A Secure Prepaid Wireless Micropayment Protocol

Supakorn Kungpisdan[1], Bala Srinivasan[2], and Phu Dung Le[3]

[1] School of Network Computing, Monash University
McMahons Road, Frankston, Victoria 3199, Australia

[2] School of Computer Science and Software Engineering, Monash University
900 Dandenong Road, Caulfield East, Victoria 3145, Australia

[3] School of Network Computing, Monash University
McMahons Road, Frankston, Victoria 3199, Australia

Abstract. In this paper, a secure prepaid micropayment protocol which is suitable for wireless networks is introduced. The proposed protocol employs a secure cryptographic technique that reduces all parties' computation and satisfies transaction security properties, including non-repudiation. This offers the ability to resolve disputes among parties. Compared to existing micropayment protocols, all parties' secret information are well-protected. Finally, we perform an analysis to demonstrate that the proposed protocol has better performance than existing micropayment protocols. As a result, the proposed protocol can be well-operated on limited capability wireless devices.
Keywords. Micropayment, mobile payment, mobile commerce, payment protocol, electronic commerce,

1 Introduction

Micropayments seem to be more widely accepted than other kinds of payments systems for wireless networks because of their lightweight, lower setup cost,and lower transaction cost. Moreover, most payment-related applications for wireless networks are conducted with small-valued goods or services e.g. downloading ring tones, operator logos, or electronic document.

Traditionally, micropayment protocols employ public-key operations and a chain of hash values such as PayWord [6] or NetCard [2]. Although these protocols work well for fixed networks, they are not suitable for applying to wireless networks due to a number of limitations of wireless environments [3, 7].

Recently, a prepaid micropayment protocol called PayFair [8] offers the ability to perform payment transactions on limited computational capability devices. It employs symmetric-key operations and keyed hash functions which reduce the computation at all engaging parties. However, PayFair lacks of transaction privacy since payment information of engaging parties is sent in cleartext during transactions. Moreover, a message sent from a client to a merchant in PayFair lacks of non-repudiation property. Furthermore, a bank is able to impersonate as its clients to perform transactions. In addition, a payment token authorized by the bank is merchant-specific in that it is still can be

Kungpisdan S., Srinivasan B. and Dung Le P. (2004).
A Secure Prepaid Wireless Micropayment Protocol.
In *Proceedings of the 2nd International Workshop on Security in Information Systems*, pages 104-113
DOI: 10.5220/0002662701040113

used to generate the coins to spend with only one specified merchant. Thus, the client is required to request the bank to issue a new payment token every time she wants to perform a payment transaction to a new merchant.

In this paper, we propose a prepaid micropayment protocol which employs a secure symmetric cryptographic technique that not only the computation at all parties, especially at the client, is reduced, but the proposed protocol also satisfies transaction security properties including non-repudiation [1]. Moreover, it offers the ability to resolve disputes among parties. Furthermore, all parties' private information such as payment information and secret keys are well-protected.

In any prepaid payment system, a client has to purchase an electronic coupon which contains spending credits and the amount paid by the client is transferred to a specified merchant before a transaction. In our proposed protocol, we present an efficient method to refund either un-spending credits or coupons. This offers the practicability to the system. Moreover, the coupon in our protocol is general-purposed in that it can be split into smaller value merchant-specific coupons to spend with many merchants.

We analyze the performance of the proposed protocol and compare with PayWord [6] and PayFair [8]. The results show that our protocol has better performance than others in terms of party's computation and the numbers of message passes. Therefore, the proposed protocol can be implemented in limited capability wireless devices with higher performance than existing micropayment protocols.

Section 2 provides overviews of PayWord and PayFair protocols. Section 3 introduces our proposed protocol. Section 4 discusses about security and performance of the proposed protocol. Section 5 concludes our work.

2 Overviews of Existing Micropayment Protocols

In this section, we outline two existing micropayment protocols: PayWord [6] and PayFair [8]. In section 2.1, PayWord is presented to provide an idea about how a micropayment protocol with public-key operations works. In section 2.2, PayFair is outlined to show how to secure transactions using symmetric-key operations.

2.1 PayWord

PayWord [6] is a postpaid micropayment protocol based on public-key cryptography. Three parties are involved in the system: *client*, *merchant*, and *bank*. The client and the merchant establish accounts with the bank. At the beginning of the protocol, the bank issues the client a *PayWord certificate* which contains authorized amount *CL* that the client is allowed to make a payment to each merchant. To make a payment to a merchant, the client generates a set of coins $c_0, ..., c_n$, where *n* = *CL*. The set of c_i is generated as follows: $c_i = h(c_{i+1})$, where $i = 1, ..., n-1$.

