# Cryptographic Protocol Analysis of AN.ON

Benedikt Westermann[1], Rolf Wendolsky[2], Lexi Pimenidis[3], and Dogan Kesdogan[1,4]

[1] Q2S*, NTNU, 7491 Trondheim, Norway
[2] JonDos GmbH, 93055 Regensburg, Germany
[3] iDev GmbH, 50672 Cologne, Germany
[4] Chair for IT Security, FB5, University of Siegen, 57068 Siegen, Germany

**Abstract.** This work presents a cryptographic analysis of AN.ON's anonymization protocols. We have discovered three flaws of differing severity. The first is caused by the fact that the freshness of the session key was not checked by the mix. This flaw leads to a situation where an external attacker is able to perform a replay attack against AN.ON. A second, more severe, error was found in the encryption scheme of AN.ON. An internal attacker controlling the first mix in a cascade of length two is able to de-anonymize users with high probability. The third flaw results from the lack of checks to ensure that a message belongs to the current session. This enables an attacker to impersonate the last mix in a cascade.
The flaws we discovered represent errors that, unfortunately, still occur quite often and show the importance of either using standardized crytpographic protocols or performing detailed security analyses.

## 1 Introduction

In recent years anonymous communications have become an important building block for privacy-preserving systems. Anonymous channels are often an unconditional requirement for e-voting, e-health or anonymous credential systems. Many techniques have been proposed in theory, for example Tarzan[1] or MorphMix[2]. However, only a few systems have been widely deployed. In terms of number of users, the two major deployed anonymization systems are Tor[3] and AN.ON/JonDonym[5][4].

In general, publications concerning anonymous communications deal with attacks on the network layer, performance improvements or the consolidation of knowledge with regards to anonymous communication in general. Unfortunately, the underlying cryptographic protocols have not received the same attention, despite the fact that anonymity strongly depends on the correct practical combination, usage and implementation of cryptographic primitives.

* *"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`

[5] Also known as "JAP", the name of the client software, or "Java Anon Proxy".

This paper takes a closer look at the cryptographic protocols used for the anonymization process of AN.ON, and discovers several vulnerabilities. The following two sections describe the basic concepts of AN.ON, the attacker model and the underlying assumptions of the system. Section 4 continues with the authentication protocol involving the user and the first server. Section 5 presents an attack on the general encryption scheme used by AN.ON. Section 6 discusses a flaw in the mix initialization protocol. The previous work in this area is presented in Section 7, following a discussion in Section 8 about the reasons for these flaws. Finally, we present our conclusions.

## 2 Description of AN.ON

*AN.ON*, short for *Anonymity Online*, is a project that provides anonymity on the network layer. More precisely, it offers sender anonymity against the receiver of a message and relationship anonymity against a local attacker, such as an eavesdropper of an Internet connection. In terms of web browsing, sender anonymity means that a web server cannot identify a user via their IP address. Relationship anonymity means that a local attacker cannot identify the sender or receiver of a message. Thus, an attacker can at most identify the sender or the receiver, but not both [5].

In order to establish such a service, AN.ON uses the so-called *mix servers* otherwise known simply as *mix*es[6]. A mix is an intermediate entity between a sender and a receiver of a message. Its task is to establish anonymity for the user. A mix accomplishes this by hiding the relation between incoming and outgoing messages. In [4] the authors propose the usage of encryption and reordering of messages to establish anonymity. However, for performance reasons the reordering of messages is typically deactivated in AN.ON.

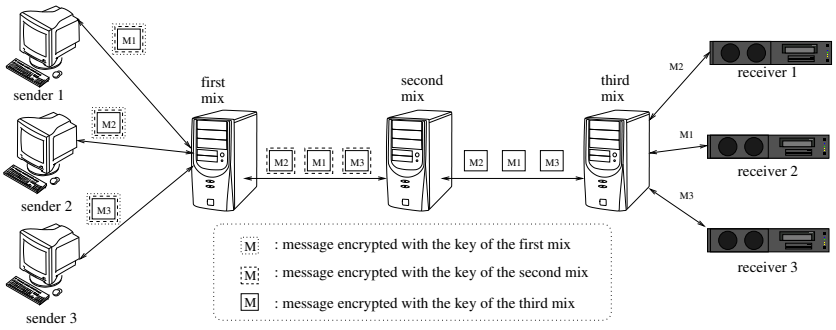


**Fig. 1.** Example for a cascade in AN.ON

To provide relationship anonymity regarding the operator of a mix, several mixes are typically chained together. Such a chain is called a *cascade* (see Figure 1). The order of the mixes is chosen by their operators and cannot be changed by

a user. Every packet which is received by the first mix in the cascade is forwarded to a second mix and so forth. By these mechanisms, AN.ON aims to establish anonymity for a user under the assumption that not all mix operators collude.

