Onions Based on Universal Re–Encryption - Anonymous Communication Immune Against Repetitive Attack [⋆] [⋆⋆]

Marcin Gomułkiewicz, Marek Klonowski, and Mirosław Kutyłowski

Institute of Mathematics, Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27
50-370 Wrocław, Poland
`Marcin.Gomulkiewicz@pwr.wroc.pl`, `klonowski@im.pwr.wroc.pl`,
`Miroslaw.Kutylowski@pwr.wroc.pl`

Abstract. Encapsulating messages in onions is one of the major techniques providing anonymous communication in computer networks. To some extent, it provides security against traffic analysis by a passive adversary. However, it can be highly vulnerable to attacks by an active adversary. For instance, the adversary may perform a simple so–called *repetitive attack*: a malicious server sends the same massage twice, then the adversary traces places where the same message appears twice – revealing the route of the original message. A repetitive attack was examined for mix–networks. However, none of the countermeasures designed is suitable for onion–routing.

In this paper we propose an "onion-like" encoding design based on universal re-encryption. The onions constructed in this way can be used in a protocol that achieves the same goals as the classical onions, however, at the same time we achieve immunity against a repetitive attack. Even if an adversary disturbs communication and prevents processing a message somewhere on the onion path, it is easy to identify the malicious server performing the attack and provide an evidence of its illegal behavior.

Keywords: anonymous communication, unlinkability, onion, universal re–encryption, repetitive attack

1 Introduction

1.1 Anonymous Communication

Providing anonymous communication in public networks is a problem of growing importance. Demands for anonymity emerge both in the personal sphere and in e-commerce. In recent years a lot of papers about safe (or even provably anonymous) communication have appeared. Many protocols have been proposed - the most significant ones are DC-networks and MIX networks introduced by David Chaum [4, 3]. Later Rackoff

[⋆] Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

[⋆⋆] Copyright: Springer-Verlag, this paper appears in Lecture Notes of Computer Science in a volume devoted to WISA'2004 (Workshop on Information Security Applications) and Simon proposed a fairly practical scheme providing anonymity based on an idea of Chaumian MIXes [14]. That was the first time when onions were explicitly used however, not under this name. This scheme was examined in details [13, 11] in different adversary models.

The idea of onions was used in a number of protocols, e.g. Babel [12] - a protocol aimed at anonymous email transfer, or Onion Routing protocol [7–9] - a protocol in which the connection between two peers is established via an anonymous path: servers on the path get information only on immediate predecessors and successors on the path. Recently, Fairbrother [6] has proposed a scheme based on onions for sending long messages (however, the scheme has turned out to be insecure). The onion mechanism is used also in TOR protocol (the second generation onion routing) [5] as one of the basic building blocks.

1.2 Active Attacks - Repetitive Attack

In many papers (i.e. [14]) it is assumed that an adversary can only eavesdrop the network and observe messages coming in and out of the servers, but cannot decode the packets, initiate new messages and/or destroy the old ones. While the assumption about an inability to read messages can be easily fulfilled using encryption techniques, preventing creation or deletion of messages is extremely difficult (for obvious reasons we cannot rely on the mechanisms such as PKI.)

If an adversary is given the possibility to send new messages of his choice, he can often compromise anonymity in the system. In order to trace the route of any given message it suffices to send it again, and search for double occurrences of identical messages, which shall ultimately reveal the identity of the final recipient. This attack is called a "repetitive attack": it was proposed to compromise mix-networks ([3].) In that case the problem is well studied and resolved: to avoid such an attack several solutions have been suggested. Unfortunately, these methods might be inadequate for the protection of the onion communication protocols.

In Onion–Routing a repetitive attack is always effective when an adversary controls all links between servers and no precautions have been used.

