A Micropayment Protocol for the WAP Environment

Ian Herwono, Markus Pütz
Chair of Communication Networks
Aachen University of Technology
D-52074 Aachen, Germany
{ian|osi}@comnets.rwth-aachen.de

Abstract

A micropayment framework suitable for the Wireless Application Protocol (WAP) environment is presented. The approach chosen focuses on the problems of scalability, modeling of a continuous cash flow and communication bandwidth. Only well known cryptographic algorithms and methods are used. The protocol will be conceptually clean as to upgrade to newer and more secure cryptographic protocols in the future. Because of the expected large number of mobile clients it is essential to the acceptance of a new protocol to be transparent to the user and the developer. We discuss the problems of mobile commerce, privacy and WAP related integration. The protocol performance regarding its timing behavior is evaluated within a standard-conformant WAP environment.

1. Introduction

The *Wireless Application Protocol (WAP)* environment is considered as an open environment expected to fit smoothly into the mobile network and Internet environment. The introduction of various WAP-based mobile information and entertainment services such as location based services or mobile games requires a flexible, secure and reliable payment mechanism which allows the content providers to charge the customers on "per-use" basis. In this paper we present a *micropayment protocol* which is specially suited for the wireless application environment with regard to its constraints like narrow bandwidth, higher network latency, or limited memory capability.

Our mobile payment protocol is chosen to be transparent to both mobile network and Internet environment. Thus the whole communication is done at the application level on top of the *Wireless Transaction Protocol (WTP)* layer of WAP stack [6]. There is no need for extra gateways and the integration at the server can be performed by application programmers without knowledge of the underlying network functionality. On the side of mobile client, e.g., mobile phone, the protocol can be handled by the WAP's *External Functionality Interface (EFI)* [7]. Furthermore, as mobile payment easily crosses national borders there is a need for transparent security even if the underlying network may be prohibited from being secure by national laws. The protocol has to be secure by itself without making use of specific functionality of the underlying network or bearer services. This brings an advantage for porting such protocol to other network infrastructures.

2. Basic Concept

To be able to pay within a computer network environment three parties are needed: *the buyer*, *the vendor* and *a broker*. The buyer and the vendor usually do not know any credentials of each other. The trust is established by a third party which is here called the (money) broker. This could be a bank or something like a credit-card company.

The vendor trusts the credentials presented to her by the broker and so does the buyer. Since there are many potential brokers a payment protocol has to take into account that there may be a trustworthy *broker network*. This leads to PKIs[1] and certificate structures. In this work we will deal only with the certificate or credentials problem, not with issues on establishing a new PKI. The proposed protocol addresses an *asynchronous mode* of operation, i.e., the broker does not have to be online to perform transactions between buyers and vendors. Hence, it is necessary to create certificates (credentials) that give both parties a certain amount of monetary trust (comparable with the maximal trust on credit for credit cards). This is achieved by using digitally signed certificates. Regarding the performance issues no public key cryptographic methods must be used, but it is recommended to use them. Nowadays there are smart cards which are capable to perform a public key signature creation or verification within under one second. As mentioned before we do not trust the carrier and so we need a tamper proof external device to perform the bulk of secure computation anyway. In our implementation we employ the *Elliptic Curve Cryptography (ECC)*. Although other algorithms like RSA could be used, elliptic curve has the best security to bandwidth ratio.

Like other electronic payment schemes, e.g., Pederson's phone ticks [4], Pay Word [5], or CAFE [1], our digital money is represented by hashes. In our case this means hash chains using one-way hash functions. The advantage

[1] Public Key Infrastructures

of using *hash chains* is that, after some overheads in the creation and the subsequent signing process, the computation of the chain is very fast. Only the end of the hash chain is signed and sent to the transaction partner. The hash seed is kept in the tamper proof external device. The initialization process is relatively slow. After that only hashes should be computed and transmitted with each hash representing a monetary value. This reduces the needed bandwidth and computation effort to a minimum and enables the modeling of a continuous cash flow. Here other protocols have the disadvantage of requiring a constant overhead per transaction. Such cash flow can be interesting in the fields of location based services like information databases or GPS-enabled guidance systems ("the longer you stay on the WAP-Site the more you pay"). On the other hand the same protocol can be used for *macropayment* by using only a single hash value, i.e., hash chain with the length of two. At this point security is provided solely by the signature process.