The basic anonymization process works in the following way: a sender encrypts a message, which includes the final destination, with a symmetric key. The key is shared between the last mix in a given cascade and the user. The result of the encryption is again encrypted with a symmetric key that is shared with the predecessor of the last mix. This procedure is repeated until the first mix in the cascade is reached and thereby the typical layered encryption is created.

Afterwards the message is sent to the first mix. This mix uses its shared symmetric key to decrypt the message and forwards the result to the next mix, which also decrypts the message. When the last mix in cascade is eventually reached, the mix performs the final decryption of the message and is thus able to get the destination of the message. Finally, the unencrypted message is forwarded to the destination.

In addition to senders, receivers and mixes, there is an additional party, the so-called *infoservice*. This service, operated by a third party, provides users of the system with the necessary information about the cascades.

## 3 Scope, Assumptions and Course

The analysis in this paper looks in detail at the protocols used in the mix server version 00.08.60 of AN.ON. This was the most recent version at the time of our analysis and was released in February 2009.

Our analysis focuses on the anonymization process and ignores the protocols that are involved in the information distribution process, which normally involves the infoservice. Due to this, we assume that a user knows the public signature keys of the mixes. Generally, we assume that every mix uses an uncompromised key pair to sign data.

Our assumed adversary has the capabilities of a *local active attacker*. Thus, the attacker is only able to eavesdrop some connections of either the user, the mixes or the final destination, but not all at the same time. The attacker is also able to add, modify, replay or drop packets passing an observed link. We further assume that the attacker can operate a single mix in the cascade. Finally, we assume that the attacker is not able to break basic cryptographic primitives, such as AES or RSA, but is assumed to possess all prior private keys that are no longer used by the mixes. Note that this attacker model is weaker than the attacker that was originally proposed in [4], in which the authors assume a global attacker. However, recent publications have shown that AN.ON is not able to resist this kind of attacker in practise[7, 8].

In order to retrieve the information about the protocols we made use of a technical report describing the anonymization process[9]. We have also examined the source code in order to find undocumented changes in the protocols. The results of the analysis were discussed with developers of the AN.ON project, who also helped us to retrieve the mix authentication protocol, see Section 6.

## 4 Authentication Protocol of the First Mix

Before we describe the protocol it is necessary to take a brief look at the so-called *descriptors* that are provided by the infoservice. A descriptor is an *XML*-based document which describes the entities of a cascade. It contains general information about the cascade, a description for each mix and a signature for the whole document. The last of these is provided by the first mix in the cascade. The description of each mix is also signed by the corresponding mix. The signature aims to prevent a malicious modification of the mixes' descriptions. The description of a mix includes different public keys, a timestamp and X509 certificates for the included public keys.

In this section of the paper we assume that when a user receives a descriptor with a valid signature of a known authority, they possess every public key of the mixes in the cascade. In addition we assume that all keys are not compromised by an adversary and that the certificates are up-to-date.

The *mix authentication protocol* aims to create a session key between the JAP, which is the client application, and the first mix in a cascade. The preconditions for the protocol are that a user knows the public signature key of the first mix. The mix owns the corresponding key pair, which is only used to produce signatures. In addition to this key pair the mix also possesses a public key pair that can only be used for encryption. However, the key which is used for the encryption is not initially known by the JAP.

Figure 2 shows a message sequence chart of the mix authentication protocol at an abstract level. We assume that the JAP knows the public signing key $K_{M^1}$ of the first mix. The *known* relation is represented by the $\ni$ sign. The mix holds two key pairs: the first key pair is $(K_{M^1}^{-1}, K_{M^1})$ which is used only to sign documents. The second key pair is $(K_{M_e^1}^{-1}, K_{M_e^1})$. Its public key can be used by the JAP to encrypt messages for the mix.

In the first message that is sent by the first mix to the user, a descriptor for the cascade is transmitted. The descriptor includes, among other data, the public encryption key $K_{M_e^1}$ of the first mix. In order to simplify the presentation we represent the remaining information by $m$. For example, the keys of the other mixes are included in $m$. The whole descriptor is signed by the first mix, which is denoted by $\{H(m, K_{M_e^1})\}K_{M^1}^{-1}$. After the message is received by the JAP, it verifies the signature. If this succeeds it also knows the public encryption key of the first mix. In the next step, the JAP generates a symmetric session key and sends the key encrypted with the public encryption key $K_{M_e^1}$ to the first mix. Since the first mix owns the corresponding private key, it is able to decrypt the message. Thus, it also knows the symmetric session key. Finally, the mix sends a confirmation to show the possession of the symmetric key to the JAP. The confirmation is basically a hash of the descriptor, the session key and the public encryption key of the first mix.