1.3 New Results

In this paper we propose a new simple encoding scheme of "onions" immune against a repetitive attack and similar attacks leading to tracing messages. We call them URE-onions, since our solution is based on an extension of Universal–Re-Encryption by Golle, Jakobsson, Juels and Syverson described in [10]. By using this technique we are fairly able to limit the possibility of a repetitive attack – if a message is sent for the second time, it is re-encrypted at random at each point of the path. Therefore, the adversary cannot detect any repetition. Moreover, in the case when a server inserts faults into messages transmitted, it can be detected with an overwhelming probability and the evidence can be provided easily.

The new way of encoding the onions does not solve the problems that arise due to the traffic analysis of dynamic connections based on onions – these problems are a major issue for anonymous communication protocols – but it seems no encoding scheme can solve them.

2 Onions And Their Weaknesses

2.1 Onion Encoding

The goal of the onions is to protect communication so that the recipients and the sender cannot be linked by an adversary analyzing the network traffic. In the scheme we consider a network consisting of n servers. We assume that each server can communicate directly with other servers (like in P2P networks.)

Each server has a public and a private key, all public keys are widely accessible. The simplest version of the onion protocol looks as follows: in order to send a message m to server D, server S chooses intermediate servers at random, say $J_1, \ldots, J_\lambda$, and then encodes m as an *onion* (Enc_X means encryption with the public key of X):

$$\text{Enc}_{J_1}(\text{Enc}_{J_2}(\ldots(\text{Enc}_{J_\lambda}(\text{Enc}_D(m), D), J_\lambda)\ldots), J_3), J_2)\ .$$

This onion is sent by S to J_1. Node J_1 decrypts the message - the plaintext obtained consists of two parts: the second one is J_2, the first one is an onion with one layer peeled off:

$$\text{Enc}_{J_2}(\ldots(\text{Enc}_{J_\lambda}(\text{Enc}_D(m), D), J_\lambda)\ldots), J_3)\ .$$

Then J_1 sends this onion to J_2. Nodes $J_2, \ldots, J_\lambda$ work similarly, the onion is "peeled off" until it finally arrives at D.

The general idea is that a server processing an onion (and an adversary tracing the traffic) cannot read the content of an onion, only the consecutive decoding with appropriate private keys gives each intermediate server sufficient information to route the message.

In fact, additional countermeasures must be taken to avoid some simple attacks (see for instance [3]):

- For an outgoing sub-onion O sent by server J_i, an adversary may attach "J_i" to O, encrypt the result with the public key of J_i, and compare the result with the messages received by J_i a step before to find out the source of O. One can prevent such an attack by attaching a random string at every layer of an onion or by using a probabilistic encryption scheme.
- The length of the onions should be fixed (otherwise the size could reveal the route of a message); some kind of padding can be used to cope with this problem.

The onion protocol acts similarly to a network of mixes: if two onions enter the same (honest) server simultaneously, an adversary cannot determine the relation between incoming and outgoing onions. Determining the number of rounds necessary for the protocol to ensure anonymity in this way is a challenging problem. Some discussion on the topic can be found in [14, 1, 11].

2.2 Adversary Model

The goal of an adversary might be a communication interruption and/or an unauthorized access to information. Our concern here is an anonymity breach, that is linking the senders and recipients of encoded messages in an anonymous communication protocol. In many papers only passive adversaries are considered: they can observe (eavesdrop) some communication links and servers, but cannot interfere with the traffic in any way. Unfortunately, in a real world, this assumption is often too strong, e.g. in a typical Ethernet network sending packets without authorization is relatively very easy. Such "pirate" packets, geared at confusing legitimate parties of the protocol, may even appear to be sent according to the original protocol.

In our paper we assume that an adversary controls all links and some servers. This is quite a pessimistic model. Moreover, the adversary is "global," which means he always has the knowledge of all corrupted system components and can use them arbitrarily. In particular he has an access to all the private keys of controlled servers.

2.3 Repetitive Attack For Onions

The goal is to establish a connection between the sender and the recipient of a message. To carry out this attack an adversary sends a traced message twice from a controlled server and then observes the traffic on all links. When a certain message show up twice somewhere, it could be the message duplicated by the adversary (partially decoded according to the onion protocol). Then an adversary can immediately see the route of the message. An adversary can also send a copy of a message any time later and trace its route by comparing traffic in controlled links when the original message and its copy were sent.

