

# Attacking and Repairing the Improved ModOnions Protocol\*

Nikita Borisov<sup>1</sup>, Marek Klonowski<sup>2</sup>,  
Mirosław Kutylowski<sup>2</sup>, and Anna Lauks-Dutka<sup>2</sup>

<sup>1</sup> Department of Electrical and Computer Engineering,  
University of Illinois at Urbana-Champaign  
nikita@uiuc.edu

<sup>2</sup> Institute of Mathematics and Computer Science, Wrocław University of Technology  
{Marek.Klonowski,Mirosław.Kutylowski,Anna.Lauks}@pwr.wroc.pl

**Abstract.** In this paper, we present a new class of attacks against an anonymous communication protocol, originally presented in ACNS 2008. The protocol itself was proposed as an improved version of ModOnions, which uses universal re-encryption in order to avoid replay attacks. However, ModOnions allowed the *detour attack*, introduced by Danezis to re-route ModOnions to attackers in such a way that the entire path is revealed. The ACNS 2008 proposal addressed this by using a more complicated key management scheme. The revised protocol is immune to detour attacks. We show, however, that the ModOnion construction is highly malleable and this property can be exploited in order to redirect ModOnions. Our attacks require detailed probing and are less efficient than the detour attack, but they can nevertheless recover the full onion path while avoiding detection and investigation. Motivated by this, we present a new modification to the ModOnion protocol that dramatically reduces the malleability of the encryption primitive. It addresses the class of attacks we present and it makes other attacks difficult to formulate.

## 1 Introduction

Mix networks have been a popular scheme for anonymous communication since they were first introduced by Chaum [2]. The basics of the design are that messages are protected by a layered (“onion”) encryption, with each mix in the path removing a layer of encryption and shuffling a batch of incoming messages before forwarding them onwards. This way, each mix only knows its predecessor and successor in the forwarding path of a message transmitted through the mix, and an outside observer cannot link inputs of the mix to the outputs due to encryption.

The idea of mixes is very appealing and became the basic component of major anonymous communication protocols. However, despite of the importance of the problem, we are still far away from providing an ultimate solution that

---

\* Partially supported by Polish Ministry of Science and Higher Education, grant N N206 2701 33.

would provide a satisfactory resilience to attempts of breaking anonymity. The main reason is that in a practical setting an adversary has many other attack possibilities than merely observing the incoming and outgoing messages of a mix. This includes issues such as traffic analysis (static and dynamic) as well as active attacks where an adversary may inject messages or modify them. So far, research on anonymous communication protocols is a step-by-step advance, where protocol proposals (dealing with certain classes of problems) are followed by new attack methods.

**Replay attack, ModOnions and detour attacks.** Two rogue mixes can carry out a *replay* attack, where one mix will re-send one or more copy of a message, and the other look for duplicate incoming messages. This way the two mixes can detect that they are on the same forwarding paths even if they are separated by many honest mixes. The traditional defense to this attack is to have each mix look for and discard message duplicates [5], requiring the mixes to maintain state. Note that most of anonymous routing protocols are not immune against replay attack. Even the most popular TOR protocol (introduced in [6]) can be affected by elaborated forms of reply attack as reported in [13]. TOR is quite different from the regular Onion Routing and to protect integrity of a message it uses enumerating packages and labeling streams. Such an approach protects to some extent from reply attacks, however makes TOR vulnerable to different statistical attacks. The fact remains, however that TOR is the most secure implemented solution. ModOnions [9] take an alternate approach, using Universal Re-Encryption (URE) [8] to re-randomize the messages (“onions”), such that duplicate copies of an onion cannot be linked. Re-encryption of such onions is possible, since instead of a single onion with a message hidden at each “layer” of the onion, there is a group of onions to be processed together, each encoding a different routing information for a path. When processed properly, each node on the path gets information from one of these onions and re-randomizes the rest. Last not least, ModOnions can be signed by some intermediate servers (for instance in order to prevent spam) and the signatures can be re-encrypted while processing [10].

ModOnions addressed the replay attacks, but it turned out that they are susceptible to the *detour attack* [4], where a ModOnion is redirected to go back to the attacker after each routing step, and a mix is used as a decryption oracle. Klonowski et al. presented a defense against the detour attacks by modifying the key management scheme and using different keys for the final decryption [11].

## 1.1 Results

**New attacks.** We show that the improved scheme presented in [11] is still vulnerable to redirection attacks that allow the recovery of the forwarding path. Our first attack uses the fact that a form of oracle decryption is still possible even in the modified scheme. It is no longer possible for an attacker to learn the next hop in a path, but he can verify a guess of a forwarding path if he controls both the first and last node. The number of guesses depends on the size

of the network and the length of the forwarding paths; the attack is feasible for a medium-sized network of 50 nodes using paths of length 5, but quickly becomes impractical for larger parameters.

Our second attack relies on *malleability* of the URE scheme, which makes it possible for an attacker to modify the encrypted plaintext of a message without knowing the key under which it is encrypted. This makes it possible to selectively modify an onion and use probes to recover its structure and learn the next hop, all while avoiding detection. This attack requires many fewer probes and is practical for most network sizes.

