Anonymous communication with on-line and off-line onion encoding ⋆

Marek Klonowski, Mirosław Kutyłowski, and Filip Zagórski

Institute of Mathematics, Wrocław University of Technology,
Marek.Klonowski@im.pwr.wroc.pl Miroslaw.Kutylowski@pwr.wroc.pl,
Filip.Zagorski@im.pwr.wroc.pl

Abstract. Anonymous communication with onions requires that a user application determines the whole routing path of an onion. This scenario has certain disadvantages, it might be dangerous in some situations, and it does not fit well to the current layered architecture of dynamic communication networks.
We show that applying encoding based on universal re-encryption can solve many of these problems by providing much flexibility – the onions can be created on-the-fly or in advance by different parties.

Keywords: anonymous communication, onion, universal re–encryption

1 Introduction

Anonymous communication Providing anonymity of communication in public networks is one of the most serious problems in computer security. Many interesting ideas have been presented so far, but still we are far away from solving the problem completely. Perhaps the most prominent proposals are Chaum's DC-networks and MIX networks [3, 2]. Later Rackoff and Simon [14] proposed a protocol in which each user chooses a random route for his message and encrypts the route and the message within a structure that resembles an onion. Due to a cryptographic encoding, the messages that meet in the same node are indistinguishable when leaving this node. This effect is called "mixing" [2] or a "conflict" [1]. So, the idea of anonymity with onions is that many messages travel around the network, meet each other and are recoded so that an adversary gradually looses control over origin points of the messages. The idea of onions became the basic component of Babel [12] and of Onion Routing [7]. (In fact, the name onion was introduced in [7].) Recently, it has been used in the TOR protocol [4].

It may happen that some ingredients of the network might be controlled or monitored by an adversary that tries to break anonymity. Many adversary models have been considered - some of the models allow an adversary to perform only passive traffic analysis based on information obtained from nodes and links under his control [1, 11]. Other models [12, 13] allow active attacks based on adding or delaying messages by an adversary.

⋆ Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS). Accepted for the SOFSEM'2005 conference and publication in Lecture Notes in Computer Science.

Connection based protocols such as Onion Routing face a fundamental problem that breaking one connection at a time effects traffic along exactly one path and therefore once a path disappears it may betray the path used. No fully satisfactory solution addressing this problem has been found yet. In this paper we are concerned with protocol aimed for sending short messages for which security proofs with respect to traffic analysis do exist.

As it was pointed in [1], anonymity level in an onion based protocol is strongly correlated with the number of messages processed by the network and the probability of a conflict/mixing of two or more messages in one node. It turns out that if an adversary may control only a fraction of links, possibilities of traffic analysis are quite limited [1, 11]. On the other hand, if an adversary controls the whole traffic, then the onions provide a low level of anonymity in the case light traffic. Simply, the onions do not meet frequently in this case, so the paths can be easily recovered by the adversary.

The regular onion encoding, as proposed in [14] and used in the later papers, has another disadvantage: a user has to know the whole network in order to be able to choose a truly random path. This assumption is unrealistic in dynamic networks. Moreover, anonymity is in a serious danger if different parties participating in the protocol use different sets of servers for intermediate nodes on the onion paths. We are aware of certain attacks possible in this situation.

Papers [5, 10] introduce new encoding techniques to onions based on universal re-encryption schemes [9]. The idea is that the components processed by the servers can be re-encrypted without any knowledge of the contents and the recipient. However, even then a protocol should be checked carefully - the scheme from [5] has been broken very fast.

New results In our paper we explore new possibilities for design of anonymous communication protocols based on onions encoded with universal re-encryption schemes [10]. First we propose an off-line protocol that allows to prepare a route of a message in advance – the onion routes (or their parts) are created by third parties as a kind of general service. Then, if an application process has to send a message, and it does not know the topology of the network, it can ask for the service mentioned. This solution is aimed for the layered communication architectures. It is also useful in a LAN if there are specialized servers responsible for anonymization messages sent to external locations.

In the second proposal (online merge onions), we show how to move responsibility of determining onion paths to specialized servers that make decisions dynamically (for instance based on the traffic load). It enables to adjust quickly to network conditions. Another important feature of this construction is that it decreases overhead of a message volume due to onion encapsulation.

