Electron Commer Res (2010) 10: 209–233
DOI 10.1007/s10660-010-9052-2

Implementation and performance evaluation of a payment protocol for vehicular ad hoc networks

Jesús Téllez Isaac · Sherali Zeadally · José Cámara Sierra

Published online: 11 July 2010

Abstract *Vehicular ad hoc networks* (*VANETs*) are envisioned to support the development of a wide range of attractive applications such as payment services which require the design of payment systems that satisfy additional requirements associated with VANETs. The wide range of scenarios (with or without connectivity restriction) arising from vehicle-to-vehicle and vehicle-to-roadside communications have opened up new security challenges which must be considered by payment system designers to achieve the same security capabilities independent of the scenario where payment occurs. We propose and implement a new payment protocol (called KCMS-VAN protocol) for those scenarios where the client cannot communicate directly with the credit card issuer (the client's financial institution) for authentication. Our proposed protocol uses symmetric-key operations which require low computational power and can be processed much faster than asymmetric ones. We also present a performance evaluation of the proposed payment protocol and the results obtained demonstrate that optimal performance can be achieved with it.

Keywords Performance evaluation · Vehicular ad hoc networks · Payment protocol · Security · Implementation

J. Téllez Isaac (✉)
Computer Science Department, FACYT, Universidad de Carabobo, Av. Universidad, Sector Bárbula, Valencia, Venezuela
e-mail: jtellez@uc.edu.ve

S. Zeadally
Department of Computer Science and Information Technology, University of the District of Columbia, Washington, DC 20008, USA
e-mail: szeadally@udc.edu

J.C. Sierra
Computer Science Department, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911, Leganés, Madrid, Spain
e-mail: sierra@inf.uc3m.es

1 Introduction

Vehicular Ad hoc NETworks (VANETs) are a form of Mobile Ad hoc NETwork (MANET) that aims to provide communications among nearby vehicles (also known as Inter-Vehicle Communication -(**IVC**)-) and between vehicles and nearby roadside base-stations (also referred to as Vehicle-to-Roadside Communication -(**VRC**)-).

The application space for vehicle-to-vehicle and vehicle-to-roadside communications is vast and opens up tremendous business opportunities and research challenges among which security is an important one. VANETs are envisioned to support the development of a wide variety of new attractive applications that can be broadly divided into two major categories [21, 24, 37]: (1) *Safety-related applications*, and (2) *Comfort applications*. Although the industry and academia have concentrated their research efforts on safety-related applications due to its importance for the automotive domain, it is expected that research on Comfort applications (that also offer great business opportunities) will continue to attract the attention of researchers and designers to develop non-safety VANET-based applications.

To enable payments in VANETs, it is necessary to build payment systems that satisfy the additional requirements associated with vehicular ad hoc networks. As mentioned previously, both vehicle-to-vehicle and vehicle-to-roadside communications open up new security challenges that must be considered by payment system designers to achieve the same security capabilities independent of the scenario where payments occur.

The above situation can be represented in the real word by the following example: a client is on the road and stops at a gas station to purchase gas or some others goods at the gas station store with some payment card. In both cases, if the client is not able to communicate with the card issuer (to authorize the payment) from the application unit (because of the absence of the infrastructure necessary such Road-Side Units (**RSUs**), or a HotSpot (**HS**)-), the client should still be able to perform the payment using the merchant's infrastructure.

An example of the above situation is the restricted connectivity scenario shown in Fig. 1 where a car (henceforth referred to as the client), with an On-Board Unit (OBU) and an Application Unit (AU) can only connect to the merchant during a payment transaction due to the lack of Internet access with its AU. This situation creates a potential security problem because the client cannot send any kind of messages directly to the issuer and has to do it through the merchant (who should not be able to change the content of the messages but must keep evidence of the payment). Note that in our scheme, the merchant takes an active role in the payment process because it acts as a proxy to allow the communication between the client and the card issuer.

Fig. 1 A restricted connectivity scenario in VANETs

On the other hand, thinking in a VANET scenario with moving vehicles, we can found the following example in the real world which represents a restricted connectivity scenario: a client is driving on a road and wants to purchase a weather forecast using some payment card. This situation creates the same potential security problem than the aforementioned example. However, even in this kind of situations, the scheme proposed in this paper could be applied (according to the ideas claimed in the Carlink project, proposed by [26]). However, we can include any experimental that show that the proposed protocol performs well when the client is on the move because this study is beyond the scope of this paper.

In order to provide authentication in electronic payment systems (including mobile commerce), many methods have been considered but symmetric and asymmetric signature methods are chosen for authentication [10]. However, traditional asymmetric signature schemes make the signature computations very expensive and are not suitable for those portable devices (such as the ones typically attached to an OBU [7]) available in the market not based on the Texas Instruments TMS320C55x processors family [11].

The protocol we propose in this work is for the case where we do not have direct communication between the client and the card issuer. As a result, the above schemes are not suitable because the client has connectivity restrictions, and consequently, communication with other parties (such a Certification Authority, for verifying a certificate) is not possible during a payment transaction. Therefore, the use of symmetric signature scheme is required to satisfy the requirements of the protocol proposed in this work. Symmetric cryptography (which employs a shared key between two parties) provides (such as asymmetric cryptography) message confidentiality, message integrity, and party authentication, and represents an alternative in the construction of secure protocols for mobile payment systems. Symmetric-key operations do not require high computational power nor do they require additional communications steps (as in the case of protocols based on public-key infrastructures where public-key certificates have to be verified by a Certificate Authority).

To address the issue of direct communication between the client and the card issuer for authentication purposes, we designed and implemented a payment protocol that allows the client to send a message to the issuer through a merchant (who will not be able to decrypt the message). The proposed protocol, called the Kiosk Centric Model payment protocol for VANETs (henceforth referred to as the KCMS-VAN Protocol), employs symmetric-key operations in all engaging parties to reduce both, the setup cost for the payment infrastructure and the transaction cost. Furthermore, it supports both credit-card and debit-card transactions, protects the real identity of the client during the payment and should be used with a portable device attached to an AU.

We analyze the performance of the KCMS-VAN protocol with actual mobile phones and PDAs as the implementation platforms. This allows us to further demonstrate that our client-side application can be installed on multiple heterogeneous Java™-enabled memory-constrained portable wireless handheld devices.

The rest of the paper is organized as follows. Section 2 presents recent related works related to our research. In Sect. 3, we propose and describe the payment protocol for vehicle-to-roadside scenarios in VANETs. Section 4 presents the implementation of KCMS-VAN in detail. We present performance evaluation results of the proposed protocol in Sect. 5. Finally, make some concluding remarks in Sect. 6.

2 Related work

In recent years, several studies have been conducted to improve the security of mobile payment systems. Many of these efforts have also been dedicated to unify concepts and scenarios into frameworks that will be useful in the design of new electronic payment systems. Many of these studies have considered the following methods to provide authentication in electronic payment systems (including mobile commerce): username/password, symmetric and asymmetric cryptography, smart card, and biometric methods. Username/password does not offer enough security for m-commerce. Biometric approaches are not feasible at present and smart card would require an external device to read the card. Symmetric and asymmetric signature have been widely used for authentication purposes.

Most of the protocols proposed in recent years are based on the Full Connectivity scenario (where all the entities are directly connected one to another [8]) and employ asymmetric-key operations [5, 9, 12, 17, 36] whereas the remaining scenarios use symmetric-key operations which is more suitable for wireless networks [15]. Unfortunately, usage of those protocols is not possible in scenarios where direct interaction among two of its parties is not allowed due to the communication restriction imposed by the model (as in the case for the Kiosk Centric Model [8]). As a result, a new trend has emerged which is to develop mobile payment systems based on restricted connectivity scenarios, achieving the same security and performance levels as the Full Connectivity Scenario. Protocols proposed by Téllez et al. [27–32] constitute examples of mobile payment protocols suitable for scenarios with communication restrictions. However, such proposals are theoretical and do not capture practical performance issues we encounter with real systems. Thus in this work, we present an implementation of our proposed secure payment protocol and evaluate its performance.

