Security Analysis of AN.ON's Payment Scheme

Benedikt Westermann

Center for Quantifiable Quality of Service*
Norwegian University of Science and Technology

Abstract. In recent years several payment schemes have emerged for anonymous communication systems such as AN.ON and Tor.
In this paper we briefly present a payment scheme that is deployed and currently used by AN.ON. The main contribution of this paper is a security analysis of the most important cryptographic protocols involved in the payment process. The analysis of the protocols shows that they contain several weaknesses that need to be addressed to provide a fair service. We show how an attacker can use the weaknesses to surf on other's credits. Finally, we propose a fix for the protocols in order to withstand the encountered attacks.

1 Introduction

Many publications in the area of anonymous communication have been published in the last decade. Most of them deal with anonymous routing mechanisms or with attacks on the latter. Besides theoretical work they have also deployed anonymization networks based on the theoretical work. Two of the most widespread deployed anonymization systems are Tor[7] and AN.ON[2].

For both systems it is understandably crucial to find operators willing to cover the costs and the possible legal consequences of providing an anonymization service. The Tor project is not commercial, therefore it can not pay the operators itself. Thus, Tor strongly depends on volunteers willing to provide servers in the network. Clearly, these circumstances make it more difficult to find operators for the Tor network. To overcome these difficulties a payment protocol for anonymous routing was proposed by Elli Androulaki et. al.[1].

In the case of AN.ON, a spin-off company named JonDos[1] was founded in 2007. Its business model is to act as broker between the operators and the users of the network. The company provides the operators of mixes, the service providing network nodes, with the possibility of collecting money from the users for the offered anonymization. This possibility creates an incentive for people to become an operator for AN.ON.

Clearly, payment protocols must fulfill various requirements. One requirement is that the payment process does not compromise the anonymity of users.

* *"Center for Quantifiable Quality of Service in Communication Systems, Center of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT.* `http://www.q2s.ntnu.no`

[1] `https://www.jondos.de`

Another requirement is that the payment process must be fair, so none of the parties should be able to betray other participants. To the best of our knowledge there is no security analysis for AN.ON's payment system available. Thus, we will take a closer look on the used cryptographic protocols.

In this paper we formalize and examine the cryptographic protocols which have been used since 2007 for the payment procedure in AN.ON. Our main focus is to check whether the protocols fulfill their basic tasks or fail to do so. In the latter case we also present some consequences of the encountered weaknesses.

This paper is divided into six sections. Section 2 describes the anonymization concept of AN.ON and the idea of its payment concept. In Section 3 we present our assumptions and our notation. Section 4 deals with the cryptographic protocols involved in the payment process, as well as some attacks on the protocols. The reasons for the attacks and possible countermeasures are discussed in Section 5. Section 6 concludes the paper.

2 Description of AN.ON

2.1 Anonymization Process

An objective of AN.ON's anonymization system is to provide *sender anonymity*[9] on the network layer for its users versus the receivers of messages. This means that a receiver of a message cannot determine the network address, for example the IP address, of a sender. In addition, AN.ON aims to provide *relationship anonymity* with respect to the operators of AN.ON. Therefore, no operator can determine which entities are communicating with each other.

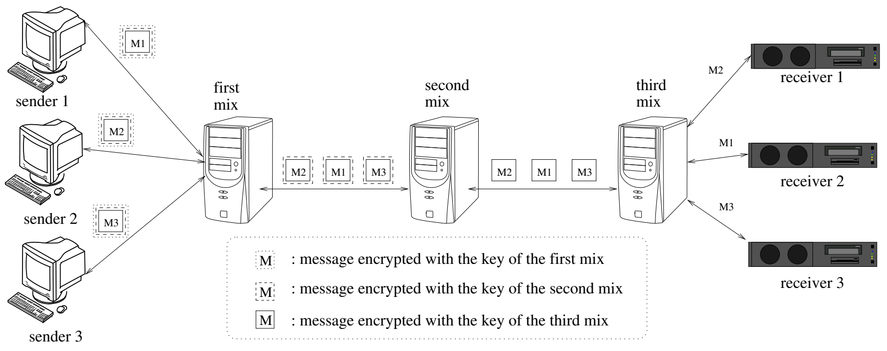

M1
sender 1
M2
sender 2
M3
sender 3
first mix
M2
M1
M3
second mix
M2
M1
M3
third mix
M2
receiver 1
M1
receiver 2
M3
receiver 3
M : message encrypted with the key of the first mix
M : message encrypted with the key of the second mix
M : message encrypted with the key of the third mix

Fig. 1. Example for a cascade in AN.ON

In order to fulfill these objectives, so-called *mix* servers, also called *mixes*, are introduced. The general task of a mix is to remove correlations between incoming and outgoing messages[6]. In the deployed AN.ON system this is achieved solely[2]

[2] Even though mixing of packets was mentioned in the original publication, it is nowadays deactivated due to performance reasons.

by multiple layers of encryption (Fig. 1). A single layer of encryption is added and removed respectively by a single mix.

