PubG}\}_{cardholder}$

[1] The digest of a message *M* is *hash(M)* where *hash()* is a cryptographic hash function[11].

$ENC(SIGN(AuthReq)_M)_{K2}+ENC(K2)_{PubG}$
$+CERT_C+CERT_M$

Pub_G *denotes the Gateway public key.*

Authorization Response The Gateway starts by verifying the merchant's certificate. It then decrypts K2 with Gateway's private key then decrypts authorization request using symmetric key K2. It then verifies the DS by computing and then comparing them. The Gateway then verifies cardholder's certificate by traversing the trust chain. It then decrypts K1 with Gateway's private key to use K1 to decrypt PI. The Gateway Ensures consistency between merchant's and customer's PI's. The Gateway then sends AuthReq through a financial network to customer's financial institution. Once validated by the financial institution, the Gateway computes the authorization response AuthRes. It then computes the capture token CapToken. The Gateway then generates two symmetric keys K3 and K4. Gateway sends to merchant:

Gateway $\rightarrow$ *Merchant:*

$ENC(SIGN(AuthRes)_G)_{K3}+ENC(K3||ACC_{inf})_{PubM}$
$+ENC(SIGN(CapToken)_G)_{K4}$
$+ENC(K4||ACC_{inf})_{PubG}+CERT_G$

PubG denotes the Gateway's public key and ACCinf the customer bank account information.

Payment Capture Phase After processing the cardholder order, the merchant requests payment from the Gateway by using the CapToken that they received in the authorization response message.

Capture Request This phase starts when the merchant generates a capture request CapReq and digitally signs it with its private key. They then randomly generate a symmetric key K5. The merchant then sends the following message:

Merchant $\rightarrow$ *Gateway:*

$ENC(SIGN(CapToken)_G)_{K4}+ENC(K4||ACC_{inf})_{PubG}$
$+ENC(SIGN(CapReq)_G)_{K5}$
$+ENC(K5)_{PubG}+CERT_M$

Capture Response When the payment Gateway receives the capture request, it decrypts the digital envelopes of K4 and K5 (using its private key). It then uses these keys to decrypt and verify the signatures of CapToken and CapReq. CapToken and CapReq are employed to generate a clearing request to the credit card issuer which is sent via a card payment system. Once authorized, the payment Gateway generates a capture response CapResp, digitally signs it. It is then encrypted using a newly generated symmetric key K6. The following message is then sent to the merchant:

Merchant $\rightarrow$ *Gateway:*

$ENC(SIGN(CapResp)_G)_{K6}+ENC(K6)_{PubG}+CERT_G$

Analysis of SET

SET relies on the use of an online trusted party: the Gateway. The SET protocol requires several security mechanisms to prevent any attack on the Gateway such as denial of service. If the Gateway becomes unavailable the whole e-commerce infrastructure is jeopardized. To prevent replay attacks, the Gateway has to store each processed transaction ID and prevent any second use of the authorization request message. As e-commerce transactions increase, preventing replay attacks becomes more delicate. This is a vulnerability that can be exploited to disrupt the whole e-commerce infrastructure.

Table1 provides a summary of the protocol's computations and communications. An integral tool within the SET protocol is the digital signature DS. It allows one to bind the order information OI with the payment information PI. Moreover, it allows the merchant and the Gateway to verify the signature without knowing the PI and OI respectively. However, the DS becomes very costly in a mobile environment, due to the nature of the devices and network environment, which demands a construction of a new tool.

Table 1: Computations in SET

	Signatures	Public Key computation	Communi-cation
Merchant	3 SG+ 3 SV	2 PKE + 2 PKD	4 S + 4 R
Customer	1 SG + 2 SV	1 PKE	2 S + 2 R

Here SG and SV denote signature generation and signature verification, respectively. S denotes a communication "send" and R denotes a communication "receive". PKE denotes a public key encryption process.