**Patches.** Using previous observations we propose a new extension to ModOnions that drastically limits the malleability of the scheme such that any modification to the plaintext will be detected with high probability. This makes the odds of success of our attacks negligible, as a large number of probes must all be modified in such a way as to escape detection. By introducing this integrity check into the ModOnion protocol, our extension should make the design of new attacks more difficult as well.

## 2 ModOnions Protocol from [11]

In this section we recall the improved version of the ModOnions protocol (Onion Routing with Universal Re-Encryption) from [11]. This protocol uses as a building block an extension of Universal Re-Encryption recalled below. At first let us recall details and properties of Universal Re-Encryption (from [8]).

### 2.1 Universal Re-encryption

Universal Re-Encryption is based on the ElGamal encryption scheme. Construction of this encryption scheme is based on a cyclic group  $G$ , where discrete logarithm, DDH problems are hard. Namely, let  $p, q$  be prime numbers such that  $p = 2q + 1$  and let  $g$  be the generator of  $G$ , which is the subgroup of  $\mathbb{Z}_p^*$  of order  $q$ . A private key is a non-zero  $x < q$  chosen uniformly at random, the corresponding public key is  $y$ , where  $y = g^x \bmod p$ . For a message  $m < p$ , a ciphertext of  $m$  is a pair  $(s, r)$ , where  $r := g^k \bmod p$  and  $s := m \cdot y^k \bmod p$  and  $0 < k < q$  is chosen at random.

The ElGamal scheme is a probabilistic one: the same message encrypted for the second time yields a different ciphertext with overwhelming probability. Moreover, given two ciphertexts, it seems to be infeasible in practice to say whether they have been encrypted under the same key (unless, of course, the decryption key is given). This property is called *key-privacy* (see [8]). ElGamal cryptosystem has yet another important property. Everyone can re-encrypt a ciphertext  $(\alpha, \beta)$  and get  $(\alpha', \beta')$  where  $\alpha' := \alpha \cdot y^{k'} \bmod p$ ,  $\beta' := \beta \cdot g^{k'} \bmod p$  for  $k' < q$  chosen at random and the public key  $y$ . Moreover, without the decryption key it is infeasible to find that  $(\alpha, \beta)$  and  $(\alpha', \beta')$  correspond to the same plaintext.

In [8] Golle et al. proposed a slightly modified version of this scheme that is called universal re-encryption scheme or *URE* for short. It consists of the following procedures:

**Setup:** A generator  $g$  of a cyclic group  $G$  of prime order is chosen, where discrete logarithm problem and DDH assumption is hard. Then  $G$  and  $g$  are published.

**Key generation:** Alice chooses a private key  $x$  at random; then the corresponding public key  $y$  is computed as  $y = g^x$ .

**Encryption:** To encrypt a message  $m$  for Alice, Bob generates uniformly at random values  $k_0$  and  $k_1$  ( $k_0, k_1 < p$ ). Then, the ciphertext of  $m$  is a quadruple:  $(\alpha_0, \beta_0; \alpha_1, \beta_1) := (m \cdot y^{k_0}, g^{k_0}; y^{k_1}, g^{k_1})$ . Let us note that this is a pair of two ElGamal ciphertexts with plaintext messages  $m$  and 1 (neutral element of  $G$ ), respectively.

**Decryption:** Alice computes  $m_0 := \frac{\alpha_0}{\beta_0^x}$  and  $m_1 := \frac{\alpha_1}{\beta_1^x}$ . Message  $m_0$  is accepted if and only if  $m_1 = 1$ .

**Re-encryption:** Two random values  $k'_0$  and  $k'_1$  are chosen. Then we compute:  $(\alpha_0 \cdot \alpha_1^{k'_0}, \beta_0 \cdot \beta_1^{k'_0}; \alpha_1^{k'_1}, \beta_1^{k'_1})$ , which is a ciphertext of the same plaintext.

From now on we assume that  $E_x(m)$  denotes an URE ciphertext of a message  $m$  for a secret decryption key  $x$ . Note that there are many possible values for  $E_x(m)$ , since URE is a probabilistic encryption scheme.

## 2.2 Extension of Universal Re-encryption

Let us assume that there are  $\lambda$  distinct servers on each routing path, each server  $s_i$  has a private key  $x_i$  and the corresponding public key  $y_i = g^{x_i}$ . To encrypt a message  $m$ , which should go through the nodes  $s_1, \dots, s_\lambda$ , first values  $k_0$  and  $k_1$  are generated at random. Then the ciphertext has the following form:

$$E_{x_1+x_2+\dots+x_\lambda}(m) = (\alpha_0, \beta_0; \alpha_1, \beta_1) := (m \cdot (y_1 y_2 \dots y_\lambda)^{k_0}, g^{k_0}; (y_1 y_2 \dots y_\lambda)^{k_1}, g^{k_1}).$$

Obviously,  $y_1 y_2 \dots y_\lambda$  is a kind of *cumulative* public key, since

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

Moreover,  $E_{x_1+\dots+x_\lambda}(m)$  is a ciphertext of  $m$  with the decryption key equal to  $\sum_{i=1}^{\lambda} x_i$ . Hence it can be re-encrypted in a regular way. Moreover, such a ciphertext can be *partially decrypted*, for instance, by the first server  $s_1$ . Namely, it computes  $E_{x_2+\dots+x_\lambda}(m)$  as the following quadruple:

$$(\alpha'_0, \beta'_0; \alpha'_1, \beta'_1) := \left( \frac{\alpha_0}{\beta_0^{x_1}}, \beta_0; \frac{\alpha_1}{\beta_1^{x_1}}, \beta_1 \right).$$

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

## 2.3 Description of ModOnions Protocol from [11]

The core idea of protocol introduced in [11] is that each server  $s$  has two different pairs of keys:

- transport keys: a private key  $x_s$  and a public key  $y_s := g^{x_s}$ ,
- destination keys: a private key  $x_s^*$  and a public key  $y_s^* := g^{x_s^*}$ .