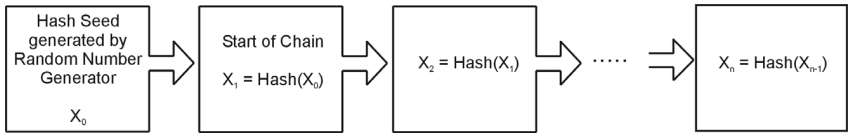

Figure 1: Hash chain of length *n*

One critical issue of micropayment protocols is the fact that only small amounts of money are to be moved. Such payments should be done very often without much overhead. When a certain amount of money is moved at one time, it is no longer possible to speak of micropayments. A secure way of payment is required even if the single transaction is only a small one. At this point the security lost by less secure, thus faster, methods in micropayment systems is restored by online requirements that usually involve the money broker. We use certificates and public key cryptography and thus have a reasonable security without online checking. In addition, the security of elliptic curve cryptography has a good effect on the scalability so that a lot of money can be securely transmitted without the need for an alternative (macropayment) protocol.

For the ongoing survival of a payment protocol it seems to be necessary to "upgrade" a protocol from time to time. Cryptographic algorithms will be broken by the computational power or mathematical insights. Therefore our protocol was designed in a way that it is possible to substitute the cryptographic algorithms with similar ones, i.e., ECC → ElGamal, RSA, etc.; AES → DES, IDEA, etc. This is surely no trivial process but it is possible without interfering with the protocol concepts and advantages.

3. EFI – External Functionality Interface

The EFI framework defines rules of co-existence with the WAP environment. Specifically, the EFI framework defines the overall structure of external system functionality, e.g., payment functionality, which should be accessible to the WAP client. Access is granted by WML-calls to URLs with the following structure:

```
"efi://<server>/<service>[?<parameter>]".
```

It is also possible to access the EFI API through dedicated *WMLScript* functions. Naming conventions are defined by extensive use of concept of "namespace" with all class names being registered with the WAP Interim Naming Authority (or equivalent institutions). The EFI has the same rights to use the man-machine interface and communication capabilities as the *Wireless Application Environment (WAE)* or the *Wireless Telephony Application (WTA)*. It is possible to implement an easy-to-use graphical user interface (GUI) which will override the normal WAP client display for user transparent transactions. Calls to the EFI are uninterruptible. The underlying functionality however may be running parallel to the rest of the WAP client.

4. The Micropayment Protocol

4.1. Electronic Cash as a Hash Chain

To create the hash chain a single random value X_0 is generated. By repeatedly applying a one-way hash function to that value a chain of length $(n+1)$ with X_n as the tail is created: $X_1 = Hash(X_0)$, $X_2 = Hash(X_1)$, etc., as shown in Figure 1. X_0 remains with the issuer of the electronic money, i.e., the smart card. To use the hashes as electronic money *a signed message*, which consists of the hash tail X_n, the number of hashes n, the creation date, the monetary value of each hash (X_n) and the recipient, is transmitted. Payment is done by transmitting X_{n-1} down to X_1.

To avoid double spending the money tokens are only valid between the two parties, i.e., buyer and vendor. The signed message is *the equivalent* to a contract that the hash values transmitted are worth the equivalent money. The number of hashes n and the creation date are safeguards against fraud. The hash chain should only be valid for a limited time. This minimizes the need for

storage of old "contracts" and the possibility to find other parts of the chain by *brute force*.

To receive the actual money the recipient (usually the vendor) has to transmit the received tokens along with the contract to the money broker who charges the sender's (buyer's) banking account and transfers the money to the recipient's account. This process may be initiated immediately after the transaction or after a prolonged time period, e.g., at the end of the day.