2 Onions

2.1 Classical Onions

In this section we briefly recall construction of onions. We assume that a network consists of n servers (called nodes); each of them has its own widely accessible public key

and the corresponding secret key. Moreover, we assume that each server can communicate directly with any other server.

A basic onion protocol looks as follows: in order to send a message m to node R, node S chooses at random intermediate nodes $J_1, \ldots, J_\lambda$, and encodes m as an *onion*:

$$\mathrm{Enc}_{J_1}(\mathrm{Enc}_{J_2}(\ldots(\mathrm{Enc}_{J_\lambda}(\mathrm{Enc}_R(m), R), J_\lambda)\ldots), J_3), J_2)$$

(Enc_X stands for encryption with the public key of X). This onion is sent by S to J_1. Node J_1 decrypts the message - the plaintext obtained consists of two parts: the second part is J_2, the first one is an onion with one layer peeled off:

$$\mathrm{Enc}_{J_2}(\ldots(\mathrm{Enc}_{J_\lambda}(\mathrm{Enc}_R(M), R), J_\lambda)\ldots), J_3)\ .$$

Then J_1 sends this onion to J_2. Nodes $J_2, \ldots, J_\lambda$ work similarly, the onion is gradually "peeled off" until it is finally received by node R.

In fact, additional countermeasures are necessary in order to avoid some simple attacks on the onion protocol (for details see for instance [1]):

- We have to use a probabilistic encryption scheme. Otherwise an adversary could establish a permutation between the input and the output of a node by a simple encryption of the whole output batch.
- The size of the onions should be fixed. A kind of padding can be used.

2.2 Universal Re-encryption

We recall universal re-encryption scheme from [9] based on ElGamal encryption. Let G be a cyclic group of order p such that the discrete logarithm problem is hard for G. Let g be a generator of G. Then a private key is a random $x < p$; the corresponding public key is $y = g^x$.

Encryption: In order to encrypt a message m for Alice, Bob generates uniformly at random values k_0 and k_1. Then, the following quadruple is a ciphertext of m:

$$(\alpha_0, \beta_0; \alpha_1, \beta_1) := \left(m \cdot y^{k_0}, g^{k_0}; y^{k_1}, g^{k_1}\right)$$

In fact, (α_0, β_0), and (α_1, β_1) are ElGamal ciphertexts of, respectively, m and 1.

Decryption: Alice computes $m_0 = \frac{\alpha_0}{\beta_0^x}$ and $m_1 = \frac{\alpha_1}{\beta_1^x}$, and accepts a message $m = m_0$, if and only if $m_1 = 1$.

As for the ElGamal scheme, this is a probabilistic cryptosystem – if we encrypt the same message twice, we get two different ciphertexts. Moreover, given two ciphertexts, it is impossible to say whether they were encrypted under the same key, provided that the private key is unknown. This property is called *key-privacy* (see [9]).

ElGamal cryptosystem has another important feature. We can re-encrypt a ciphertext (α, β) so that any relation between the old and the new ciphertext (α', β') is hidden for the observer that has no access to the private key. For the scheme presented above even a **public key** is not necessary – for this reason, it is called *universal re-encryption*, or *URE* for short. The re-encryption procedure looks as follows: First, random values k_0' and k_1' are chosen. Then a re-encrypted version of a ciphertext $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ is obtained as:

$$\left(\alpha_0 \cdot \alpha_1^{k_0'}, \beta_0 \cdot \beta_1^{k_0'}; \alpha_1^{k_1'}, \beta_1^{k_1'}\right)\ .$$

Let $\mathrm{URE}_x(m)$ stand for a ciphertext of m obtained with universal re-encryption scheme, where x is the private decryption key.

2.3 Onions Based on Universal Re-encryption

In [10] the following method of encoding a message m going from A to $B = J_{\lambda+1}$ through path $J_1, \ldots, J_\lambda$ is presented. Let (y_i, x_i) be the pair of public and private key of J_i for $i \leq \lambda+1$. An URE-onion consists of ciphertexts $\mathrm{URE}_{x_1}(J_1)$, $\mathrm{URE}_{x_1+x_2}(J_2), \ldots,$ $\mathrm{URE}_{x_1+\ldots+x_\lambda}(J_\lambda)$ and $\mathrm{URE}_{x_1+\ldots+x_{\lambda+1}}(m)$. These ciphertexts are obtained with the public keys, respectively, $y_1, y_1 \cdot y_2, \ldots, y_1 \cdot \ldots \cdot y_{\lambda+1}$. After creating, these ciphertexts are permuted at random.