A closer look at the protocol shows that a mix has no guarantee that the message containing the session key was created in the current session. An attacker can send, instead of a new message, a recorded message of an old session. This is known as a *replay*. The mix cannot check, in this protocol, whether the session

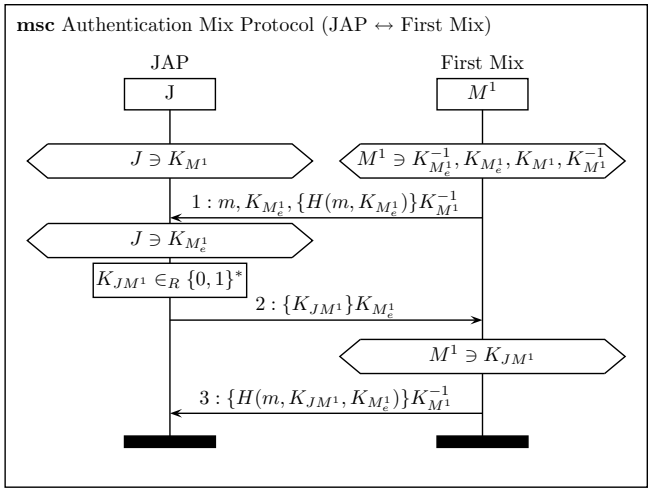


**Fig. 2.** Mix Authentication Protocol

is a replay or not. Thus, the whole anonymization process may be vulnerable to replay attacks.

As we did not find any replay protection in the other protocols, we tried to replay the whole session including several HTTP requests. We used the existing service "Dresden" in order to check if a replay is possible. To this end, we recorded a session of a short sequence of website queries and their replies. For testing purposes, we retrieved a website which was hosted on a server under our control. The second and third website were hosted on foreign servers. The last website again was located on a server which was under our own control.

To replay the session we simply connected to the mix and sent all the previously recorded raw packets in the mix. Shortly after the replay was started we were able to observe HTTP requests on both of our web servers. Both HTTP requests were sent by the last mix of the "Dresden" cascade. Thus, in the version we evaluated there was no protection against replay[6]. This lack of protection leads to various different attacks threatening not only the anonymity of a user with respect to a global attacker[6]. For instance, an attacker could replay a post command which modifies data on the web server in order to threaten data integrity.

The replay attack can be limited to an internal attacker if the mix ensures that the received key is fresh. The simplest solution seems to be to establish a TLS connection between the user and the first mix. However, whether this solution suits AN.ON's requirements is not within the scope of this paper.

This change only protects against an external attacker. In order to protect against replay by an internal attacker other countermeasures are necessary, such

[6] Replay protection is under development, but takes place at a higher level

as the solution described in [10]. Here the authors try to find a trade-off between storing every used key in a database, performance and security. Their idea is to use small parts of a symmetric key to mark a time period. Some parts of the key are predictable, which subsequently lowers the strength of the symmetric key. An identifier of every used key is stored in a database during a given time period. Only keys that are within the current time period, and that are not already stored in the database, are accepted for a new connection.

## 5 Attack on AN.ON's Encryption/Decryption Scheme

In this section we concentrate on the encryption and decryption of exchanged messages.

### 5.1 Structure of Mix Packets

As briefly described in Section 2, a user forwards packets along a chain of mixes. For each hop, the user adds a layer of encryption around the message which is later removed by the corresponding mix.

To analyse the protocol it is necessary to examine the structure of the packets. Figure 3 illustrates the structure of a packet which is sent by a user to the first mix. The first part is the mix packet header. Its size is 6 bytes and it contains a channel ID and some flags. The remaining part can be used to transfer data from the user to the last mix in the cascade. It is important to notice that only the last 992 bytes travel along the whole cascade. The first 6 bytes (channel ID and flags) can be completely modified at each hop. It is therefore necessary to encrypt different parts of the packet in different ways with different keys.

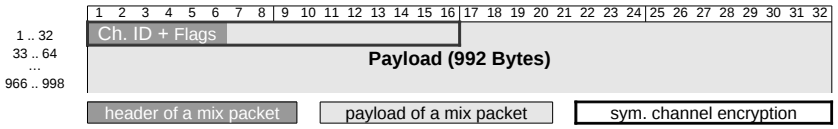


**Fig. 3.** The Structure of a Mix Packet

One encryption layer is added to protect the confidentiality of the channel ID and the flags during the transmission between adjacent entities. Obviously, the symmetric key for the encryption is shared by exactly two adjacent parties. For purposes of speed optimization the whole packet is not encrypted, only the first 16 bytes (128 bit). Thus, this layer of encryption includes the channel ID, the flags and the remaining 10 bytes of the payload of the message. For encryption, AES is used in the *output feedback mode* (OFB, see Section 5.2)).