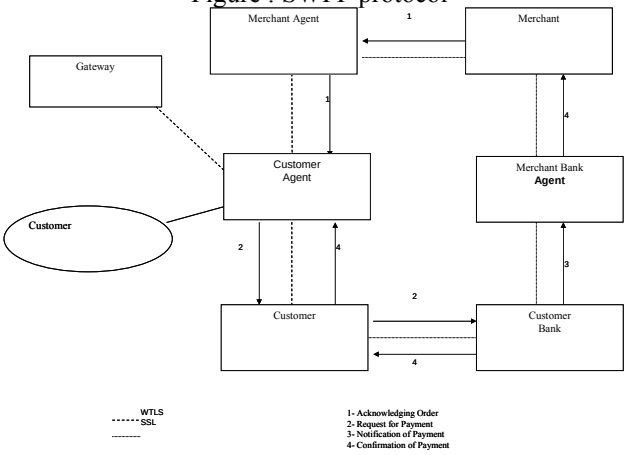
Clearly SET is expensive (resourcewise) in a mobile wireless environment; this is especially true for the merchant. With an additional requirement of a trusted gateway, it is clear SET would be impractical solution for a general m-commerce protocol. That is, SET is organized so the merchant is required to perform more computations that the customer (client) and has a persistent internet connectivity, assumptions that run contrary to our vision of the necessary capabilities for merchants in the future.

2.2 Secure Wireless Payment Protocol

In the previous section we discussed the SET protocol and how costly SET is in terms of bandwidth and computations in a wireless environment. In this section we will discuss a protocol that could be used in wireless transactions. Wireless Payment Protocol [1] (WPP) was introduced in 2000 to provide an efficient protocol for m-commerce. However, WPP falls short on security. In 2002, Secure Wireless Payment Protocol [2] (SWPP) was designed to address the security weaknesses of WPP. SWPP and WPP have the same design except for that SWPP does implement security mechanisms such as WTLS and digital signatures.

The major phases in the SWPP protocol are: acknowledgment order, request for payment, notification of payment, confirmation of payment. We will describe each of the phases in the coming sections. Based on [1],[2] we were able to construct an outline of the major phases and message flows in the protocol.

Figure : SWPP protocol



Acknowledgment Order

The protocol starts when the customer agent initiates a WTLS class 3 (thus both client/customer and server/merchant are authenticated) connection with the merchant. After browsing the merchant's products, the customer creates a list of the products that they request to purchase in the order information OI. Note that OI is not signed, later we will discuss what implications this will have on the protocol's security.

Stage 1*: Customer → Merchant*
OI

The merchant agent then generates the invoice, the payment information PI and merchant payment information MBI. The merchant then signs MBI and PI before encrypting the result with the customer public. This will conceal the MBI and PI data from being revealed to anyone other than the customer bank. Stage 2 represents the information sent by the merchant to the customer. Again, note that Invoice is not signed.

Stage 2: *Merchant →Customer*
$ENC(SIGN(PI+MBI+MII)_{PrivM})_{PubCB}$ *+Invoice*

where $Priv_M$ denotes the merchant private key and Pub_{CB} denoted the customer bank's public key.

Request for Payment

In this stage the customer verifies the invoice's data before initiating a new secure WTLS communication with the customer bank. The customer possesses a SWIM card where all confidential data such as bank information CBI and private keys are stored. A SWIM card is the WIM (specifications that provide the security elements necessary for e-commerce transactions) implementation on the SIM card [12]. To be able to retrieve the CBI, the customer agent requests it from the SWIM in stage 3, and the SWIM will respond by signing CBI. The customer then proceeds by forwarding the following:

Stage 3: *Customer →Customer Bank*
$\{ENC(SIGN(PI+MBI+MII)_{PrivM})_{PubCB}\}$ *from Merchant*
$SIGN(Invoice2+CBI+CID+PRN)_{PricvC}\}$

Confirmation of Payment

Once the merchant bank receives the request to transfer the funds, they verify the PI and MBI information. If validated, the merchant bank transfers the funds to the merchant account, signs the transaction confirmation, and sends it to the merchant agent.