Processing an URE-onion consists of two phases: a partial decryption and a re-encryption phase. For instance, J_1 performs the following steps: each URE-ciphertext $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ is replaced during partial decryption by

$$(\alpha_0/\beta_0^{x_1}, \beta_0, \alpha_1/\beta_1^{x_0}, \beta_1)\ .$$

It is easy to see that if $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ is $\mathrm{URE}_{x_1+\ldots+x_i}(w)$, then after the partial description we get $\mathrm{URE}_{x_2+\ldots+x_i}(w)$. One of the ciphertexts obtained is in fact of the form $(J_2, \beta_0, 1, \beta_1)$ and indicates the next destination. Then all ciphertexts (including this of J_2) are re-encrypted and permuted at random before being sent to J_2.

It is easy to see that the encoding described above guarantees that only processing along the path chosen by A guarantees delivery of the URE-onion. Any malicious processing (re-direction, detours, changing the contents) can be detected with high probability and the malicious server can be identified.

3 Features and Protocols Based on URE-onions

In this section we present several important features of URE-onions that can be used in design of anonymity protocols.

3.1 Basic Features

Plaintext insertion after encryption Universal re-encryption inherits the remarkable property of ElGamal encryption scheme: the plaintext may be determined after essential part of encryption computation. Indeed, first we prepare a ciphertext of 1. It has the form $\left(1 \cdot y^{k_0}, g^{k_0}; y^{k_1}, g^{k_1}\right)$. Then we can convert it to a ciphertext of m simply by multiplying the first component by m.

Navigators An onion encoding a special void message $-$, with a starting point A and destination B can be used to encode only a "path". If a node obtains after decoding an onion a message "$-$", it knows that the onion has reached the end of its path. Such an onion will be called a *navigator from A to B* and denoted $\mathrm{Nav}[A, B]$.

Navigators are particularly handy for URE-onions: a so-called *URE-navigator* consists of two parts: the first one is a navigator, say $\mathrm{Nav}[A, B]$, the second part is an

URE-ciphertext obtained with some public key (not necessarily the key of the destination node B) encoding some additional information. Immediately after creation of an URE-navigator the ciphertext encodes 1. Afterwards, when the URE-navigator is used and re-coded, we can replace 1 with an arbitrary message m, as described above. We use notation Nav$[A, B]$ URE$_x(m)$ for such a URE-navigator, where x is the decryption key of the ciphertext of m.

Let us remark that for traditional onions we can add external layers to a navigator Nav$[A, B]$: afterwards the path of the onion would lead from a chosen C to A, and then follow the route defined by Nav$[A, B]$. For standard onion constructions such a modification is possible even, if we get Nav$[A, B]$ from a third party and we cannot disassemble it. For URE-onions such a manipulation is impossible.

3.2 Plain Off-line Scheme

In order to send anonymously a message m from S to R we can simply send an URE-navigator Nav$[S, R]\, URE_{x_R}(m)$. Subsequent servers from the path "peel off" the navigator and re-encrypt message m. Node R can decrypt the ciphertext and retrieve m.

Such a URE-navigator can be called an *off-line onion*, since an empty navigator can be created in advance and as soon as a message m to be sent is ready at application level, an URE-navigator encoding m is created by inserting m into the URE-ciphertext, as described above, and by re-encrypting all ciphertexts of the URE-navigator immediately afterwards (in order to hide m and the navigator used from the party that constructed the navigator).

Replacement Attack Assume that an active adversary controls (actively) the beginning and the end of a path encoded in the navigator. At the beginning of the path, he replaces the URE-ciphertext of the off-line onion by an URE-ciphertext of a random string r encrypted with his own key. Of course, he can trace such a modified onion while it moves through the network. Simply, he decrypts all URE-ciphertexts of the onions with his decryption key – re-encryption does not prevent retrieving r. Once the message arrives at the end of the path, the adversary replaces the URE-ciphertext of r back by the original one and re-encrypts it. The destination node obtains a proper ciphertext and has no idea that the connection was under attack.

