# The Light Signatures Protocol

Eric Pommateau*,‡ Frédéric Bellaiche* Jacky Montiel*,†

*NTSys
64, chemin des mouilles
69130 Ecully
FRANCE

†Ecole Normale Supérieur de Lyon
46, Alle d'Italie
69364 Lyon Cedex 07
FRANCE

‡CEDRIC - CNAM
292 Rue St Martin
75141 Paris Cedex 03
FRANCE

**Abstract:** *We present a new client-server security protocol that includes a high-performance signatures mechanism. The protocol design relies on the two following ideas: 1) in a client-server relation, the permanence of the parties allows to share session keys for signing successive messages; 2) the session keys can be set using standard strong cryptography between the two parties. Afterwards, they are used for signing the messages within a predefined cycle of exchanges.*

*This concept has been applied to the $\mu$-COMM micro-payment system developed by NTSys and leads to a performance gain of the order of one hundred.*

*Keywords:* micro-payment, cryptographic protocol, authentication, signatures, performances.

## 1 Introduction

The design of light signatures has been motivated by the development of the $\mu$-Comm micro-payment system (Fig. 1). Payment techniques on Internet are mostly based on SSL/TLS [10], with high-performance alternatives brought by hardware devices. Nevertheless, the messages authentication scheme is a crucial stake for micro-payment and must be cost efficient. Our work followed previous works on micro-payment [3, 4, 5, 6], with the objective of designing a software highly powerful and cost effective solution. The protocol of light signatures presented here, is built on a cycle of client/server exchanges which requires a reciprocal authentication. It is faster than the use of a standard strong signature for each message and provides the same level of security. This mechanism is particularly interesting with regard to micro-payment.

$\mu$-COMM uses an aggregation mechanism managed by a confidence party that arbitrates transactions between content resellers and Internet end-users. The on-line control/registration of transactions requires authentication mechanisms which, in the case of micro-payment, must be cost effective. Using standard RSA or elliptic curves cryptography becomes critical to manage signatures in such systems when the end-user traffic increases, since the confidence party is a bottleneck. The system is improved by light signatures between the confidence party and the content servers.

The light signatures protocol has been initially imagined by Jacky Montiel[1]. The refinements and development have been carried out in the $\mu$-COMM+ extension project, supported by the French research public program OPPIDUM on security, in co-operation with the École Normale Supérieure de Lyon.

## 2 Principles

The protocol requires an initialization and uses a sequence of $N$ client/server exchanges of messages, each sequence is called, in a sequel, a cycle.

[1] The use of light signatures in micro-payment systems is patented by NTSys.

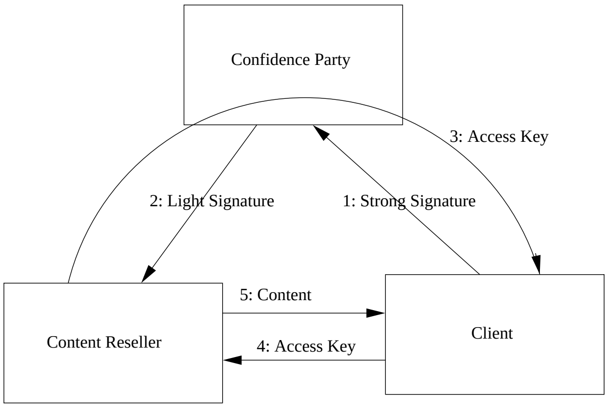


Figure 1: Example of on-line transaction

When each cycle begins, a seed is sent by the client with the first message. It is ciphered with a secure public key cipher (RSA, El Gammal).

In a first step, a sequence of nonce (the nonce sequence) is generated by the client, starting with the seed. Each element of the sequence is obtained as a one way hash function of the previous one. The sequence has $N$ elements. $N$ is a bound of the number of messages exchanged between the client and the server. When the server receives the first messages, it deciphers the seed and generates the same nonce sequence. In the next steps the odd elements of nonce sequence are added to the messages sent from the client to the server and even elements are added to the messages sent from the server to the client. Each message is also signed by another hash function applied to the whole message including the nonce. As the client and the server know the whole nonce sequence, each one is able to verify the signature of each received message. This protocol can be used upon a non reliable lower protocol layer (such as UDP or at the IP level): the nonce sequence is used in a decreasing order to protect upon loss of messages sequence[2] and the sequence order can be used as a synchronization point in the client-server exchange.