Analysis of SWPP

Observe that the Transaction ID is not necessarily unique. The Merchant could take advantage of this by sending the same transaction ID's to two different customers and send the merchandise to one of them. Since the invoice does not include the customer unique identifier, the two invoice copies will be identical. Therefore, the merchant can blame one of the customers for making a copy of the invoice.

A major problem with the protocol is the transaction commitment since the invoice is not signed. The use of WTLS ensures integrity of the data exchanged through out the whole connection. However, this does not prevent any party from forging messages received if they are not signed. The customer could buy the product then renege having received the one he ordered. Moreover, the merchant can sell a product and ship a different one. Thus, any party can forge the invoice at the end of the transaction.

In addition, the order information OI is not signed. This could allow the merchant to forge it by changing the product that the customer requested. Consequently, they can ship a different product once the transaction has been processed.

Even if the invoice was signed, another commitment problem could arise. When the protocol is launched at Stage 1, the customer sends a customer order to the merchant. This results in a commitment problem i.e. a lack of commitment from both parties. That is the merchant is committed to selling the product, because they send the invoice, whereas the customer is not committed to buying it. Consequently *Company A* could attack the product availability of the *Company B* by repeatedly sending order requests for the product. The products will be held in reserve until the invoices expire. Since the customer is not

committed to buying it, they can continually request the product without necessarily buying it.

In the protocol design, we can assume that the banks are considered to be trusted parties. In reality, this is not completely accurate: banks should not be required to blindly trust each other and the customer does not trust an unsigned confirmation message from its bank. Giving additional trust responsibilities to the bank will complicate the protocol even more. In fact, creating a party that assumes all financial responsibilities and trusted by all parties at the same time will make it more vulnerable to attacks. We feel banks should be treated the same as any commercial entrepreneur and not require any special trust properties.

Table 2 underlines the major computations and communication that SWPP requires. As previously described, the protocol has some security breaches that any malicious merchant, customer or intruder could take advantage of. In order to improve the protocol security, the protocol should be modified and several signatures should be added such as the merchant signing the invoice and the customer signing the OI. Since the bank is not a trusted party, confirmations sent by the merchant's bank should be signed as well. This will result in the addition of: "One signature generation and two signature verifications for the merchant and one signature generation and one signature verification for the customer." These additional computations are included in Table 2.

Table 2 SWPP Computations and Communication Comparison

	Signatures	Public Key computation	Communi-cation
Merchant	2 SG* + 2 SV*	1 PKE	1 S + 2 R
Customer	2 SG* + 2 SV*	1 PKE	2 S + 2 R

Here SG and SV denote signature generation and signature verification, respectively. S denotes a communication ``send'' and R denotes a communication ``receive''. PKE denotes a public key encryption process. ``*'' denotes that additional computations have been added to make the protocol secure.

SWPP does not address all the security requirements that are needed. Further even if we improve the security features of SWPP, SWPP does not address fair commitment. Lastly, as our work will demonstrate improvements in bandwidth can be made.

3. An Improved Simple Secure M-Commerce Protocol (SSMCP*)

Here we address several problems concerning m-commerce. First we address the failure to achieve a fair commitment between the merchant and the customer. As well, our protocol is constructed in an attempt to reduce the amount of communications. The result is an improved protocol SSMCP*. A concept introduced in the protocol SSMCP* is the way the merchant receives the e-cheque from the customer. This enhancement is the creation of a public bank URL that is specific to each individual customer. As opposed to the ECheq passing through the customer to reach the merchant, the bank uploads the ECheq to the customer designated URL, denoted by $CUST_{URL}$, where the merchant can then download it. If the ECheq is timestamped at a date later than the invoice expiry date, then the transaction is canceled. This is based on a similar procedure that was used in the WTLS protocol where the URL for a WTLS-certificate is sent instead of the actual certificate [4]. Those communication steps that are required to make the *SSMCP* M-commerce transaction* are displayed in Figure 3.