In the first payment, the client sends the merchant a *commitment*, which contains the PayWord certificate and c_0, digitally signed by the client. Later on, in each payment, the client sends the coin c_i to the merchant. The merchant can infer the value of the coin by applying a number of hash functions to c_i. At the end of the day, the merchant sends the highest value of c_i together with the commitment to the bank. The bank then

deducts the money from the client's account and transfers the money to the merchant's account.

However, PayWord is not suitable for applying to wireless environments because it has high client's computation due to public-key operations. Moreover, a certificate verification process leads to additional communication passes [3]. In addition, payment information, c_0 and c_i, is readable by any party who holds the client's public key. Thus, any party is able trace the client's spending.

2.2 PayFair

PayFair [8] is a prepaid micropayment protocol which employs symmetric-key operations and hash functions. The details of PayFair are shown as follows:

Phase A: Prepaid Phase

$$\mathbf{C} \rightarrow \mathbf{B}: \quad ID_C, O_C, h(O_C, K_C) \qquad (a)$$

$$\mathbf{B} \rightarrow \mathbf{C}: \quad \{\{N, RN\}_{SK}, RT\}_{K_C}, N, h(\{N, RN\}_{SK}, N, O_C, K_C) \qquad (b)$$

Where *SK* is the secret known only to the bank. K_C is shared between the client and the bank. The client requests the bank by sending order number O_C containing the requested amount. The bank returns the message containing a payment token $\{N, RN\}_{SK}$, which is later used to generate coins. *RN* is a random number generated from the serial number *N* and the secret SK_{RN} known only by the bank. The client generates a set of coins $w_i, i = 0, ..., n$, where $w_n = \{N, RN\}_{SK}$, from the process: $w_i = h(w_{i+1})$.

Phase B: Micropayment Phase

$$\mathbf{C} \rightarrow \mathbf{M}: \quad w_0, N, h(w_0, ID_M, K_C) \qquad (c)$$

$$\mathbf{M} \rightarrow \mathbf{B}: \quad w_0, N, ID_C, R_M, h(w_0, ID_M, K_C) \qquad (d)$$

$$\mathbf{B} \rightarrow \mathbf{M}: \quad w_0, ID_C, ID_M, YES, h(w_0, ID_C, K_M, R_M, YES) \qquad (e)$$

The client sends the message *(c)* containing w_0 to the merchant. The merchant then forwards $h(w_0, ID_M, K_C)$ with relevant information to the bank in *(d)*. After receiving the message, the bank can generate w_n from w_0, *N*, and its own *RN* and *SK*. It then transfers the amount *n* to the merchant's account and sends the response to the merchant in *(e)*. The client can start a payment transaction with the merchant as follows:

$$\mathbf{C} \rightarrow \mathbf{M}: \quad w_i \quad where \quad i = 1, ..., n \qquad (f)$$

However, in PayFair, the problem about revealing payment information occurred in PayWord still exists since, in the messages *(c)* and *(f)*, w_0 and w_i are sent in cleartext. In addition, although Yen [8] claimed that payment token w_n is general-purposed, it is still merchant-specific when used, that is, although the coins is merchant-independently generated, they are still can be used to pay only one specific merchant. Thus, the client needs to request the bank for a new payment token every time she wants to make a payment to a new merchant. Moreover, in *(c)*, the bank can impersonate as the client to perform transactions with the merchant.

3 The Proposed Protocol

3.1 Overview of the Proposed Protocol

There are three parties involved in our protocol: *client*, *merchant*, and *bank*. At the beginning of the protocol, a client requests a bank for an authorization to perform micropayment transactions. The bank checks the validity of the client's account and issue a *Bank Coupon* containing the amount requested by the client.

To make payment to a merchant, the client generates a *Merchant Coupon* containing the value specified to the merchant. The value of the merchant coupon must not exceed the value of the bank coupon. This coupon has to be validated by the bank. To validate the merchant coupon, the client generates a set of coins, attaches them into the merchant coupon, and sends to the bank. After the validation, the bank transfers the money with the requested value from the client's account to the merchant's account. The client then can make payments to the merchant up to the amount specified in the merchant coupon. In our protocol, a bank is trusted by its clients to generate correct numbers and values of coins for coin validation purpose, but it is not trusted to create payment initialization requests to merchants by itself. This is because the bank itself can generate the sets of coins. It is possible to generate fake requests on behalf of its clients.