3 Proposed secure payment protocol for vehicle-to-roadside scenarios in VANETs

3.1 The kiosk centric model payment protocol for VANETs (KCMS-VAN) model

The KCMS-VAN protocol was designed taking into consideration the model suggested by Abad-Peiro et al. [1] such that it can be applied by many different payment methods. Thus, based on the General Payment Model of Abad-Peiro et al. [1], our proposed secure payment protocol uses the following entities:

1. *Client:* a user who wants to purchase goods or services from the merchant. In our proposed protocol, the client is a userside entity equipped with an On-Board Unit (called **OBU**) and/or an Application Unit (called **AU** that may use the OBU's communication capabilities). An AU can be an integrated part of a vehicle and be permanently connected to an OBU or could be a portable device such as a Personal Digital Assistant (PDA), a mobile phone or a gaming device that can dynamically attach to and detach from an OBU [7].

2. *Merchant:* an entity that has products or services to offer or sell. This entity could be a computational one (such as a normal web server, a roadside computing station or an intelligent vending machine) or a physical one (such as a gas station that makes it possible to pay from within an AU) which the user can connect to using a short range link (using wireless technologies such as Wi-Fi or Bluetooth). Moreover, this entity connects with the Payment Gateway (an entity which provides the necessary infrastructure to allow a merchant to accept credit card and other forms of electronic payment) through a secure channel allowing the client to communicate with the issuer using this connection.
3. *Acquirer:* is the merchant's financial institution. It verifies the validity of the deposited payment instructions and manages the merchant's account including fund transfers.
4. *Issuer:* is the client's financial institution. It provides electronic payment instructions to the client to use in a payment and manages the client's account including fund transfers.
5. *Payment gateway:* an additional entity that acts as a medium between acquirer/issuer on the bank's private network side and the client/merchant at the Internet side [15].

The parties of the former model communicate with each other regarding fund transfers, using the following 3 primitive payment transactions:

– In *Payment*, the client transfers the value to the merchant.
– In *Value Subtraction*, the client requests that the payment gateway (on behalf of issuer) deducts the money from the client's account.
– In *Value Claim*, the merchant requests that the payment gateway (on behalf of acquirer) transfers money to the merchant's account.

The five entities in KCMS-VAN and their interactions are shown in Fig. 2. Note that there is no direct interaction between the client and the issuer. Moreover, the connection between the client and the merchant (denoted as the dotted arrow) is set up through a wireless channel (such as IEEE 802.15.1, Bluetooth, IEEE 802.11a/b/g).

Interaction between the merchant and the payment gateway (depicted as solid arrow in Fig. 2) should be reliable and secure against passive and active attacks. Therefore, the connection should be established through a secure wired channel using a well-known security protocol such as secure socket layer/transport layer security (SSL/TLS). Note that the issuer, the acquirer and the payment gateway operate in the bank private network. The security of the messages exchanged among them is beyond of the scope of this paper.

Before receiving payment services, the client must register with an Issuer. Generally, this registration can be done either personally at the issuer's premises or via the issuer's website. During the client's registration, the following steps are performed:

1. The Client shares his/her credit- and/or debit-card information (CDCI) with the issuer (who will not reveal it to any merchant).
2. The Issuer assigns several nicknames to the client. Those nicknames are known only to the client and the issuer [10].

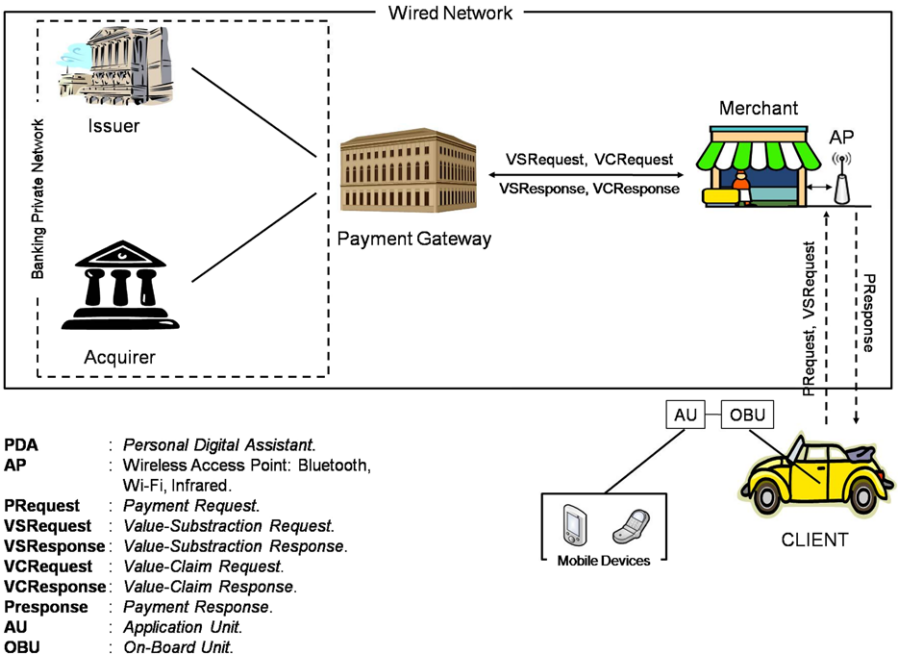

Fig. 2 KCMS-VAN architecture

3.2 Notations

All the entities involved in our proposed payment protocol are called parties and communicate through wireless and wired networks. The symbols C, M, PG, I, A are used to denote the names of the parties Client, Merchant, Payment Gateway, Issuer and Acquirer, respectively. The following symbols are used to represent other messages and protocols:

- **ID_P:** The identity of party P that contains the contact information of P.
- **NID_C:** Client's nickname, temporary identity.
- **TID:** Identity of transaction that includes time and date of the transaction.
- **TST_P:** Timestamp generated by P.
- **Stt:** The status of transaction (Stt = {Accepted, Rejected}).
- **OI:** Order information ($OI = \{TID, OD, h(OD,\text{Price})\}$) where OD and Price are order descriptions and its amount.
- **TC:** The type of card used in the purchase process ($TC = \{\text{Credit, Debit}\}$).
- **TIDReq:** The request for TID.
- **MIDReq:** The request for ID_M.
- **SEC_{A-B}:** The master secret shared between parties A and B.
- **$\{M\}_X$:** The message M symmetrically encrypted with the shared key X.

Table 1 Details of key generating technique

Generating KS_{C-M_i}	$KS_{C-M_1} = h(1\text{-bit-shift-of-}KS_{C-M})$, $KS_{C-M_2} = h(2\text{-bit-shift-of-}KS_{C-M}), \ldots,$ $KS_{C-M_n} = h(n\text{-bit-shift-of-}KS_{C-M})$
Generating KS_{M-PG_k}	$KS_{M-PG_1} = h(1\text{-bit-shift-of-}KS_{M-PG})$, $KS_{M-PG_2} = h(2\text{-bit-shift-of-}KS_{M-PG}), \ldots,$ $KS_{M-PG_n} = h(n\text{-bit-shift-of-}KS_{M-PG})$
Generating KS_{C-I_z}	$KS_{C-I_1} = h(1\text{-bit-shift-of-}(CDCI, KS_{C-I}))$, $KS_{C-I_2} = h(2\text{-bit-shift-of-}(CDCI, KS_{C-I})), \ldots,$ $KS_{C-I_n} = h(n\text{-bit-shift-of-}(CDCI, KS_{C-I}))$

- $h(M)$**:** The one-way hash function of the message M.
- **MAC(X,K):** Message Authentication Code of the message X with the key K.
- KS_{A-B_t}**:** The session key shared between parties A and B, generated applying a hash function with t-bit cyclic shifting (either left shift or right shift) of KS_{A-B}. More details about this technique is given in the next section.

3.3 Session key generation technique

Two efficient key generation techniques are employed in KCMS-VAN to generate the sets of session keys used in transactions. In both of them, the main idea is to apply a hashing algorithm with one-bit cyclic shift of a master secret each time a session key is generated [16]. As a result, the performance of the protocol is increased due to the reduced frequency of the key update processes.

The key set KS_{C-M_i} (with $i = \{1, \ldots, n\}$), is generated from the secret key KS_{C-M} and stored in the client's device and merchant's terminal. The set KS_{M-PG_k} (with $k = \{1, \ldots, n\}$), is generated from the secret key KS_{M-PG} and stored both in the merchant and Payment gateway terminals. The set KS_{C-I_z} (with $z = \{1, \ldots, n\}$), is generated from the key secret KS_{C-I} and is stored in the client's device and the issuer's terminals.