Protocol Description

After browsing the merchant's products, the customer creates a list of the product requests along with their unique identifiers. At this point, starts the secure communication provided by the TLS/WTLS [4]. Having TLS/WTLS protocol in lower layers will provide integrity and authentication. In the communication between the merchant and the client (as well as between the bank and the customer), we will be using a TLS client authentication or a WTLS class 3, this requires that both the server (merchant) and the client (customer) authenticate themselves to each other. The customer requests to purchase the products by sending a *Purchase Request* that includes the invoice request $INVOICE_{Req}$ that contains the $CUST_{URL}$ of the customer (see Table3 for more details) and a customer nonce N_C. The customer then signs $INVOICE_{Req}$ concatenated with N_C.

Table 3 Invoice Request

$INVOICE_{REQ}$	Description
1- Customer's name, ID_C	Customer's full name and identifier
2- Merchant's name, ID_M	Merchant's full name and identifier
3- $CUST_{URL}$	Customer's personal bank URL where e-cheques are uploaded
4- $ID_{PRODUCT}$	The unique product ID (it could be repeated as many times as the number of products to be purchased)

5- Timestamp	Date and time of response

Stage SSMCP1: *Customer: Purchase Request →Merchantr*
$ENC(SIGN(PI+MBI+MII)_{PrivM})_{PubCB}$ +Invoice

Purchase Request=$Invoice_{Req}+N_C+$
$Sign_{Cpriv}(Invoice_{Req}||N_C)+CERT_C$
We will use symbols +, as well as || to represent concatenation.

After the merchant receives the *Purchase Request*, the merchant generates a merchant nonce N_M and *hashes* the concatenation of the nonces $N_M||N_C$ (we use the symbol || to denote concatenation). The resulting hash $h(N_M||N_C)$ will be transaction identifier ID_{Trans}. This will allow both parties to be sure that the transaction ID is unique and has never been used before. The merchant then generates the invoice, which is constructed in a manner very similar to the one in [2]. The invoice contains the ID_{Trans}, the (list of the) product Identifier(s) ID_{Prod} and a more detailed description of the product(s) in the list (Table 4).The merchant replies by sending the *Purchase Response* (Stage SSMCP1).

Table 4 Invoice

Invoice	Description
1- ID_{TRANS}	The unique transaction ID
2- ID_{PROD}	The unique product ID
3-Product description	The product detailed description
4- Item's number	The number of items of the product requested
5- Unit price	The unit price of the product
6-Steps 2-5 could be repeated	Repeated if the invoice contains multiple products
7- ID_M, $CERT_M$,	ID_M is the unique merchant identifier as it appears on the certificate.$CERT_M$ is the url of the merchant' s certificate
8- ID_C , $CERT_C$	ID_C is the unique Client identifier as it appears on the certificate $CERT_C$ is the url of the customer's certificate
9- Timestamp	Date and time of the invoice creation
10- Expiry date	Expiry date where the invoice becomes invalid

Stage SSMCP2 *Merchant:Purchase Response →Customer*

Purchase Response =$Invoice+N_M+Sign_{Mpriv}(Invoice||N_C||N_M)$

When the *Purchase Response* is received, the customer verifies the invoice and the ID_{Trans} by computing the hash of the concatenated nonces $N_M||N_C$. Provided that all information is verified, the customer will start a secure communication channel with their Bank, the communication will satisfy a WTLS class 3 connection (both client (customer) and server (bank) are authenticated). The customer then starts to generate the customer payment information PI_C (see Table 5).

The PI_C contains the customer's detailed bank account information. The customer then initiates Stage SSMCP3 and signs $PI_C||ID_{Trans}||ECheq_{Req}||CERT_M$ where $ECheq_{Req}$ is the request for e-cheque (see Table 6).