Now the blocks encoding intermediate nodes on the routing path  $s_1, s_2, \dots, s_\lambda$  are constructed as follows:

$$\begin{aligned}
 & E_{x_{s_1}^*}(\text{send to } s_2), \\
 & E_{x_{s_1} + \dots + x_{s_{i-1}} + x_{s_i}^*}(\text{send to } s_{i+1}) \quad \text{for all } 2 \leq i \leq \lambda - 1, \\
 & E_{x_{s_1} + \dots + x_{s_{\lambda-1}} + x_{s_\lambda}^*}(m, t), \quad \text{where } t \text{ is a current time.}
 \end{aligned}$$

Note that:

- $\lambda$  is a static parameter of the protocol,
- $E$  denotes encryption scheme with properties as in [9, 11],
- for each block one of destination keys  $x_i^*$  is used and only once; moreover, it is the destination key of the final recipient of the information stored in the block.

**Routing.** When a server  $s$  gets Modified ModOnion the following steps of the protocol are executed:

1. Server  $s$  copies all blocks of a ModOnion. Then it decrypts all blocks with its private destination key.
2. If every previous server on the path is honest, then after decryption exactly one of the blocks should contain a correct message.

**Case 1:** one of the decryptions yields a correct name of the next server  $s'$ .

Then:

- (a) All blocks (as obtained from the previous server), except for the one containing  $s'$ , are decrypted with the private transport key of  $s$ . The blocks obtained are then re-encrypted in the regular way.
- (b) A random block replaces the block containing  $s'$ .
- (c) The resulting blocks are permuted at random.
- (d) The ModOnion obtained in this way is sent to  $s'$ .

Note that the number of blocks in a ModOnion remains unchanged.

**Case 2:** after decryption  $s$  obtains one meaningful message with destination  $s$ . Then the message is delivered.

**Case 3:** the result of decryption for all blocks is meaningless. Then the investigation procedure is launched.

**Investigation procedure.** Investigation procedure is a part of the protocol launched by the node detecting dishonest behavior of other nodes. In this procedure consecutive nodes from the path proves correct processing of the ModOnion. It is assumed that malicious node, once caught is permanently excluded from the protocol. Detailed description can be found in [9].

### 3 Attacks on the ModOnions Protocol from [11]

In this section we present two attacks on the improved ModOnions Protocol described in the previous section.

### 3.1 Attack 1: Guessing the Path

The modification to the ModOnions ensured that a router no longer acts as a decryption oracle for the destination key, thus making it impossible to carry out the detour attack. However, the router still acts as a decryption oracle for the transport key. This lets the attacker verify a guess for the path the onion will take. Suppose that an onion is sent to  $s_6$  along a path  $s_1, s_2, s_3, s_4, s_5$ , with  $s_1$  and  $s_5$  being malicious. The onion that arrives at node  $s_1$  will have the following form (we skip the permutation of blocks for clarity of presentation):

$$\begin{aligned} & E_{x_{s_1}^*}(\text{send to } s_2), \\ & E_{x_{s_1}+x_{s_2}^*}(\text{send to } s_3), \\ & E_{x_{s_1}+x_{s_2}+x_{s_3}^*}(\text{send to } s_4), \\ & E_{x_{s_1}+x_{s_2}+x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\ & E_{x_{s_1}+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_6), \\ & E_{x_{s_1}+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(m, t). \end{aligned}$$

$s_1$  can no longer use  $s_2$  to reveal the next router in the path. However, it can use it as a decryption oracle to partially decrypt the message using the transport key  $x_{s_2}$ . If  $s_1$  then guesses that  $s_3$  and  $s_4$  are the next routers on the path, it can use them as decryption oracles as well. After this, the onion will include the block  $E_{x_{s_5}^*}(\text{send to } s_6)$ . By performing a trial decryption with key  $x_{s_5}^*$  (which is available to the attacker since  $s_5$  is compromised), the attacker can learn both that the guess of  $s_3$  and  $s_4$  is correct and that  $s_6$  is the ultimate destination of the message.

We next describe the attack in more detail.

1. After receiving the onion,  $s_1$  partially decrypts it with  $x_{s_1}^*$  to learn the destination  $s_2$ .
2. Then  $s_1$  uses  $x_{s_1}$  to partially decrypt the remaining blocks and gets:

$$\begin{aligned} & E_{x_{s_2}^*}(\text{send to } s_3), \\ & E_{x_{s_2}+x_{s_3}^*}(\text{send to } s_4), \\ & E_{x_{s_2}+x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\ & E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_6), \\ & E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(m, t). \end{aligned}$$

Let us call this onion  $O_1$ .