The structure and the encryption of the payload depend on whether a mix packet is the first in an anonymous channel, or a packet of an already opened channel. If a mix receives a packet it checks, based on the channel ID and the

flags, if the packet opens a new stream. Such a packet is called a *channel-open packet*. In this case, the mix decrypts the first 128 bytes by using its RSA private key[7]. The first part of the decrypted bytes contains a symmetric key (16 bytes). This is used for decrypting both the remaining 864 bytes of the packet and the subsequent packets of the anonymous channel. If the mix is an intermediate mix, the remaining part of the packet is encrypted for the next mix (see Figure 4). Thus, the mix needs to forward the packet to the next mix. Before the packet is forwarded, the mix removes its key (16 bytes) from the packet and adds 16 bytes of random data to the end in order to preserve the length of the packet. Finally, the mix forwards the payload together with a matching header to the next mix. If the mix is the last mix in a cascade it decrypts the first 128 bytes with its private RSA key and the remaining bytes with the symmetric key which is stored in the first 16 bytes of the packet. In addition to the payload a mix packet for a last mix contains a header field after the key. This field indicates how many of the 992 bytes for the payload are used for data, as well as the type of the data. The remaining bytes are random data (see Figure 5).

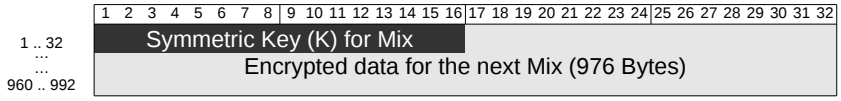


**Fig. 4.** The structure of the payload for intermediate mixes for the initial packet.

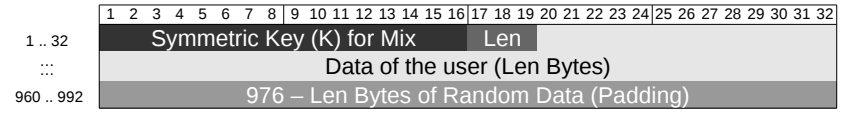


**Fig. 5.** The structure of the payload for the last mix for the initial packet.

In the case of subsequent packets only one encryption scheme is used. The whole payload is encrypted symmetrically with the key that was sent within the channel-open packet. Each mix uses its key to decrypt the 992 bytes of payload. When the packet is decrypted by the last mix, the packet can again be divided in three parts: the header, the data and random data.

In order to process replies, every mix encrypts the data received by the successor mix and forwards it to the predecessor mix. The structure of the mix packet is equal to the structure a subsequent packet. Thus, the whole packet is symmetrically encrypted. The same symmetric keys are used by the mixes for both transmission directions. The algorithm used is AES in OFB-mode.

[7] In the case of the first mix, this part is symmetrically encrypted. Since this is not important for the attack we omit a further explanation.

### 5.2 Output Feedback Mode (OFB)

AN.ON uses AES in OFB mode for its symmetrical encryption operations. The objective of the OFB mode is to produce a infinite key stream. To this end, it uses an initialization vector and a key. The initialization vector is encrypted with AES, which uses the key. The result of this is used twice: first, it is XORed with the plain text. Second, it is used in the subsequent round instead of the initialization vector. Figure 6 illustrates the mode of operation of OFB.

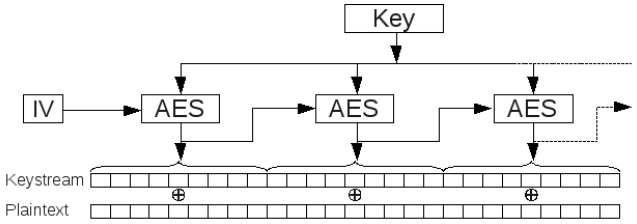


**Fig. 6.** Sketch of the output feedback mode

### 5.3 The Attack

As mentioned above, a mix uses the same key to decrypt messages in the sending direction as well as to encrypt the messages in the receiving direction. Moreover, a mix uses the same initialization vector for both directions. Therefore, a mix produces the same key stream for both directions.

We denote with $k_i^m$ the byte of the $i$-th position in the key stream of mix $m$. Let $d_i$ denote the $i-th$ data byte in the data stream[8] and with $c_i$ the final encrypted message byte at position $i$. To distinguish the sending and receiving direction we use either the superscript $r$ or $s$ for $c_i$ and $d_i$ respectively.