The details of the generation of the different sets of session keys are shown in Table 1.

3.4 Our proposed kiosk centric model payment protocol for VANETs (KCMS-VAN)

The KCMS-VAN Protocol is composed by two sub-protocols: the *KCMS-VAN Merchant Registration Protocol (MRP)* and the *KCMS-VAN Payment Protocol (PP)*.

For the *KCMS-VAN Merchant Registration Protocol*, the client has to register with the merchant to send the master key KS_{C-M}. The protocol has to be executed every time the client wants to perform transactions with a merchant. The details of the protocol are shown as follows:

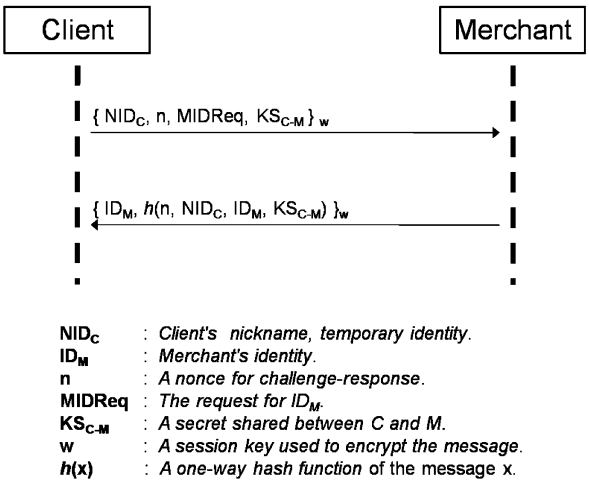

Fig. 3 Merchant registration protocol message exchange

$$\mathbf{C} \rightarrow \mathbf{M}: \quad \{NID_C, n, MIDReq, KS_{C-M}\}_w$$
$$\mathbf{M} \rightarrow \mathbf{C}: \quad \{ID_M, h(n, NID_C, ID_M, KS_{C-M})\}_w$$

Client **C** generates KS_{C-M} which is to be shared with the merchant and sends it with her/his nickname NID_C, a nonce n for challenge-response and $MIDReq$ to **M**. After the merchant **M** receives the message, she/he sends $h(n, NID_C, ID_M, KS_{C-M})$ and the merchant's identity (ID_M). Note that both messages (from the client to the merchant and viceversa) are encrypted with the session key w. The transmitted messages between the client and the merchant during the *Merchant Registration Protocol* are shown in Fig. 3.

Once **C** and **M** have exchanged the necessary information, they can generate a new set of KS_{C-M_i} using the same key generation technique. The client may then start the *KCMS-VAN Payment Protocol*.

For the *KCMS-VAN Payment Protocol*, the client purchases goods from the merchant and pays for them using her/his credit-card or debit-card. This protocol is formalized as follows:

$$(1)\ \mathbf{C} \rightarrow \mathbf{M}: \quad NID_C, i, TIDReq$$
$$\mathbf{M} \rightarrow \mathbf{C}: \quad \{TID, ID_M\}_{KS_{C-M_i}}$$

$$(2)\ \mathbf{C} \rightarrow \mathbf{M}: \quad \{OI, Price, NID_C, ID_I, TST_C, z, h(KS_{C-I_z}), VSRequest\}_{KS_{C-M_i}}, MAC[(OI, Price, NID_C, ID_I, TST_C, z), KS_{C-M_{i+1}}].$$

$$VSRequest = (MAC[(Price, h(OI), TST_C, TC, ID_M), KS_{C-I_z}], TC, TST_C)$$

$$(3)\ \mathbf{M} \rightarrow \mathbf{PG}: \quad \{VCRequest, ID_M, z\}_{KS_{M-PG_k}}, k, MAC[(ID_M, k, z, VCRequest), KS_{M-PG_{k+1}}]$$

$$VCRequest = (VSRequest, TST_M, h(OI), TID, Price, NID_C, ID_I)$$

(4) Under banking private network,

(4.1) **PG** → **I:** $NID_C, ID_M, VSRequest, TID, h(OI), z,$
$Price, h(KS_{M-PG_{k+1}})$

(4.2) **PG** → **A:** $Price, ID_M$

(4.3) **I,A** → **PG:** $VSResponse, Stt, h(Stt, h(OI))$

$$VSResponse = \{Stt, h(OI)\}_{KS_{C-I_z}}$$

(5) **PG** → **M:** *VCResponse*

$$VCResponse = \{Stt, VSResponse, h(Stt, h(OI), h(KS_{C-I_z}))\}_{KS_{M-PG_{k+1}}}$$

(6) **M** → **C:** *PResponse*

$$PResponse = \{VSResponse\}_{KS_{C-M_{i+1}}}$$

Step 1: The client **C** and merchant **M** exchange the information necessary to start the protocol by performing the following sub-step.

1-1: **C** sends his/her nickname (NID_C), the index i (that will be used to generate the session key between the client and the merchant) and the request for the transaction identity (*TIDReq*) to **M**.

1-2: **M** receives the request and sends back its identity (ID_M) and *TID* to **C**, encrypted with KS_{C-M_i}.

Step 2: Client **C** creates a *Payment Request* (referred to as the General Payment Model as described in [1, 15]) in the following sub-steps.

2-1: A *Value-Subtraction Request* (called *VSRequest*) is created and it includes $MAC[(Price, h(OI), TST_C, TC, ID_M), KS_{C-I_z}]$, TST_C and TC.

2-2: A new message is created which includes $C's$ nickname, $I's$ identity, *Price*, *OI* (used to inform **M** about the goods and prices requested), *VSRequest*, the index z, timestamp TST_C read from $C's$ clock and $h(KS_{C-I_z})$ (used to prevent the payment from modifying the approval result in Step 4-5).

2-3: The message created in the previous sub-step (henceforth referred to as the *Payment Request*) is encrypted with the session key KS_{C-M_i}.

2-4: The *Payment Request* is sent to the merchant.

Step 3: The merchant **M** generates the *Value-Claim Request* (called *VCRequest*) by performing the following sub-steps.

3-1: The message received from **C** is decrypted to extract *OI* and TST_C.

3-2: The timeliness of the *Payment Request* is verified. If the check is successful, the following sub-steps will be performed.

3-3: The *VCRequest* is prepared, and contains the *VSRequest*, TST_M, *h(OI)*, identity of transaction, order's amount, $C's$ nickname, and $I's$ identity.
3-4: The *VCRequest*, the $M's$ identity and the index z, are encrypted with KS_{M-PG_K}.
3-5: The encrypted message is sent in sub-steps 3-4 to **PG** with k.

Step 4: Using the private network of the banking institution, the Payment Gateway (**PG**) performs the following sub-steps to verify the payment.
4-1: The *VCRequest* is decrypted to retrieve *VSRequest* and the others fields.
4-2: The timeliness of *VCRequest* is verified. If the check is successful, the following steps are executed.
4-3: The *VSRequest* and other important, such as: $h(OI)$, *TID*, ID_M, *Price*, z and $h(KS_{M-PG_{k+1}})$ are forwarded to the issuer (**I**) where it is decided whether to approve or reject the transaction.
4-4: ID_M and the requested price *Price* are sent to confirm to the Acquirer **A** that the client is the party to whom the requested amount *Price* will be transferred to.
4-5: The approved result (*Stt*) and Value-subtraction Response (called *VSResponse* and encrypted with KS_{C-I_z}) are received from the issuer **I**. It is worth noting that the *VSResponse* is prepared by the issuer after (a) checking the timeliness of *VSRequest* and the validity of the client's account, and (b) after transferring the total amount of *OI* to the merchant's account.

Step 5: The Payment Gateway **PG** generates the *Value-Claim Response* (called *VCResponse*) in the following sub-steps.
5-1: A *VCResponse* is created that includes *Stt*, *VSResponse* (which will be forwarded to the client **C**) and $h(Stt, h(OI), h(K_{C-I_z}))$.
5-2: The *VCResponse* is encrypted with $KS_{M-PG_{k+1}}$ and sent to **M**.