A single mix between a sender and a receiver is not sufficient in order to protect the users against the operator of a mix, since an operator is able to link incoming and outgoing messages. Thereby he can uncover the relationship between sender and receiver. This is avoided by chaining several mixes together (see Fig. 1). Such a chain of mixes is called a *cascade*. Only if all operators in a cascade collude they are able to revoke the user's relationship anonymity.

It is important to notice that the order of the mixes in a chain is fix and is chosen by the operators of the mixes. Therefore, a user cannot freely choose the order of the mixes in a cascade. This has various advantages as well as disadvantages[3]. One advantage is that all mixes in the cascade process the same amount of packets. Another advantage is that each packet is sent along the same route through the cascade. Both properties are mandatory for AN.ON's payment scheme.

2.2 Payment Concept

Cascades can be divided into free cascades and premium cascades. Free cascades can be used by every user without any fee. However, they are often only operated by one single operator. The free cascades do not guarantee any quality of service. By contrast, premium cascades promise a minimum bandwidth for their users. Beyond that premium cascades are more secure due to distributed and dedicated servers as well as certified and independent operators.

Certainly, the additional service of the premium cascades is not for free. To collect money from the users AN.ON has integrated a payment system which bases upon a prepaid concept[10,8]. It is similar to the concept which is used in a phone booth. When users connect to the cascade, they pay a small amount of money in advance (up to 3 EuroCent) to the operators in order to transfer some bytes (3 MB) over the cascade. Once they have almost consumed the bought traffic volume they are able to purchase new traffic volume on the fly at the cascade. The purchase of new traffic volume is possible without the termination of existing connections.

However, the transfer of such small amounts of money is not efficient because of the overhead and the lack of digital money. In order to solve the problem, users buy some credit[3] at a so-called *payment instance*. The payment instance is operated by a trusted third party. Afterwards, users can spend their bought credits on a premium cascade.

The first mix in a premium cascade is responsible for the accounting. Thus, we call the first mix also accounting instance[4]. Due to the fact that every mix in a cascade transfers the same amount of data, it is possible to implement the accounting only at the first mix. Hence, behind the first mix the packets which are related to the traffic of users are indistinguishable from packets in a free cascade.

[3] Current rates are between 2 Euro for 200 MB up to 40 Euro for 6,5 GB.

[4] Accounting instances and payment instances are operated by different operators.

3 Assumptions and Notations

We assume that the payment instance is trustworthy with respect to the payment. However, it is not trustworthy with respect to the anonymization process. Furthermore, we assume that every participant in the protocol knows the public key of the payment instance as well as the public key of the accounting instance. Concerning the accounting instance, we assume that it may act malicious in the payment process.

Cryptographic keys in this paper are denoted with prefix K. K_x represents the public key of a principal X. K_x^{-1} is the corresponding private key of the principal X. In particular we assume that if a principal X owns K_x^{-1} he automatically possesses K_x. K_{xy} denotes a session key between a principals X and Y. The term $X \ni a, b, c$ denotes that the principal X is in possession of the items a, b, c.

We assume a non-global but active adversary who cannot break cryptographic primitives. Our assumed attacker model is equal to the attacker AN.ON can withstand in its current implementation.

4 A Closer Look on the Protocols

4.1 Overview over the Protocols

AN.ON involves various cryptographic protocols in order to establish the payment process. This section describes in detail the most important protocols with their assumptions, intentions and weaknesses.

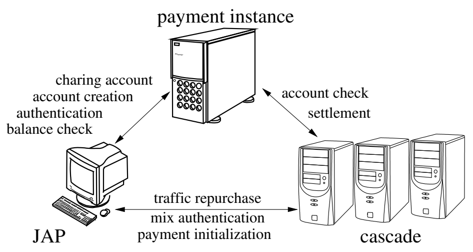


Fig. 2. Account creation protocol

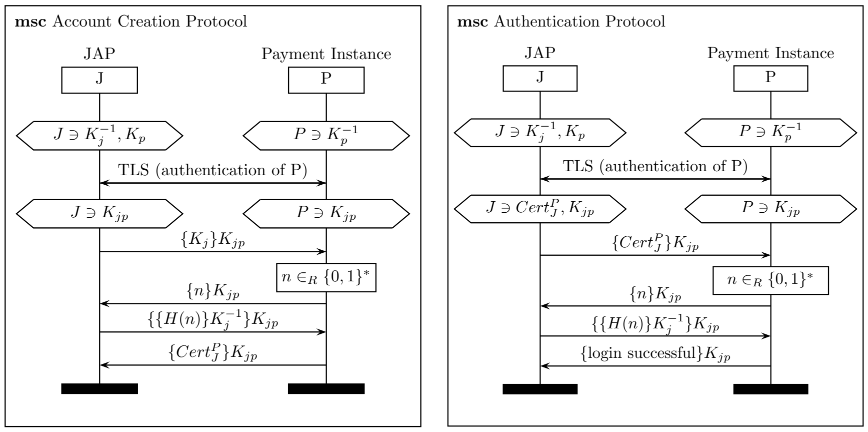
Fig. 2 gives an overview over some protocols and parties which are involved in the payment process. The protocols between the payment instance and the accounting instance are integrated within the *payment initialization protocol* and the *traffic repurchase protocol* respectively. Thus, both protocols are not described separately. Please note, there are also protocols concerning the payment process between mixes, but those remain unaccounted for. The following text provides a detailed description of the protocols mentioned in Fig. 2.