One can prevent this attack: instead of URE$_{x_R}(m)$ the sender transmits ciphertext URE$_{x_1+\ldots+x_\lambda}(m)$, where $x_1, \ldots, x_\lambda$ are private keys of the subsequent nodes on the path from S to R. Now, each intermediate node has to decrypt partially (and re-encrypt) the URE-ciphertext obtained. It is easy to see that after this modification the attack described above fails - the URE-ciphertext must be processed by all intermediate servers indicated in the navigator. So the destination node would retrieve a different message. Also due to the partial decryption, the adversary would not detect its own message inserted at the beginning of the path. Indeed, it is difficult to detect any connection between the ciphertexts of the form $(m \cdot (yz)^k, g^k)$ and $(m \cdot z^{k'}, g^{k'})$ knowing the public keys y, z only.

Advantages of the scheme The main point is that the scheme separates encoding the message from encoding the route. It may be useful in many ways:

1. The onions can be prepared in advance.
2. If a sender does not know topology of the network or its knowledge is not up to date, it is better to use navigators offered by trusted servers. In this way we can delegate the chores of creating the routes to a special well protected and administered server. This is quite advantageous since if some users choose intermediate servers in a different way than the rest of the world, then traffic analysis might become easy. Note that all results on traffic analysis [14, 1, 11] require that the intermediate nodes are chosen by all users with the same probability distribution.
3. An empty off-line onion (i. e. one encoding the message 1) can be delivered as a regular message to any node. Then this node can use it as anonymous return-address and send it back without knowing the address of the request source. Of course, such an anonymous reply scheme is possible also with the traditional onions [1], however the present solution does not require the intermediate servers to memorize any values.

The main disadvantage of the scheme is that the server preparing a navigator has to know all pairs (source, destination) used (of course, the user can fetch much more navigators that it uses and in this way hide a particular connection). Hence the solution might be suited for a company, but it is not aimed for a general use.

3.3 Merging Navigators

Using plain off-line onions becomes dangerous, when navigators were created by a server cooperating with an adversary. Even if a direct identification of a navigator in the traffic transmitted is impossible, traffic analysis might provide valuable information. This would be facilitated by the fact that the adversary might know all random paths encoded in the navigators generated by a certain server.

In order to avoid such a situation we propose *merge onions* (MO for short); our protocol shows how to combine navigators from different sources into an onion with a longer path. If the navigators come from different and non-cooperating sources, the resulting onion cannot be traced by an adversary collaborating with only some of these sources.

Creating MO For the sake of simplicity we describe how to compose a MO from two parts (Fig. 1 presents the case in which 4 navigators are used). A sender S wishing to transmit a message m to destination R executes the following steps:

- it chooses two navigators at hand, say $\text{Nav}[J_1, J_\delta], \text{Nav}[L_1, L_\delta]$.
- it composes a merge-onion containing the following components:

$$\text{Nav}[J_1, J_\delta], \text{URE}_x(L_1), \text{URE}_x(\text{Nav}[L_1, L_\delta]), \text{URE}_{x+y}(R), \text{URE}_{x+y+x_R}(m)$$

  where, respectively, x and y are the sums of the description keys related to the navigators $\text{Nav}[J_1, J_\delta], \text{Nav}[L_1, L_\delta]$, and x_R is the decryption key of R.

The way of processing such an onion is clear: first it is sent to J_1. Then it is processed according to the navigator $\text{Nav}[J_1, J_\delta]$; at each step all remaining components

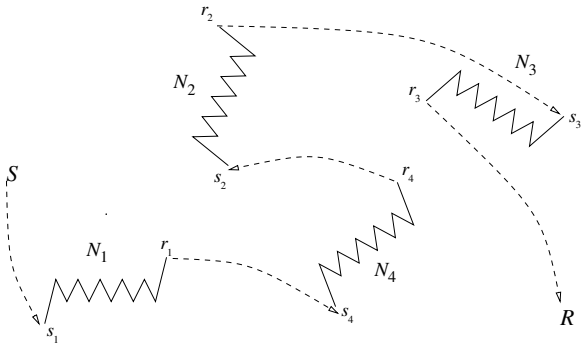

Fig. 1. Composing a merge-onion path from navigators N_1, N_2, N_3, N_4

are partially decrypted and re-encrypted. This lasts until we reach the end of the first navigator. Then the second component reveals L_1 and the last three components are sent to L_1. Then the message follows the route encoded by the navigator Nav$[L_1, L_\delta]$ until it reaches L_δ. Then R is retrieved and the last component (which is a ciphertext of m with the decryption key x_R) is sent to R.