Our proposed protocol is composed of 6 sub-protocols: *Setup, Payment Initialization, Payment, Extra Credit Request, Coupon Cancellation,* and *Coin Return* protocols. Section 3.2-3.7 demonstrate the details of the protocols.

3.2 Setup Protocol

A client **C** requests a bank **B** for an authorization on making a micropayment transaction with the amount CL_T as follows:

$$\mathbf{C} \rightarrow \mathbf{B}: \quad ID_C, CL_T, T_{CP}, h(CL_T, T_{CP}, Y) \tag{1}$$

Note that CL_T stands for total credits that the client is allowed to spend in the system. T_{CP} is the timestamp when generating the request. $h(CL_T, T_{CP}, Y)$ is used to protect the integrity of the message. The bank checks the validity of the client's account and then deducts the amount CL_T from the client's account. Bank then sends the client a *Bank Coupon* that can be used to perform transactions as follows:

$$\mathbf{B} \rightarrow \mathbf{C}: \quad \{CL_T, T_T, T_{CP}, SN, c\}_Y \tag{2}$$

The bank coupon has unique serial number *SN* assigned by the bank and contains authorized credits CL_T. T_T stands for timestamp when issuing CL_T, and *c* is a random number generated by the bank used for generating coins. With this bank coupon, the client can make payments to many merchants repeatedly up to CL_T. After running out of the credits, the client needs to run this protocol to request the bank for a new CL_T again.

3.3 Payment Initialization Protocol

To make a payment to a merchant **M**, the client generates a set of coins $c_i, i = 0, ..., n$, where $n = CL_T$, as follows:

$$c_n = \{c, T_G\}$$
$$c_i = h(c_{i+1}) \quad where \quad i = 0, ..., n-1$$

The client specifies the amount CL_M to spend with the merchant. The client attaches the coins and CL_M into a *Merchant Coupon*, and sends it to the bank:

$$\mathbf{C} \rightarrow \mathbf{B}: \quad h(c_0, T_G, CL_M, X), h(ID_M, c_0, T_G, CL_M, CL_T, T_T, SN, Y), T_G \qquad (3)$$

Where T_G stands for the timestamp when generating a set of coins $c_0, ..., c_n$. Note that the client can either spend the whole credits to only one merchant or spend some credits to a merchant and spend the rest to other merchants. We can see that *h(c_0, T_G, CL_M, X)* is the payment request from the client to the merchant which is unreadable by the bank. The bank retrieves CL_T and CL_M from h($ID_M, c_0, T_G, CL_M, CL_T, T_T$, *SN, Y)* and checks whether $CL_T < CL_M$. If so, it rejects the request. If $CL_T > CL_M$, the bank calculates the client's remaining credits CL_{TR}, where $CL_{TR} = CL_T - CL_M$. It then maintains the table of CL_{TR} to prevent over-spending problem. At this stage, the bank transfers CL_M to the merchant's account. Then the bank sends the following messages to the client and the merchant:

$$\mathbf{B} \rightarrow \mathbf{M}: \quad \{c_0, T_G, SN, CL_M, h(ID_M, SN, CL_{TR}, T_{TR}, Y)\}_Z, \\ h(c_0, T_G, CL_M, X) \qquad (4)$$
$$\mathbf{B} \rightarrow \mathbf{C}: \quad h(ID_M, SN, CL_{TR}, T_{TR}, Y), T_{TR} \qquad (5)$$

Where T_{TR} stands for timestamp when the bank updates CL_{TR}. Note that T_T is updated to T_{TR} after calculating CL_{TR}. The merchant retrieves c_0 and CL_M from the encrypted message. She knows that the client has requested to make the payment to her from $h(c_0, T_G, CL_M, X)$, and the client's request has been authorized by the bank from the message encrypted with *Z* shared between the bank and herself. After receiving the message (5), later on, the client can use $\{CL_{TR}, T_{TR}\}$ to make payment to another merchant.