4.2. Authentication and Encryption

The authentication process includes the exchange of certificates from both buyer and vendor. These certificates express *the trust* the money broker puts into both parties. One necessary certificate attribute would be the trust on credit (creditability). If necessary or appropriate the X.509 standard may be used. Both certificates should have relatively short expiration dates so that there is no reasonable need for certificate revocation list (CRL), as such lists are difficult to be implement in a smart card device with very limited memory capability. Here a tradeoff has to be found between security and possible fraud damage. When making extensive use of CRL on the client side frequent connections to the broker would be necessary which we need to avoid.

After the certificate exchange each party computes a shared secret *S*. If public key approach is not used, a challenge response procedure can be employed. We investigate the employment of two public key based key agreement schemes. The first one is the *authenticated Diffie-Hellman (DH) key agreement* (ECKAS-DH1[2] with ECSVDP-DHC[3]) [3]. It uses cofactor multiplication to address the small subgroup attack problem. The second one is a variant of that algorithm developed by *Menezes/Qu/Vanstone* (ECKAS-MQV[4] with ECSVDP-MQVC[5]) which provides *forward secrecy* in case the original private key is compromised and uses cofactor multiplication too.

The authenticated DH key agreement is basically similar to mechanism of the original DH algorithm. Instead of calculating a random value for the key generation the private key is used. The second input is the peer's public key. Authentication is thus implicit as the public key is accompanied by its certificate which has to be validated before any transaction takes place. To vary the shared secret the key agreement scheme uses a *key derivation value* which has to be public to both peers. The final secret key is derived by the hashed value of the concatenation of both computed DH-secret key (master key) and the key derivation value. ECSVDP-MQVC has the disadvantage of requiring the generation of a second key-pair. This also means that the "master" shared secret changes with every key-exchange and is therefore not cacheable for the certificate lifetime. When using short lived certificates this security advantage is negated by the already changing shared keys due to new key-pairs. As key derivation function we use SHA-1[6]. The shared secret *S* is then used by a symmetric encryption algorithm to protect the ongoing transaction against eavesdropping. We propose the use of AES[7] algorithm.

In a real smart card implementation the choice of algorithms would strongly depend on the capabilities of the smart card processor. For current smart cards however there are already native implementations of the suggested algorithms available.

4.3. Protocol Scheme

Before any transaction can take place both parties have to retrieve their certificate if they do not have a recent one yet. They do that by authenticating themselves to the money broker which in turn delivers the certificate. This process is done transparently to the user by the EFI when a transaction is requested. It should also be possible to initiate that process by the user or in the initialization process of the EFI in order to minimize the time for transaction startup.

The transaction protocol scheme is described as follows (see also Figure 2):

1. Buyer **B** browses through the WAP-site of vendor **V** and requests for information/content.
2. In turn, **V** requests payment from the (unknown) buyer **B** by using an EFI-URL reference on its WAP-site. **V** transmits (using the EFI call) its "transaction reply URL", certificate ($Cert_V$), suggested hash unit value (X_{h1}) and the number of required money units, i.e., number of hashes.
3. **B** checks $Cert_V$ offline and sends her own certificate ($Cert_B$) to **V**.

[2] Elliptic Curve Key Agreement Scheme, Diffie-Hellman
[3] Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman, cofactor
[4] Elliptic Curve Key Agreement Scheme, Menezes-Qu-Vanstone
[5] Elliptic Curve Secret Value Derivation Primitive, Menezes-Qu-Vanstone, cofactor