For the protocol described, the adversary can see what is the number of remaining navigators to be used until the end of the path. In order to hide this information we may introduce a simple modification of the protocol. The server, which retrieves the next navigator to be used, does not remove its ciphertext, but re-encrypts it and moves behind the last ciphertext of a navigator.

Advantages of Merge Onions The size of MO grows moderately with the number of navigators used. Each navigator (except one) is represented by a single URE-ciphertext of the navigator and a URE-ciphertext of the starting node of the next navigator.

3.4 Online Merge Onions

Online Merge Onion scheme (OMO), in contrast to Merge Onion scheme, demands from the sender knowledge of a few stable servers in the network that remain working all the time. Navigators are chosen online by the servers selected by the sender. We can think about OMO as a scheme in which sender "asks" some servers to provide anonymity of his message by sending it along routes with many conflicts.

Creating OMO A sender S wishing to transmit a message m to R executes the following steps:

- it chooses k servers $A_1, A_2, ..., A_k$ at random (from a common public list), and creates a navigator $N = \text{Nav}[A_1, ..., A_k]$ encoding the path $A_1, ..., A_k$;
- it inserts a message "to R" into N,
- it creates $\text{URE}_{x_R}(m)$, where x_R is the decryption key of R,

– it chooses an URE-navigator U from a set of available navigators and inserts a message: "to A_1" into it,
– finally, it sends a message:

$$U(\text{to } A_1), \text{URE}_{x_{A_1}}(\text{Nav}[A_1, ..., A_k](\text{to } R)), \text{URE}_{x_R}(m)$$

to the starting node of the navigator U.

Processing OMO There are two cases. If a server D receiving the onion is not on the list $(A_1, ..., A_k)$, then it processes it according to the navigator standing in front of the message and re-encrypts the remaining parts. If $D = A_i$, then

– it decrypts $\text{URE}_{x_{A_i}}(\text{Nav}[A_i, A_{i+1}, ..., A_k])$, so it gets $\text{Nav}[A_{i+1}, ..., A_k]$ and the message: "to A_{i+1}",
– it encrypts the navigator obtained with the public key of A_{i+1}, that is, it gets $\text{URE}_{x_{A_{i+1}}}(\text{Nav}[A_{i+1}, ..., A_k])$,
– it chooses an URE-navigators M and inserts the message "to A_{i+1}" into it.
– it re-encrypts the last part of the message, which is $\text{URE}_{x_R}(m)$,
– it sends concatenation of these parts:

$$M(\text{"to } A_{i+1}\text{"}), \text{URE}_{A_{i+1}}(Nav[A_{i+1}, ..., A_k]), \text{URE}_{x_B}(m)$$

to the starting node of the navigator M.

Protocol features

Adapting onion length to network load Servers can dynamically adopt their behavior to message traffic independently of the senders. It is often believed that using dummy messages to increase the traffic to the maximal amount and keeping lengths of the onion paths fixed is a proper answer to network dynamics. However, proving resilience to traffic analysis when a fraction of servers and lines is malicious depends on how often the onions are processed through honest servers and links[11]. So dummies do not help that much, as one may hope at the first look.

The protocol OMO gives the freedom to adopt the lengths of the paths on-the-fly. So, as shown on picture below, server A can assign to packet m arriving at time t not only a different path in a navigator, but also a different path length. For instance, on the figure below a packet corresponding to the same message m, but arriving at time, say t', will reach server B in 5 steps instead of 11. Moreover, due to re-encryption, those two packets look completely different.

Traffic reduction If a message m to be transmitted is small, then the volume of routing information contained in an onion containing m might be high compared to the volume of m. This disadvantage can be relaxed somewhat through online merge onions: while the total length of the path along which a message is processed is long (preventing a traffic analysis), all the time the message transmitted contains only two (much shorter) navigators.

