Attacking and Repairing the Improved ModOnions Protocol-Tagging Approach

Nikita Borisov[1], Marek Klonowski[2], Mirosław Kutyłowski[2] and Anna Lauks-Dutka[2]
[1] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, USA
[e-mail: nikita@uiuc.edu]
[2] Institute of Mathematics and Computer Science, Wrocław University of Technology, Poland
[e-mail: {Marek.Klonowski, Miroslaw.Kutylowski, Anna.Lauks}@pwr.wroc.pl]
*Corresponding author: Anna Lauks-Dutka

Received April 1, 2010; revised June 1, 2010; accepted June 2, 2010;
published June 30, 2010

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 exploits 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 modifications to the ModOnion protocol that dramatically reduce the malleability of the encryption primitive. It addresses the class of attacks we present and it makes other attacks difficult to formulate.

Keywords: Onion routing, tagged encryption, universal re-encryption, replay attack

This paper is an extended version of the paper "Attacking and Repairing the Improved ModOnions Protocol" published in the Proceedings of ICISC'2009. In particular, we propose here a new tagging technique. This research was partially supported by Polish Ministry of Science and Higher Education, grant N N206 2701. The fourth author was partially supported by the fellowship co-financed by European Union within European Social Fund.

DOI: 10.3837/tiis.2010.06.011

1. Introduction

Mix networks have been a popular scheme for anonymous communication since their introduction by Chaum [1]. 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 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 outcoming 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 [2], 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 [3]) can be affected by elaborated forms of reply attack as reported in [4]. 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 [5] take an alternate approach, using Universal Re-Encryption (URE) [6] 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 [7].

ModOnions addressed the replay attacks, but it turned out that they are susceptible to the *detour attack* [8], 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 [9].

1.1 Results

New attacks. We show that the improved scheme presented in [9] 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 [9]

In this section we recall the improved version of the ModOnions protocol (Onion Routing with Universal Re-Encryption) from [9]. This protocol uses as a building block an extension of Universal Re-Encryption. At first let us recall details and properties of Universal Re-Encryption from [6].

2.1 Universal Re-Encryption

Universal Re-Encryption is based on the ElGamal encryption scheme. Construction of this scheme is based on a cyclic group *G*, where discrete logarithm and 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 Z_p^* of order *q*. A private key $p < q$ is a non-zero element 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 [6]). ElGamal cryptosystem has yet another important property. Everyone can re-encrypt a ciphertext (α,β) and get (α',β'), where $\alpha' := \alpha \cdot y^{k'} \bmod p$, $\beta' := \beta \cdot g^{k'} \bmod p$ for $k' < q$ chosen at random and *y* is the public key user for encryption. Moreover, without the decryption key it is infeasible to find that (α,β) and (α',β') correspond to the same plaintext.

In [6] 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 is hard and DDH assumption holds. 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 $y = g^x$ in *G*.

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 $m_0 := \alpha_0 / \beta_0{}^x$ and $m_1 := \alpha_1 / \beta_1{}^x$. 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 λ 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 $s_1, \ldots, s_\lambda$, first values k_0 and k_1 are generated at random. Then the ciphertext has the following form:

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

Obviously, $y_1 y_2 \ldots y_\lambda$ is a kind of *cumulative* public key. Since

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

$E_{x_1+x_2+\ldots+x_\lambda}(m)$ 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 $E_{x_2+\ldots+x_\lambda}(m)$ 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 [9]

The core idea of the protocol introduced in [9] 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 d_s and a public key $D_s = g^{x_s^*}$.