3.4 Payment Protocol

After completing payment initialization, the client can start the payment to the merchant by sending the coin as follows.

$$\mathbf{C} \rightarrow \mathbf{M}: \quad c_j \qquad (where \quad j = 1, ..., n) \qquad (6)$$

The merchant verifies the requested amount by comparing with c_0. After the verification, she provides goods or services to the client. After each payment, CL_M is deducted. The client is allowed to make the payments up to CL_M without any payment

authorization from the bank. If the remaining credits are not enough to make another payment, the client can request the bank for extra credits by running *Extra Credit Request Protocol*.

3.5 Extra Credit Request Protocol

Normally, when a client spends the credits up to CL_M, she needs to run *Setup Protocol* to issue a new bank coupon. In our protocol, we reduce the frequency of doing this process by running *Extra Credit Request (ECR) Protocol* instead. With *ECR Protocol*, the numbers of message passes are reduced. Before the next payment, the client checks whether $j > CL_M$. If so, she still can purchase the goods but she needs to request for extra credits from the bank. The client realizes that, if her request has been approved, her total credits CL_{TR} will be deducted by CL_M. To request for extra credits, the client sends the following message:

$$\mathbf{B} \rightarrow \mathbf{M}: \quad c_j, CL_M, h(ID_M, CL_M, T_G, SN, CL_{TR}, T_{TR}, Y) \tag{7}$$

At this stage, CL_M stands for new credits to spend with specified merchant. The merchant retrieves CL_M and forwards the following message to the bank:

$$\mathbf{M} \rightarrow \mathbf{B}: \quad ID_M, h(ID_M, CL_M, T_G, SN, CL_{TR}, T_{TR}, Y) \tag{8}$$

The bank retrieves CL_{TR} and CL_M, and then calculates a new CL_{TR}, where $newCL_{TR} = currentCL_{TR} - CL_M$. The bank transfers CL_M to the merchant's account, and then sends the response to the merchant as follows:

$$\begin{aligned} \mathbf{B} \rightarrow \mathbf{M}: \quad & h(ID_M, SN, CL_{TR}, T_{TR}, Y), T_{TR}, YES, h(YES, CL_M, T_{TR}, Z) \\ & \text{if approved} \\ (\text{ or } & \textit{Rejected} \quad \text{if client has not enough credits }) \end{aligned} \tag{9}$$

The merchant checks whether the authorized CL_M in $h(YES, CL_M, T_{TR}, Z)$ is equal to CL_M received from the client in *(7)*. If so, the merchant sends the client the following message:

$$\mathbf{M} \rightarrow \mathbf{C}: \quad h(ID_M, SN, CL_{TR}, T_{TR}, Y), T_{TR} \tag{10}$$

The client expects to receive the updated CL_{TR}, where $updatedCL_{TR} = current\ CL_{TR} - CL_M$. She calculates CL_{TR} and compares with the received CL_{TR}. If they are matched, the client can infer the updated bank coupon from CL_{TR}. The above message is considered as a notification of the client's remaining total credits. Note that, to make the payment to a new merchant, the client repeats *Payment Initialization Protocol* with the updated bank coupon without running *Setup Protocol* as that in existing protocols.

3.6 Coupon Cancellation Protocol

In our protocol, a client is able to refund an un-used bank coupon previously purchased from a bank by sending the following message to the bank:

$$\mathbf{C} \rightarrow \mathbf{B}: \quad SN, T_{CR}, h(SN, CL_T, T_T, T_{CR}, Y) \tag{11}$$

Where T_{CR} is timestamp when requesting for coupon cancellation. The bank removes the coupon with the serial number *SN* from its database. This coupon will be no longer used in the system. The bank transfers the amount CL_T to the client's account and sends the response of the client's request to the client as follows:

$$\mathbf{B} \rightarrow \mathbf{C}: \quad CancelOK, (CancelOK, SN, T_{CR}, Y) \tag{12}$$

3.7 Coin Return Protocol

In some situation, a client may want to end transaction with a merchant after spending some credits and request merchant to return her the un-spending credits. This process can be done in the proposed protocol as follows:

$$\mathbf{C} \rightarrow \mathbf{M}: \quad c_{j_{max}}, T_G, h(ID_M, c_0, T_G, Y) \tag{13}$$

Where $c_{j_{max}}$ is the highest-value coins currently spent to the merchant. The merchant checks whether the received $c_{j_{max}}$ is equal to $c_{j_{max}}$ that she has. If they are matched, the merchant forwards the following message to the bank:

$$\mathbf{M} \rightarrow \mathbf{B}: \quad ID_M, c_{j_{max}}, T_G, h(ID_M, c_0, T_G, Y) \tag{14}$$

