Hiding Data Sources in P2P Networks

Marek Klonowki, Miroslaw Kutylowski, Bartłomiej Rozanski

2005
Hiding Data Sources in P2P Networks

Marek Klonowski, Mirosław Kutylowski, and Bartłomiej Różański

Wrocław University of Technology
Institute of Mathematics and Computer Science

Abstract. We propose a peer-to-peer (P2P) architecture where the identity of nodes holding data remains hidden, but the information itself can be efficiently fetched. This architecture can be used to protect P2P networks against malicious attacks towards nodes holding important data. In particular, our protocol can be used for maintaining access to revocation lists, blacklists and similar data, which are of growing importance for modern P2P protocols.

Keywords. peer-to-peer, anonymous communication, blacklist

1. Introduction

Peer-to-peer (P2P) technology is an attractive basis for diverse application areas, not solely limited to file sharing environments. The advantages of P2P systems are, among others: robustness to node failures, high dynamics of nodes and scalability. Despite its features P2P paradigm had not yet drawn significant interest from commercial and business applications. The reason behind this situation is that the vast majority of modern P2P protocols implement no or very weak mechanisms to protect data flow between network nodes against adversaries monitoring the traffic. Usually, very little effort is required to track down source, destination and intermediate route of the packages. Therefore the adversary can shut down a given connection very easily, i.e. by performing a Denial of Service (DoS) attack. Moreover, if an adversary can eavesdrop connections between nodes, he can easily analyze traffic to deduce user preferences. This may lead to simple attacks against certain data and allow blocking of shared contents.

1.1. P2P based secure data servers

Obviously, the idea of implementing high security data servers in P2P networks is very appealing. Robustness to network dynamics and failures, as well as self-organization makes such servers very attractive and less expensive to operate – provided that they are protected against adversaries. Let us mention two application areas:

1Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS)
2Corresponding author: Institute of Mathematics and Computer Science, Wrocław University of Technology, Wybrzeże Wyspińskiego 27 PL-50-370 Wrocław, e-mail: miroslaw.kutylowski@pwr.wroc.pl
P2P infrastructure for revocation lists  Implementing Public Key Infrastructure through P2P networks is an appealing idea [1]. One of the major problems in PKI infrastructures is to provide reliable sources of certificate revocation lists. In a typical client-server architecture there is a server (or servers) holding such data. All participants of the protocol are aware of the place of storing such a sensitive data. Therefore, all over the time it requires strong protection against extensive attacks. Our experience shows that such attacks performed by skilled adversaries are not just hypothetical, but indeed very real and dangerous. Hence, dedicated PKI servers (resistant against such attacks) become very expensive and form substantial cost of applying X.509 PKI infrastructures.

A distributed system for revocation services would alleviate problems with hardware and communication failures. An example of such a system is Cornell Online Certification Authority (COCA) [2] based on a pool of P2P servers. Usual difficulty in this case lies in maintaining information consistency – which becomes non-trivial issue in large distributed networks (COCA is based on the concept of a quorum). Obviously, much easier solution would be to store data at just one or very few nodes. These few data holders would be much better secured, if they remained hidden from adversaries. At first glance, this may seem to be an infeasible task, since we also want a (seemingly) contradictory feature, i. e. efficient fetching of data. As we show further in this paper, these two features can be conciliated in a protocol which forms a solid building-block for modern P2P architectures.

Blacklists in a P2P network  Selfish and unfair peer behavior is one of most significant problems of P2P networks nowadays. Lack of proper incentives may become an acute issue for commercial usage of P2P networks: commercial users may drop altruistic attitude and become selfish – they would use services intensively but provide very poor service. The way to solve this problem is to somehow enforce fair behavior.

In order to cope with this problem many trust mechanisms have been designed (see for instance [3]). The general idea is that a peer behaving correctly collects credentials. This this is a white-list approach. The same goal can be achieved with so called blacklists, where records of selfish peer behavior are stored and can afterwards be checked by other peers. The obvious problem with such blacklists is that they are stored at some network node(s) which is known to everyone. Hence, powerful but dishonest parties need little effort to block unfavorable information. Obviously, no security can be guaranteed here as long as the blacklist are stored at publicly known and accessible nodes.

1.2. Previous systems

Different aspects of secure access in P2P networks have been studied due to vulnerability of P2P networks to DoS attacks. A lot of work have been done in order to provide anonymous access to network resources (so this is the opposite goal: data sources are known, and the goal is to hide who is fetching them). The systems like CROWDS and Tarzan [6] were proposed with this goal in mind.