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

$$E_{d_{s_1}}(\text{"send to"}\, s_2)$$

$$E_{x_{s_1}+\ldots+x_{s_{i-1}}+d_{s_i}}(\text{"send to"}\, s_{i+1}) \qquad \text{for all} \quad 1 \le i \le \lambda - 1,$$

$$E_{x_{s_1}+\ldots+x_{s_{\lambda-1}}+d_{s_\lambda}}(m,t), \qquad \text{where } t \text{ is a current time.}$$

Note that:

- λ is a static parameter of the protocol,
- E denotes encryption scheme with properties described in [5][9],
- for each block one of destination keys d_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 contains 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 the name of 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. This is a part of the protocol launched by the node detecting dishonest behavior of other nodes. In this procedure consecutive nodes from the path prove correct processing of the ModOnion. It is assumed that malicious node, once caught is permanently excluded from the protocol. A detailed description can be found in [5].

3. Attacks on the ModOnions Protocol from [9]

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 disregard a random order of blocks for clarity of presentation):

$$E_{d_{s_1}}(\text{"send to"} s_2)$$

$$E_{x_{s_1}+d_{s_2}}(\text{"send to"} s_3)$$

$$E_{x_{s_1}+x_{s_2}+d_{s_3}}(\text{"send to"} s_4)$$

$$E_{x_{s_1}+x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"} s_5)$$

$$E_{x_{s_1}+x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"} s_6)$$

$$E_{x_{s_1}+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

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 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 $E_{d_{s_5}}(\text{"send to"} s_6)$. By performing a trial decryption with key d_{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 d_{s_1} to learn the destination s_2.
2. Then s_1 uses x_{s_1} to partially decrypt the remaining blocks and gets:

$$E_{d_{s_2}}(\text{"send to"} s_3)$$

$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_4)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

Let us call this onion O_1.

3. URE has the property that given a ciphertext $E_x(m)$ encrypted with key x, 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:

$$E_{d_{s_2}}(\text{"send to"}s_1)$$
$$E_{x'+d_{s_2}}(\text{"send to"}s_3)$$
$$E_{x'+x_{s_2}+d_{s_3}}(\text{"send to"}s_4)$$
$$E_{x'+x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x'+x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$
$$E_{x'+x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

4. This onion is then sent to s_2. Node s_2 partially decrypts with d_{s_2} to learn that s_1 is the next hop.[1] 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:

random

$$E_{x'+d_{s_2}-x_{s_2}}(\text{"send to"}s_3)$$
$$E_{x'+d_{s_3}}(\text{"send to"}s_4)$$
$$E_{x'+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x'+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$

[1] 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.

$$E_{x'+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

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

$$\begin{array}{l} \text{random} \\ E_{d_{s_2}-x_{s_2}}(\text{"send to"}s_3) \\ E_{d_{s_3}}(\text{"send to"}s_4) \\ E_{x_{s_3}+d_{s_4}}(\text{"send to"}s_5) \\ E_{x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6) \\ E_{x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t) \end{array}$$

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 $O_2 = D_{x_{s_2}}(O_1)$ [2].

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 $O_{s_i} = D_{x_{s_i}}(O_2)$ [3].
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:

$$\begin{array}{l} \text{random} \\ E_{d_{s_2}-x_{s_2}-x_{s_3}-x_{s_4}}(\text{"send to"}s_3) \\ E_{d_{s_3}-x_{s_3}-x_{s_4}}(\text{"send to"}s_4) \\ E_{d_{s_4}-x_{s_4}}(\text{"send to"}s_5) \\ E_{d_{s_5}}(\text{"send to"}s_6) \\ E_{x_{s_5}+d_{s_6}}(m,t) \end{array}$$

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

[2] This process is very similar to the oracle decryption proposed by Danezis in [5].

[3] A minor caveat is that O_2 is one block longer than O_2: 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.

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 perform trial decryption 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 $E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{tag})$ 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. When using the decryption oracles, no investigation will be started.

Complexity. If the distance between the malicious nodes on the path is $k+1$, then in average $0.5 \cdot \binom{N-1}{k}$ oracle decryptions are required, where *N* is the number of honest routers in the network. For a network with 50 routers, $k=2$, 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 $k+3$ the complexity of the attack grows to $\Omega(N^k)$.