Step 6: Merchant **M** performs the following sub-steps.
6-1: The *VCResponse* is decrypted to retrieve *VSResponse* and other fields.
6-2: The merchant's own *OI* is compared with the received *h(OI)*. If they do not match, then the client performs Sub-step 6-3a, otherwise the client performs Sub-step 6-3b.
6-3a: A message is sent to the Payment Gateway to notify it of the response failure. The Payment Gateway then starts a recovery procedure or resends the message.
6-3b: The *Payment Response* (called *PResponse* and represents the result of the client's request) is created and includes the *VSResponse*.
6-4: The *PResponse* is encrypted with $KS_{C-M_{i+1}}$ and sent to **C**.

3.5 Security analysis

We analyze the security of our proposed secure Payment Protocol in this section. In the case of the KCMS-VAN protocol, the client uses a nickname NID_C (a temporary

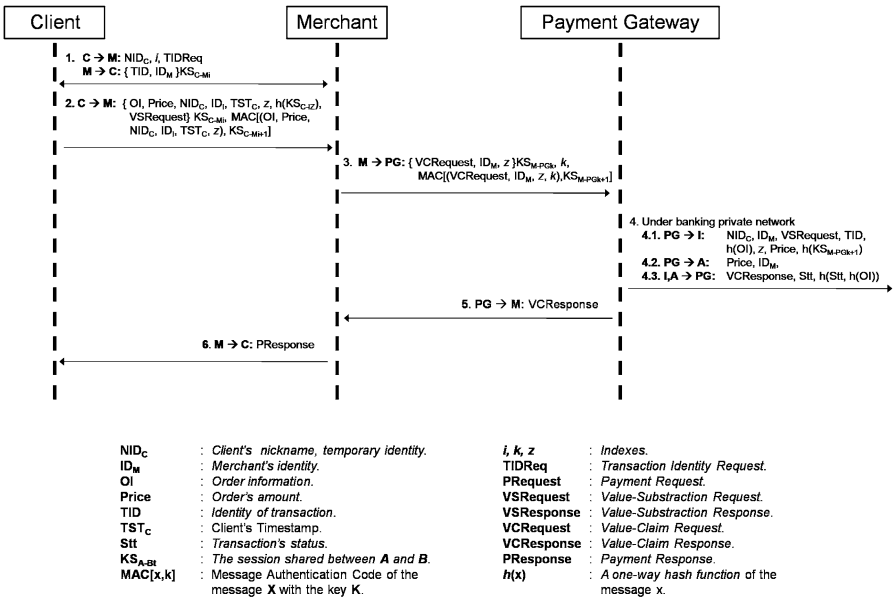

Fig. 4 KCMS-VAN payment protocol message exchanges

identity known only to the client and the issuer) instead of his/her real identity. As a result neither the merchant nor the payment gateway can map the nickname to the client's true identity. This anonymity protects relevant information from third parties but not unrestrained anonymity [2].

The Confidentiality of messages transmitted in each transaction should be protected while in transit, specially in wireless networks where anybody can easily eavesdrop the transmitted messages. In our protocol, we protect those confidential messages by employing symmetric cryptography which uses a secret shared between two parties (called sender and receiver) that wish to communicate safely without revealing details of the message. Moreover, the encryption key allows receiver and sender to authenticate each other. We also use the Message Authentication Code (MAC) to maintain the integrity of important messages. The proposed scheme provides integrity and identity authentication.

Although, generally, in any transaction a party should not trust others unless they can provide a proof of trustworthiness [15], in our protocol we assume the trust relationship between the client and the issuer because the client has a credit- and/or debit-card issued by the issuer who will not reveal it to any other party.

Since KS_{C-I_Z} could be generated only by the client or issuer but not by merchant, she/he is able to provide a non-repudiable evidence to prove to other parties that client has sent a message or requested merchant to perform transaction. Thus, Non-repudiation of transaction is ensured by KS_{C-I_Z} in the proposed protocol.

In the following paragraphs, we adopt the realistic analysis method proposed by [3] to discuss two possible attacks against our proposed protocol:

– **Replaying attack:** If an intruder *E* wants to impersonate a legal user by replaying the user's transmitting contents, the timestamp included in the transmitted message ensure the freshness of the message and avoids replaying attacks. Thus, our proposed protocol is secure against replaying attack.
– **Key guessing attack:** It is very difficult for any intruder *E* to get information related to the secret key through analyzing intercepted data because the authentication key is dynamic (exploiting, on each transaction, different session keys based on the same master secrets). Then, our proposed protocol is secure against the key guessing attack.

4 System design and implementation of KCMS-VAN protocol

4.1 Environment settings

The various applications (client, merchant, payment gateway and issuer) in our KCMS-VAN implementation have been developed in Java [34], using JavaME [35] for the implementation of the client in mobile devices. Since the JaveME Mobile Information Device Profile (MIDP) version 2.0 does not have the necessary security support, we have used security APIs from [33], a light-weight API suitable for use in any environment (including the newly released JavaME).

In Table 2 we show the hardware configuration of the devices used to run and benchmark all the applications.

4.2 Chosen cryptographic operations

Security and *computational* requirements are two important aspects which should be considered in order to choose suitable algorithms for encryption and hash functions

Table 2 Hardware specifications of system used for performance evaluation

Protocol Party	Device	Features
Client	Nokia™ N95	- 332 MHz Texas Instruments OMAP 2420 (ARM11-based). - 160 MB of RAM. - Symbian OS 9.2, S60 rel. 3.1.
	Palm™ T|X	- Intel 312 MHZ ARM-based processor. - 100 MB of RAM. - Palm OS® Garnet 5.4. - IBM's Java Virtual Machine.
Merchant Payment Gateway Issuer Acquirer	Sony™ Vaio VGN-SZ450N	- Intel Core 2 Duo (2 GHz). - 2 GB of RAM. - Windows Vista Business. - Sun's Java Virtual Machine.

in our application. From past results discussed in [23] and [22], computational requirement is directly related to energy consumption of operating devices in that the higher computation the algorithm requires, the higher energy it consumes.

We present and discuss the cryptographic algorithms we used for our implementation bellow:

– **Symmetric-key algorithm:** The results presented in [23] and [22] shows that the *Advanced Encryption Standard* algorithm (called *AES* [20, 25]) requires less energy consumption and computation compared to the *Triple Data Encryption Standard* algorithm (called *3DES* [18, 19]). Therefore, AES is more suitable to operate on power-constrained mobile devices. But, comparing both algorithms in terms of security, 3DES with 112-bit key length provides the equivalent security of RSA public-key algorithm with 2,048-bit key length, while AES with 128-bit key length provides the security equivalent to RSA public-key algorithm with 3,072-bit key length [6].

  We deployed the AES algorithm (with 128-bit key) as the symmetric-key encryption algorithm for our implementation because it provides higher security, faster operation, and lower energy consumption compared to 3DES.

– **Hash function:** In our implementation, we chose MD5 algorithm [18] because it requires less computation and consumes less energy than *Secure Hash Algorithm version 1* (Called *SHA-1* [18]) as shown in citeravi2002 and [22]. Moreover, MD5 can produce the same length of output to the length of an AES key (128 bits).

– **Keyed-hash function:** The *Hashed Message Authentication Code with Message Digest 5* algorithm (called *HMAC-MD5* [14]) is used in our application to perform keyed-hash operation because the key length of HMAC-MD5 is equivalent to the length of each session key. Moreover, HMAC-MD5 is considered a secure keyed-hash algorithm available for wireless networks since no attacks have emerged on HMAC even when it is implemented with hash functions that are not weak collision-resistance [4].

4.3 The KCMS-VAN software at the client side

The KCMS-VAN protocol requires a software (henceforth referred to as the KCMS-VAN Wallet) at the client's side for purchase transactions. The client can obtain the above application by connecting to the issuer's web site to download it or sending a request to the issuer to receive it by mail. After the client has downloaded or received the KCMS-VAN wallet, she/he should install it on her/his mobile device. Note that during the installation process, the client is requested to set up her/his own password (henceforth referred to as *KWP* password) to protects unauthorized users to: a) open the program, b) authorize the payment transaction, or c) access client's information and key file. Figure 5 illustrates snapshots of the main screen and main menu of KCMS-VAN wallet on both Nokia™ N95 (top) and Palm™ T|X devices (bottom). The left screen shows the main screen of KCMS-VAN wallet whereas the right screen shows the main menu.