4.2 Account Creation

The *account creation protocol* involves a payment instance and a user which is in AN.ON normally called *JAP*. The objective of this protocol is to create an account for the user. An account is bound via a private key to the user. The private key is created by the user prior to the protocol run. Only with the private key a user is later on able to prove her ownership of the account and subsequently buy traffic volume from an accounting instance.

In order to attest that a public key belongs to a valid, but maybe uncharged account, the payment instances issue a certificate for the public key. The certificate should only be issued by the payment instance if it is convinced that a corresponding private key exists.

Fig. 3(a) presents a message sequence chart of the account creation protocol. We assume the following preconditions for a protocol run. Firstly, the payment instance possesses its own key pair (K_p, K_p^{-1}). Secondly, a user holds the public key (K_p) of the payment instance. Lastly, she also owns a key pair which has been freshly generated by herself.



(a) Account creation protocol (b) Authentication process

Fig. 3. Two important protocols between a user and a payment instance

The protocol starts with an establishment of a TLS connection. We assume that an establishment of a TLS connection results in a fresh symmetric key which is known to both parties. It is important to mention that only the user authenticates the payment instance.

After the establishment process of the TLS connection the user sends her public key encrypted with the session key to the payment instance. The payment instance checks if the key has already been used. If the public key is unknown to the payment instance, it generates a random challenge which is encapsulated in a XML message. After its encryption the encrypted XML messages is sent back

```
<Challenge><DontPanic version="1.0">iNYpjACH7[..]</DontPanic></Challenge>
```

Fig. 4. An example for an unencrypted challenge used in the protocols

to the user. The user hashes the decrypted message including the `<DontPanic>` element of the challenge (see Fig. 4). In the following step the user signs the result with her private key and sends it back to the payment instance. The response to the challenge aims to convince the payment instance that a private key exists which corresponds to the transmitted public key. Thus, if the payment instance is able to verify the signature of the hash, it can assume that a private key exists which corresponds to the received public key. However, this does not necessarily mean that the corresponding communication party owns that key. If the signature is verified successfully the payment instance creates an account and a corresponding account certificate (Fig. 5) which is signed by the payment instance with its private key K_p^{-1}. The account certificate includes the ID of the payment instance, an ID representing the account, a timestamp and the public key K_j. The certificate is sent to the user encrypted with K_{jp}. We denote the account certificate with $Cert_J^P$. The subscript indicates the owner of the certificate/private key and the superscript shows who has signed the certificate.

```
<AccountCertificate version="1.0">
    <AccountNumber>160520966610</AccountNumber>
    <BiID>ECD365B98453B316B210B55546731F52DA445F40</BiID>
    <CreationTime>2009-04-01 13:37:29.968</CreationTime>
    <JapPublicKey version="1.0">[..]</JapPublicKey>
    <Signature>[..]</Signature>
</AccountCertificate>
```

Fig. 5. An example of an account certificate

A closer look on the protocol shows that the user never proves the possession of the session key. This leads to the problem that the session key and the user's public are independent. Thus, a payment instance cannot know if the person who possesses the session key also owns the key pair (K_j, K_j^{-1}). The general problem is depicted in Fig. 6 for the authentication protocol which is discussed in the following part.

4.3 Authentication at the Payment Instance

The first *authentication* protocol takes place between a payment instance and a user. Its objective[5] is to do a mutual authentication between both parties. As

[5] The objectives of the protocols are not stated in the literature, therefore we need to assume reasonable objectives for the protocols.

precondition we assume that a user possesses an account certificate. Additionally, we assume the payment instance to be responsible for the user's account. Fig. 3(b) shows the protocol with the help of a message sequence chart.

The protocol starts, similar to the previous protocol with the establishment of a TLS connection. During the establishment phase only[6] the user authenticates her communication partner. The execution of a TLS handshake results in a fresh session key known to both parties.

In the next step the user sends the signed account certificate to the payment instance which verifies the certificate. If this was successful the payment instance creates a challenge and sends it encrypted with K_{jp} to the user. When the user receives the message she decrypts the challenge, creates a hash of the message and signs the result with her private key. Afterwards the user sends the result encrypted with K_{jp} to the payment instance. After the payment instance has checked the result, an encrypted confirmation is sent back to the user. This facilitates the authentication process between the payment instance and the JAP.