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

$$E_{d_{s_2}}(\text{"send to"}s_3)$$
$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_4)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

to the following form:

$$E_{d_{s_2}}(\text{"send to"}s_3)$$
$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$

$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

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 performing partial decryption, the onion has the following form:

$$E_{d_{s_2}}(\text{"send to"}s_3)$$
$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_4)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_6)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(m,t)$$

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:

$$E_{d_{s_2}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(\text{"send to"}s_1)$$

   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:

$$E_{x'+x_{s_2}}(tag)$$
$$E_{d_{s_2}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_1)$$
$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$

$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_1)$$

$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(\text{"send to"}s_1)$$

for some random key x'.

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

$$E_{x'}(tag)$$

random

$$E_{d_{s_3}}(\text{"send to"}s_1)$$

$$E_{x_{s_3}+d_{s_4}}(\text{"send to"}s_5)$$

$$E_{x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_1)$$

$$E_{x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(\text{"send to"}s_1)$$

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:

$$E_{x'+x_{s_2}}(tag)$$

$$E_{d_{s_2}}(\text{"send to"}s_3)$$

$$E_{x_{s_2}+d_{s_3}}(\text{"send to"}s_1)$$

$$E_{x_{s_2}+x_{s_3}+d_{s_4}}(\text{"send to"}s_1)$$

$$E_{x_{s_2}+x_{s_3}+x_{s_4}+d_{s_5}}(\text{"send to"}s_1)$$

$$E_{x_{s_2}+x_{s_3}+x_{s_4}+x_{s_5}+d_{s_6}}(\text{"send to"}s_1)$$

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:

$$E_{x'-x_{s_3}}(tag)$$

random

random

$$E_{d_{s_4}}(\text{"send to"}s_1)$$

$$E_{x_{s_4}+d_{s_5}}(\text{"send to"}s_1)$$

$$E_{x_{s_4}+x_{s_5}+d_{s_6}}(\text{"send to"}s_1)$$

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 Based on Tagged Encryption

As it was shown so far 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. In the conference version of this paper ([10]) we showed that context-sensitive encryption scheme, i.e. the encryption in which all the keys depends on the path of the onion, allows to prevent any leakeage of the information about the path. The only problem with the encryption construction proposed is time and complexity of the decryption phase. Namely, during the decryption procedure a node needs to perform many trial decryptions in order to find a proper destination key. Here we propose the improvement of the context-sensitive encryption method that allows to avoid this problem.

In this paper we use a notion of tag non-malleability, as defined by MacKenzie et al. [11]. This notion associates a tag with a ciphertext and ensures that no modifications may be made to the tag while retaining a meaningful message. We use this construction to embed the message path (or fragments thereof) into a tag. Therefore, however the adversary tampers with the message, the tags ensure that the message traverses the intended path, eliminating the possibility of the above and other attacks. We first present MacKenzie et al.'s construction of a tag non-malleable scheme and its extensions to support universal re-encryption and then describe an updated context sensitive encryption protocol that uses it.

Each router s_i will have a pair of destination keys (sk_i^*, pk_i^*) and a collection of n distinct pairs of transport keys $(sk_1, pk_1),\ldots,(sk_n, pk_n)$. Whenever a router needs to use a transport key, it picks the key based on context by selecting $sk_{i,F(s_{i-1}\|s_i\|s_{i+1})}$, where $F:\{0,1\}^* \to Z_n$ and s_{i-1} and s_{i+1} are the identifiers of the nodes preceding and following s_i in the onion path. Function F should have all properties of a secure hash function except that its range is very small. As F we can for instance a hash function with its values truncated to a few bits.

4.1 Tagged encryption

Tagged encryption augments standard public-key encryption definition to include a tag in the encryption and decryption operations. A tagged ciphertext can only be decrypted with the correct tag; otherwise, the decryption returns an error.