3. URE has the property that given a ciphertext encrypted with key  $x$ ,  $E_x(m)$ , it is possible to produce a new ciphertext encrypted under the key  $x+x'$ , as long as  $x'$  is known. (One can think about this as a partial decryption under the key  $-x'$ .) This allows  $s_1$  to wrap the blocks of  $O_1$  in an extra layer of encryption so that these blocks are blindly partially decrypted and passed along by  $s_2$ . The new onion includes the original blocks of  $O_1$  (after blinding them) as well as a new destination block:

$$\begin{aligned}
& E_{x_{s_2}^*}(\text{send to } s_1), \\
& E_{x'+x_{s_2}^*}(\text{send to } s_3), \\
& E_{x'+x_{s_2}+x_{s_3}^*}(\text{send to } s_4), \\
& E_{x'+x_{s_2}+x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\
& E_{x'+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_6), \\
& E_{x'+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(m, t).
\end{aligned}$$

4. This onion is then sent to  $s_2$ . Node  $s_2$  partially decrypts with  $x_{s_2}^*$  to learn that  $s_1$  is the next hop.<sup>1</sup> Note that the second block will remain encrypted under  $x'$  and so  $s_2$  will only find one plaintext block, therefore no investigation will be started.
5.  $s_2$  partially decrypts the onion with  $x_{s_2}$  and forwards the following blocks to  $s_1$ :

$$\begin{aligned}
& \text{random,} \\
& E_{x'+x_{s_2}^*-x_{s_2}}(\text{send to } s_3), \\
& E_{x'+x_{s_3}^*}(\text{send to } s_4), \\
& E_{x'+x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\
& E_{x'+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_6), \\
& E_{x'+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(m, t).
\end{aligned}$$

6.  $s_1$  partially decrypts the onion with  $x'$ , obtaining a new onion  $O_2$ :

$$\begin{aligned}
& \text{random,} \\
& E_{x_{s_2}^*-x_{s_2}}(\text{send to } s_3), \\
& E_{x_{s_3}^*}(\text{send to } s_4), \\
& E_{x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\
& E_{x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_6), \\
& E_{x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(m, t).
\end{aligned}$$

Note that  $O_2$  contains all the blocks of  $O_1$ , partially decrypted with the key  $x_{s_2}$ . We will call the steps 3–6 an oracle decryption, writing  $O_2 = D_{x_{s_2}}(O_1)$ .<sup>2</sup>

7. Now the attacker can proceed with guessing the path. For each (honest) router  $s_i$ , with  $i \neq 2$ , the attacker performs an oracle decryption to obtain  $O_{s_i} = D_{x_{s_i}}(O_2)$ .<sup>3</sup>

<sup>1</sup>  $s_2$  may get suspicious about forwarding an onion back to  $s_1$ , but technically,  $s_5$  or any other malicious node can act as the next hop to avoid this problem.

<sup>2</sup> This process is very similar to the oracle decryption proposed by Danezis [4].

<sup>3</sup> A minor caveat is that  $O_2$  is one block longer than  $O_1$ :  $s_1$  cannot distinguish the random block from the others and discard it. For the sake of simplicity of description we assume that ModOnions may have variable length. However, if we assume that the ModOnions always contain the same number of blocks, then  $s_1$  will need to split  $O_2$  into two onions and obtain oracle decryption of both.

8. For each pair  $(i, j)$ , with  $j \neq i$  and  $j \neq 2$ , the attacker performs another oracle decryption to obtain  $O_{s_i, s_j} = D_{x_{s_j}}(O_{s_i})$ .
9. For a correct guess of  $s_3, s_4$ , the onion  $O_{s_3, s_4}$  will have the form:

$$\begin{aligned}
 & \text{random,} \\
 & \text{random,} \\
 & \text{random,} \\
 & E_{x_{s_2}^* - x_{s_2} - x_{s_3} - x_{s_4}}(\text{send to } s_3), \\
 & E_{x_{s_3}^* - x_{s_3} - x_{s_4}}(\text{send to } s_4), \\
 & E_{x_{s_4}^* - x_{s_4}}(\text{send to } s_5), \\
 & E_{x_{s_5}^*}(\text{send to } s_6), \\
 & E_{x_{s_5} + x_{s_6}^*}(m, t).
 \end{aligned}$$

By performing a trial decryption of all blocks with  $x_{s_5}^*$ , the attacker can learn  $s_6$ , which is the final destination of the message in this example.

This attack will work whenever  $s_1$  and  $s_5$  are at the beginning and end of the path. If the attacker has compromised more nodes, he can trial decrypt with the destination key of every compromised node to detect whether they lie on the path. (A trial decryption should also be performed on the intermediate onions  $O_{s_i}$  to detect cases where a compromised node is in the fourth position on the path.)

Note that there is another easy way to test a guess for a path:  $s_1$  can follow the protocol, but insert  $E_{x_{s_2} + x_{s_3} + x_{s_4} + x_{s_5}^*}(tag)$  instead of a random block into the onion forwarded to  $s_2$ . However, to test multiple guesses, onions would need to be replayed and the destination would get multiple copies of the same message, launching an investigation. Using the decryption oracles, no investigation will be started.

**Complexity.** The adversary needs to perform at most  $(N - 1) + \binom{N-1}{2}$  oracle decryptions, where  $N$  is the number of honest routers in the network. For a network with 50 routers, this is a little over 1000 oracle decryptions, making the attack expensive, but feasible. (To speed up the attack, decryptions at multiple routers can be carried out in parallel.) However, with larger networks the attack becomes infeasible; similarly, by increasing the path length from 5 to  $k$ , the complexity of the attack grows to  $\Omega(N^{k-3})$ .