At first we formalize the receiving situation where a packet travels from the last mix along the cascade to the user. Let $p$ denote the position of the last mix that processed the packet and let $n$ be the number of mixes in the cascade. Thus, if the user receives a packet, $p$ is equal to 1 since the first mix was the last involved mix. Let $i$ denote the byte position in the byte stream of a packet. Based on the notation we can describe the $i$th encrypted byte in the receiving direction by:

$$c_i^r(p) = d_i^r \bigoplus_{j=p}^{n} k_i^{m_j} \qquad i \geq 0 \ \wedge \ 1 \leq p \leq n \tag{1}$$

However, if we take a look at the sending direction the situation is slightly different due to the payload structure of the initial packet in a data stream. Recall

[8] The data stream includes the 3 byte header of each packet as well as the padding.

that if an intermediate mix in a cascade receives a channel-open packet in a data stream, the first 128 bytes are asymmetrically encrypted and the following bytes symmetrically. During processing, an intermediate mix removes its key from the payload (and adds 16 bytes to the end). Now the packet is forwarded to the next mix who expects again that the first 128 bytes of the payload are encrypted asymmetrically and the remaining bytes are encrypted symmetrically. Thus, the key streams of the mixes need to be shifted by a user by 16 byte per hop. The reason for the shift is the symmetric key that is stored in the first 16 bytes of the initial data stream packet.

By formalizing the encryption, we result in:

$$c^s_{i+128}(p+1) = d^s_{i+128} \bigoplus_{j=p+1}^{n} k^{m_j}_{i+(n-j)*16} \qquad 0 \leq p \leq n-1 \wedge i \geq (n-p-1)\cdot 16 \quad (2)$$

for the $i$-th encrypted byte.

The equations 1 and 2 become interesting if we consider the first mix in a cascade of length 2. By using both formulas we result in:

$$c^r_i(2) = d^r_i \oplus k^{m_2}_i \quad (3)$$

$$c^s_{i+128}(2) = d^s_{i+128} \oplus k^{m_2}_i \quad (4)$$

The xor of both encrypted values result in:

$$c^r_i(2) \oplus c^s_{i+128}(2) = d^s_{i+128} \oplus d^r_i \quad (5)$$

In case of AN.ON most of the traffic that is transfered over the cascade is normal HTTP traffic, which includes the HTTP header. Moreover, the content of an HTTP header is partially known by the adversary. Thus, an attacker can use the known parts in order to decrypt unknown parts of the HTTP header with help of Formula 5. This becomes critical in AN.ON if the attacker is able to reconstruct the *request line* or the *Host* field in an HTTP request. The former includes the requested URL and the latter the queried host. Therefore, if an attacker is able to reconstruct parts of either the *Host* field or the request line he has deanonymized the user.

For a proof of concept, we have recorded the payload parts of the packets that were either sent from the first mix to the second mix ($p = 1$) or sent from the second mix to the first mix ($p = 2$) in the *Dresden-Dresden* cascade. We therefore assume an internal attacker on the first mix. In order to correct the offset of the sending stream we removed the first 128 bytes of the stream. The result of this was XORed with the receiving stream. To omit the 3 bytes of the payload header we removed the first 3 bytes of the result and XORed it with the most probable HTTP response line *"HTTP/1.0 200 OK\r\n"*[9]. This procedure resulted in the string "st: www.google.de" which is the last part of the *Host* field in the original HTTP request header. Thus, we uncovered the destination of the

[9] There are only a few different possibilities for the response line.

request simply through the use of two recorded encrypted packets that were sent and received by the first mix. Since the first mix in the cascade also knows the IP address of the user, an internal attacker is able to revoke the relationship anonymity without the help of the second mix. Clearly, this contradicts to the objectives of AN.ON.

For our attack we have assumed a local internal attacker. Nevertheless, an external attacker is also able to perform the attack, even though it is slightly more difficult. This difficult is for three reasons: firstly, the attacker does not know the mapping between incoming and outgoing messages. Thus, they cannot map directly the IP address of the sender to the uncovered receiver. Secondly, the attacker cannot use the first 7 bytes of the HTTP response in the payload due to the channel encryption. Thus, they have less information available. Thirdly, the attacker cannot easily recognize which received packet belongs to which sent packet. These constraints are not, in our opinion, a significant challenge. The mapping can be received due to the fact that the packets are processed in a FIFO order. The fact that the attacker misses 7 bytes merely lowers the probability of success slightly. The last challenge can be addressed by probing which received packet leads to a useful output with respect to a recorded sent packet. If we assume the external attacker is able to master the first and third challenge, they are able to deanonymize the user in our example. The attacker is able to retrieve ".google.de" without any further guesses.