Key generation: as in standard public-key encryption, key generation produces a public and a secret key:

$$(sk, pk) \leftarrow \text{keygen}(1^k)$$

Encryption: Encryption takes an extra tag parameter *t*:

$$C \leftarrow \text{enc}_{pk}(M, t)$$

Decryption: Decryption takes the same parameter *t*:

$$M' \leftarrow \text{dec}_{sk}(C, t)$$

The scheme should satisfy the following conditions:

$$\text{dec}_{sk}(\text{enc}_{pk}(M,t),t) = M$$

$$\text{dec}_{sk}(\text{enc}_{pk}(M,t),t') = \perp \qquad \text{for } t' \neq t$$

The formal security definition of tag non-malleability parallels the simulation-based non-malleability definition, with the difference that queries with the same tag are forbidden. For more details, see [11]. MacKenzie et al. provide a construction based on Cramer-Shoup scheme that will be the foundation of our new protocol:

Key generation: Given a group G_q of order q where the DDH assumption holds, and g a generator of G_q, generate $g_2 \in G_q$ and $a,b,c,d,e \in G_q$ at random. Compute $U := g^a \cdot g_2{}^b$; $V := g^c \cdot g_2{}^d$; $W := g^e$. Then (g, g_2, U, V, W) is the public key and (a,b,c,d,e) is the secret key.

Encryption: Pick $r \in Z_q$ at random. Compute $x := g^r$, $y := g_2{}^r$, $w := m \cdot W^r$, $v := U^r V^{rt}$. Return $\text{enc}_{\langle g, g_2, U, V, W \rangle}(m,t) = (x, y, w, v)$ as the ciphertext.

Decryption: Check that $v = x^{a+ct} y^{b+dt}$. If so return $\text{dec}_{\langle a,b,c,d,e \rangle}(\langle x,y,w,v \rangle, t) = \dfrac{w}{x^e}$, else return $\text{dec}_{\langle a,b,c,d,e \rangle}(\langle x,y,w,v \rangle, t) = \perp$.

Roughly, this replaces the hash of *x, y, w* with *t* in the definition of Cramer-Shoup scheme.

4.2 Tag-based Universal Re-encryption

To use the scheme for our purposes, we need to extend it to support universal re-encryption. Fortunately, the double-strand technique that is used in the ElGamal-based construction of Golle et al. can be applied to this scheme as well.

Encryption of a message *m* with the tag parameter *t* for some user with the secret key $sk = (a,b,c,d,e)$ and the public key $pk = (g, g_2, U, V, W) = \left(g, g_2, g^a g_2{}^b, g^c g_2{}^d, g^e\right)$ takes the form:

$$\mathrm{enc}_{pk}(m,t) = (x_0, y_0, w_0, v_0, x_1, y_1, w_1, v_1) = \left(g^{r_0}, g_2{}^{r_0}, m \cdot W^{r_0}, U^{r_0}V^{r_0 t}, g^{r_1}, g_2{}^{r_1}, W^{r_1}, U^{r_1}V^{r_1 t}\right)$$

where values $r_0, r_0 \in Z_q$ are chosen at random.

Now let us regard the cumulative encryption. Le us assume that we have a set of *n* key pairs $(sk_1, pk_1), \ldots, (sk_n, pk_n)$, where for each $i = 1, \ldots, n$:

- $sk_i = (a_i, b_i, c_i, d_i, e_i)$ is the private key,
- $pk_i = (g, g_2, U_i, V_i, W_i) = \left(g, g_2, g^{a_i} g_2{}^{b_i}, g^{c_i} g_2{}^{d_i}, g^{e_i}\right)$ is the public key.