Table 5 Payment Information

PI_C	Description
1- Bank Name	Customer's bank
2- Customer's name, ID_C	Customer's full name and identifier
3- Customer's account number	Customer's bank account number
4- Timestamp	Date and time of request

Stage SSMCP3 *Customer: E-cheque Request→ Bank*

E-cheque Request=$PI_C+ID_{Trans}+ECheq_{Req}+CERT_M+$
$Sign_{Cpriv}(PIC||ID_{Trans}||ECheq_{Req}||CERT_M)$

The bank verifies the customer's signature as well as their bank account information extracted from the PI_C. Provided that all information is valid, the bank will check for fund availability and credit the customer's account with the corresponding transaction's amount. The bank, then, constructs the e-cheque and digitally signs it. The bank generates a unique e-cheque identifier ID_{ECheq} that will prevent any replay attack by the merchant. The e-cheque, denoted by ECheq, will be created and will contain the ID_M (extracted from the merchant certificate), $CERT_M$, the transaction's amount, and the ECheq expiry date (see Table 7).

Table 6 Request for e-cheque

$ECheq_{Req}$	Description
1- Bank name	Customer's bank
2- ID_C	Customer's unique identifier
4- Amount	Amount of money for fund transfer
5- Timestamp	Date and time of request

Table 7 E-cheque

ECheq	Description
1- ID_{ECheq}	Unique serial number for the token generated by the Gateway
2- ID_{Trans}	Transaction unique identifier
3- ID_{Issuer}	E-cheque's Issuer unique identifier
4- ID_{Bearer}	E-cheque's bearer unique identifier

5- ID_B	E-cheque bearer's bank unique identifier
6- Amount	Amount of money to be paid
7- Timestamp	Date and time of request
8- Expiry Date	E-cheque expiry date. When the e-cheque expired becomes invalid

Therefore the bank stores ID_{Trans} along with the ID_{ECheq} in its database. It will be deleted when the ECheq has expired or has been cashed out. If the ECheq has expired, the bank will re-credit the customer's account with the original transaction amount. The bank signs the ECheq as well as the $ECheq_{Resp}$ (see Table 8) that contains the bank response regarding the success of the transaction.

Table 8 E-cheque Response

$ECheq_{Resp}$	Description
1- Bank identifier	Bank unique identifier
2-ID_{ECheq}	E-cheque unique identifier (optional). It will be sent just in case the transaction is approved by the bank
3- ID_{Trans}	Unique transaction ID
4- Response	Indicates whether the transaction was valid or not. If not it will indicate the reasons of failure
5- Timestamp	Date and time of response

Stage SSMCP4: *Bank: E-cheque Response → Customer*

E-cheque Response =$ECheq_{Resp}$+ $Sign_{Bpriv}(ECheq_{Resp})$

The bank then constructs the ECheq and signs it. After signing the ECheq, the bank uploads it to the customer's URL $CUST_{URL}$ in *E-cheque Upload* (Stage SSMCP5).

Stage SSMCP5 *Bank: E-cheque Upload → $CUST_{URL}$*
E-cheque Upload = *upload* ECheq+$CERT_B$+$Sign_{Bpriv}$(ECheq)

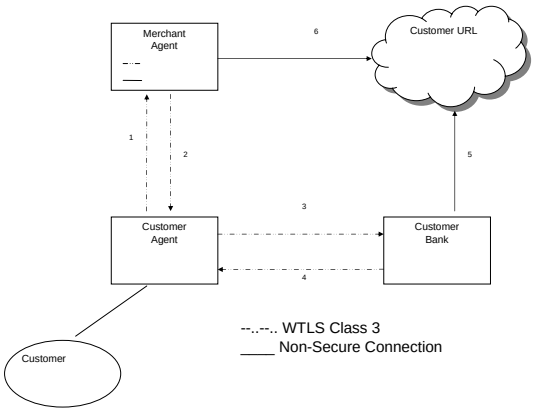
At the invoice expiration date, the merchant downloads the ECheq from the URL in *E-cheque Verification* (Stage SSMCP6).