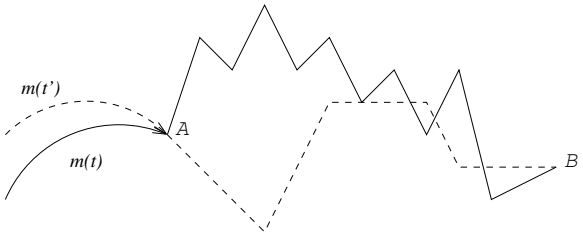

Fig. 2. Adjusting the path length by OMO

Enforcing conflicts and constructing navigators The way of choosing the mix servers is the following. The procedure requires a global list $\mathcal{A}$ of mix servers, known to every protocol participant. When a server A_i has to choose a navigator leading to a server A_{i+1}, it uses an approximation of the number of onions in the network to determine the length of the navigator, say t. The number t can be found through observation of the traffic passing through in the preceding moments. Assume that a mix can process at most z onions at once. Then A_i takes s such that $z = \Theta(t/s + \log s/ \log\log s)$. The mix servers for the navigator constructed by A_i are chosen uniformly at random from the prefix of list $\mathcal{A}$ of length s. Standard "bin and balls" arguments may be applied here to show that a large number of conflicts at mix servers would be generated in this way. We skip a detailed analysis here.

Conclusions

We have shown that universal re-encryption provides many new interesting features:

- possibility to prepare onions in advance,
- adaptiveness to network traffic,
- size reduction of the auxiliary parts of onion messages,
- possibility to process the onions through arbitrary chosen mixes,
- implementing onions in a layered architecture of a distributed, dynamic system.

Let us compare the parameters used by the schemes. Necessary path length λ for each of the schemes depends on assumptions about adversary model. If an adversary can corrupt only a constant fraction of navigator sources, essentially the same analysis applies as in the case of [11]. So we consider the same (global) path length λ for 3 schemes considered below.

	Classical Onions	Merge Onions	Online Merge Onions
message size	$O(\lambda+\|m\|)$	$O(\lambda+\|m\|)$	$O(k+\lambda/k+\|m\|)$
end-user encoding cost	$O(enc(\lambda\|m\|))$	$O(k\cdot enc(\lambda/k)+enc(\|m\|))$	$O(enc(\lambda/k+\|m\|))$
preprocessing possible	no	yes	partially
processing cost at a server	$O(enc(\lambda+\|m\|))$	$O(enc(\lambda+\|m\|))$	$O(enc(\lambda/k+\|m\|))$
messages tracing*	easy	easy	hard
repetitive attack**	easy	easy	harder
traffic change	–	moderate increase	decrease
required knowledge of network topology	full	none	limited
traffic adaptiveness	no	no	yes

* at low traffic, by a passive adversary who controls all links

** at any traffic, by an active adversary

References

1. Berman R., Fiat A., Ta-Shma A.: Provable Unlinkability Against Traffic Analysis, Financial Cryptography 2004, LNCS , Springer-Verlag
2. Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms, CACM 24(2) (1981), 84-88
3. Chaum, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability, Journal of Cryptology 1.1 (1988), 65-75
4. Dingledine R., Mathewson N., Syverson P., *Tor: the Second Generation Onion Router*, USENIX Security, 2004
5. Fairbrother, P.: An Improved Construction for Universal Re-encryption, Privacy Enhancing Technologies'2004, LNCS , Springer-Verlag.
6. Frankling, M., Haber, S.: Joint Encryption and Message-Efficient Secure Computation, Journal of Cryptology 9.4 (1996), 217-232.
7. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding Routing Information. Information Hiding '1996, LNCS 1174, Springer-Verlag, 137-150.
8. Golle, P.: Reputable Mix Networks, Privacy Enhancing Technologies '2004, LNCS , Springer-Verlag.
9. Golle, P., Jakobsson, M., Juels, A., Syverson, P.: Universal Re-encryption for Mixnets, RSA-CT'2004, 163-178.
10. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: Anonymous Communication Immune against Repetitive Attack, Workshop on Information Security Applications (WISA)'2004, LNCS , Springer-Verlag, to appear.
11. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: Provable Unlinkability Against Traffic Analysis already after $\mathcal{O}(\log(n))$ Steps!, Information Security Conference (ISC)'2004, LNCS 3225, Springer-Verlag, 354-366.
12. Gülcü, C., Tsudik, G.: Mixing E-mail with BABEL, ISOC Symposium on Network and Distributed System Security, IEEE 1996, 2-16.
13. Kesdogan D., Egner J., Büschkes R.: Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System, Information Hiding '98, LNCS 1525, Springer-Verlag, 83-98.
14. Rackoff, C., Simon, D.R.: Cryptographic Defense Against Traffic Analysis, ACM STOC25 (1993), 672-681.