Then the cumulative encryption (using *n* public keys) and *n* tags $t_1, t_2, \ldots, t_n$ of a message *m* takes the form:

$$\mathrm{enc}_{\{pk\}_{i=1}^n}\left(m, \{t_i\}_{i=1}^n\right) = (x_0, y_0, w_0, v_0, x_1, y_1, w_1, v_1)$$

$$= \left(g^{r_0}, g_2{}^{r_0}, \prod_{i=1}^n m \cdot W_i^{r_0}, \prod_{i=1}^n U_i^{r_0} V_i^{r_0 t_i}, g^{r_1}, g_2{}^{r_1}, \prod_{i=1}^n W_i^{r_1}, \prod_{i=1}^n U_i^{r_1} V_i^{r_1 t_i}\right)$$

Re-encryption is also possible. Partial decryption with the key $sk_n = (a_n, b_n, c_n, d_n, e_n)$ using the proper tag t_n takes the form:

$$\mathrm{dec}_{sk_n}\left((x_0, y_0, w_0, v_0, x_1, y_1, w_1, v_1), t_n\right) = \left(x_0, y_0, \frac{w_0}{x_0^{e_n}}, \frac{v_0}{x_0^{a_n + c_n t_n} y_0^{b_n + d_n t_n}}, x_1, y_1, \frac{w_1}{x_1^{e_n}}, \frac{v_1}{x_1^{a_n + c_n t_n} y_1^{b_n + d_n t_n}}\right)$$

We can use these primitive to wrap an onion in *n* layers of encryption and unwrap them one-by-one at each router.

4.3 Context-Sensitive Tagged Onion Construction

Preliminaries. Given a group G_q of order *q* where the DDH assumption holds, a generator *g* of G_q the generator $g_2 \in G_q$ is chosen at random. Then each node s_i in the system computes a pair of destination keys (sk_i^*, pk_i^*) and *n* distinct pairs of transport keys $(sk_{i,1}, pk_{i,1}), \ldots, (sk_{i,n}, pk_{i,n})$ such that:

$$sk_i^* = \left(a_i^*, b_i^*, c_i^*, d_i^*, e_i^*\right)$$

$$pk_i^* = \left(g, g_2, U_i^*, V_i^*, W_i^*\right) = \left(g, g_2, g^{a_i^*} g_2{}^{b_i^*}, g^{c_i^*} g_2{}^{d_i^*}, g^{e_i^*}\right)$$

$$sk_{i,1} = (a_{i,1}, b_{i,1}, c_{i,1}, d_{i,1}, e_{i,1})$$

$$pk_{i,1} = (g, g_2, U_{i,1}, V_{i,1}, W_{i,1}) = \left(g, g_2, g^{a_{i,1}} g_2{}^{b_{i,1}}, g^{c_{i,1}} g_2{}^{d_{i,1}}, g^{e_{i,1}}\right)$$

$$\vdots$$

$$sk_{i,n} = \left(a_{i,n}, b_{i,n}, c_{i,n}, d_{i,n}, e_{i,n}\right)$$

$$pk_{i,n} = \left(g, g_2, U_{i,n}, V_{i,n}, W_{i,n}\right) = \left(g, g_2, g^{a_{i,n}} g_2^{b_{i,n}}, g^{c_{i,n}} g_2^{d_{i,n}}, g^{e_{i,n}}\right)$$

where $a^*_i, b^*_i, c^*_i, d^*_i, e^*_i, a_{i,1}, b_{i,1}, c_{i,1}, d_{i,1}, e_{i,1}, \dots, a_{i,n}, b_{i,n}, c_{i,n}, d_{i,n}, e_{i,n} \in Z_q$ are chosen independently at random. The hash function H and pseudorandom function $F : \{0,1\}^* \to Z_n$ are known to all nodes.