### 5.4 Discussion of the Attack

The attack presented above is based on several problems. Firstly, the plain text of the encrypted message is partially known. Secondly, the encryption is a XOR encryption and therefore an encrypted bit only depends on a single bit of the plain text as well as the key stream. Thirdly, the same parameters are used for both directions. The first and the second fact are difficult to avoid due to the design of AN.ON. Thus neither the *cipher feedback (CFB)* nor the *cipher block chaining (CBC)* mode can be used , due to the processing of the initial packet in a data stream. The *electronic codebook (EBC)* mode is also not suitable as it does not hide data patterns. Thus, it is only possible to change the parameters of the encryption, preferably the key. The key streams of both stream directions thus become different. In a conversation with the developer[11] it was mentioned that AN.ON recently became aware of these risks and that changes had been made in order to use different keys. This was independent of our analysis. Hence, the mix software has already been updated to reflect this issue.

## 6 Attack on the Mix Authentication Protocol

In this section we look at the cascade initialization protocol between mixes. The protocol aims to exchange a key with adjacent mixes in the cascade. In addition, it should also mutually authenticate the mixes.

Let $m_1, \ldots, m_n$ the mixes in a cascade. The protocol begins with the establishment of a TCP connection between the mixes $m_i$ and $m_{i+1}$. Note that $m_i$ initiates the connection to $m_{i+1}$.

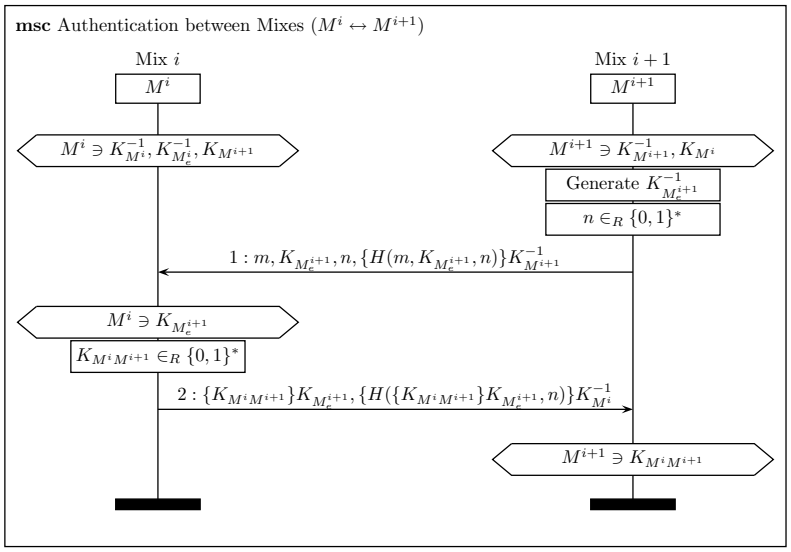


**Fig. 7.** Mix Authentication Protocol

Figure 7 depicts the protocol between two mixes. It starts with the generation of a nonce ($n$) and an asymmetric encryption key ($K^{-1}_{M^{i+1}_e}$) by the mix $i+1$. Afterwards, mix $i+1$ transmits its description ($m$), its public encryption key ($K_{M^{i+1}_e}$), the generated nonce and a signed hash of the triple $(m, K_{M^{i+1}_e}, n)$ to mix $i$. Mix $i$ checks whether the received signature is valid with respect to the known key of mix $i+1$ or not. In the former case mix $i$ generates a session key and encrypts the session key with the received key $K_{M^{i+1}_e}$. The result is sent together with a signature of the encrypted key and the nonce to the mix $i+1$. Mix $i+1$ checks the validity of the signature with respect to the configured public key of mix $i$. If it succeeds mix $i+1$ uses the received key as symmetric key for the channel encryption between the two mixes.

A problem arises when an attacker compromises the private encryption key of the last mix in the cascade, possibly at a later point in time. In this case the attacker is able to replace the certified mix with his own mix. To this end, the attacker needs a recorded session of the authentication protocol in which the compromised key was used. In order to mount the attack the attacker redirects the TCP connection from the certified mix to its own mix. Afterwards, the mix replays the first message of the previous session to the mix $i$. Due to the fact that the signing key is usually changed only once a year, the mix $i$ will most

probably accept the signature of the "certified" mix. In correspondence with the protocol, mix $i$ generates a session key and sends the encrypted session key together with its signature back to the attacker. The attacker who knows the private encryption key can now decrypt the session key and is therefore authenticated in the cascade as certified mix even though he does not possess the signature key. A user is unable to distinguish the attacker from the certified mix at a later date via the existing protocols. Therefore, the attacker is able to eavesdrop on all the outgoing data of the users, which may contain identifying information. However the attacker is not able to deanonymize users solely based on this attack.