The bank retrieves $c_{j_{max}}$ and c_0 and calculates returned amount, where $returned\ Amount = CL_M - j_{max}$. Bank then transfers the returned amount to the client's account and updates the client's bank coupon with the new CL_{TR}, where $updatedCL_{TR} = currentCL_{TR} + returnedAmount$. The bank updates the entry in the list at the record containing T_G and c_0, and then sends the acknowledgement to the merchant.

$$\begin{aligned} \mathbf{B} \rightarrow \mathbf{M}: \quad & h(returnedAmount, ID_M, c_0, T_G, CL_{TR}, T_{TR}, Y), \\ & h(returnedAmount, ID_C, c_0, T_G, T_{TR}, Z), T_{TR} \end{aligned} \tag{15}$$

The merchant is notified that the returned amount has been withdrawn and transferred to the client's account from $h(returnedAmount, ID_C, c_0, T_G, T_{TR}, Z)$. Also, she is notified that the set of coins starting with c_0 is no longer valid. The merchant then sends the following message to the client.

$$\mathbf{M} \rightarrow \mathbf{C}: \quad h(returnedAmount, c_0, T_G, CL_{TR}, T_{TR}, Y), T_{TR} \tag{16}$$

The client expects to receive the updated CL_{TR}, where $updatedCL_{TR} = current\ CL_T + returnedAmount$, and $returnedAmount = CL_M - j_{max}$. The client compares CL_{TR} with the received one. If they are matched, she can infer the updated

CL_{TR}. Later on, the client can use the bank coupon with the updated CL_{TR} to make a payment to another merchant.

4 Discussions

4.1 Transaction Security Properties

In this section, we show that the simple cryptographic technique applied to our proposed protocol satisfies the above transaction security properties. The following message demonstrates how the technique works:

$$\mathbf{B} \rightarrow \mathbf{M}: \quad \{c_0, TG, SN, CL_M, h(ID_M, SN, CL_{TR}, T_{TR}, Y)\}_Z, \\ h(c_0, T_G, CL_M, X) \tag{4}$$

We can see that all transaction security properties for payment systems [1, 5] are satisfied as follows:

1. ***Party authentication*** is ensured by symmetric encryption and *Y* shared between the client and the bank. The encryption ensures that either the bank or the merchant has originated the message, and *Y* ensures that the bank is the originator of the message.
2. ***Transaction privacy*** is guaranteed by symmetric encryption.
3. ***Transaction integrity*** is guaranteed by $h(c_0, T_G, CL_M, X)$ forwarded from the client.
4. ***Non-repudiation of transactions*** is ensured by $h(ID_M, SN, CL_{TR}, T_{TR}, Y)$ in that the bank cannot deny that it did not generate $\{c_0, T_G, SN, CL_M, h(ID_M, SN, CL_{TR}, T_{TR}, Y)\}_Z$ since it is the only party that holds both *Z* and *Y*.

4.2 Dispute Resolution

Our proposed protocol provides offers the ability to resolve disputes among engaging parties in both direct and indirect manners. According to direct dispute resolution, consider the message (5) in *Payment Initialization Protocol*, we can prove that bank is the originator of this message since $h(ID_M, SN, CL_{TR}, T_{TR}, Y)$ can be retrieved by only the client and the bank, but the client does not have the secret *Z*. Thus, the client is not the originator of the message. However, some messages provide indirect dispute resolution. Consider the message (10) sent from the merchant to the client in *Extra Credit Request Protocol*, although the client can generate this message by herself, she cannot modify the content of the message since it will be later detected by the bank.

4.3 Private Information

In any payment system, the information that is known only by relevant parties such as secret keys, bank account information, price, or goods descriptions is considered as *Private Information* [4]. Revealing such information offers the opportunity to perform various kinds of attacks or to trace the client's spending behavior.

In our proposed protocol, c_0 and $c_{j_{max}}$ are sent in encrypted forms compared to signed messages in PayWord and cleartext in PayFair. Moreover, only c_j is sent from the client to the bank over the air. The bank can infer c_0 from $c_0 = h(c, T_G)$, where *n* stands for the current CL_{TR} and later sends c_0 to the merchant in the message (4). Therefore, the secrecy of the requested amount is preserved.

4.4 Performance Analysis

To demonstrate the practicability of the proposed protocol, we compare our protocol with PayWord [6] and PayFair [8] in terms of performance by focusing on the computation and the numbers of message passes of engaging parties.