Onion Construction. The construction of the context-sensitive tagged onion is similar to the ModOnion construction, but it uses tags and the context-sensitive keys to keep track of the path. Suppose a node s_0 wishes to send a message to s_5 through the following a path: $s_0, s_1, s_2, s_3, s_4, s_5$. We assume that it has the public keys for all routers of this path. The first block of the onion is:

$$\text{enc}_{\left\{pk^*_1\right\}}\left(\text{"send to"} s_2, \left\{H\left(s_0 \| s_1 \| s_2\right)\right\}\right).$$

The next block would be:

$$\text{enc}_{\left\{pk_{1, F(s_0 \| s_1 \| s_2)}, pk^*_2\right\}}\left(\text{"send to"} s_3, \left\{H\left(s_0 \| s_1 \| s_2\right), H\left(s_1 \| s_2 \| s_3\right)\right\}\right)$$

Proceeding in this manner, the last block will have the form:

$$\text{enc}_{\left\{pk_{1, F(s_0 \| s_1 \| s_2)}, \dots, pk_{4, F(s_3 \| s_4 \| s_5)}, pk^*_5\right\}}\left(m, \left\{H\left(s_0 \| s_1 \| s_2\right), \dots, H\left(s_3 \| s_4 \| s_5\right) H\left(s_4 \| s_5\right)\right\}\right)$$

Forwarding of the onions proceeds in a similar fashion as before, except that context-sensitive keys are used and the tag verification needs to be performed. Thus when node s_i receives the onion, then

1. s_i copies the onion and decrypts all the blocks of the copy with the destination key sk^*_i to find out the one that decrypts correctly and is consistent with the tag. Namely, for each block of the form $\left(x_0, y_0, w_0, v_0, x_1, y_1, w_1, v_1\right)$ node s_i
    - computes $m' = \dfrac{w_0}{x_0^{e^*_i}}$; if m' is a proper message, it gets the knowledge about the next hop on the path s_{i+1},
    - computes $t_i = H(s_{i-1} \| s_i \| s_{i+1})$ and checks if $v_0 = x_0^{a^*_i + c^*_i \cdot t_i} y_0^{b^*_i + d^*_i \cdot t_i}$ and $v_1 = x_1^{a^*_i + c^*_i \cdot t_i} y_1^{b^*_i + d^*_i \cdot t_i}$

- if only one block decrypts correctly, then from all other blocks of the original the encryption layer with the transport key sk_{i,t_i} and the tag t_i is taken off,

2. s_i replaces the destination block with a random one, re-randomizes and shuffles the blocks, and forwards the new onion to s_{i+1},
3. if there is any discrepancy, s_i discards the onion and starts an investigation.

5. Security Analysis of the Modified ModOnion Scheme

Defending against the Two-Hop Attack. The two-hop attack makes extensive use of modifying the onion contents. However now, thanks to the tagged based encryption, any deviation from the original path is detected during the tag verification. The attacker can still replace the plaintext of the message (this part is malleable), but cannot modify the tag. Thus, during an unsuccessful probe, s_2 would get an onion with the block:

$$\text{enc}_{\left\{pk_2^*\right\}}\left(\text{"send to"}s_1, \left\{H\left(s_1 \| s_2 \| s_3\right)\right\}\right).$$

The odds that

$H\left(s_1 \| s_2 \| s_1\right) = H\left(s_1 \| s_2 \| s_3\right)$ are negligible. Therefore, s_2 will launch an investigation with high probability. Similarly, for a successful probe, s_3 will receive an onion with:

$$\text{enc}_{\left\{pk_3^*\right\}}\left(\text{"send to"}s_1, \left\{H\left(s_2 \| s_3 \| s_4\right)\right\}\right)$$

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

The tagged based encryption acts as integrity check on the message contents, effectively preventing any modifications of the onion. For the two-hop attack to succeed without detection, apart from equality of tags, it must be the case that:

$$F\left(s_1 \| s_2 \| s_3\right) = F\left(s_1 \| s_2 \| s_1\right), \quad F\left(s_2 \| s_3 \| s_4\right) = F\left(s_2 \| s_3 \| s_1\right)$$
$$F\left(s_3 \| s_4 \| s_5\right) = F\left(s_3 \| s_4 \| s_1\right), \quad F\left(s_4 \| s_5\right) = F\left(s_4 \| s_5 \| s_1\right)$$

otherwise one of the nodes will notice the attack and start an investigation. The odds of this are n^{-4}.