[6] Secure Hash Algorithm
[7] Advanced Encryption Standard

4. If this is the first transaction between **B** and **V**, **V** checks $Cert_B$ offline and the associated trust on credit online if possible (otherwise relies on the certificate attributes).
5. **B** and **V** compute a shared secret *S* using a key agreement scheme, e.g., ECSVDP-DHC.
6. **B** generates X_0 and X_n. *n* is determined by a standard value or by **B** using the EFI man-machine interface (depending on the implementation).
7. **B** generates X_h so that $X_{h1} = k * X_h$. *k* is also determined by standard or the user.
8. **B** computes the contract $C = \{X_n, n, <\text{creation date}>, X_h\}$ and the corresponding signature *Sign(C)*.
9. **B** encrypts *C* and *Sign(C)* with *S* (using symmetric key cryptography) and transmits the information to the reply address specified earlier by *V* (in 2.).
10. **V** verifies *Sign(C)* and X_h and replies with a status message to the EFI .
11. If the status is valid **B** will pay by sending the hash values encrypted with the shared secret *S*, and in turn **V** will deliver the paid information/content. Optionally, the encrypted hash values can be transmitted together with the signed contract (in 9.) so that the paid content can be delivered right after accepting the contract.
12. On another content request**,** **V** again requests payment from now known buyer. **B** is identified by cookie, communication parameters or others. **V** transmits hash unit value (X_{h1}), number of required money units (hashes) and identification (by certificate or **V**'s serial number, etc.).
13. The following process can happen:
    a. **B** pays to the known **V** by transmitting the next hash in the appropriate chain, or
    b. **B** does not know **V** anymore and returns a re-initialization request (→2.), or
    c. **B** has no or not enough hashes left and returns a new contract request (→7.), or
    d. **B** has no hashes with the appropriate X_h left and returns a new contract request (→7.).