The KCMS-VAN wallet provides the following functionalities:

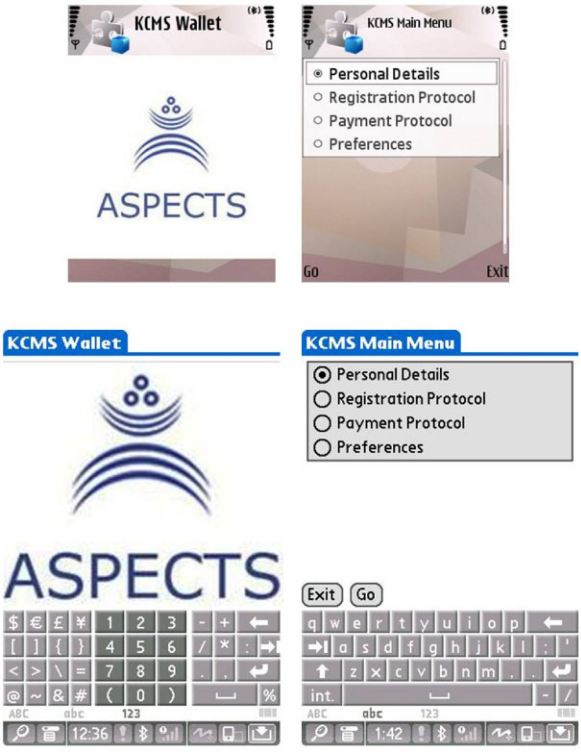

Fig. 5 Snapshots of main screen and main menu of the KCMS-VAN wallet on both Nokia N95 and Palm T|X devices

– **Personal Details:** The Client's personal information is used in both *Merchant Registration* and the *Payment Fulfillment*. To prevent the client from being prompted for her/his information in both phases, the KCMS-VAN wallet allows client to store in a file (protected by the KCMS Wallet Password -*KWP*-) his/her personal information (such as name, contact information, issuer ID and credit-and/or debit-card information) for further uses. To achieve that, the KCMS-VAN wallet prompts the user to enter the above information and store it at the client's mobile device. Figure 6 illustrates the KCMS-VAN wallet prompting for the client to enter her/his details on both Nokia™ N95 and Palm™ T|X devices.
– **Key Generation:** To generate a new session key from the master key, the KCMS-VAN wallet call *generateItemofSet()* with two values: a master key and a random number *j*. Upon receiving the master key, its BigInteger representation is created and the number of zeros according to the value of *j* is added to the right to the master key, using the *shiftleft()* to perform the left-cyclic. Then, the new value will be put through a MD5 has function to produce a new 128-bit session key. The java code for *generateItemofSet()* is shown in Listing 1.

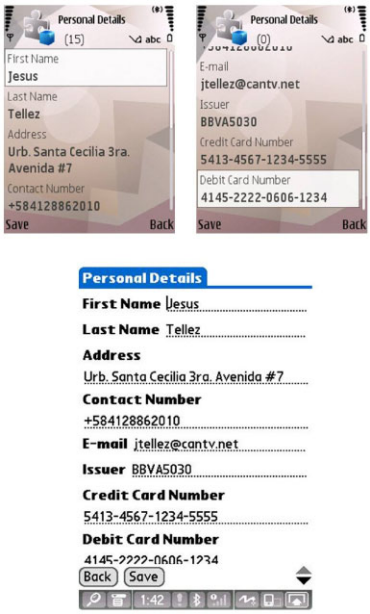

Fig. 6 Snapshots of KCMS wallet personal details form on both Nokia N95 (*top*) and Palm TlX (*bottom*) devices

```
public String generateItemofSet(byte[] secret,int index)
   {
       String key_Base = new String(secret);
       String cadr = "", resul = "";
       BigInteger cadb;
       long valcar;
       char[] characters = key_Base.toCharArray();

       for (int i = 0;i < characters.length; i++)
          {
             valcar = characters[i];
             cadr   = cadr + Long.toString(valcar, 2);
          }

       cadb = new BigInteger(cadr, 2);
       cadb = cadb.shiftLeft(index);
       cadr = cadb.toString(16);
       resul = new String(this.hashfunc(cadb.toString(16),"MD5"));
       return (resul);
   }
```

Listing 1 Java code for *generateItemofSet()*.

The above key generation technique can be directly applied to generate the sets of session keys KS_{C-M_i} and KS_{M-PG_k} from the key KS_{C-M} and KS_{M-PG}, respectively. To generate the set of session keys KS_{C-I_z}, the credit- and/or debit-card information (CDCI) is treated as the BigInteger and added to the value of KS_{C-I} before doing the *shiftleft()* operation. Figure 7 shows an example of the proposed session key generation technique.

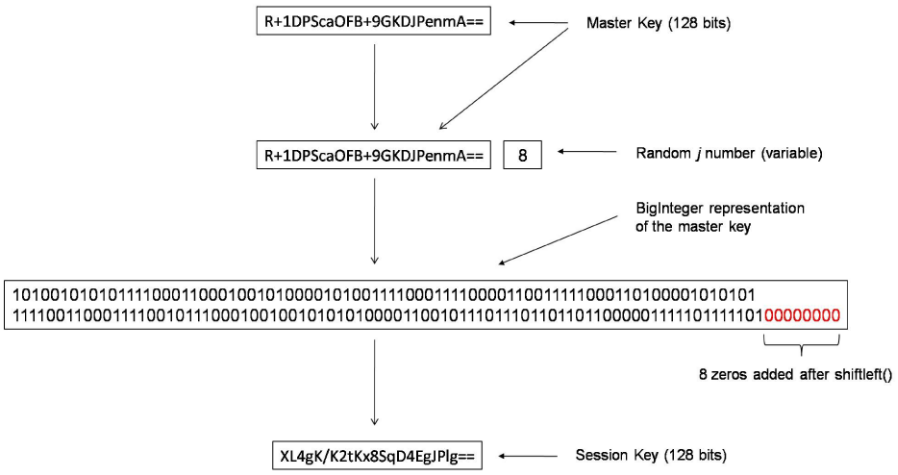

Fig. 7 Example of the KCMS wallet key generation technique

```
public String generateAESkey(int skey)
    {
        byte[] AESkey = this.generateBytes(skey);

        return (new String(Base64.encode(AESkey)));
    }
```

Listing 2 Java code for *generateAESKey()*.

– **Merchant Registration:** In order to make payments to a merchant, the client has to run the *Merchant Registration Protocol* to register with the merchant to share the master key KS_{C-M}. First, the client is prompted to introduce the *KWP* password to retrieve her/his personal information (such as name, address, contact number and email address) stored on the client's device. The KCMS-VAN wallet then generates the secret KS_{C-M} and the session key by using *generateAESKey()* [13]. The java code used for *generateAESKey()* is presented in Listing 2.

After the keys have been generated, the KCMS-VAN wallet encrypts the client's personal information, the secret KS_{C-M}, a nonce and a request for the merchant's identity. All these data are encrypted with the session key w using *AESEnc()*, before being sent to the merchant to register the client. The format of the client registration message is shown in Table 3 whereas the java code of *AESEnc()* is shown in Listing 3.

Upon receipt of the message, the merchant decrypts it using the *AESDec()* operation (see the java code in Listing 4) to retrieve the client's information including the secret KS_{C-M}. The merchant then encrypts, with the session key w and using *AESEnc()*, her/his identity and $h(n, NID_C, ID_M, KS_{C-M})$. The encrypted message is sent to the client to confirm the registration.

Table 3 Client registration message format

Fields	Size (bits)
Client Data	Variable (Max. 1064)
KS_{C-M}	128
Nonce	128
MIDReq	48

```
public String AESEnc(String mkey,String StToEnc)
   {
        String resul = null;

        byte[] key    = Base64.decode(mkey.getBytes());
        byte[] input = StToEnc.getBytes();
        BlockCipher engineAES          = new AESLightEngine();
        BufferedBlockCipher cipherAES =
              new PaddedBlockCipher(new CBCBlockCipher(engineAES));
        cipherAES.init(true, new KeyParameter(key));
        byte[] cipherText =
              new byte[cipherAES.getOutputSize(input.length)];

        int outputLen =
         cipherAES.processBytes(input,0,input.length,cipherText,0);

        try
           {
               cipherAES.doFinal(cipherText, outputLen);
               resul = new String(Base64.encode(cipherText));
           }
        catch (CryptoException ce)
           {
               resul = "";
           }
        return(resul);
   }
```

Listing 3 Java code for *AESEnc()*.