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 $\text{enc}_{\left\{pk_3^*\right\}}\left(\text{"send to"}s_1, \left\{H\left(s_1 \| s_3 \| s_1\right)\right\}\right)$ as the new destination block), it would receive an onion where all blocks have been partially decrypted with the key $sk_{3,F(s_1 \| s_3 \| s_1)}$. However, the blocks in the real path would have in fact been encrypted with $pk_{3,F(s_2||s_3||s_4)}$, 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:

$$\mathrm{enc}_{\left\{pk^*_2\right\}}\left(\text{"sendto"}s_3,\left\{H\left(s_1 \| s_2 \| s_3\right)\right\}\right)$$

$$\mathrm{enc}_{\left\{pk_{2,F(s_1 \| s_2 \| s_3)}, pk^*_3\right\}}\left(\text{"sendto"}s_4,\left\{H\left(s_1 \| s_2 \| s_3\right),H\left(s_2 \| s_3 \| s_4\right)\right\}\right)$$

$$\mathrm{enc}_{\left\{pk_{2,F(s_1 \| s_2 \| s_3)}, pk_{3,F(s_2 \| s_3 \| s_4)}, pk^*_4\right\}}\left(\text{"sendto"}s_5,\left\{H\left(s_1 \| s_2 \| s_3\right),H\left(s_2 \| s_3 \| s_4\right),H\left(s_3 \| s_4 \| s_5\right)\right\}\right)$$

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'*:

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 $sk_{2,F(s_1||s_2||s_3)} + sk_{3,F(s_2||s_3||s_4)} + sk_{4,F(s_3||s_4||s_5)}$. If the path guess is correct, s_5 will receive an onion that will contain a block which can be decrypted with the key $x' + sk_{5,F(s_4||s_5||s_6)}$. 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 by 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 λ times, where λ 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. As you could see in the previous section, the risk of the successful tagging attack is limited. Moreover only the first method of tagging attacks presented above poses a serious threat. Thus, we concentrate on preventing a node from inserting any 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 information on sending time 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 a node should construct a ciphertext of *nil* using public key *P*,
- when a ModOnion is transmitted from node s_i to s_j, the node s_j may check if the ModOnion contains a ciphertext of *nil*. We suggest to perform these procedure with some fixed probability p, independently on other events. Two methods can be applied:
    - s_i presents to s_j a zero-knowledge proof that one of the blocks of the ModOnion is a ciphertext of *nil*. The proof must not show which of the blocks contains the ciphertext of *nil*.or
    - s_j stores the ModOnion. When the time window *T* for processing the ModOnion expires, s_j may ask s_i to reveal the exponents used for creating the ciphertext of *nil*. The proof is given by presenting the random exponent used for creating the ciphertext of *nil*.

The second method requires that a node either should store the random exponents used, or should create them in a pseudorandom way so that the generator can be re-run in order to re-compute the output of the generator. Note that in the proposed solution, each node is required to retain only a very small amount of data for a short time, instead of whole traffic processed by particular node.

6. Conclusions

We presented two new types of attacks on the ModOnions protocol showing that, despite modifications in [9], the protocol is still insecure against more sophisticated attacks. However, we introduce a patch that successfully defends against these new attacks without providing any new information to intermediate nodes. Our main design feature is to use encryption mode that depends on information known to the nodes anyway (that is, the previous and the next nodes on the routing path). Note that any new information would endanger the scheme due to replay attacks.

The main remaining challenge seems to be the tagging attack. The countermeasure proposed is quite dependable, but requires storing temporarily some data by the nodes processing messages.

References

[1] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", in *Communications of the ACM*, vol. 24, no. 2, pp. 84-88, 1981.