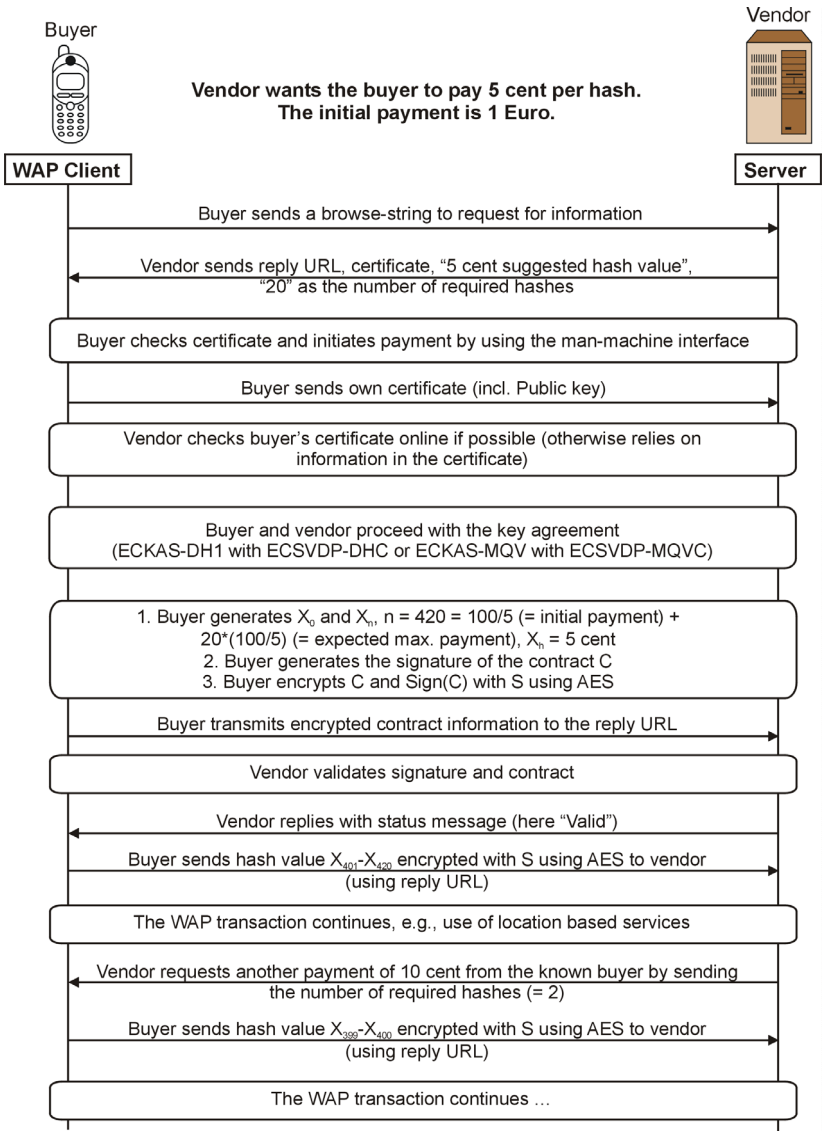

Figure 2: An exemplary micropayment transaction scenario using hash chains

5. Performance Evaluation

The proposed micropayment protocol has been formally specified in SDL[8] and coded in C++. Most of the implementations concerning cryptographic computations originate from the free C++ class library Crypto++ 4.1[9]. In order to investigate the protocol performance within a WAP environment regarding its specific communication overheads and delays, the micropayment protocol has been combined with our prototypical, standard-conformant implementation of the relevant WAP layers, i.e., WTP, WTLS[10], WDP[11] and IP, as depicted in Figure 3. Further we consider the transport layer security provided by WTLS to allow information privacy on the air interface, i.e., encryption of exchanged data or content between WAP client (buyer) and server (vendor). The security features of WTLS and its performance regarding encryption throughputs and key exchange (handshake) durations are discussed in [2]. We assume an underlying IP-based mobile bearer service like GSM/GPRS[12] which is characterized by its effective network throughput on the air interface.

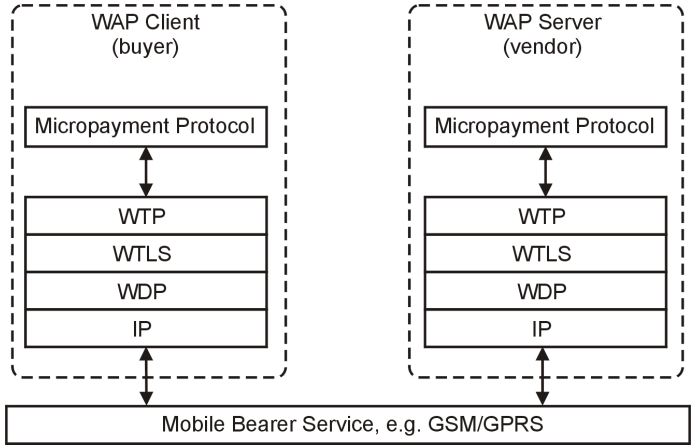

Figure 3: The micropayment simulation system

[8] Specification and Description Language

[9] Please refer to http://www.eskimo.com/~weidai/cryptlib.htm for further information.

[10] Wireless Transport Layer Security

[11] Wireless Datagram Protocol

[12] Global System for Mobile communication/General Packet Radio Service

5.1. Security Parameters

The ECC system parameters F_q, E, P and n are common for the whole system. Point compression is used and a key length of 160 bit is chosen. The used parameters correspond to the basic ECC parameters with assigned number 7 as proposed in the WTLS specification. As the ECC key length of 160 bit is commonly considered equal to the 1024-bit RSA key we chose this ECC domain as sufficient for medium security. Depending on the desired scalability to larger transactions and the point of time of implementation other key length and ECC domains would be required. For the symmetric part AES with a key length of 128 bit is used. As hash algorithm MD5 and SHA-1 are utilized.

5.2. Transaction Scenarios

Differentiating by the current state of a transaction, e.g., whether the buyer's identity is already known to the vendor[13], three scenarios have been investigated using our simulation system. The second and third scenarios can thus be seen as parts of the first (full) scenario. The transaction is finished right after a confirmation of the initial payment has been sent by the vendor. The paid content is to be delivered by the vendor afterwards (not considered in our evaluation).

In the following the scenarios are described and the size of messages that are forwarded to the underlying WTP layer of the WAP protocol stack, is given:

1. **Scenario 1 (full transaction for initial payment):**
   a.) Buyer sends a browse-string requesting content from vendor (200 byte).
   b.) Vendor responds with its certificate and other payment information in accordance with the chosen key agreement scheme (DHC: 324 byte; MQVC: 345 byte).
   c.) Buyer verifies the vendor's certificate, extracts the vendor's public key, and computes the shared secret.
   d.) Buyer sends her own certificate (DHC: 218 byte; MQVC: 239 byte).
   e.) Vendor verifies the buyer's certificate, extracts the public key, and computes the shared secret.
   f.) Vendor sends a short message confirming the acceptation of buyer's certificate (4 byte).