[2]this mechanism is similar to OTP [9]

# 3 The Protocol

## 3.1 Notation

All trough this document, we define :

- $Clt$: the client,
- $Srv$: the server,
- $Q_i$: a request of the client,
- $A_i$: a response from the server,
- $Ind_{Clt}$ and $Ind_{Srv}$: two integers maintained by the client and the server (respectively),
- $N$: the maximum number of messages to be sent,
- $X, Y$: the concatenation of message $X$ and message $Y$,
- $P_A$ and $S_A$: a public-private key pair owned by the actor A,
- $\{X\}_K$: the message $X$ ciphered with the key $K$,
- $H_i(X)$: $X$ hashed with the hash function $H_i$,
- $\alpha$: a seed created by Clt (e.g. a pseudo-random value),
- $E$: an error code, defined below,
- $Sign(X, E, k)$: the signature of message $X$ concatenated with $E$ and the indice $k$ (see formula below).

## 3.2 Generating the Nonces Sequence

The seed $\alpha$, is created randomly by the client and sent to the server with the use of heavy cryptography. First, the client ciphers the seed with its private key to ensure the authentication and then with the server's public key to ensure the confidentiality.

This seed is used to create the nonce sequences by iterating a hash function on this seed; a sequence of the form:

$$\alpha, H_1(\alpha), H_1^2(\alpha), \dots, H_1^{2N}(\alpha)$$

is obtained. We note $N_i^\alpha$, a nonce corresponding to the *ith* hash of $\alpha$ (e.g. $N_i^\alpha = H_1^i(\alpha)$).

The client and server must store the whole sequence (see performances).

## 3.3 Signature's Function

The function used to sign a message is:

$$Sign(M, E, i) = H_2(M, E, N_{2N-1-i}^\alpha)$$

where $M$ is the content of the message ($Q_i$ or $A_i$, depending of the actor who signs the message).

$H_2$ hashes the message. Without this, any intruder can change the content and sign the message with another valid signature. In practice, $H_1$ and $H_2$ can be the same.

## 3.4 Initialization

The protocol requires an initialization phase which is not defined in the protocol specification.

- Client and server must have both a public-private key pair. The client must know the server's public key and the server must know all the client public keys. It can rely on a Public Key Infrastructure (PKI) [11].
- $N$ must be the same for client and server. $N$ influences the speed-up of the protocol (see performance).
- Client and Server have an unique identifier (Clt, Srv), all the actors must know these identifiers.

[3]" represents an empty string

## 3.5 Protocol Data Units

### 3.5.1 First Message, Issued by the Client

The first message is:

| |
|---|
| ACTOR, INDEX, SEED, CONTENTS, SIGNATURE |

- ACTOR is the entity who sends the message, here ACTOR = Clt,
- INDEX, represents the rank of the nonce being currently used. (here INDEX=0),
- SEED, represents the seed ciphered with the client's private key end then with the server's public key,
- CONTENTS is the first request of the client. Here it is equal to $Q_0$,
- SIGNATURE is the result of the Sign (CONTENTS,",INDEX)[3] function.

Note that there is no error message. During the first message, no message is transmitting between the client and the server.

### 3.5.2 Other Message, Issued by The Client or by the Server

Any message of the protocol sent either by the server or the client is of the form:

| |
|---|
| ACTOR, INDEX,CONTENTS, ERROR, COMPLEMENT_ERROR, SIGNATURE |

where:

- ACTOR is the one who sends the message (the identifier of the client for example, and a unique code for the server).
- INDEX represents the rank of the nonce in use.
- ERROR is an acknowledgment or a code for a protocol violation.
- COMPLEMENT_ERROR is an extra information requested by some error.
- SIGNATURE is obtained by the function: Sign(CONTENTS,ERROR,INDEX)
- CONTENTS is either a client's request or a server response.