### 3.2 Attack 2: The Two-hop Attack

Our second attack exploits the fact that URE allows an attacker to replace the plaintext of a message without knowing the encryption key as well as the plaintext removed. So, for example, given a block  $E_{x_{s_i}}(\text{send to } s_j)$ , an attacker can change it to  $E_{x_{s_i}}(\text{send to } s_1)$ . Using this technique, an attacker  $s_1$  could change the received blocks

$$\begin{aligned}
 &E_{x_{s_2}^*} \text{ (send to } s_3), \\
 &E_{x_{s_2}+x_{s_3}^*} \text{ (send to } s_4), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}^*} \text{ (send to } s_5), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*} \text{ (send to } s_6), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*} (m, t)
 \end{aligned}$$

to the following form:

$$\begin{aligned}
 &E_{x_{s_2}^*} \text{ (send to } s_3), \\
 &E_{x_{s_2}+x_{s_3}^*} \text{ (**send to } s_1), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}^*} \text{ (send to } s_5), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*} \text{ (send to } s_6), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*} (m, t).
 \end{aligned}**$$

If  $s_1$  sent these blocks to  $s_2$ , they would travel two hops, over to  $s_3$  and back to  $s_1$  (in a transformed form).  $s_1$  would then learn that  $s_3$  follows  $s_2$  in the path of the onion. However, to adjust the onion properly,  $s_1$  would need to know the order of the blocks. Since this is unknown,  $s_1$  recovers the order by probing, as explained below in detail:

1.  $s_1$  receives an onion to be forwarded. After picking out the destination and partial decryption, the onion has the following form:

$$\begin{aligned}
 &E_{x_{s_2}^*} \text{ (send to } s_3), \\
 &E_{x_{s_2}+x_{s_3}^*} \text{ (send to } s_4), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}^*} \text{ (send to } s_5), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*} \text{ (send to } s_6), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*} (m, t).
 \end{aligned}$$

2.  $s_1$  replaces the plaintext of all but one of the blocks with the directive “send to  $s_1$ ,” obtaining an onion like the following:

$$\begin{aligned}
 &E_{x_{s_2}^*} \text{ (**send to } s_1), \\
 &E_{x_{s_2}+x_{s_3}^*} \text{ (**send to } s_1), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}^*} \text{ (send to } s_5), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*} \text{ (**send to } s_1), \\
 &E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*} \text{ (**send to } s_1).
 \end{aligned}********$$

In this case, the third block is left unmodified.

3. This new onion is sent to  $s_2$ , along with a tag block inserted as the random block:

$$\begin{aligned}
& E_{x_{s_2}+x'}(tag), \\
& E_{x_{s_2}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(\text{send to } s_1).
\end{aligned}$$

for some random key  $x'$ .

4. The onion received back from  $s_2$  has the following form:

$$\begin{aligned}
& E_{x'}(tag), \\
& \text{random}, \\
& E_{x_{s_3}^*}(\text{send to } s_1), \\
& E_{x_{s_3}+x_{s_4}^*}(\text{send to } s_5), \\
& E_{x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_1), \\
& E_{x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(\text{send to } s_1).
\end{aligned}$$

$s_1$  notices the tag and declares this probe to be a failure.

5.  $s_1$  starts with the original onion and once again replaces all but one of the blocks with “send to  $s_1$ ”, this time leaving a different block unmodified:

$$\begin{aligned}
& E_{x_{s_2}+x'}(tag), \\
& E_{x_{s_2}^*}(\text{send to } s_3), \\
& E_{x_{s_2}+x_{s_3}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}^*}(\text{send to } s_1), \\
& E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+x_{s_6}^*}(\text{send to } s_1).
\end{aligned}$$

6.  $s_1$  once again sends the onion to  $s_2$ , along with a tag.  $s_2$  now finds  $s_3$  to be the destination of the onion, and forwards it there.  $s_3$  then forwards the onion back to  $s_1$ , after all partial decryptions and inserting random blocks it has the form:

$$\begin{aligned}
& E_{x'-x_{s_3}}(tag), \\
& \text{random}, \\
& \text{random}, \\
& E_{x_{s_4}^*}(\text{send to } s_1), \\
& E_{x_{s_4}+x_{s_5}^*}(\text{send to } s_1), \\
& E_{x_{s_4}+x_{s_5}+x_{s_6}^*}(\text{send to } s_1).
\end{aligned}$$

7. Note that  $s_1$  cannot decrypt any of the blocks of the onion, so it suspects that  $s_3$  is the hop following  $s_2$  in the onion path. To double check, it can resend the onion with a tag of  $E_{x_{s_2}+x_{s_3}+x'}(tag)$ . This time, the onion will come back with  $E_{x'}(tag)$  as one of its blocks.

After these steps,  $s_1$  learns the identity of the next hop in the path. It can now use  $s_2$  as a decryption oracle to obtain a copy of an onion that would have been sent by  $s_2$  to  $s_3$  during normal forwarding. Thus it can assume the role of  $s_2$  and repeat this attack with  $s_3$  to learn the identity of  $s_4$ , and so on.