An analysis of the protocol shows that it faces the same problem as the *account creation protocol* (see Fig. 4.2). It does not guarantee that the session key and the private key are owned by the same principal. Due to this a third party can claim the ownership of a user's account even if she does not own the private key (see Sec. 4.6). Fig. 6 points out the general problem of the protocol.

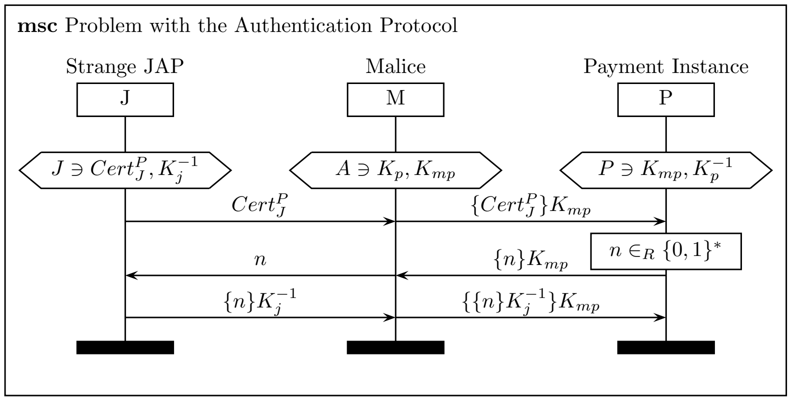


Fig. 6. The general problem of the authentication protocol.

4.4 Charging of the Account & Balance Check

The *charging account* as well as the *balance check* protocol are built on top of the authentication protocol. Thus, both protocols assume that each party owns

[6] Please note, we are not the authors of this protocol, we only describe the protocol. Thus, the reasons for the design are unknown to us.

the session key. Additionally, they require that both participants are mutually authenticated. All messages which are exchanged during the protocol runs are encrypted with the afore negotiated session key.

The purpose of the *charging account* protocol is to provide the user with a transaction ID. The transaction ID can be used by the user as subject for the necessary money transfer, e.g. by a bank transfer. In the rest of the paper this protocol remains unaccounted for.

In the case of the *balance check* the user sends a *balance check request* to the payment instance. As a result she obtains a document stating the current status of the account. The document includes amongst others the amount of used traffic, the remaining credits and a signature of the payment instance confirming the correctness of the data.

4.5 Mix Authentication

The protocol takes place between users and an accounting instance. Since this paper deals with the payment protocols of AN.ON, an analysis of the underlying anonymization and authentication protocols is out of scope. We assume that they work correctly and that the authentication protocol authenticates the mix. As for the TLS protocol we assume that it results in a shared and valid session key between both parties. In addition we assume that only the mix side is authenticated. If the assumptions for the mix authentication protocol are correct is subject of further research.

For the mix authentication only the public keys of the payment instance and the accounting instance are required on the user's side. It does not require any secret on the side of the user.

4.6 Payment Initialization

Presentation of the Payment Initialization Protocol The *payment initialization* protocol requires a successful run of the *mix authentication* protocol.

A run of this protocol aims to verify that a user is in the possession of a valid and charged user account. Hence, a user needs to provide an accounting instance with the user's *account certificate*. In addition, a user needs to prove the possession of the corresponding private key. In order to verify that a user's account is charged the protocol requires also the involvement of a payment instance. This is due to the fact that an accounting instance does not have the knowledge about the current status of a user's account.

Fig. 7 presents the protocol between an accounting instance and a user. The protocol requires a user to possess an own key pair, an account certificate for the public key, the public key of the payment and the accounting instance. A payment instance only knows the public key of the accounting instance, the user's public key and its own key pair. Lastly, an accounting instance needs to own its key pair and the public key of the payment instance. In Fig. 7 it also holds a so-called *cost confirmation*. Such a cost confirmation[7] attests that the user has

[7] It can also be seen as a digital coin.

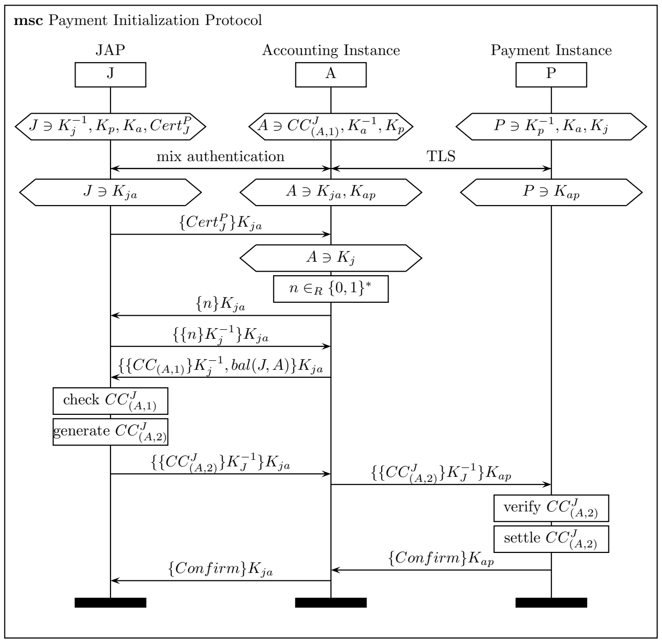


Fig. 7. The payment initialization protocol.

bought traffic from the accounting instance. It contains the overall traffic which a user has consumed on a cascade, an ID of the user's account, an ID of the responsible payment instance and an ID of the cascade for which the certificate was issued. Additionally, it contains a list of hashes referring to *price certificates* of the mixes in a cascade. Finally, it includes a signature of the user.