### 3.6 Error Code

The error codes are:

- *OK*,
- *BAD_SIGNATURE*,
- *SERVER_EARLY*,
- *SERVER_LATE*,
- *CLIENT_EARLY*,
- *CLIENT_LATE*,
- *TIME_OUT_CLIENT*,
- *TIME_OUT_SERVER*,
- *LAST_MESSAGE*.

The complement is empty if the code does not need an extra information ("OK" for example). All errors are included in the signed part of the message (light ciphered), otherwise any intruder could, without modifying a message, change the error code and make strong attacks like DoS[4] attacks: for example sending the error code "LAST_MESSAGE". The complement is used to have information about the desynchronization and to have extra information for intrusion detection.

### 3.7 Description of the Protocol

The error codes and error complements are omitted in the messages for lisibilities issues. Each found error is stored localy and sent in blocks with the next message. When a message is sent, the error code is nulled.

#### 3.7.1 Client's First Request

At the beginning of the protocol, $Ind_{Clt}$ is 0.

$$\boxed{\begin{array}{l} Clt \rightarrow Srv : \\ Clt, 0, \{\{\alpha\}_{S_{Clt}}\}_{P_{Srv}}, Sign(M,0), M. \end{array}}$$

At this moment, the nonce sequence derived from the seed $\alpha$ must be stored in a protected area on both the client and server side. If the sequence is not stored, the performance of the protocol is worse then a classical protocol.

[4]Denial of Service

#### 3.7.2 Client's Next Request

$Ind_{Srv}$, is the last index received by the client from the server. $Ind_{Clt} \neq 0$ and $Ind_{Clt} < N$:

- if $Ind_{Srv} = Ind_{Clt}$:

$$\boxed{\begin{array}{l} Clt \rightarrow Srv : \\ Clt, 2 * Ind_{Clt}, Sign(Q_{Ind_{Clt}}, \\ 2 * Ind_{Clt}), Q_{Ind_{Clt}}. \end{array}}$$

- else if $Ind_{Srv} < Ind_{Clt}$ (Old message of the server):

  The client ignores this response.
  $\text{ERROR}+ = \mathit{SERVER\_LATE}$

- else $Ind_{Srv} > Ind_{Clt}$:

$$\text{ERROR}+ = \\ \mathit{LAST}\_\text{MESSAGE} + \mathit{SERVER\_EARLY}$$

$$\boxed{\begin{array}{l} Clt \rightarrow Srv : \\ Clt, 2N, Sign('\text{ERROR}', 2N), \\ '\text{ERROR}' \end{array}}$$

If $Ind_{Srv} = 0$, the message is not processed ($\mathit{SERVER\_LATE}$). If $Ind_{Srv} \geq N$, the message is ignored (invalid signature).

#### 3.7.3 Server's Response

At the beginning of the protocol, $Ind_{Srv}$ equals zero. When the client's first message arrives, all the nonce sequence is stored by the server.

Depending of the value of $Ind_{Clt}$ received, the server answers according to:

- if $(Ind_{Clt} = Ind_{Srv})$:

$$\boxed{\begin{array}{l} Srv \rightarrow Clt : Srv, 2 * Ind_{Srv} + \\ 1, Sign(A_{Ind_{Srv}}, 2 * Ind_{Srv} + \\ 1), A_{Ind_{Srv}} \end{array}}$$

  The variables are then updated: $Ind_{Srv} = Ind_{Srv} + 1$, $Ind_{Clt} = Ind_{Clt} + 1$ (when the client receives the answer from the server).

- else if $(Ind_{Clt} > Ind_{Srv})$:
  $\text{ERROR}+=\mathit{CLIENT\_EARLY}$

$$\boxed{Srv \rightarrow Clt : Srv, 2 * Ind_{Clt} + 1, Sign(R, 2 * Ind_{Clt} + 1), A_{Ind_{Srv}}}$$

The variables are then updated:
$Ind_{Srv} = Ind_{Clt} + 1$, $Ind_{Clt} = Ind_{Clt} + 1$(when the client receives the answer of the server)

- else if $(Ind_{Clt} < Ind_{Srv})$:

  This request is ignored.
  ERROR+=$CLIENT\_LATE$

  No update is necessary.