At a first glance, the attack looks impractical due to the fact that an attacker needs to compromise one of the private encryption keys. However, if we consider, for example, the recent OpenSSL bug in the Debian Linux distribution[10][12], this attack becomes more practical. The mix software relies on OpenSSL, and thus any asymmetric encryption key generated by a mix which used a vulnerable OpenSSL library is potentially compromised. This means that an attacker can immediately retrieve the private key from a given public key generated by a vulnerable OpenSSL version. Hence, if an attacker once recorded a session in which a mix used a vulnerable key, he is able to impersonate the mix with the attack described above. The only way to circumvent the attack in the current version is to replace every signature key that has been used with a vulnerable OpenSSL version. Obviously, the protocol also needs to be fixed to counter the described attack.

It is worth noting that this protocol is based on the "Key Transport Mechanism 4" of the ISO/IEC 11770-3:2008 standard[13]. The author of the AN.ON protocol modified it slightly in order to authenticate $mix_{i_+1}$ as well. The author therefore included a signed version of the $mix_{i+1}$'s encryption key as well as the descriptor of the mix. In addition he omitted the identity of mix from the encrypted secret, which could lead to other attacks. This example shows how dangerous it is to modify standardized cryptographic protocols to apply them beyond their intended use.

For the protocol in Section 4 we see no reason why a custom-made or a modified standard protocol is necessary for the authentication and encryption. TLS supports client and server authentication via X509 certificates and is additionally able to secure data transmitted later. This protocol should therefore be suitable for the communication and authentication between the mixes. One reason to choose another protocol might be performance, as some data is encrypted unnecessarily in this scheme.

## 7 Related Work

This paper is the first cryptographic protocol analysis of AN.ON's anonymization process. In 2009, Westermann[14] performed a security analysis of AN.ON's payment system, but did not take the anonymization process into account.

[10] A vulnerable version can only generate a limited number of keys

For I2P[11], an anonymously developed anonymization service, there is no published description of the anonymisation protocol available and to the best of our knowledge also no publicly available security analysis.

In contrast to AN.ON and I2P, the cryptographic protocols of the Tor system have been analyzed[3] with the NRL protocol analyzer[15]. In 2006 Goldberg proved that the Tor authentication protocol is secure in the random oracle model[16]. In general, this does not guarantee that the implementation has no flaws with respect to the implementation of the protocol and the cryptographic primitives. An examples of this is that, due to the lack of AES in counter mode in early OpenSSL versions, the Tor developers were forced to implement their own version. Unfortunately, there was a bug in this implementation that caused the counter to be reset after 16 bits. This clearly threatened the security of the system[17].

## 8 Discussion

In the field of low-latency anonymous communications, the main research focus seems to be on mechanisms that establish anonymity or improve performance. Many publications deal only with the general mechanisms for establishing anonymity by using idealized underlying protocols, and omit a clear and detailed cryptographic protocol description. However, most mechanisms are not implemented and thus this lack of detail is a minor problem. As soon as a protocol is implemented, however, it is crucial to publish and analyse the protocols that are composed or invented by the authors.

Tor is a good example of the right way to achieve this. The developers described and analysed the cryptographic protocols in a early stage of the project. Possible changes to the cryptographic protocols are published before they are implemented in Tor. In [18] the authors propose a more efficient way to establish circuits, however to the best of our knowledge this is not implemented yet, but is in discussion to be introduced in a later version.

In general, it is almost always a good idea to use standard cryptographic protocols for a product. However, building an anonymity network solely on standardized protocols, while possible, introduces a number of constraints[19]. In the case of high-latency anonymity networks, with regard to the protocols and mechanisms proposed so far, it seems almost unavoidable to compose cryptographic primitives and invent cryptographic protocols for novel, specific purposes. However, it seems that this area enjoys a stronger focus on proving the correctness of protocols compared to the field of low-latency networks.

Our analysis shows that referring to a technical report for cryptographic protocols is risky. We claim that a technical report is mostly read by developers, who are not necessarily cryptographic protocol experts. As a consequence, weaknesses in the protocols are more likely to be overlooked.

[11] `http://www.i2p2.de`

## 9 Conclusion

In this paper we have analysed the cryptographic protocols of AN.ON and discovered three flaws of differing severity. The first flaw is caused by the fact that the freshness of the session key was not checked by the mix. This flaw leads to a situation where an external attacker is able to perform a replay attack against AN.ON. However, when the replay detection techniques that are currently under development are integrated, the internal as well as the external attacker will no longer able to replay a session. Nevertheless, the flaw in the authentication protocol must be addressed.