## 4 Defense: Context-Sensitive Encryption

A core problem that is exploited by all the attacks on ModOnions is that the onion construction is highly malleable: an attacker can make extensive modifications to the encrypted onion and still produce a valid result. To defend against them, we must reduce or eliminate this ability.

A non-malleable encryption scheme [7], such as Cramer–Shoup [3], would ensure that any modification to the ciphertext will result in a “random looking” plaintext upon decryption. However, a non-malleable scheme will, by definition, prevent the re-randomization used in re-encryption. Canetti et al. defined a relaxed version of non-malleability, called Re-randomizable Chosen Ciphertext Adversary security (RCCA) [1]. Prabhakaran and Rosulek later produced an RCCA-secure scheme, based on Cramer–Shoup and the “double strand” construction used in the Frikken–Golle universal re-encryption scheme we use in this paper [12]. However, this scheme cannot be used with ModOnions since it does not support key privacy, which is a requirement for anonymous message forwarding.

Instead, we propose a new modification to the ModOnions protocol that dramatically reduces the malleability of the scheme. Since all attacks on ModOnions rely on redirecting the onion to follow a different path, our modification centers around making such redirection impossible. It entangles the encryption construction with the onion path, such that if the path is modified, the decryption produces an invalid result and thus an attack is detected. Our main approach is to use a *context-sensitive* encryption/decryption key. Each router will have a collection of  $n$  distinct transport and  $n$  distinct destination keys,  $x_{s_i,1}, \dots, x_{s_i,n}$  and  $x_{s_i,1}^*, \dots, x_{s_i,n}^*$ . Whenever a router needs to use a transport or a destination key, it picks the key based on context by selecting  $x_{s_i, H(s_{i-1}||s_i||s_{i+1})}$ , where  $H : \{0,1\}^* \rightarrow \mathbb{Z}_n$  is a hash function and  $s_{i-1}$  and  $s_{i+1}$  are the identifiers preceding and following  $s_i$  in the onion path. (In fact,  $n$  need not to be large. Even  $n = 2$  yields probability of 0.5 of detection of a malicious node in most cases, so it prevents systematic attacks).

### 4.1 Context-Sensitive Onion Construction

We now describe our construction in more detail. Suppose a node  $s_0$  wishes to send a message to  $s_6$ , following a path  $s_1, s_2, s_3, s_4, s_5$ . We assume that it has the

public keys for all routers, i.e.,  $y_{s_i,j} = g^{x_{s_i,j}}$ .  $s_0$  creates a ModOnion as before, but using context-sensitive keys. The first block of the onion would be:

$$E_{x_{s_1,H(s_0||s_1||s_2)}}^* (\text{send to } s_2).$$

The next block would be:

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}}^* (\text{send to } s_3)$$

Notice that the transport key for  $s_1$  is made context-sensitive as well. For  $s_2$ 's destination key, the context used is the part of the path known to  $s_2$ . Proceeding in this manner, the complete onion has the following form:

$$E_{x_{s_1,H(s_0||s_1||s_2)}}^* (\text{send to } s_2)$$

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}}^* (\text{send to } s_3)$$

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}+x_{s_3,H(s_2||s_3||s_4)}}^* (\text{send to } s_4)$$

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}+x_{s_3,H(s_2||s_3||s_4)}+x_{s_4,H(s_3||s_4||s_5)}}^* (\text{send to } s_5)$$

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}+x_{s_3,H(s_2||s_3||s_4)}+x_{s_4,H(s_3||s_4||s_5)}+x_{s_5,H(s_4||s_5||s_6)}}^* (\text{send to } s_6)$$

$$E_{x_{s_1,H(s_0||s_1||s_2)}+x_{s_2,H(s_1||s_2||s_3)}+x_{s_3,H(s_2||s_3||s_4)}+x_{s_4,H(s_3||s_4||s_5)}+x_{s_5,H(s_4||s_5||s_6)}+x_{s_6,H(s_5||s_6)}}^* (m, t)$$

Forwarding of the onions proceeds in a similar fashion as before, except that context-sensitive keys are used. Notice that when  $s_1$  receives the onion, it does not yet know what the next hop in the path will be. We nevertheless want to use context-sensitive encryption here to reduce the possibility of attack. To resolve this problem,  $s_1$  uses a brute-force search for all possible destination keys. That is, it performs a trial decryption of each block with the keys  $x_{s_1,1}^*, x_{s_1,2}^*, \dots, x_{s_1,n}^*$ . This search will certainly slow down the decryption process, but even for  $n = 100$ , the trial decryptions should take less than a second on a modern PC, and of course,  $n$  is a tunable parameter.

After a trial decryption succeeds with key  $x_{s_1,j}^*$  for some  $j$ ,  $s_1$  verifies that the contents of the decrypted message ("send to  $s_2$ ") matches the recovered context  $s_0||s_1||s_i$ , i.e., that  $H(s_0||s_1||s_2) = j$ . If there is a discrepancy, the onion is discarded and an investigation is started. Otherwise, it replaces the destination block with a random one, partially decrypts the rest of the onion with the key  $x_{s_1,H(s_0||s_1||s_2)}$ , re-randomizes and shuffles the blocks, and forwards the new onion to  $s_2$ .

If the decrypted plaintext contains a message, rather than a redirection directive, the router once again verifies that the correct context is used, i.e., that  $H(s_0||s_1) = m$  and discards a message otherwise.