At the client side, the client retrieves the confirmation of her/his registration by decrypting the message using *AESDec()*. Then, the KCMS-VAN wallet stores the secret KS_{C-M} in a key file protected with the *KWP* password. Once the registration is done, the merchant stores on her/his device the client's information together with the secret KS_{C-M} and the key KS_{C-M_i} (generated from KS_{C-M}).

Figure 8 illustrates the KCMS-VAN wallet during the *Merchant Registration* phase on both Nokia™ N95 (top) and Palm™ T|X devices (bottom). On the left screen, the KCMS wallet shows the client's details whereas the right screen shows the successful registration.

– **Payment Execution:** To make a payment to the merchant, the client is first prompted to enter the Order Description, the Product ID, the Price and the Type of Card to use (*Debit* or *Credit*). After filling the information, the KCMS-VAN wallet application generates the keys KS_{C-M_i} and KS_{C-I_z} based on the random numbers i and z, respectively. Then, the client sends his/her nickname, the index i, and the transaction ID request (**TIDReq**). Upon receipt of the message, the merchant generates KS_{C-M_i} (based on the i value) and sends his/her identity ID_M) and the

```
public String AESDec(String mkey,String StToDec)
   {
       String resul = null;

       byte[] key   = Base64.decode(mkey.getBytes());
       byte[] input = Base64.decode(StToDec.getBytes());
       BlockCipher engineAES          = new AESLightEngine();
       BufferedBlockCipher cipherAES = new
              PaddedBlockCipher(new CBCBlockCipher(engineAES));
       cipherAES.init(false, new KeyParameter(key));
       byte[] cipherText =
              new byte[cipherAES.getOutputSize(input.length)];

       int outputLen =
       cipherAES.processBytes(input,0,input.length,cipherText,0);

       try
          {
              cipherAES.doFinal(cipherText, outputLen);
              resul = new String(cipherText);
          }
       catch (CryptoException ce)
          {
              resul = "";
          }
       return(resul);
   }
```

Listing 4 Java code for *AESDec()*.

```
public String HmacMD5(String kyeenc, String st)
   {
       String resul = "";
       HMac hmac;

       hmac             = new HMac(new MD5Digest());
       byte[] macValue = new byte[hmac.getMacSize()];
       byte[] stbyt     = st.getBytes();
       hmac.init(new KeyParameter(Base64.decode(kyeenc)));
       hmac.update(stbyt, 0,stbyt.length);
       hmac.doFinal(macValue, 0);
       hmac.reset();
       resul = new String(Base64.encode(macValue));

       return resul;
   }
```

Listing 5 Java code for *HmacMD5()*.

transaction ID (*TID*) to the client, encrypted with the session key KS_{C-M_i} using *AESEnc()*.

The client then creates the *Value-Subtraction Request* (**VSRequest**) to be sent to the issuer. An authenticated hash of this message is computed using HMAC-MD5 algorithm with the key KS_{C-I_z} by using the *HmacMD5()*. Note that the merchant will not able to decrypt or create the request since the merchant does not have the key KS_{C-I} which is known only by the client and the issuer. The java code used for the *HmacMD5()* is presented Listing 5.

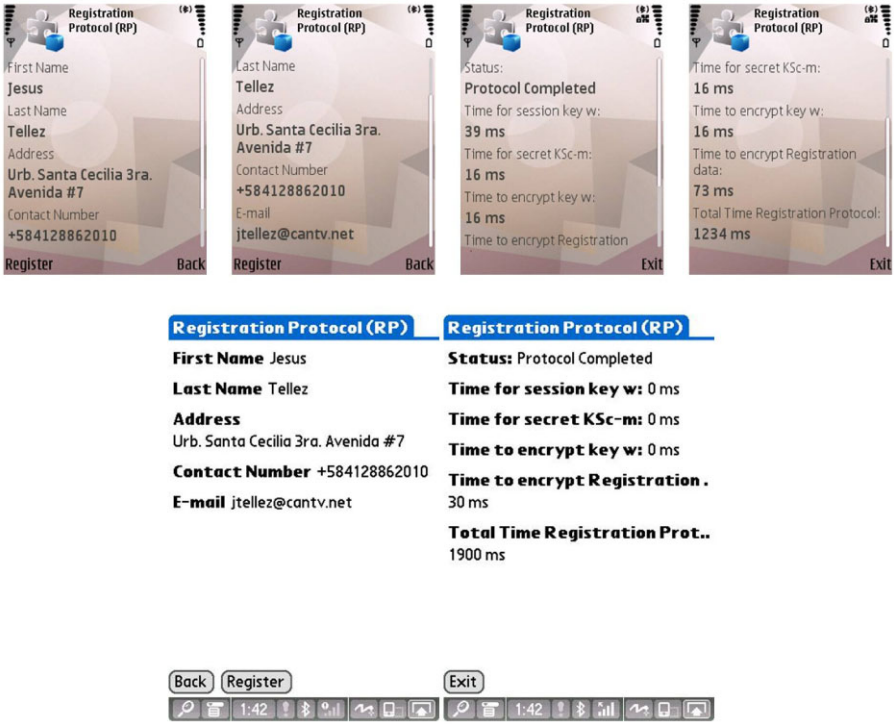

Fig. 8 Screenshots of KCMS-VAN wallet during the execution of the merchant registration protocol phase on both Nokia™ N95 and ™ TlX devices

Once the **VSRequest** has been created, the client prepares the Payment Ordering request (called **PRequest**) to be sent to the merchant. This message includes **VSRequest**, the order information (**OI**), price, client nickname ($\mathbf{NID}_C$) and issuer identity ($\mathbf{ID}_I$). The encryption of *PRequest* with KS_{C-M_i} using *AESEnc()* ensures its confidentiality.

Upon receipt of the request from the client, the merchant (who has the KS_{C-M}) decrypts the message using *AESDec()*. The merchant then combines the *VSRequest* received from the client and the amount payable by the client with the necessary information to create the *VCRequest*. This message is encrypted with KS_{M-PG_k} before being sent to the Payment Gateway.

The Payment Gateway (PG) decrypts the message received using *AESDec()*. Then, PG sends the *VSRequest* to the issuer and the *VCRequest* to the acquirer. Upon receipt of the approval response, the PG prepares *VCResponse* (which includes *VSResponse* encrypted with KS_{C-I_z}) and encrypts it with KS_{M-PG_k} using *AESEnc()*. The Payment Gateway then sends *VCResponse* to the merchant which in turn sends it to the client, encrypting $KS_{C-M_{i+1}}$ using *AESEnc()*. The message sent from the merchant to the client represents the Payment Ordering Response (*PResponse*). Note that although the flow information between the payment gateway, the issuer and the acquirer (step 4 of KCMS-VAN protocol) exists within a private banking network (beyond the scope of KCMS-VAN Protocol). We im-

Table 4 Format of payment primitive transactions: *PRequest*, *VSRequest*, *VCRequest*, *VCResponse* and *VSResponse*

Primitive	Fields	Size (bits)
Payment Request (*PRequest*)	*OI*	Variable
	Price	
	Client Data	
	Issuer Data	
	Timestamp	
	hmac(*VSRequest*)	128
Value-Subtraction Request (*VSRequest*)	Price	Variable
	Merchant Data	
	$h(OI)$	128
Value-Subtraction Response (*VSResponse*)	$h(OI)$	128
	Response (Yes/No)	Variable
Value-Claim Request (*VCRequest*)	VSRquest	Variable
	Price	
Value-Claim Response (*VCResponse*)	$h(OI)$	128
	Response (Yes/No)	Variable

plement the issuer and acquirer to have a better idea of the performance of the whole payment system even if it could be assumed that all transactions relevant to acquirer and issuer are successful within a limited time.