[13] please refer to the transaction protocol scheme (section 4.3).

g.) Buyer generates the electronic contract according to the payment information received from vendor.
h.) Buyer sends the encrypted contract together with a single encrypted hash value for the initial payment (116 byte).
i.) Vendor verifies both the contract and the hash value and returns a payment confirmation message (4 byte); the transaction is completed.

2. **Scenario 2 (subsequent payment with contract renewal):**
   a.) Buyer sends a browse-string requesting content from vendor (200 byte).
   b.) Vendor responds with its certificate and other payment information in accordance with the chosen key agreement scheme (DHC: 324 byte; MQVC: 345 byte).
   c.) Buyer generates the contract according to the received payment information.
   d.) Buyer sends the encrypted contract together with a single encrypted hash value for the requested payment (116 byte).
   e.) Vendor verifies both the contract and the hash value and returns a payment confirmation message (4 byte); the transaction is completed.

3. **Scenario 3 (subsequent payment w/o contract renewal):**
   a.) Buyer sends a browse-string requesting content from vendor (200 byte).
   b.) Vendor responds with its certificate and other payment information in accordance with the chosen key agreement scheme (DHC: 324 byte; MQVC: 345 byte).
   c.) Buyer sends a single encrypted hash value for the requested payment (40 byte).
   d.) Vendor verifies the hash value in accordance with the known contract and returns a payment confirmation message (4 byte); the transaction is completed.

In the second step (b.) – even for subsequent payments – the vendor always sends its certificate to the buyer. This is done to facilitate the secure identification of which vendor the payment should be made for, since the buyer might make business with another vendor each time another (type of) content is requested. We make use of the small-size WTLS certificates in our evaluation. In general the certificate – for subsequent payments – can also be replaced with a unique identifier of the vendor. Note that the payee must not be the vendor itself; it could be another payment service provider authorized by the vendor (content provider).

5.3. Results

All following measurements are the results of tests carried out on a SUN Enterprise server equipped with 1664 Mbyte RAM and using one single *dedicated* processor of 400 MHz clock frequency. The results mainly comprise the times needed to perform the cryptographic computations (at client and server) and to exchange the transaction messages across the air interface depending on the message sizes – including overheads – and available network throughputs in accordance with the three scenarios described in section 5.2.

In Figure 4 the ascertained transaction times of the particular scenarios at different effective network throughputs are given. In case that no WTLS connection for securing the air interface between both parties has been established previously, additional time is needed for performing the WTLS handshake prior to starting with the actual transaction. We chose WTLS class 3 (with client authentication) and ECDH with 160-bit ECC key as its handshake algorithm.

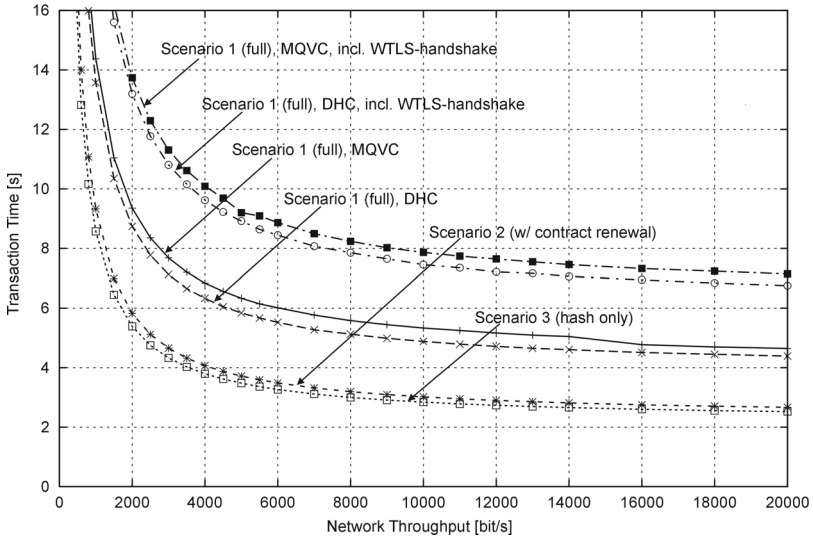

Figure 4: Transaction time according to the scenarios at different effective network throughput