## 5 Security Analysis of the Modified ModOnion Scheme

**Defending against the Two-Hop Attack.** As the two-hop attack makes extensive use of modifying the onion contents, it is severely impacted by the

use of context-sensitive encryption. In the case of an unsuccessful probe,  $s_2$  will receive an onion with the block:

$$E_{x_{s_2, H(s_1||s_2||s_3)}}^* (\text{send to } s_1).$$

After a brute-force search, the decryption will succeed using a key  $x_{s_2, i}^*$  for some  $i$ , but the odds that  $i = H(s_1||s_2||s_1)$  are only  $1/n$ . Therefore,  $s_2$  will launch an investigation with probability  $\frac{n-1}{n}$ . Similarly, for a successful probe,  $s_3$  will receive an onion with:

$$E_{x_{s_3, H(s_2||s_3||s_4)}}^* (\text{send to } s_1)$$

and launch an investigation whenever  $H(s_2||s_3||s_4) \neq H(s_2||s_3||s_1)$ .

The context-sensitive encryption acts as an integrity check on the message contents, effectively preventing any modifications of the onion. For the two-hop attack to succeed without detection, it must be the case that:

$$H(s_1||s_2||s_3) = H(s_1||s_2||s_1), \quad H(s_2||s_3||s_4) = H(s_2||s_3||s_1), \quad H(s_3||s_4||s_5) = H(s_3||s_4||s_1), \\ H(s_4||s_5||s_6) = H(s_4||s_5||s_1), \quad H(s_5||s_6) = H(s_5||s_6||s_1)$$

otherwise one of the nodes will notice the attack and start an investigation. The odds of this are  $\frac{1}{n^5}$ , so even very small values of  $n$  provide an effective defense.

**Defending against the Path-Guessing Attack.** The path guessing attack does not modify any onion plaintext and thus is not immediately thwarted by context-sensitive encryption. However, using a router as a decryption oracle becomes significantly more complicated. Consider, for example, using  $s_3$  as a decryption oracle in the path-guessing attack. If  $s_1$  were to follow the same steps as in Section 3.1 for oracle decryption (but using  $E_{x_{s_3, H(s_1||s_3||s_1)}}^*$  as the new destination block), it would receive an onion where all blocks have been decrypted with the key  $x_{s_3, H(s_1||s_3||s_1)}$ . However, the blocks in the real path would have in fact been encrypted with  $x_{s_3, H(s_2||s_3||s_4)}$ , and thus the oracle decryption would be useless (unless the hashes happen to match).

However,  $s_1$  can perform a trial to verify a guess of an entire path. It would start by creating the following destination fields:

$$E_{x_{s_2, H(s_1||s_2||s_3)}}^* (\text{send to } s_3) \\ E_{x_{s_2, H(s_1||s_2||s_3)} + x_{s_3, H(s_2||s_3||s_4)}}^* (\text{send to } s_4) \\ E_{x_{s_2, H(s_1||s_2||s_3)} + x_{s_3, H(s_2||s_3||s_4)} + x_{s_4, H(s_3||s_4||s_5)}}^* (\text{send to } s_5)$$

Then it would wrap all the existing blocks of the original onion ( $O_1$  in Section 3.1) in a layer of encryption with random key  $x'$ :

$$E_x(m) \rightarrow E_{x+x'}(m).$$

When the onion is sent to  $s_2$ , it will be forwarded along the path  $s_2, s_3, s_4, s_5$ , with the corresponding context-sensitive decryptions happening along the way.

In other words,  $s_5$  will receive blocks that has been partially decrypted with the key  $x_{s_2, H(s_1||s_2||s_3)} + x_{s_3, H(s_2||s_3||s_4)} + x_{s_4, H(s_3||s_4||s_5)}$ . If the path guess is correct,  $s_5$  will receive an onion that will contain a block  $E_{x' + x_{s_5, H(s_4||s_5||s_6)}}^*$  (send to  $s_6$ ), which it could locate by trial decryption. If the path guess is incorrect,  $s_5$  would receive an onion with blocks that it cannot decrypt.

This attack is slower than the (already slow) path guessing attack. In our particular case ( $s_3, s_4$  honest,  $s_5$  corrupted) it requires  $(N - 1)(N - 2)C$  path guesses, where  $N$  is the number of honest nodes in the network and  $C$  is the number of nodes collaborating with  $s_1$ . Possibility to parallelize this process is limited due to the fact that every probe must pass through  $s_2$  first, limiting the rate that can be used without arousing suspicion. If the size of the network is such that the attack is still practical, increasing the path length can easily eliminate it.

**Possibility of Tagging Attacks.** The modified construction of onions and the context-sensitive encryption still potentially allow the adversary to replace the plaintext (the address of the next server or the message) in a single block or put something in the place of the random block. To some extent, this can be useful for a tagging attack. Let us consider the following scenario: an onion gets to some corrupted node  $s_i$ . This node wants to add a special tag, so that if this onion arrives at another node controlled by the adversary, then he will be able to recognize this event.