2.4 Ways To Protect Against Repetitive Attack

In the case of onions, invariant elements are the onions that have to be sent further and the random strings (included in the onion to cope with the attack that was mentioned at the end of Section 2.1.) Unfortunately, we cannot remove the random strings or recode all layers of the onion simultaneously – at least for the classical onions. The reason is that such a recoding procedure should be performed without the knowledge of public keys, and certainly must be performed without the knowledge of private keys. Moreover, re-coding should be performed on internal parts of an onion, while on the other hand an intermediate server has no access to the internal parts of the onion processed. In fact, this is a fundamental feature of the encoding scheme.

For mixing networks there are some simple countermeasures against a repetitive attack. The first one requires the sender to prove his knowledge of what he is sending; obviously, since the layers are peeled off one by one, and we do not want to disclose their contents in any way, such a proof is not possible directly in the case of onions.

The second countermeasure is discarding any duplicate messages. Unfortunately, since an adversary may re-send a message at any later time, this would require from every server a lot of space for storing all traffic (or at least some fingerprints) that passes through it. There would also be a time overhead in processing the traffic - each single onion would be checked for re-occurrence. Perhaps the most worrying aspect is that recording traffic by servers would make eavesdropping much easier than before – coping the records could be much easier.

The second technique can be enhanced by appending to each package some information about the intervals of time when it should arrive at subsequent servers. In this solution packages "out of date" are simply discarded, so the servers can collect data from a short period of time only. This approach presented in [2] demands quite precise synchronization of time (otherwise one can mount an attack based on observing which messages get discarded).

MIX-networks work in "rounds." In such systems we can propose to change keys or parameters in order to make a repetitive attack impossible. Unfortunately, it is not a practical solution for distributed communication systems which have to work continuously.

Handshake based solutions, like the one used in TOR protocol, have a disadvantage of a large latency and bidirectional communication.

3 Onions Based on Universal Re-Encryption

3.1 Universal Re–Encryption

Let us recall El-Gamal encryption scheme: p is an appropriate prime number (with a hard discrete logarithm problem), g is a generator of $\mathbb{Z}_p^*$, a random, nonzero $x < p-1$ is the private key, the corresponding public key is y, where $y = g^x \bmod p$. A message $m < p$ is encrypted in the following way. First a number k, $0 < k < p-1$, is chosen uniformly at random. Then we put $r := g^k \bmod p$ and $s = m \cdot y^k \bmod p$. The pair (s, r) is a ciphertext of m.

The El-Gamal cryptosystem has a very useful property: the same message encrypted for the second time yields a different ciphertext. Moreover, given two ciphertexts, it is impossible to say whether they were encrypted under the same key (unless, of course, the decryption key is given). This property is called *key-privacy* (see [10]). El-Gamal cryptosystem has yet another interesting feature. Everyone 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 not equipped with the decryption key. Namely, if y is the public key used for ciphertext creation, one can choose k' at random and set $\alpha' := \alpha \cdot y^{k'} \bmod p$, $\beta' := \beta \cdot g^{k'} \bmod p$. Obviously, (α', β') is a ciphertext of the same plaintext as before, but both its parts are "blinded" by random factors $y^{k'}$ and $g^{k'}$.

It is an astonishing feature that the above re-encryption trick can be modified slightly so that the public key does not need to be known ([10]): the inventors of the scheme, Golle, Jakobsson, Juels and Syverson, call it *universal re-encryption*, or *URE* for short. The scheme looks as follows:

– **Preliminaries** A cyclic group G is chosen such that the discrete logarithm problem is computationally hard (e.g. $\mathbb{Z}_p^*$ for an appropriate prime number p). An arbitrary generator of G (say g) is chosen. Then G and g are published.
– **Key setup** Alice chooses a private key x at random; then the corresponding public key y is computed as $y = g^x$.