When the client receives the message, it retrieves the Payment Ordering Response using *AESDec()* to decrypt the message. The format of PRequest, VSRequest, VSResponse, VCRequest and VCResponse are shown in Table 4.

Figure 9 illustrates the KCMS-VAN wallet during the *Payment* phase on both Nokia™ N95 (top) and Palm™ T|X devices (bottom). On the left screen, the KCMS-VAN wallet shows the Order description whereas the right screen shows the successful payment.

5 Performance evaluation

5.1 Discussions of empirical results

5.1.1 Execution time of merchant registration protocol

In this section, we report on the empirical results obtained for our implementation of the KCMS-VAN protocol. In particular, we focus on the time taken to perform various parts of the KCMS-VAN protocol and the overall time taken to complete a payment transaction. The results were collected by performing 10 executions with different sets of data and the timings taken were done using the *getTime* method in the **Date** class of J2SE.

The average times required by the client and the merchant to perform the Merchant Registration Protocol are shown in Tables 5 and 6, respectively. Note that the time taken by the client to input the data has not been included into the time taken by

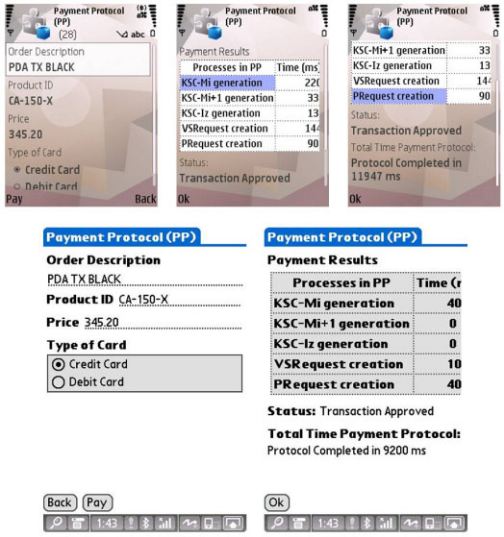

Fig. 9 Screenshots of the payment phase on both Nokia™ N95 and Palm™ T|X devices

Table 5 Time (ms) Taken at Client to perform the *Merchant Registration Protocol* (**TCRP**)

Processes	Device	
	Nokia N95	Palm T|X
AESGen$_w$	39.90	1.00
AESGen KS_{c-m}	28.80	1.00
AESEnc Data	81.90	5.10
AESDec Data	50.10	2.00
TCRP	1118.50	727.10

Table 6 Time (ms) Taken at Merchant to perform the *Merchant Registration Protocol* (**TMRP**)

Processes	Client's Device	
	Nokia N95	Palm T|X
AESEnc Data	< 1	< 1
AESDec Data	6.40	5.90
TMRP	34.70	43.40

the application to perform the *Merchant Registration Protocol*. Moreover, when we tested our implementation by entering the maximum number of characters allowed into each individual field, we found that the maximum times taken by the client (1196 milliseconds for Nokia N95 and 740 milliseconds for the Palm T|X) and the merchant (35 milliseconds for Nokia N95 and 76 milliseconds for the Palm T|X) to perform the *Merchant Registration Protocol* are not much different from the average times calculated in Table 5 and Table 6.

Table 7 Time (ms) taken at client (Nokia N95), merchant, payment gateway, and issuer on performing *Payment Protocol* (**TPP**)

Processes	Nokia N95	Merchant	PG	Issuer
KS_{C-M_i} generation	253.90	6.70	–	–
$KS_{C-M_{i+1}}$ generation	29.90	< 1	–	–
KS_{C-I_z} generation	15.00	–	–	< 1
KS_{M-PG_k} generation	–	–	6.80	–
VSRequest creation	145.30	–	–	–
PRequest creation	522.00	–	–	–
PRequest decryption	880.00	–	–	–
VCRequest creation	2.10	–	–	–
VCRequest decryption	–	–	< 1	–
VSResponse creation	–	–	–	4.80
TPP	6842.20	2742.10	322.50	12.80

Table 8 Time (ms) taken at client (Palm T|X), merchant, payment gateway, and issuer on performing *Payment Protocol* (**TPP**)

Processes	Palm T|X	Merchant	PG	Issuer
KS_{C-M_i} generation	66.00	5.70	–	–
$KS_{C-M_{i+1}}$ generation	–	< 1	–	–
KS_{C-I_z} generation	–	–	–	< 1
KS_{M-PG_k} generation	–	–	6.00	–
VSRequest creation	15.00	–	–	–
PRequest creation	653.00	–	–	–
PRequest decryption	1145.00	–	–	–
VCRequest creation	2.10	2.40	–	–
VCRequest decryption	–	–	< 1	–
VSResponse creation	–	–	–	4.70
TPP	5657.00	2461.90	310.80	12.80

5.1.2 Execution time of payment protocol

The average total time that the client has spent on performing the first transaction with the merchant (both the Merchant Registration Protocol and the Payment Protocol) was 7.96 seconds ($1.12 + 6.84 = 7.96$ seconds) using the Nokia N95 device compared to 6.39 seconds ($0.73 + 5.66 = 6.39$ seconds) when the palm t|x device was used. In next payment transactions, the total average time to complete each transaction will reduce to only 6.84 seconds on Nokia N95 device and 5.66 seconds on palm t|x device because the client does not have to execute the Merchant Registration Protocol. This minimal amount of time to complete a transaction reveals the potential of KCMS-VAN protocol to execute payment transactions in wireless environments with minimal delays.

Table 9 KCMS-VAN wallet software size and its memory usage

Device	Internal memory	Application size	% Memory used
Nokia™ N95 mobile phone	163840 kb	68 kb	0.042 %
Palm™ T|X handheld	131072 kb	132 kb	0.100 %

5.1.3 Application size

as mobile devices have limited memory space, application size is an important issue which should be considered when developing applications for such devices. We focus our proposed payment protocol on the client's side because it is the only party in our scheme that uses a mobile device that interacts with the system.

Table 9 shows the KCMS-VAN wallet software size and its memory usage in the mobile devices used during our performance evaluation measurements.

6 Conclusions and further work

A protocol for secure on-line payments in a vehicle-to-roadside Restricted Scenario in VANETs (where the client cannot directly communicate with the issuer) was proposed in this research. Our protocol employs symmetric cryptographic techniques which allow clients to make purchases without disclosing private information and using a short range link (such as that provided by Bluetooth or Wi-Fi) to communicate with the merchant. Moreover, the chosen cryptography scheme has low computation requirements at both parties (since no public-key operation is required).

The empirical results from the performance evaluation results on the implementation have proven that payment transactions over a wireless network can be conducted by our proposed KCMS-VAN protocol. The chosen lightweight, secure cryptographic algorithms are suitable for our VANET application. Deploying such algorithms in KCMS-VAN results in the reduction of messages exchanged and computation costs at the client's mobile device. As we have seen from the implementation, a payment transaction by KCMS-VAN can be completed within average of 6.84 second using a Nokia N95 device and 5.66 second using palm T|X device.

The Client's KCMS-VAN wallet has a small size which makes it very suitable for memory-constrained mobile devices. The KCMS-VAN requires only 68 Kilobytes to be stored on a Nokia mobile phone and 123 kilobytes on a Palm Pilot 123 kb to be stored on a Palm Pilot.

In the future, we will explore the possibility to accommodate other cryptographic algorithms (such as elliptic-curve cryptography, digital signature scheme with message recovery using self-certified public keys) to our system. We also plain to reformulate and implement the proposed protocol into other restrictive connectivity scenarios in VANETs (such as a scenario where the merchant cannot communicate directly with the acquirer).

Acknowledgements We thank the anonymous reviewers for their useful comments that helped us to improve the quality and presentation of this paper. Sherali Zeadally was supported by an NSF award (No. 0911969) during this work whilst Jesús Isaac Téllez and José Cámara Sierra were supported in part by I-ASPECTS- Project (TIN2007-66107), but all ideas expressed represent the view of the authors.