There are two ways in which the adversary may try to achieve that goal. According to the first method,  $s_i$  can copy some block of the onion, change the original plaintext there into some “tag” message, blind the ciphertext with some additional key  $x'$  and use it as the random block created himself. The adversary will be able to recognize the tag if and only if the block used is a block addressed to a node under control of the adversary. This occurs with probability  $f$ , where  $f$  is the fraction of nodes controlled by the adversary in the network. If the adversary replaces some other blocks with the blocks constructed as above, then the chance to detect a tag may increase at most  $\lambda$  times, where  $\lambda$  is the number of blocks. So, the probability remains small for the case when  $f$  is reasonable. However, if the tag is not recognized by some adversary node, then it will be detected that some block is missing and an investigation will be started.

According to the second method, the adversary creates a ciphertext that encodes some “tag” by using the public keys of arbitrarily chosen nodes and some blinding factor  $x'$  and replaces some original block with this ciphertext. In this case tagging remains undetected by honest nodes as long as the tag is inserted in the random block created by  $s_i$ . The tag can be read by an adversary node only if the path for the tag is guessed correctly. If this path has length  $k$ , then this probability equals  $\frac{f}{(N-1)^{k-1}}$ , where  $f$  is the fraction of nodes controlled by the adversary and  $N$  is the overall number of nodes. The adversary may increase this probability by replacing more blocks by the blocks with tags. However, in

most cases the tags are undetected, but the block removed will be found missing leading to an investigation.

**Countermeasures against Tagging Attacks.** Since only the first method of tagging attacks presented above is a serious threat is a real threat, we concentrate on preventing a node to put some extra information in the random block. For this purpose we need a public key  $P$ . Peculiarity of this public key is that it has no owner holding the corresponding private key. We also assume that each message contains sending time information and that each message will be delivered in time  $T$  - otherwise we talk about an irregularity that has to be investigated. The protocol may look as follows:

- instead of a random block the node should construct a ciphertext of 1 using public key  $P$ ,
- when a ModOnion is transmitted from node  $n_i$  to  $n_j$ , the node  $n_j$  starts a checking procedure with a probability  $p$ , independently of other events (in particular, whether checking has been already initiated for another copy of the same message). The checking procedure consists of the following steps:
  1. the message is stored by  $S_j$ ,
  2.  $s_i$  is asked to resend the same message (re-encryption applies),
  3.  $s_j$  waits time  $T$  and afterwards demands a proof from  $s_i$  that one of the blocks of the ModOnion is a ciphertext 1 created as described above. The proof can be given by presenting the random exponent used for creating the ciphertext.

Alternatively, instead of revealing the encryption exponent,  $s_i$  may provide a zero-knowledge proof that one of the ciphertexts of the onion is a ciphertext of 1 under key  $P$  (without revealing which). In this case we do not have to wait with the proof, but the proof might be too intensive computationally.

Note that no proof should reveal which block is random, unless the message cannot be sent anymore without starting an investigation. Indeed, otherwise  $s_j$  would know two random blocks and could insert a tag in one of them without possibility to be caught.

## 6 Conclusions

We presented two new types of attacks on the ModOnions protocol showing that, despite modifications in [11], the protocol is still insecure. However, we introduce a patch that successfully defends against these new attacks without providing any new information to intermediate nodes. The main remaining threat seems to be the tagging attack.

## Acknowledgment

We thank the anonymous reviewers for the comments, especially for the remarks concerning the tagging attack.

## References

1. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003)
2. Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM* 24(2), 84–88 (1981)
3. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
4. Danezis, G.: Breaking Four Mix-Related Schemes Based on Universal Re-encryption. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 46–59. Springer, Heidelberg (2006)
5. Danezis, G., Dingleline, R., Mathewson, N.: Mixminion: design of a type III anonymous remailer protocol. In: IEEE Symposium on Security and Privacy, pp. 2–15 (2003)
6. Dingleline, R., Mathewson, N., Syverson, P.F.: Tor: The Second-Generation Onion Router. In: USENIX Security Symposium 2004, pp. 303–320 (2004)
7. Dolev, D., Naor, M.: Non-malleable cryptography. In: ACM Symposium on Theory of Computing, pp. 542–552 (1991)
8. Golle, P., Jakobsson, M., Juels, A., Syverson, P.F.: Universal Re-encryption for Mixnets. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 163–178. Springer, Heidelberg (2004)
9. Gomułkiewicz, M., Klonowski, M., Kutylowski, M.: Onions Based on Universal Re-encryption — Anonymous Communication Immune Against Repetitive Attack. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 400–410. Springer, Heidelberg (2005)
10. Klonowski, M., Kutylowski, M., Lauks, A., Zagórski, F.: Universal Re-encryption of Signatures and Controlling Anonymous Information Flow. In: WARTACRYPT 2004 Conference on Cryptology, vol. 33, pp. 179–188. Tatra Mountains Mathematical Publications (2006)
11. Klonowski, M., Kutylowski, M., Lauks, A.: Repelling Detour Attack against Onions with Re-Encryption. In: Bellare, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 296–308. Springer, Heidelberg (2008)
12. Prabhakaran, M., Rosulek, M.: Rerandomizable RCCA encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 517–534. Springer, Heidelberg (2007)
13. Pries, R., Yu, W., Fu, X., Wei Zhao, W.: A New Replay Attack Against Anonymous Communication Networks. In: Proceedings of IEEE International Conference on Communication, pp. 1578–1582 (2008)
14. Tsiounis, Y., Yung, M.: On the Security of ElGamal Based Encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998)