[2] G. Danezis, R. Dingledine and N. Mathewson, "Mixminion: design of a type III anonymous remailer protocol", in *Proc. of IEEE Symposium on Security and Privacy,* pp. 2-15, 2003.

[3] R. Dingledine, N. Mathewson and P.F. Syverson, "Tor: The Second-Generation Onion Router", in *Proc. of USENIX Security Symposium*, pp. 303-320, 2004.

[4] R. Pries, W. Yu, X. Fu and W. Wei Zhao, "A New Replay Attack Against Anonymous Communication Networks", in *Proc. of IEEE International Conference on Communication 2008*, pp. 1578-1582, 2008.

[5] M. Gomulkiewicz, M. Klonowski and M. Kutylowski, Onions Based on Universal Re-encryption - Anonymous Communication Immune Against Repetitive Attack", in *Proc. of WISA 2004*, LNCS 3325, Springer-Verlag, pp. 400-410, 2004.

[6] P. Golle, M. Jakobsson, A. Juels and P.F. Syverson, "Universal Re-encryption for Mixnets", in *Proc. of CT-RSA 2004*, pp. 163-178, 2004.

[7] M. Klonowski, M. Kutylowski, A. Lauks and F. Zagorski, "Universal Re-encryption of Signatures and Controlling Anonymous Information Flow", in *Proc. of WARTACRYPT 2004 Conference on Cryptology*, Tatra Mountains Mathematical Publications, pp. 179-188, 2006.

[8] G. Danezis, "Breaking Four Mix-Related Schemes Based on Universal Re-encryption", in *Proc. of International Security Conference - ISC 2006*, LNCS 4176, Springer-Verlag, pp. 46-59, 2006.

[9] M. Klonowski, M. Kutylowski and A. Lauks, "Repelling Detour Attack against Onions with Re-Encryption", in *Proc. of Applied Cryptography and Network Security Conference 2008*, LNCS 5037, Springer-Verlag, pp. 296-308, 2008.

[10] N. Borisov, M. Klonowski, M. Kutylowski and A. Lauks-Dutka , "Attacking and Repairing the Improved ModOnions Protocol", in *Proc. of* ICISC 2009, LNCS, Springer-Verlag, 2009.

[11] P. Mackenzie, M.K. Reiter and K. Yang, "Alternatives to non-malleability: Definitions, constructions and applications", in *Proc. of Theory of Cryptography Conference 2004*, LNCS 2951, Springer-Verlag, pp. 171-190, 2004.

Nikita Borisov is an assistant professor at the University of Illinois at Urbana-Champaign. His research interests are online privacy, network security, and Internet-scale distributed systems. He is the co-designer of the "off-the-record" (OTR) instant messaging protocol and was responsible for the first public analysis of 802.11 security. He is a recipient of the National Science Foundation CAREER award in 2010; he has also served as co-chair of the Privacy Enhancing Technologies Symposium in 2007 and 2008. He received his Ph.D. from the University of California, Berkeley in 2005 and his B.Math from the University of Waterloo in 1998.

Marek Klonowski has received Ph.D. in computer science (Adam Mickiewicz University, 2005) and in mathematics (Wroclaw University of Technology, 2009). His research interests include security in distributed systems and analysis of algorithms. He is an assistant professor at Wroclaw University of Technology.

Miroslaw Kutylowski is a professor of computer science at Wroclaw University of Technology. He received PhD in mathematics on theory of computing in 1986 from Wroclaw University. It was followed by Habilitation degree in 1992. He was a Humboldt Fellow in TU Darmstadt, and a faculty member in Heinz Nixdorf Institut in Paderborn. His current research interests include distributed algorithms, especially for ad hoc systems, computer security as well as legal problems of IT systems. He is involved in projects concerning e-government.

Anna Lauks-Dutka is a Ph.D. student at Wroclaw University of Technology, Poland. She also received her MS degree in computer science from Wroclaw University of Technology, Poland. Her research interests are focused on cryptography and include anonymity, e-voting and digital signatures.