Price certificates or the hashes of them are needed for the accounting process between a payment instance and the operators of a cascade. It indicates how much money each operator of the cascade receives by anonymizing a GB of traffic. However, the price a user pays for each anonymized GB does not depend on the certificates, it solely depends on the rate which was agreed between the user and the payment instance. Thus, we do not discuss this in greater detail.

For the protocol run in Fig. 7 we assume that the user already bought some traffic on the cascade in a prior session. Thus, the accounting instance already holds an old cost confirmation of the user. A cost confirmation is denoted by $CC_{(X,n)}^Y$. It means that the cost confirmation is issued by a principal Y for a principal X and n is a sequence number for the confirmations. The higher the sequence number the more current is the confirmation.

In the first part of the protocol two session keys are created between the participants. The first session key (K_{ja}) is established between the JAP and the accounting instance with the help of the *mix authentication protocol*. The second one (K_{ap}) is created with a TLS handshake between the accounting instance and the payment instance.

Please note, only the accounting instance verifies its communication partner during the second key establishment process. Thus, the payment instance does not know if it talks with the accounting instance. For both keys (K_{ap}, K_{ja}) we assume that they are valid and uncompromised.

In the following steps the user sends her account certificate encrypted with the session key to the accounting instance. The accounting instance checks the signature of the certificate with the public key of the payment instance. If this succeeds the accounting instance generates a random challenge and sends it to the user. After the user has received the challenge she signs the challenge with her private key (K_j^{-1}). Subsequently, she sends the result back to the accounting instance which verifies the user's signature. If this also succeeds the accounting instance needs to distinguish two cases.

In the first case the user has already used the cascade. Thus, the user has already issued a cost confirmation which can be fetched from the database of the accounting instance. Additionally, the accounting instance retrieves the amount of unused traffic $(bal(J, A))$ which has been bought in a previous session. Both information will be sent encapsulated in a so-called *pay request* to the user. When the user receives the pay request she compares the provided information with her own information. If both information correspond to each other the user creates a new signed cost confirmation on basis of the information.

In the second case the user uses the cascade for the first time. Due to this the accounting does not hold any former cost confirmation which it can present to the user. Instead of an old signed cost confirmation the accounting instance sends a *unsigned cost confirmation*. The unsigned cost confirmation informs the user about some information which needs to be included in the signed version. Basically, this includes the price certificates of the mixes within the cascade, the account number, the ID of the cascade and the number of bytes the user has to buy in advance. The latter needs to be verified by the user if it is a valid (and acceptable) value. If this value exceeds an upper bound[8] the user will not continue the payment initialization process. In addition to the proposed values the user needs to add the ID of the responsible payment instance to the unsigned cost confirmation. Now she only needs to sign the prepared document. Once this is also done, the user has created the initial cost confirmation.

After everything is checked, created and signed the user sends the cost confirmation back to the accounting instance. The accounting instance verifies the signature of the confirmation and checks if the provided information matching the minimum requirements with respect to the amount of prepaid bytes.

Due to the fact that the accounting instance is not able to check the balance of the user's account, it needs to contact the payment instance. The payment instance can decide with the help of a cost confirmation if the account has enough credits left. In order to consult the payment instance, the accounting instance forwards the cost confirmation to the payment instance.

The payment instance verifies the cost confirmation and checks if the account of the user has enough credits left to settle the received cost confirmation. If

[8] Currently this upper bound is at 3 MB.

this is the case the payment instance sends back a confirmation that the cost confirmation was valid and is now settled. When the accounting instance has received a confirmation that the cost confirmation was valid, it confirms this to the user by sending a confirmation message: `<LoginConfirmation code="0">AI login successful</LoginConfirmation>`. This confirmation is also encrypted with the session key.

Analysis of the payment initialization protocol During the payment initialization phase the accounting instance provides the user with the unused but paid traffic volume. This amount of bytes is represented by $bal(J, A)$ in Fig. 7. A possible problem arises if the user does not track the number of bytes which are paid but haven't been used. In this case the accounting instance is able to betray the user. However, we will not address this possible problem in this paper.