– **Encryption** To encrypt message m for Alice, Bob generates numbers $0 < k_0, k_1 < |G|$ uniformly at random. Then, the ciphertext of m is computed as a quadruple:

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

In fact, this is an El-Gamal encryption made twice: the encrypted messages are m and 1, respectively.

– **Decryption** Alice computes

$$m_0 := \frac{\alpha_0}{\beta_0^x} \quad \text{and} \quad m_1 := \frac{\alpha_1}{\beta_1^x} \ ,$$

and accepts message $m = m_0$, if and only if $m_1 = 1$.

– **Re-encryption** Random values k_0' and k_1' are chosen. Re-encrypted message is described by the following formula:

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

During re-encryption all four components of a ciphertext change in a provably secure way (see [10]).

3.2 Extension of Universal Re–Encryption

Let us assume that we have chosen a path of λ servers. We would like to encrypt a message so that it must be processed by the servers from the path and according to the order on the path. Simultaneously, we would like to retain the outstanding features of regular URE.

– **Key setup** Let x_i be the private key of the ith server ($1 \leq i \leq \lambda$). Let $y_i = g^{x_i}$ be the corresponding public key; y_i is published. Obviously each server determines his keys on its own and does not need to cooperate with others in this phase.

– **Encryption** To encrypt a message m, two random values k_0 and k_1 are generated. The ciphertext has the following form:

$$\begin{aligned} E_{x_1,x_2,\ldots,x_\lambda}(m) &= (\alpha_0, \beta_0; \alpha_1, \beta_1) = \\ &= \left(m \cdot (y_1 y_2 \ldots y_\lambda)^{k_0}, g^{k_0}; (y_1 y_2 \ldots y_\lambda)^{k_1}, g^{k_1}\right) \end{aligned}$$

Hence,

$$E_{x_1,x_2,\ldots,x_\lambda}(m) = \left(m \cdot g^{k_0 \cdot \sum\limits_{i=1}^{\lambda} x_i}, g^{k_0}; g^{k_1 \cdot \sum\limits_{i=1}^{\lambda} x_i}, g^{k_1}\right) \ .$$

So $E_{x_1,\ldots,x_\lambda}(m)$ is a ciphertext with decryption key $\sum_{i=1}^{\lambda} x_i$, and therefore it can be re-encrypted in the usual way. At any moment such a ciphertext can be partially decrypted. For instance, the first server can do it as follows:

$$E_{x_2,\ldots,x_\lambda}(m) = \left(\frac{\alpha_0}{\beta_0^{x_1}}, \beta_0; \frac{\alpha_1}{\beta_1^{x_1}}, \beta_1\right)$$

It is obvious that it is still a correct URE ciphertext with decryption key $\sum_{i=2}^{\lambda} x_i$, and therefore it can also be re-encrypted as it was mentioned above.

4 Modified Onion Protocol

In this section we show how to introduce non-determinism into processing of onions. For this purpose we modify the encoding used and propose so called *URE-onions*. We use a notation similar to those used in the previous section. Let x_i denote a secret key of server S_i; the corresponding public key $y_i = g^{x_i}$ is widely known. Also, g is a public parameter, it is a generator of a group such that finding discrete logarithms is computationally hard. Let $E_x(m)$ denote a ciphertext of a message m obtained with a public key corresponding to x according to schema of Golle *et al.*

As a first step, a random path of servers is chosen: $S_{i_1}, S_{i_2}, \ldots, S_{i_\lambda}$. Then a URE-onion consists of λ ciphertexts, called *blocks*. The jth block, for $1 \leq j \leq \lambda - 1$, has the form:

$$E_{x_{i_1}+\cdots+x_{i_j}}(\text{“send to } S_{i_{j+1}}\text{”})$$

The last block has the form

$$E_{x_{i_1}+\cdots+x_{i_\lambda}}(m)$$

The main difference between URE-onions and the classical onions is that we deviate from the original encapsulation idea: the messages for different routing steps are included in separate ciphertexts.