As expected, more time – but not significantly – is required when MQVC-based key agreement scheme is chosen. The impact of the different cryptographic algorithms becomes negligible as the network throughput decreases, and the size of messages transferred during the transaction gets more important. The use of small-size WTLS certificates within the transaction scenario is thus advantageous. In turn, at higher throughputs the total transaction time is mostly affected by cryptographic processes and static network latencies. In case that a shared secret has been agreed, the transaction times of subsequent payments (at the same vendor) are halved. It is obvious that renewals of contract do not increase the transaction time significantly. However the described transaction scenarios also include the transmission of the arbitrary "browse-string" and "payment information" messages, whose sizes strongly depend on the system implementation at vendors, e.g., which content management system is used. Hence, we also consider the *subjective transaction times* by neglecting the corresponding arbitrary messages in our measurements (Figure 5). For the particular scenario it means that:

- Scenario 1 comprises step c.) through i.),
- Scenario 2 comprises step c.) through e.), and
- Scenario 3 comprises step c.) and d.).

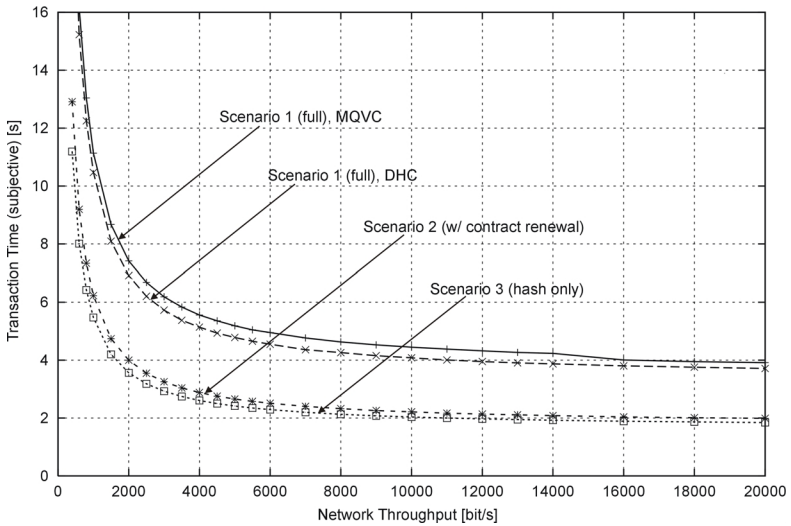

Figure 5: Subjective transaction time according to the scenarios at different effective network throughput

The subjective transaction times are in particular useful for determining the protocol performance on a continuous cash flow, e.g., for paying streaming content like music or videos, where no user interaction is needed for subsequent payments. At higher network throughputs (> 6000 bit/s) around 2 seconds are needed for each micropayment consisting of a single hash value (scenario 3).

Furthermore we have investigated the impact of the number of hashes included in a single payment on the resulting transaction time. This is important for the system implementation to determine the optimal monetary value of each hash (X_{h1}), the length of the hash chain (n), and thus the frequency of contract renewals in accordance with the pricing of each content. Figure 6 depicts the resulting subjective transaction times depending on the number of hashes included in a single transmission at a network throughput of 4000 bit/s. As shown in the figure, the time required to send eight hashes within a single payment slightly exceeds the required time for a contract renewal *and* payment with a single hash value, which might have the same monetary value as the eight hashes have, i.e., $X_{h1,new} = 8 * X_{h1,old}$. However it has to be noted that in case of implementation on smart card processors with less computational capability than the one used in our evaluation platform, more time will be needed to perform a contract renewal so that the corresponding threshold, i.e., 8 hashes at 4000 bit/s, might change significantly.