A second, more severe, error was found in the encryption scheme of AN.ON. An internal attacker controlling the first mix in a cascade of length two is able to de-anonymize users with high probability. The error was introduced by the reuse of keys with the same initialization vector. As of November 15, 2009, this error is fixed in version 00.08.84 by a slight protocol change. Due to compatibility reasons with older clients, some mix operators have still not updated, but plan to do so soon.

The third flaw discovered is, at a first glance, more theoretical than practical. It does, however, have practical relevance due to the OpenSSL flaw in Debian. The missing check for a message to belong to the current session enables an attacker to impersonate the last mix in a cascade. However, this can only be done if the attacker has compromised a private encryption key of the mix that was signed by the last mix in an older session.

The flaws we discovered represent errors that, unfortunately, still occur quite often. This again shows the importance of using standardized cryptographic protocols. As discussed in Section 8 it is not always possible to use a standard cryptographic protocol due to special requirements. In this case, a composition of cryptographic protocols and primitives becomes necessary. This does not necessarily lead to a secure system, as various examples and attacks in the past have shown[20, 21]. Therefore, a proof or detailed analysis should be provided, as it has been given by Tor or by Sphinx[22].

## References


1. Freedman, M.J., Morris, R.: Tarzan: a peer-to-peer anonymizing network layer. In Atluri, V., ed.: ACM Conference on Computer and Communications Security, ACM (2002) 193–206
2. Rennhard, M., Plattner, B.: Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002), Washington, DC, USA (November 2002)
3. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: USENIX Security Symposium, USENIX (2004) 303–320
4. Berthold, O., Federrath, H., Köpsell, S.: Web MIXes: A system for anonymous and unobservable Internet access. In: Designing Privacy Enhancing Technologies. Volume 2009/2001 of Lecture Notes in Computer Science., Springer (2001) 115–129

5. Pfitzmann, A., Hansen, M.: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management - a consolidated proposal for terminology (February 2008) v0.31.
6. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **4**(2) (February 1981) 84–88
7. Kesdogan, D., Agrawal, D., Pham, V., Rautenbach, D.: Fundamental limits on the anonymity provided by the mix technique. In: SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2006) 86–99
8. Berthold, S., Böhme, R., Köpsell, S.: Data retention and anonymity services – introducing a new class of realistic adversary models. In: The Future of Identity in the Information Society. Volume 298/2009 of IFIP Advances in Information and Communication Technology. (2009) 92–106
9. Köpsell, S.: AnonDienst - Design und Implementierung. Technical report, TU Dresden University (2004)
10. Köpsell, S.: Vergleich der Verfahren zur Verhinderung von Replay-angriffen der Anonymisierungsdienste AN.ON und Tor. In Dittmann, J., ed.: Sicherheit. Volume 77 of LNI., GI (2006) 183–187
11. Köpsell, S.: Private discussion with the developer (May 2009)
12. Common Vulnerability and Exposure: CVE-2008-0166. http://www.cve.mitre.org (2008) Last visited: 15.12.2009.
13. ISO/IEC 11770-3:2008: Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques. ISO, Geneva, Switzerland
14. Westermann, B.: Security analysis of AN.ON's payment scheme. In Jøsang, A., Maseng, T., Knapskog, S.J., eds.: NordSec 2009. Volume 5838/2009 of Lecture Notes in Computer Science., Springer (October 2009) 255 – 270
15. Meadows, C.: The NRL protocol analyzer: An overview. The Journal of Logic Programming **26**(2) (1996) 113–131
16. Goldberg, I.: On the security of the Tor authentication protocol. In Danezis, G., Golle, P., eds.: Privacy Enhancing Technologies. Volume 4258 of Lecture Notes in Computer Science., Springer (2006) 316–331
17. Dingledine, R.: Security and Anonymity Vulnerabilities in Tor: Past, Present, and Future. Talk at DefCon 16 (August 2008)
18. Øverlier, L., Syverson, P.: Improving efficiency and simplicity of Tor circuit establishment and hidden services. In Borisov, N., Golle, P., eds.: Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007), Ottawa, Canada, Springer (June 2007)
19. Panchenko, A., Westermann, B., Pimenidis, L., Andersson, C.: Shalon: Lightweight anonymization based on open standards. In: Proceedings of 18th Internatonal Conference on Computer Communications and Networks, San Francisco, CA, USA (Aug 2009)
20. Simmons, G.J.: Cryptanalysis and protocol failures. Communications of the ACM **37**(11) (1994) 56–65
21. Gligoroski, D., Andova, S., Knapskog, S.J.: On the importance of the key separation principle for different modes of operation. In Chen, L., Mu, Y., Susilo, W., eds.: ISPEC. Volume 4991 of Lecture Notes in Computer Science., Springer (2008) 404–418
22. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (2009) 269–282