Stage SSMCP 6 *$CUST_{URL}$: E-cheque Verification} →Merchant*
E-cheque Verification = Merchant *downloads*
ECheq+$CERT_B$+$Sign_{Bpriv}$(ECheq)

The merchant then verifies the e-cheque's information ID_{Trans}, ID_M, amount, *valid expiration date*} and *time stamp* and the bank signature on it. Observe that if the e-cheque is time stamped after the invoice expiry, the merchant has the right to cancel the transaction. The merchant can cash the ECheq at any time before its expiration. This stage can actually be completed by the financial agent of the merchant. The download can actually occur at a later time. The merchant would need to be contacted (via some authenticated channel) by the merchant's financial agent that the ECheq is valid. Thus a further savings in bandwidth could be achieved by having the financial agent download and verify the ECheq.

The entire SSMCP* protocol is illustrated in Figure 3.

Figure 3 SSMCP* protocol



Arbitration--judging a transaction which has a malicious or faulty party

If two signed invoices are found with the same ID_{Trans}, then the merchant will be held responsible for it. Remember the merchant is the last party to choose the nonce. Due to the use of a cryptographic hash function it is infeasible to produce a collision of two hashes. In the case of potential fraud of ID_{Trans}, each customer should present their signed copy of the invoice to the judge. The customer should keep the invoice as a proof of the item purchased and its price since the merchant committed to it by signing it. If the item shipped was somehow improper, the customer can show the invoice to the judge. The judge will verify the ECheq and make sure that the ID_{Trans} corresponds to the IDTrans on the invoice. The judge will then verify the ID_M, ID_C, ID_{Prod} and the description of the product(s) on the invoice as well as the invoice signature (by using the public key corresponding to the ID_M on the invoice). Lastly, any replay attack by the merchant to attempt cashing the ECheq more than once will be detected by the bank since the ECheq has a unique identifier ID_{ECheq} (to keep track of those ECheq that have been cashed).

In a typical e-commerce protocol the price commitment made by the merchant is not complete until the merchant cashes the check, which is the formal acknowledgement of acceptance. Realize that under our view of m-commerce, both merchant and customer communicate using a wireless medium, but neither may have a persistent internet

connection. Thus there may be a delay between the cashing of the check. Such a delay, under a typical e-commerce protocol, would cause a delay in a "commitment". In our view of the m-commerce, because the protocol could be used with a merchant who does not necessary have a persistent internet capability, deadlines/expiration dates will need to allow some flexibility. Consequently if the merchant has flexibility in the expiration date, then the a merchant could purposely delay the cashing of a customer's check, in hopes of finding a better deal, if they do find a better deal, they can claim the check was lost or simply let the deal expire. Thus the merchant has an upper hand and there is no price commitment by the merchant until the very hand. In a court of law, there would be no material proof of the merchant's receipt of the check. This problem has been addressed in our protocol SSMCP*. The merchant assigns a period of time for the customer to submit the ECheq by determining an expiration date for the invoice. In this manner, the merchant is unable to cancel the transaction unless the customer uploads the ECheq after the invoice expiration date. The fact that there is material proof of the exact time of receipt of the ECheq (the time stamp the bank applies to the signed ECheq) can be used to resolve conflicts in a judiciary setting. As banks are federally regulated institutions there is trust that the fields generated by the bank inside ECheq are accurate. This trust is created once one verifies the bank signature on the ECheq.

Assessing SSMCP*

A major improvement is that the new version of the protocol removes the ability of the merchant to renege the price of the e-cheque and delay the transaction.

Table 9 provides a comparison between SSMCP*, SET and SWPP and highlights the major communication's and computation's improvements. In addition, the protocol replaces sending the entire certificates by sending their URL which reduces bandwidth.