Another feature that differs this approach from the classical one is that we exclude any random contents from the intermediate messages. Obviously, random strings included in a message would betray duplication of messages and hence also some information on the route. The general rule is that auxiliary messages may contain only information that could be available for an adversary analyzing the traffic.

4.1 Routing

First, all blocks described above are sent together to server S_{i_1}. When a server S_j receives a URE-onion, it partially decrypts, re-encrypts, and changes the order of its blocks:

Partial Decryption Phase: each block $(\alpha_0, \beta_0; \alpha_1, \beta_1)$ is replaced by

$$D_{x_j}\left(E_{x_j}(m_i)\right) = \left(\frac{\alpha_0}{(\beta_0)^{x_j}}, \beta_0; \frac{\alpha_1}{(\beta_1)^{x_j}}, \beta_1\right) .$$

Re-Encryption Phase: now S_j re-encrypts each block. So in place of the original block $(\alpha_0, \beta_0; \alpha_1, \beta_1)$ we obtain for some randomly chosen k_1, k_2:

$$\left(\frac{\alpha_0}{(\beta_0)^{x_j}} \cdot \left(\frac{\alpha_1}{(\beta_1)^{x_j}}\right)^{k_1}, \beta_0 \cdot (\beta_1)^{k_1}; \left(\frac{\alpha_1}{(\beta_1)^{x_j}}\right)^{k_2}, (\beta_1)^{k_2}\right) .$$

Permutating Phase: All blocks are permuted at random.

After the decryption phase, exactly one block should contain the next destination S, unless the URE-onion has reached its target. It is easy to notice that the length of the URE-onion remains fixed and the server processing a URE-onion cannot say how many hops remain. Also, re-encryption guarantees that the address of S, the next server on the route, remains hidden for all servers except S_j.

At the next step the URE-onion is sent to the destination S retrieved from a block at a partial decryption phase.

4.2 Immunity against Repetitive Attack

Let us argue shortly why a repetitive attack does not work for the proposed protocol. Assume that there is at least one honest server "on the path" between two malicious servers controlled by an adversary. Then the adversary cannot detect the repetition of an onion, since the honest server re-encrypts each onion processed at random.

We have also eliminated all the information available for intermediate servers except the next destination. Sending a URE-onion multiple times would only increase the number of URE-onions with message "send to S_i," and so an adversary can only hope to provide additional data to the traffic analysis, which is not our concern here. Note that a URE-onion provides no more data for the traffic analysis than the regular onions.

4.3 Attempts to Change the Route

Since the blocks on a URE-onion are given, a malicious server can reorder them, eliminate some of them, or inject its own blocks:

Reordering: Since at each phase (except the last one) there is exactly one block that represents a valid server name, the order of the blocks is irrelevant. This has no effect on the protocol security.

Inserting own blocks: certainly, one can inject a number of blocks encoding initial servers on a path. This is possible, since encoding is based on public keys only. Then the URE-onion will be routed through such a "detour". The problem is that if at least one server on this detour is honest and performs partial decryption then the original blocks will be partially decrypted unnecessarily. Consequently, the blocks of the original URE-onion become unreadable. If the additional blocks are inserted somewhere in the middle then the processing will go on until the first inserted block is encountered. Then the inserted block will be unreadable as well due to partial decryptions that have occurred in between.

Removing a block: if a block is removed then at some point some server, say S_{i_j} will not find the ciphertext

$$E_{x_{i_j}}(\text{“send to } S_{i_{j+1}}\text{”})$$

in the delivered blocks. Certainly, the server S_{i_j} cannot find the next server on the path so it must stop processing this URE-onion. Potentially an adversary can also remove some blocks and then insert new ones. We address this problem in the next subsections.

Modification of a block: in fact, it is possible to change the contents of a block without decrypting it. We also discuss this problem in the next subsection.

4.4 Multiplicative Attack

An adversary can carry out an a bit more sophisticated attack than a repetitive attack. We call it a “multiplicative attack.”