#### 3.7.4 Other Events

- *TIME_OUT_CLIENT*: If the server does not answer in a defined amount of time, the client resends its request, signed with a new nonce. It sends the $TIME\_OUT\_CLIENT$ too and the number of time-out as a complement. When a fixed time-out number is reached, another sequence is generated by the client.
- *TIME_OUT_SERVER*: This situation happens because of the limited time-life of the nonce and the key. Furthermore, the server might want to do revisions of its protocol. This mechanism is also useful when the server analyzes that an intruder has stolen its keys.

The time scales of the two events can not be compared: the first time scales is within the order of a minute and the second within the order of a week or a month.

# 4 Claims

## 4.1 Cryptographics Mechanisms

The mechanisms used in this protocol are:

1. *The public-private key scheme*
   This method ensures that no one but the server can read the seed. Only the client and the server know the sequence.
2. *one-way hash function*
   $H_1$ and $H_2$ are functions which are impossible to reverse with a computer (with the knowledge of $H_i(x)$ we do not have the knowledge of $x$).
3. *Changing nonce*
   The sequence of nonce is used to protect against replay of messages attack.
4. *Computation of the sequence with a one way function and keeping the nonce in reverse side*

   If the computation function of the sequence was trivial, the knowledge of one nonce could allow any intruder to sign all the messages until the sequence is empty. Otherwise, the signature must be light to compute, so the verification time does not get too long, thus heavy functions like 3-DES are not used. One-way functions are a good compromise.

## 4.2 Attacks Analysis

An intruder is defined by its ability to:

- Read all the messages on the network
- Send messages on the network by taking the identity of another actor.
- Block messages

Nevertheless, he can not within a reasonable time:

- break a RSA key
- Reverse a one-way function.

#### 4.2.1 Fatal Attacks

The tree actions below are fatal to the protocol:

- The intruder acquires $\alpha$, so he can compute the sequence and send as many signed messages as he wants.
- The intruder acquires $S_{Srv}$, so he can play the role of the server for the protocol.

- The intruder acquires $S_{Clt}$, so he can play the role of the client for the protocol.

We prevent these sorts of attacks by using a RSA algorithm. If the intruder wants to acquire one of the listed data, he must break RSA algorithm.

#### 4.2.2 Non-fatal attacks

Two sorts of cryptanalysis attacks are considered:

1. *On-line attacks*: An intruder gets and blocks the progression of a message. He tries to falsify this message before the actor emits a TIME_OUT. The one-way function protects this protocol against this form of attack. The reasonable time necessary to reverse this function is normally greater than the time set to TIME_OUT_CLIENT

2. *Off-line attacks*: The intruder, having intercepted a message, has enough time to compute the next nonce. Getting the nonce in the reverse way prevents the intruder from using his knowledge. He has to end his calculation before the end of the protocol (TIME_OUT_SERVER).

#### 4.2.3 Denial of service

The intruder can overflow the server by sending invalid messages or trying to change error codes. The server protected itself by signing the error codes and not answering to the first invalid messages.

### 4.3 Performance Evaluation

#### 4.3.1 Notation

Let $T(RSA)$ be the mean time to cipher or decipher with a public or a private key using RSA algorithm.
We note $T(H_i)$, the time to compute the result of the one-way function $H_i$.
All the functions can be identical (or have the same characteristics). We will define $H_1 = H_2 = H$ in the sequel.
We assume that the time to compute a hash function on the seed and on a message is the same.
We compare the time used to process $N$ exchanges between the client and the server in two cases. In the first case all messages use a classical RSA signature:

$$\begin{array}{c} Clt \rightarrow Srv : M, \{H_2(M)\}_{P_{Srv}} \\ Srv \rightarrow Clt : R, \{H_2(M)\}_{P_{Clt}} \end{array}$$

This time is denoted $T_{RSA}(N)$.
In the second case, our protocol is used. The corresponding time is denotes $T_{SL}(N)$. In both cases, exchanges are assumed to be free of errors.