Table 9 Comparison between E-commerce

	Merchant Comp.	Merchant Comm.	Customer Comp.	Customer Comm
SET	3 SG+3 SV +2 PKE +2 PKD	4 S+4 R	1 SG +2 SV + 1 PKE	2 S+2 R
SWPP	2 SG+2 SV +1 PKE	1 S+2 R	2SG+2SV + 1 PKE	2 S+2 R
SSMCP*	1 SG+2 SV	1 S+1 R	2SG+2SV	2 S+2 R

Here SG and SV denote signature generation and signature verification, respectively. S denotes a communication ``send'' and R denotes a communication ``receive''. PKE and PKD denote a public key encryption and decryption, respectively.

SET is considered to be a secure protocol; however its efficiency can fall short in a mobile environment. SSMCP* presents a better fit in such an environment by eliminating the need of the costly (in terms of resources) dual signatures. Another major improvement is that SET requires the online involvement of a trusted party: the Gateway. The Gateway has to keep track of all transaction processed to prevent replay attacks. On the other hand in SSMCP*, banks have to store the transaction ID until the e-cheque is cashed only.

Furthermore, SSMCP* provides several security improvements over SWPP. In fact, transaction ID is not guaranteed to be unique in SWPP creating several security issues as described in previous sections. Another security issue with SWPP is that the protocol does not prevent customer's replay attacks.

Suggested cryptographic tools

An m-commerce transaction, because it involves finances, will require one to choose a suitably secure (size) digital signature key. In the RSA cryptosystem, the public key is (e,N) such that modulus N=p * q, where p and q are distinct primes.The private key is d such e * d =1 mod $\varphi(N)$ where $\varphi(N) = (p-1)(q-1)$. The security of RSA is related to the difficulty of factoring, and the essential cryptographic computation is an exponentiation modulo the composite N. The signing computation will involve the use of the exponent d and the verification will utilize e. The problem concerning the size of e and the size of d has been well researched [3]. The public key e may be selected to be small, but in order to avoid serious attacks [3], the private key (signing key) d needs to be large and will be approximately the same size as N. Today a 1024 bit RSA modulus is considered secure, but financial institutions require significantly stronger keys, so one would probably choose a 2048 or 3072 bit RSA key for an m-commerce transaction.

Elliptic curve cryptography ECC is often viewed as a panacea for the problems of a low-complexity device implementing public key cryptography (due to limited computing resources and bandwidth). An elliptic curve is determined by the collection of all ordered pairs, from a finite field that satisfy a particular equation, together with the point of infinity. For example, when using the field $GF(2^n)$, the equation $y^2+xy=x^3+ax^2+b$ determines an elliptic curve, where a and b are fixed field elements which would be characterized as elliptic curve parameters. The curve is constructed so that it has a subgroup of large prime order. An addition can be defined on the elliptic curve, and the basic cryptographic computation is to compute the scalar multiple $kP=P+P+\ldots+P$ of a point P which belongs to the elliptic curve. The public-key is a scalar multiple of a point kP, and the private key is the scalar k. The keysize for an ECC cryptosystem does not need to be anywhere near as large as an RSA key. The ECDSA is a federally approved ECC digital signature