Considering the party's computation, we mainly focus on the numbers of cryptographic operations applied to engaging parties. Table 1 demonstrates the numbers of cryptographic operations applied to our protocol, PayWord, and PayFair, respectively. Note that n stands for the computations for generating a set of coins.

Table 1. The number of cryptographic operations of SET, iKP, and KSL protocol at client, merchant, and payment gateway, respectively

Cryptographic Operations		Our Protocol	PayWord	PayFair
1. Signature	C	-	1	-
	M	-	-	-
	B	-	1	-
2. Signature verifications	C	-	1	-
	M	-	2	-
	B	-	1	-
3. Symmetric operations	C	1	-	1
	M	1	-	-
	B	2	-	2
4. Hash functions	C	n	n	n
	M	n	n	n
	B	n	n	n
5. Keyed-hash functions	C	4	-	3
	M	1	-	4
	B	3	-	5

From Table 1, we can see that in our protocol, only symmetric-key operations and hash functions are applied, compared to public-key operations in PayWord [6]. It infers that our protocol has better performance than PayWord. Compared to PayFair [8], the proposed protocol also has less party's computation. Moreover, in PayFair, a client is required to contact a bank for issuing a new coupon and generate a new set of coins every time she runs out of credits whereas the coupon in our proposed protocol is issued only once and can be used to make payments with many merchants. This greatly reduces the computational load at the client. These features result in better performance than PayFair.

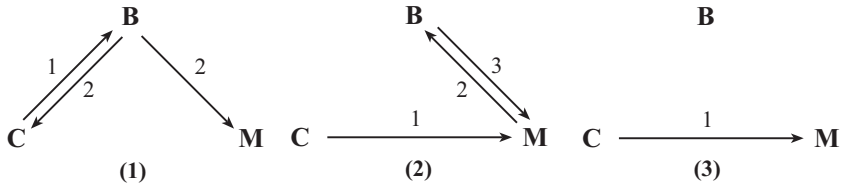

Fig. 1. The numbers of message passes in *Payment Initialization Protocol* of (1) the proposed protocol, (2) PayFair, and (3) PayWord

According to the numbers of message passes, from Fig.1, we can see that the proposed protocol has less numbers of message passes than PayFair which infers better performance. Compared to PayWord, the proposed protocol has higher numbers of message passes. However, PayWord is operated in postpaid mode which a client does not require any payment authorization from a bank in *Payment Initialization Protocol*.

5 Conclusion

We pointed out the problems of existing micropayment protocols when applied to wireless environments due to poor performance and security flaws. We then proposed a prepaid micropayment protocol for wireless networks which solves the above problems. We applied symmetric cryptographic technique which not only reduces parties' computation, but also satisfies transaction security properties. We also performed performance analysis to show that our protocol has better performance than PayWord [6] and PayFair [8] which results in more applicable to limited capability wireless devices.

As our future works, we aim to extend the proposed protocol to perform postpaid micropayments and compare the its results with existing postpaid micropayment protocols including PayWord [6].

References

1. Ahuja, V.: Secure Commerce on the Internet. Academic Press (1996)
2. Anderson, R., Manifavas, C., Sutherland, C.: NetCard - A Practical Electronic Cash System. Lecture Notes in Computer Science, Vol. 1189 (1995)
3. Kungpisdan, S., Srinivasan, B., Le, P.D.: Lightweight Mobile Credit-Card Payment Protocol. Lecture Notes in Computer Science, Vol. 2904 (2003) 295–308
4. Kungpisdan, S., Permpoontanalarp, Y.: Practical Reasoning about Accountability in Electronic Commerce Protocols. Lecture Notes in Computer Science, Vol. 2288 (2002)268–284
5. Park, D.G., Boyd, C., Dawson, E.: Micropayments for Wireless Communications. Lecture Notes in Computer Science, Vol. 2015 (2000) 192–205
6. Rivest, R., Shamir, A.: PayWord and MicroMint: Two Simple Micropayment Schemes. Cryptobytes, Vol. 2(1) (1996) 7–11
7. Romao, A., da Silva, M.: An Agent-based Secure Internet Payment Systems. Lecture Notes in Computer Science, Vol. 1402 (1998) 80–93
8. Yen, S.M.: PayFair: A Prepaid Internet Micropayment Scheme Ensuring Customer Fairness. IEE Computers and Digital Techniques, Vol. 148(6), (2001) 207–213