Let a URE-onion contain blocks $E_{k_i}(m_i)$ for $i \leq \lambda$. Each of them, except a single one, is dedicated to a particular server and keeps information about its successor on the path. Let

$$E_{k_i}(m_i) = (\alpha_0, \beta_0; \alpha_1, \beta_1) = (m_i \cdot y^{k_0}, g^{k_0}; y^{k_1}, g^{k_1})$$

A malicious server processing the URE-onion can choose some block blindly, say the block $E_{k_i}(m_i) = (\alpha_0, \beta_0; \alpha_1, \beta_1)$, and replace α_0 by $\alpha_0 \cdot \gamma$ for an arbitrary γ. In such a situation one of the further servers on the path obtains $\gamma \cdot m_i$ instead of m_i. If this server is also under the adversary’s control, it knows the value γ so it can easily recover the value m_i and can carry on the protocol. If this server is not under control of the adversary, then m_i remains scrambled and the server finds that the address decoded is faulty.

During the attack described an adversary can manage to get some information about an onion path. However, if the first of the attacking servers misses an opportunity to disturb a proper block, the URE-onion will not be delivered to the final destination. So the multiplicative attack is less efficient than the repetitive attack for the regular onion protocol, but it is still unacceptable. For this reason we propose an “investigation” subprotocol to defend the scheme against the mentioned attack.

4.5 Investigation – Finding out Dishonest Servers

If an honest server obtains an invalid URE-onion (i.e. none of blocks or more than one decrypted block represent the name of the next server on the route or a valid message) it can complain about the previous server from the path. In such a situation both servers - the previous server as well as the complaining one – must prove that they have behaved correctly, otherwise one of them is recognized guilty. If they manage to prove their compliance with the protocol, the next predecessor on the path is interrogated. The procedure is repeated until a cheater is detected. The main goal is to build an appropriate procedure for verifying a server. We assume that each server knows from whom it gets each packet and that it can prove it to other servers. The evidence might come for instance from the signed hash values of the traffic transmitted.

Let us consider a single server S_j from the path. It has a private key x_j such that $y_j = g^{x_j}$. Each block $(\alpha_0, \beta_0; \alpha_1, \beta_1)$ of the URE-onion should be processed by S_j in two phases – a partial decryption phase and a re-encryption phase. Assume also that S_j is asked to prove its honest behaviour. It must show that the URE-onion obtained from $(\alpha_0, \beta_0; \alpha_1, \beta_1)$ through the partial decryption and re-encryption is correctly built i.e. it has the form:

$$(\widehat{\alpha_0}, \widehat{\beta_0}; \widehat{\alpha_1}, \widehat{\beta_1}) = \left(\frac{\alpha_0}{(\beta_0)^{x_j}} \left(\frac{\alpha_1}{(\beta_1)^{x_j}} \right)^{k_1}, \beta_0(\beta_1)^{k_1}; \left(\frac{\alpha_1}{(\beta_1)^{x_j}} \right)^{k_2}, (\beta_1)^{k_2} \right)$$

for some randomly chosen k_1, k_2. For verification, the numbers k_1 and k_2 are revealed, as well as

$$(\alpha'_0, \beta_0; \alpha'_1, \beta_1) = \left(\frac{\alpha_0}{(\beta_0)^{x_j}}, \beta_0; \frac{\alpha_1}{(\beta_1)^{x_j}}, \beta_1 \right) .$$

but of course x_j must remain secret. The re-encryption phase can be checked in a straightforward way. For examining partial decryption we use a zero-knowledge protocol for showing the equality of discrete logarithms[15]: recall that the aim of a protocol called EQDL$(A, B, C; a, b, c)$ is to prove that there is a number x such that $A = a^x, B = b^x, C = c^x$. So the server proving its behaviour presents a proof

$$\text{EQDL}(\alpha_0/\alpha'_0, \alpha_1/\alpha'_1, y_j; \beta_0, \beta_1, g) .$$

Since $y_j = g^{x_j}$ the proof should convince that $\alpha_0/\alpha'_0 = \beta_0^{x_j}$ and $\alpha_1/\alpha'_1 = \beta_1^{x_j}$ which was our goal.