Another issue arises due to the fact that in the current protocol version the confirmation of the payment instance is not linked to the cost confirmation which was handed in by the accounting instance. Thus, a replay might be possible under some special conditions, e.g.:

1. the accounting instance hands in multiple cost confirmations during a single TLS session, and
2. the block cipher operates in the *electronic code book(ECB)* mode.

Under these conditions an attacker is able to replay an old confirmation of the payment instance to betray the accounting instance. We believe that it is risky to rely on such details. A possibility to address this is to return a document which refers uniquely to the cost confirmation. The document is signed by the payment instance. This improves the protocol in two ways. Firstly, it circumvents a replay attack, sine the message cannot be used to acknowledge another cost confirmation. Secondly, a signed document gives the operator the possibility to proof later on that the payment instance has confirmed the cost confirmation. Another advantage might be that the encryption between the payment instance and the accounting instance becomes unnecessary. However, this needs to be checked in more detail.

A closer look on this protocol shows that it uses the same authentication process as the authentication protocol between the user and the payment instance. In combination with the problem sketched in Fig. 6 the accounting instance is able to claim the identity of a user.

In order to mount the attack, the accounting instance needs to wait until a user connects to the cascade. Fig. 8 sketches the attack. As in the normal protocol, the process starts with the mix authentication process. The accounting instance authenticates itself to the user. Up to here the accounting instance behaves like an honest one. The attack starts after the user has provided the accounting instance with the user's account certificate. Instead of generating a random challenge, the accounting instance establishes a TLS connection to the payment instance which is referred to in the account certificate. Note, for this operation the malicious accounting instance does not need any secret data.

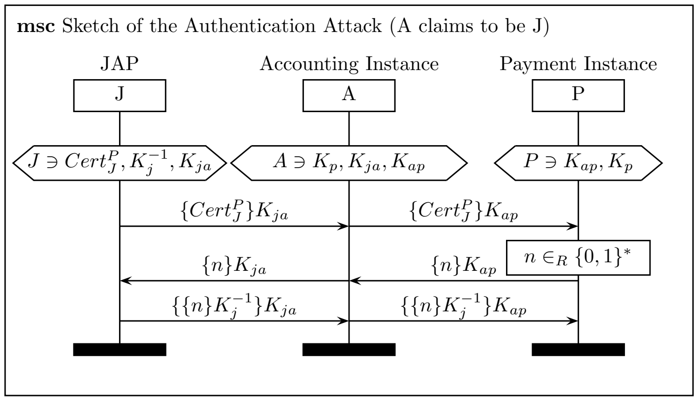

msc Sketch of the Authentication Attack (A claims to be J)
JAP
Accounting Instance
Payment Instance
J
A
P
$J \ni Cert_J^P, K_j^{-1}, K_{ja}$
$A \ni K_p, K_{ja}, K_{ap}$
$P \ni K_{ap}, K_p$
$\{Cert_J^P\}K_{ja}$
$\{Cert_J^P\}K_{ap}$
$n \in_R \{0,1\}^*$
$\{n\}K_{ja}$
$\{n\}K_{ap}$
$\{\{n\}K_j^{-1}\}K_{ja}$
$\{\{n\}K_j^{-1}\}K_{ap}$

Fig. 8. Accounting instances claims to be the user.

When the TLS connection is established the accounting instance simply forwards the received account certificate of the user to the payment instance. The payment instance answers in conformance with the protocol with a challenge. The challenge is forwarded by the accounting instance to the user who subsequently provides the accounting instance with a signed version of the challenge. The answer is forwarded by the accounting instance to the payment instance. Since the response of the challenge includes a valid signature of the user, the payment instance will confirm the successful login to the accounting instance. Thus, the accounting instance is authenticated in the name of the user even though it is not in possession of the user's private key.

Due to the authentication at the payment instance the attacker is now able to retrieve the user's account status. Such an account status contains the number of bytes the user has consumed on the different cascades. In addition it includes the last cost confirmations the user has issued for the different cascades. If any of the cost confirmation has been created without the consumption of traffic[9], the accounting instance is able to use the cost confirmation to consume the bought traffic of the user. In order to do this the malicious accounting instance needs to wait until the user connects again to the cascade. Afterwards it can mount a similar attack as the one described by Fig. 8. The only difference is that the accounting instance does not connect to the payment instance, but rather to the accounting instance which is referred to in the cost confirmation. In section 5 we will deal with the underlying reasons for the attacks.

4.7 Data Exchange and Repurchase of Traffic Volume

The protocol is built upon the payment initialization protocol. Thus, it assumes a shared session key and some amount of confirmed prepaid volume for a user.

[9] This is the case, if the user connects to the cascade but does not send any further packet.

Therefore the user is able to transfer mix packets over the cascade. The client as well as the accounting instance keep track of the number of packets which are transmitted over the cascade.

When the amount of prepaid bytes drops under a predefined *lower limit* the accounting instance sends a new pay request. The pay request is similar to the pay request which is sent during the very first connection to the cascade. Thus, the pay request contains the account number, the ID of the payment instance, the price certificates of the cascade and the (minimum) number of bytes a user needs to confirm. The pay request notifies the client to send a new cost confirmation in order to further use the cascade.