scheme [7]. For an elliptic curve defined over the Galois field $GF(2^n)$, a suitable sized n (for precise details of the elliptic curve see [7]) for an elliptic curve equivalent (in terms of security) to a RSA modulus of 2048 bits is n=233 bits, and a suitable sized n for an elliptic curve equivalent to a RSA modulus of 3072 bits is n=283 bits. Table 10 describes the computational time for computing the most time consuming operation of RSA and ECC, for RSA it represents a modular exponentiation and for ECC it represents the time to compute a scalar multiple of a point. It demonstrates that ECC would provide significant time improvement over RSA. In addition to the computational complexity advantage that ECC holds over RSA, there is also a bandwidth advantage. An ECDSA signature consists of an ordered pair (r,s) where both r and s belong to the finite field. For example, the elliptic curve defined over $GF(2^{233})$ is an equivalent to an RSA modulus of 2048 bits and the elliptic curve defined over $GF(2^{283})$ is an equivalent to an RSA modulus of 3072 bits. Thus the elliptic curve signature will be significantly smaller than the RSA signature. In the former case, ECC defined over $GF(2^{233})$, 466 bits are required for an ECC signature which is equivalent to a 2048 bit RSA signature. Thus there will be bandwidth savings in the signature. There will be additional bandwidth improvement when one considers the size of an ECC certificate versus an RSA certificate (transmission of public-key certificates is a necessity). For example, when transmitting an elliptic curve point, one can use point compression to transmit the point, a result which reduces the bandwidth complexity by a half [6].

Lastly, with ECC cryptosystem one can standardize the security level as well as the elliptic curve. Thus one can optimize the implementation and tailor it for the given parameters, whereas in RSA no two parties can reuse the same modulus of their private-key/public key selection.

Consequently, the ECDSA is the signature algorithm that we recommend for use in our protocol, and we recommend the use of ECC for any of the necessary public-key cryptographic operations.

Table 10 RSA vs ECC comparison at same security level

RSA N vs ECC $GF(2^n)$	RSA N	ECC over $GF(2^n)$
N=2048 bits n=233 bits	2081 millisecs	57.58 millisecs
N=3072 bits n = 283 bits	7540 millisecs	114.37 millisecs

4. Conclusion

In this paper, we have defined a low-complexity m-commerce protocol that is constructed in such a way that it minimizes both bandwidth as well as computational complexity for both the merchant and the consumer. The realization that we will have low-complexity merchants in the future is important, and the limitations of low-complexity merchants have not been addressed in other e-commerce protocols. Lastly, we have provided comparisons of our protocol SSMCP* with other e-commerce protocols.

References

[1] Allen C., Dierks T. The TLS Protocol Version 1.0, RFC2246 http://www.faqs.org/rfcs/rfc2246.html January 1999

[2] Beadle H., Gonzalez R., Safavi-Naini R., Bakhtiari S. "A Review of Internet Payments Schemes", In Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC'96), Melbourne, Australia, December 1996, pp.486-94

[3] Boneh, D. "Twenty years of attacks on the RSA cryptosystem". In Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203– 213, 1999.

[4] Fourati A., Ben Ayed H., Kamoun F., and Benzekri A.. "A SET Based Approach to Secure the Payment in Mobile Commerce". 27th Annual IEEE Conference on Local Computer Networks (LCN'02), Nov 2002, pp. 136-140.

[5] Hall J., Killbank S., Barbeau M., Kranakis E. WPP: A Secure PaymentProtocol for Supporting Credit- and Debit-Card International Conference on Telecommunications, Romania, Bucharest, June 4-7, 2001

[6] King, B. "A Point Compression Method for Elliptic Curves Defined over GF(2n)". Workshop on Public Key Cryptography 2004, LNCS, Springer-Verlag, pp. 333-345

[7] NIST. "Recommended elliptic curves for federal use". http://www.nist.gov.

[8] Stallings W. Cryptography and Network Security, Prentice-Hall, New Jersey. 1999.

[9] Wang H., Kranakis E. "Secure Wireless Payment Protocol". International Conference on Wireless Networks 2003. pp. 576-582.

[10] WAP Forum WAP WTLS: Wireless Application Protocol Wireless Transport Layer Security Specification http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf

[11] Stinson D. Cryptography. Theory and Practice. CRC Press. Boston. 2002.

[12] Friis-Hansen S., Stavenow B., "Secure Electronic Transactions-The Mobile Phone Continues", Ericsson Review, no. 4, 2001. http://www.ericsson.com/about/publications/review/200104/files/2001041.Nake