#### 4.3.2 Results

We get:

- $T_{RSA}(N) = 2N[T(H) + T(RSA)]$
  The message is first hashed and then ciphered with an RSA key, so the formula is:
- $T_{SL}(N) = 2 * T(RSA) + 3 * N * T(H)$
  The first message needs two RSA ciphering. $N$ hashes are needed for the computation of the sequence and only two hashes for each message sent.

We can compute the speed-up ratio $T_{RSA}/T_{SL}$.

$$T_{RSA}/T_{SL} = (A + 1)/(2A + 1/N)$$

where $A = T(H)/T(RSA)$

#### 4.3.3 Application

MD5 and SHA1 are the two hash functions used in almost all Internet security protocols. We choosed MD5, for it is faster than SHA1.

A MD5-hash is 1000 times faster than a 512 bits RSA signature verification [2]. The length of $\alpha$ can be the same as the result of hash a message, so we take 160 bits for the length of $\alpha$. With this data, we obtain a speed-up equal to 45 for $N$=50 and 83 for $N$=100. The maximum speed-up found is 500.50.

## 5 Conclusion and Further Work

We have presented a fast and reliable protocol of authentication designed to sign messages in a client-server model. It relies upon strong cryptography for signatures computation in a preliminary exchange of seed and it is accelerated by the classical use of one-way function. The light signatures are thus especially useful in a scheme of authentication which must be at the same time fast and inexpensive.

The principal contributions of the light signatures protocol are:

- It can send messages signed in a fast way
- It does not require any particular hardware (like expensive SSL cards) for equivalent performances.
- It offers the same level of security as a heavy signature.

The light signatures protocol has shown its performance in micro-payment distributed applications. Other possible fields of application are:

- smart cards and mobile terminals identifying themselves to a central equipment (PC or craddle).
- authentication in multi-part systems architecture.

An implementation of the protocol developed by Eric Pommateau is available under Open Source License. The protocol is used in the framework of the Internet Payment System ($\mu$-COMM).

Complementary work is currently carried out at ENSL, as part of $\mu$-COMM+ project, on the formalization of the protocol, in particular by using formal techniques of Paulson [8] or Bolignano [7], in order to demonstrate formal security properties. We thanks Prof. Pierre Lescanne, D. Hirschcoff and P. Dargenton from ENSL for their contribution to improve the protocol. We thanks also Prof. Stéphane Natkin and Prof. Gérard Florin (CNAM) for their advices. Eric Pommateau is a PhD student in Laboratoire CEDRIC - Conservatoire National des Arts et Métiers (CNAM).

## References

[1] Bruce Schneier, *Cryptographie appliquée*, seconde édition, éditions Wiley, 1996.

[2] Mostafa Sherif *La monnaie électronique*, Editions Eyrolles, 1999.

[3] Ronald L. Rivest, Adi Shamir, *Payword and Micromint: Two simple micropayment scheme*, 1996.

[4] Ronald L. Rivest, *Electonic lottery Tickets as Micropayment*,Proceedings of Financial Cryptography '97, 1997.

[5] Steve Glassman, Mark Manasse, Martin Abadi, *The Milicent Protocol for Inexpensive Electronic Commerce*

[6] Mihir Bellar, *iKP, A Family of Secure Electronic Payment Protocols*, 1995.

[7] Dominique Bolignano, *An approach to the formal verification of cryptographic-protocols*, Third ACM Conference of Computer and communication Security, pages 106-118, ACM press, 1996.

[8] L C Paulson, *The inductive approach to verifying cryptographic protocols.* J. Computer Security 6,pages 85-128, 1998

[9] N. Haller, C. Metz, P.Nesser, M. Straw *A One-Time Password System (RFC 2289)*, February 1998

[10] T. Dierks, C. Allen *The TLS Protocol Version 1.0 (RFC 2246)*, January 1999

[11] S. Chokhani, W. Ford *Internet X* , March 1999