As soon as the client receives such a pay request it checks the received data, adds the ID of the responsible payment instance and compares the number of bytes which need to be confirmed. Normally, the user has already transmitted additional packets since the pay request was issued. Therefore the user usually confirms more than the pay request requires. The result of this process is signed with the user's private key and sent back to the accounting instance.

In contrast to the payment instance the accounting instance can choose whether it forwards the new cost confirmation directly to the payment instance or it waits to submit a later cost confirmation to the payment instance. In the latter case the accounting instance only needs to hand-in the latest cost confirmation. Obviously, this is a trade-off between resources which are needed and the possible loss which arises if a client is malicious. However, the safest way is to submit the cost confirmation directly. Hence, the protocol is equal to the last part of the protocol in Fig. 7. Although, there are some interesting problems, for example what happens if the confirmation of the payment instance arrives after the client's volume is consumed. We will not further discuss them, since they do not directly affect the protocol.

5 Reasons for the Attack

In the previous section we pointed out some weaknesses of the protocols. The first weakness of all the authentication protocols is that a payment instance and an accounting instance respectively does not know if the session key is possessed by the same person who also owns the private key to the account. The second problem is the fact that the challenge-response procedure is transferable to other protocols. The combination of both weaknesses enables an accounting instance to claim the user's identity and at a later stage consume some of the bought traffic. However, the attack is not only restricted to the accounting instance. Also a malicious payment instance is able to mount the attack. In order to fix the protocols both weaknesses must be addressed.

The transferability of the challenge-response procedure can be avoided by the modification of the response. We propose to add the ID of the challenger and an identifier of the protocol which uniquely identifies the protocol and its version. We justify this with the following reasons. Firstly, an ID of the protocol avoids that the challenge can be transfered to another protocol or to an older versions of the protocol. Secondly, the ID of the challenger is added to the response to avoid that another malicious instance can use the response in another run of the protocol, for example the malicious instance can choose the same key as the JAP and therefore replay the messages. This attack would be similar to the one in Fig. 8. However, we do not add the responder's ID since this would not strengthen the properties of the protocol. This is due to two reasons. Firstly, an attacker who is able to read the challenge is probably also able to arbitrarily change the unprotected challenge. Hence, he can also substitute the ID of the responder by the false one. Secondly, the signature in the response implicitly decodes the ID of the responder. If the verifier of the response is honest then he would reject the response to the challenge if it is not verifiable with responder's public key. If the verifier is malicious he can accept the response anyway. We added the information to the response and not to the challenge, since an additional information in the challenge brings no additional security. Nevertheless, it might be useful to add the information also to the challenge due to practical reasons.

In the following we address the problem that arises due to the independence of the session key and the private key. One solution is to use the client authentication capabilities of TLS. Unfortunately, this is not a solution for every used protocol of the payment concept. For instance the payment initialization protocol (and the underlying mix authentication protocol) is not based on TLS.

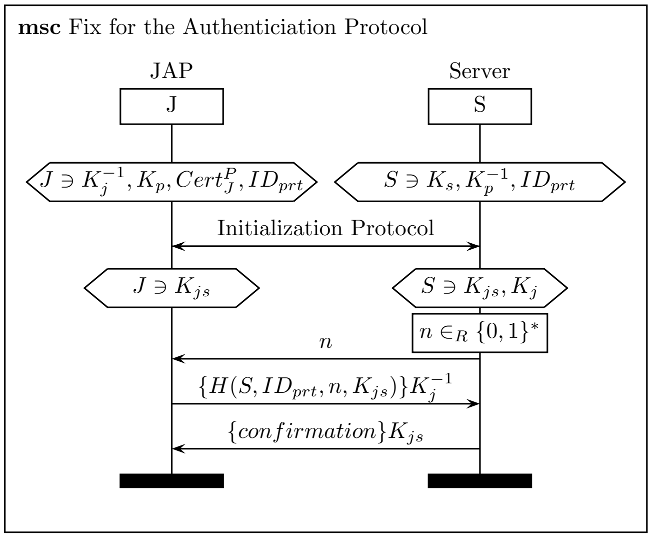

msc Fix for the Authenticiation Protocol
JAP
J
Server
S
$J \ni K_j^{-1}, K_p, Cert_J^P, ID_{prt}$
$S \ni K_s, K_p^{-1}, ID_{prt}$
Initialization Protocol
$J \ni K_{js}$
$S \ni K_{js}, K_j$
$n \in_R \{0,1\}^*$
n
$\{H(S, ID_{prt}, n, K_{js})\}K_j^{-1}$
$\{confirmation\}K_{js}$

Fig. 9. A fix for the (user) authentication protocols

Another solution is to modify the response in the challenge-response procedure in order to bound the session key to the user's public key. Instead that the user hashes only the "normal" response, she can additionally include the session key to the information which she needs to hash and sign. The modified version with all of our changes is shown in Fig. 9.