Let us note that the EQDL proof scheme is not really interactive, so it is better suited for showing honesty afterwards.

A server is immediately rejected from the protocol if it fails to prove its correct behaviour. The only drawback of this method is the necessity of storing all random parameters that have been used by re-encryption (or ability to reconstruct them from a random seed). Fortunately, the time of storing them can be limited to some time bound within which an onion normally reaches its target.

5 Concluding Remarks

Thanks to Universal Re-Encryption scheme URE-onions are immune to a repetitive attack. Any attempt of a multiplicative attack can be detected with a probability proportional to the ratio of honest servers in the whole network. Also changing a block leads to detection of a dishonest server with a significant probability so active tracing of an URE-onion becomes a very risky business. Moreover, URE-onions do not require additional interaction except for the case of “cheating investigation,” and even in this case the interaction is kept minimal. Of course the described scheme does not automatically ensure security against all theoretically possible active attacks.

The new onions might be more expensive than the original ones regarding processing time: each server must perform λ decryptions of λ blocks instead of one decryption of a large block, as it happens for the regular onion protocol.

A serious disadvantage is that an URE-onion cannot be combined with symmetric encryption in the same way as it can be done for regular onions. So it might be suited for small (e.g. control) messages only. On the other hand, URE-onions offer much more flexibility that can be used for diverse purposes.

References

1. Berman, R., Fiat, A., Ta-Shma, A.: *Provable Unlinkability Against Traffic Analysis*, Financial Cryptography’2004, Lecture Notes in Computer Science 3110, Springer-Verlag
2. Büschkes, R., Egner, J., Kesdogan, D.: *Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System*, Information Hiding Workshop ’1998 Lecture Notes in Computer Science 1525, Springer-Verlag, pp. 83-98
3. Chaum, D.: *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*, Communications of ACM 24(2) (1981) pp. 84-88

4. Chaum, D.: *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*, Journal of Cryptology 1.1 (1988), 65-75
5. Dingledine, R., Mathewson, N., Syverson, P.: *Tor: the Second Generation Onion Router*, USENIX Security, 2004
6. Fairbrother, P.: *An Improved Construction for Universal Re-encryption*, Privacy Enhancing Technologies '2004, Lecture Notes in Computer Science , Springer-Verlag.
7. Goldschlag, D. M., Reed, M. G., Syverson, P. F.: *Hiding Routing Information*, Information Hiding Workshop '1996, Lecture Notes in Computer Science 1174, Springer-Verlag, 137-150
8. Goldschlag, D. M., Reed, M. G., Syverson, P. F.: *Private Web Browsing*, Journal of Computer Security, Special Issue on Web Security 5 (1997), 237-248
9. Goldschlag, D. M., Reed, M. G., Syverson, P. F.: *Anonymous Connections and Onion Routing*, IEEE Journal on Selected Areas in Communication, 1998, 16(4):482-494
10. Golle, P., Jakobsson, M., Juels, A., Syverson, P.: *Universal Re-encryption for Mixnets*, RSA Conference, Cryptographers' Track, 2004, 163-178
11. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: *Provable Unlinkability Against Traffic Analysis Already After $\mathcal{O}(\log(n))$ Steps!*, 7th Information Security Conference (ISC'2004), Lecture Notes in Computer Science , Springer-Verlag
12. Gülcü, C., Tsudik, G.: *Mixing E-mail with BABEL*, ISOC Symposium on Network and Distributed System Security, IEEE 1996, 2-16
13. Jakobsson, M., Juels, A.: *An optimally robust hybrid mix network*, 20 ACM Symposium on Principles of Distributed Computing 2001, 284-292
14. Rackoff, C., Simon, D. R.: *Cryptographic Defense Against Traffic Analysis*, 25 ACM Symposium on Theory of Computing (1993), pp. 672-681
15. Schnorr, C.P.: *Efficient signature generation by smart cards*, Journal of Cryptology 4, 1991: 161-174