References

1. Abad Peiro, J. L., Asokan, N., Steiner, M., & Waidner, M. (1997). Designing a generic payment service. *IBM Systems Journal*, *37*(1), 72–88.
2. Asokan, N. (1994). Anonymity in Mobile computing environment. In *Workshop on mobile computing systems and applications* (pp. 200–2004).
3. Bella, G., & Bistarelli, S. (2005). Information assurance for security protocols. *Computers and Security*, *24*(4), 322–333.
4. Bellare, M. (2006). New proofs for NMAC and HMAC: security without collision-resistance. In *The 26th annual international cryptology conference (Crypto 2006)* (pp. 602–619).
5. Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Van Herreweghen, Els., & Waidner, M. (2000). Design, implementation and deployment of the *i*KP secure electronic payment system. *IEEE Journal on Selected Areas in Communication*, *18*(4), 611–627.
6. Certicom (2003). The next generation of cryptography. *Code and Cipher*: *Certicom's Bulletin of Security and Cryptography*, 1 (1).
7. Car2Car Communication Consortium (2007). *Overview of the C2C-CC system* (Technical Report version 1.0). Car2Car Communication Consortium.
8. Chari, S., Kermani, P., Smith, S., & Tassiulas, L. (2001). Security issues in M-commerce: a usage-based taxonomy. In *E-commerce agents* (pp. 264–282).
9. Hassinen, M., Hyppönen, K., & Haatajam, K. (2006). An open, PKI-based mobile payment system. In *Emerging trends in information and communication security, international conference (ETRICS'2006)* (pp. 86–100).
10. Hu, Z., Liu, Y., Hu, X., & Li, J. (2004). Anonymous micropayments authentication (AMA) in mobile data network. In *23rd Annual joint conference of the IEEE computer and communications societies (IEEE INFOCOM)* (pp. 46–53).
11. Hwang, R., Su, F., & Huang, L. (2007). Fast firmware implementation of RSA-like security protocol for mobile devices. *Wireless Personal Communications*, *42*(2), 213–223.
12. J. Hall, J., Kilbank, S., Barbeu, M., & Kranakis, E. (2001). WPP: a secure payment protocol for supporting credit-and debit-card transactions over wireless networks. In *International conference on telecommunications (ICT 2001)*.
13. Juntao, M. (2003). *Enterprise J2ME: developing mobile Java applications*. New York: Prentice Hall PTR.
14. Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: keyed-hashing for message authentication, RFC 2104.
15. Kungpisdan, S., Srinivasan, B., & Dunf Le, P. (2004). A secure account-based mobile payment protocol. In *International conference on information technology: coding and computing (ITCC'04)* (pp. 35–39).
16. Kungpisdan, S., Srinivasan, B., & Dung Le, P. (2003). Lightweight mobile credit-card payment protocol. In *4th International conference on cryptology in India (progress in cryptology—INDOCRYPT 2003)* (pp. 295–308).
17. Lei, Y., Chen, D., & Jiang, Z. (2004). Generating digital signatures on mobile devices. In *18th International conference on advanced information networking and applications (AINA'04)* (pp. 532–535).
18. Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1997). *Handbook of applied cryptography*. Boca Raton: CRC Press.
19. NIST (1999). FIPS PUB 46-3 Data Encryption Standard (DES). http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.
20. NIST (2001). FIPS PUB 197 Advance Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
21. Papadimitratos, P., Kung, A., Hubaux, J.-P., & Kargl, F. (2006). Privacy and identity management for vehicular communication systems: a position paper. In *Workshop on standards for privacy in user-centric identity management*.
22. Potlapally, N., Ravi, S., Raghunathan, A., & Jha, N. (2003). Analyzing the energy consumption of security protocols. In *2003 International symposium on low power electronics and design (ISLPED'03)* (pp. 30–35).
23. Ravi, S., Raghunalhan, A., & Potlapally, N. (2002). Securing wireless data: system architecture challenges. In *15th International symposium on system synthesis* (pp. 195–200).
24. Raya, M., & Hubaux, J.-P. (2005). The security of vehicular ad hoc networks. In *3rd ACM workshop on security of ad hoc and sensor networks (SASN'05)* (pp. 11–21).

25. Sanchez-Avila, C., & Sanchez-Reillol, R. (2001). The Rijndael block cipher (AES proposal): a comparison with DES. In *35th IEEE international Carnahan conference on security technology* (pp. 229–234).
26. Sukuvaara, T., & Pomalaza-RÃ¡ez, C. (2009). Vehicular networking pilot system for vehicle-to-infrastructure and vehicle-to-vehicle communications. *International Journal of Communication Networks and Information Security*, *1*(3), 1–10.
27. Téllez, J., & Sierra, J. (2007). A secure payment protocol for restricted connectivity scenarios in M-commerce. In *EC-Web* (pp. 1–10).
28. Téllez, J., & Sierra, J. (2007). Anonymous payment in a client centric model for digital ecosystems. In *Inaugural IEEE international conference on digital ecosystems and technologies (IEEE-DEST 2007)* (pp. 422–427).
29. Téllez, J., & Sierra, J. (2007). An anonymous account-based mobile payment protocol for a restricted connectivity scenario. In *DEXA workshops* (pp. 688–692).
30. Téllez, J., Sierra, J., Izquierdo, A., & Carbonell, M. (2006). Payment in a kiosk centric model with mobile and low computational power devices. In *Computational science and its applications (ICCSA 2006)* (Part V, pp. 798–807).
31. Téllez, J., Sierra, J., Izquierdo, A., & Torres, J. (2006). Anonymous payment in a Kiosk centric model using digital signature scheme with message recovery and low computational power devices. *Journal of Theoretical and Applied Electronic Commerce Research*, *1*(2), 1–11.
32. Téllez, J., Sierra, J., Zeadally, S., & Torres, J. (2008). A secure vehicle-to-roadside communication payment protocol in vehicular ad hoc networks. *Computer Communications*, *31*(10), 2478–2484.
33. The Legion of the Bouncy Castle (2008). The Legion of the Bouncy Castle Java cryptography APIs version 1.4. http://www.bouncycastle.org/, 2008.
34. Sun Microsystem (2008). Java platform, micro edition (Java SE) v 1.6.0, API specification. http://java.sun.com/javase/index.jsp.
35. Sun Microsystem (2008). Java platform, micro edition (Java ME), API specification. http://java.sun.com/javame/index.jsp.
36. Wang, H., & Kranakis, E. (2003). Secure wireless payment protocol. In *International conference on wireless networks* (pp. 576–582).
37. Yousefi, S., Mousavi, M., & Fathy, M. (2006). Vehicular ad hoc networks (VANETs): challenges and perspectives. In *6th International conference on ITS telecommunications* (pp. 761–766).

Jesús Téllez Isaac is a System Engineer graduated at the Central Technological University (UNITEC), Venezuela, and he is a doctor candidate at the University Carlos III of Madrid in the Computer Science Department. He is an Associate Professor in the Computer Science Department of the University of Carabobo. He has served as a Technical Program Committee member for various international conferences. His research interests include Internet Security, performance evaluation of systems, mobile computing, mobile payment systems. *Jesús Téllez Isaac, Computer Science Department, FACYT, Universidad de Carabobo, Av. Universidad, Sector Bárbula, Valencia, Venezuela.*

Sherali Zeadally received his Bachelor's Degree in Computer Science from University of Cambridge, England, and the Doctoral Degree in Computer Science from University of Buckingham, England in 1996. He is an Associate Professor at the University of the District of Columbia. He currently serves on the Editorial Boards of over 15 international journals. He has been serving as a Co-Guest editor for over a dozen special issues of various peer-reviewed scholarly journals. He is a Fellow of the British Computer Society and a Fellow of the Institution of Engineering Technology, UK. His research interests include computer networks including wired and wireless networks, network and system security, mobile computing, ubiquitous computing, RFID, performance evaluation of systems and networks. *Sherali Zeadally, Department of Computer Science and Information Technology, University of the District of Columbia, Washington, DC, 20008, USA.*

José Cámara Sierra is an Associate Professor in the Computer Science Department of the University Carlos III of Madrid. He holds a Ph.D. in Computer Science and an M.Sc. in Business Administration. His research work focuses on Internet Security. He has participated in numerous research projects, and had published articles in various journals in the area of security. *José María Sierra Cámara Universidad Carlos III de Madrid, Department of Computer Science, Avda. de la Universidad, 30, Leganés (Madrid), 28911, España.*