In Fig. 9 the *initialization protocol* represents the messages which are necessary to authenticate the server and to transmit the user's public key. The server is either the payment instance or the accounting instance. The fixed version of the challenge-response procedure should replace the challenge-response parts of the protocols presented in previous section. It mainly fixes the two encountered weaknesses of the authentication protocols. In addition we have skipped unnecessary cryptographic operations. Neither the challenge nor the response needs to be encrypted with the session key K_{js}. The *confirmation* message represents the different possible replies of the server. Obviously, it depends on the specific protocol which kind of reply the server sends to the user.

If we assume that the client does not blindly signs data and the server side knows that the public key belongs to the user, then the server side can conclude that the user owns the session key and the private key which corresponds to the account certificate. This can also be formalised with the help of the BAN logic[5,4]. We assume the following four preconditions:

1. $S \mid\equiv \#(n)$: the server (S) believes that his challenge is fresh.
2. $S \mid\equiv \overset{K_j}{\mapsto} J$: the server believes that K_j is the public key of the user J.
3. $S \triangleleft J \overset{K_{js}}{\longleftrightarrow} S$: the server has once seen the session key.
4. $S \triangleleft n$: the server has once seen the challenge he has generated.

Finally, we need to model the idealized response from the client to the server. We skipped to model the other messages, since they are not needed for the claim we want to show. Please note, we have skipped the outer encryption with the session key, it does not add any more security. The idealized version of the response is:

$$J \rightarrow S : \{H(J \overset{K_{js}}{\longleftrightarrow} S, n)\}K_j^{-1} \tag{1}$$

By applying the postulates of the BAN logic we can show $S \mid\equiv J \mid\equiv J \overset{K_{js}}{\longleftrightarrow} S$:

$$\cfrac{\cfrac{\cfrac{\cfrac{S \mid\equiv \overset{K_j}{\mapsto} J, S \triangleleft \{H(J \overset{K_{js}}{\longleftrightarrow} S, n)\}K_j^{-1}}{S \mid\equiv J \mid\sim H(J \overset{K_{js}}{\longleftrightarrow} S, n)}, S \triangleleft J \overset{K_{js}}{\longleftrightarrow} S, S \triangleleft n}{S \mid\equiv J \mid\sim (J \overset{K_{js}}{\longleftrightarrow} S, n)}, \cfrac{S \mid\equiv \#(n)}{S \mid\equiv \#(J \overset{K_{js}S}{\longleftrightarrow} S, n)}}{S \mid\equiv J \mid\equiv (J \overset{K_{js}S}{\longleftrightarrow} S, n)}}{S \mid\equiv J \mid\equiv J \overset{K_{js}}{\longleftrightarrow} S}$$

Basically, this is the condition we want to achieve with our modification. Now the server side can assume that the client believes in the session key. Certainly, the result is only valid if prior server authentication protocol results in the assumptions 2 and 3.

6 Conclusion

AN.ON's payment concept utilizes the property of a cascade, every mix in a cascade transfers the same amount of traffic, to establish their prepaid payment concept. In this paper we analysed the most important cryptographic protocols of this interesting approach. The analysis of the protocols showed that they contain several weaknesses which need to be addressed to provide a fair service. In addition we showed some practical implications of the weaknesses. For instance we showed that an accounting instance can abuse its position to consume some portion of the traffic the user has bought on different cascades. Beside this we also propose a solution which fixes the encountered weaknesses of the protocol.

Acknowledgements

We want to thank Rolf Wendolsky and the JonDos GmbH for their help to retrieve the current protocol version and Stig Mjølsnes for his PhD-course in *Cryptographic Protocols and Their Applications* as well as his fruitful comments.

References

1. E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. M. Bellovin. Par: Payment for anonymous routing. In N. Borisov and I. Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 219–236, Leuven, Belgium, July 2008. Springer.
2. O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2001.
3. O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free mix routes and how to overcome them. In *Designing Privacy Enhancing Technologies*, volume 2009/2001 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2001.
4. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, Digital Equipment Corporation Systems Research Center, Palo Alto, Calif., Februar 1989. Revised on 22 February 1990.
5. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. volume 8, pages 18–36, New York, NY, USA, 1990. ACM.
6. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2):84–88, February 1981.
7. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.
8. A. Müller. Entwurf und Implementierung einer Zahlungsfunktion für einen Mix-basierten Anonymisierungsdienst unter Berücksichtigung mehrseitiger Sicherheitsanforderungen. Master's thesis, TU Dresden, Germany, 2002.
9. A. Pfitzmann and M. Hansen. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management - a consolidated proposal for terminology, Feb. 2008. v0.31.
10. R. Wendolsky. A volume-based accounting system for fixed-route mix cascade systems. In *Bamberger Beiträge zur Wirtschaftsinformatik und angewandten Informatik*, pages 26–33, 02 2008.