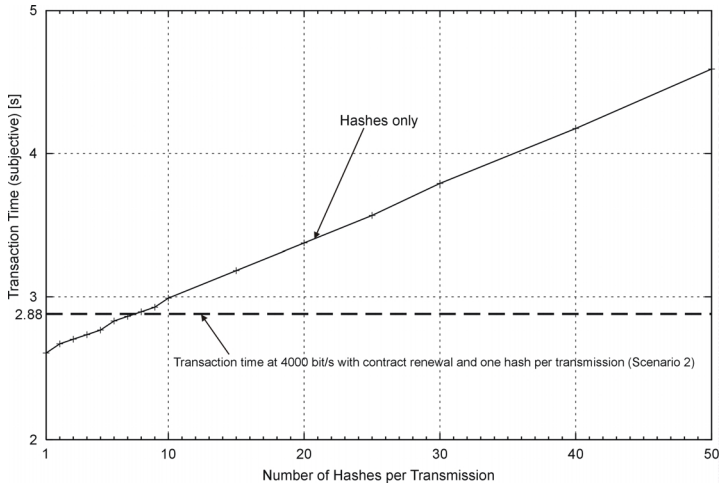

Figure 6: Impact of the number of hashes included in a single payment at a network throughput of 4000 bit/s

6. Conclusions

Our protocol is originally designed for frequently recurring small payments. However in case that a small number of higher amount payments are to be made, an appropriate level of security can also be provided by the protocol. In general, we have identified a minimum threshold of the effective network throughput for our payment protocol at 2000 bit/s. A mobile user would not accept to wait for more than 5 seconds prior to completion of a payment transaction. Thus a drawback of our protocol is when only individual payments at different vendors are performed since the payments need around 5 seconds in average at 4000 - 8000 bit/s network throughput to complete. In contrast to that, subsequent payments can be completed in under 3 seconds (subjectively).

Further works will focus on the comparison of the presented micropayment protocol with others like *Payword* or *Millicent* within the wireless environment.

References

[1] Boly, J., Bosselaers, A., Cramer, R., Michelsen, R., Mjolsnes, S., Muller, F., et al., "The ESPRIT Project CAFE – High Security Digital Payment Systems", in Computer Security – ESORICS '94, LNCS 875, Springer-Verlag, Berlin, 1994.

[2] Herwono, I., Liebhardt, I., "Performance of WTLS and its Impact on an M-Commerce Transaction", (to appear) Proceedings 3rd International Conference on Information and Communications Security (ICICS'01), November 2001.

[3] IEEE, "P1363/D13 – Standard Specifications for Public Key Cryptography", November 1999.

[4] Pederson, T., "Electronic Payment of Small Amounts", Security Protocols, LCNS 1361, M. Lomas, Ed., Springer-Verlag, Berlin, 1997.

[5] Rivest, R., Shamir, A., "Payword and Micromint: Two simple Micropayment schemes", Security Protocols, LCNS 1361, M. Lomas, Ed., Springer-Verlag, Berlin, 1997.

[6] WAP Forum, "Wireless Application Protocol Architecture Specification", http://www.wapforum.org, April 1998.

[7] WAP Forum, "External Functionality Interface Framework", Proposed Version, http://www.wapforum.org, May 2001.

Authors' Biographies

Ian Herwono received his bachelor and master (Dipl.-Ing.) degree both in Electrical Engineering from the Aachen University of Technology, Germany, in July 1996. In October 1996 he joined the Chair of Communication Networks at the same University, where he is working towards his PhD. His research interests are in the areas of development and performance evaluation of mobile commerce services, network security architecture and wireless ad-hoc networks.

Markus Pütz is a student of the Faculty Electrical Engineering at the Aachen University of Technology, Germany. He has long practical experience with network and system security in heterogeneous network environments. Currently he is working on his master's thesis at the Chair of Communication Networks dealing with the development and evaluation of mobile micropayment protocols.