TURTLE system [4] proposes to use an overlay network based on trusted connections between friends. DoS attack becomes hard in such a system, since communication is restricted to friends. Fetching data is less efficient, it is implemented through a search in the network.
1.3. Proposed architecture

We present an architecture for hiding data sources in P2P networks, so that high protection against attacks towards network contents can be achieved. Our solution is based on universal re-encryption technique as well as dynamic data access structures. We show that our protocol is resistant to the adversary who can tap nodes and monitor their traffic.

1.4. Paper organization

In Section 2 we present message encoding methods used in our protocol. In Section 3 we describe the main protocol. In Section 4 we present an analysis of an adaptive attack performed by adversaries trying to detect data sources.

2. Encoding schemes

In this section we present cryptographic tools which will be used further in this paper to provide secure communication. We first describe Onion Routing protocol that allows anonymous communication over large scale networks using anonymity paths. Then Universal Re-Encryption (URE for short) is presented – a technique that allows partial decryption of ciphertexts. The combination of both methods – URE-Onions, which provide secure communication and protection against repetitive attacks, as well as allows partial ciphertext decryption is presented further in this section. Finally, a special type of URE-Onion called Navigator is presented – an URE-Onion with one element encoding. This allows “injection” of messages through nodes on the anonymity path in a secure way, i.e. no other node except the very recipient is aware of message transmission.

2.1. Onions

So-called onions are commonly used as a building-block for anonymous communication protocols in large scale networks [7,8,9]. Let us recall the basic mechanism. Each node in the network has a pair of keys. The public keys are widely known. To send a message $m$ to node $D$, a server $S$ chooses some random path of intermediate nodes, say $J_1, \ldots, J_\lambda$, called anonymity path. Let $\text{Enc}_X$ denote a ciphertext of $m$ computed with the public key of $X$. We assume that encryption scheme $\text{Enc}$ is probabilistic. The onion encoding $m$ has the following form (for simplicity we omit some less relevant details):

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

At the beginning this onion is sent by $S$ to $J_1$. Node $J_1$ decrypts this ciphertext. It obtains a plaintext consisting of two parts: the second part is $J_2$ and the first part is the onion with one encryption layer “peeled off”:

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

Now $J_1$ sends this ciphertext to $J_2$. The nodes $J_2, \ldots, J_\lambda$ perform similar operations, and the onion is subsequently peeled off until it finally reaches $D$. It can be easily seen that peeling off a single layer changes the onion thoroughly. Therefore, if two onions meet
conflicts. Rigid mathematical analysis of protocols based on onions in case of passive adversary is often called a conflict. In fact, anonymity of protocols based on onions is closely related to conflicts. Rigid mathematical analysis of protocols based on onions in case of passive adversary is given in [10,11,12].

2.2. Universal Re-Encryption

Let us recall El-Gamal encryption scheme: let G be a cyclic group of order q, for which discrete logarithm problem is hard. Let g be a generator of G. A private key is a random \( x < q \), and the corresponding public key is \( y = g^x \). A message \( m \) is encrypted in the following way: first some \( k, 0 < k \leq p - 1 \), is chosen uniformly at random. Then we put \( r := g^k \) and \( s := m \cdot y^k \). The pair \((s, r)\) is the ciphertext of \( m \).

El-Gamal cryptosystem has an interesting feature: everyone can re-encrypt ciphertext \((\alpha, \beta)\) so that the relation between \((\alpha, \beta)\) and a new ciphertext \((\alpha', \beta')\) is hidden for every party which does not know the private decryption key. Namely, if \( y \) is the public key used for ciphertext creation, one can choose some random \( k' \) and compute \( \alpha' := \alpha \cdot y^{k'} \), \( \beta' := \beta \cdot g^{k'} \). It is easy to see that the resulting pair \((\alpha', \beta')\) is a valid ciphertext of the same message. Golle et al. [13] presented a slightly modified El-Gamal scheme for which re-encryption of a message does not even require knowledge of the public key. It is called universal re-encryption (URE for short). In their scheme the ciphertext of message \( m \) (called URE-ciphertext of \( m \)) has the following form:

\[
(\alpha_0, \beta_0; \alpha_1, \beta_1) = (m \cdot y^{k_0}, g^{k_0}; y^{k_1}, g^{k_1}),
\]

where \( k_0 \) and \( k_1 \) are picked uniformly at random. This is obviously a pair of El-Gamal ciphertexts of messages \( m \) and 1 respectively. Decryption of this ciphertext is performed just as in the case of the original El-Gamal scheme: \( m_0 := \alpha_0/\beta_0^{\alpha_1}, m_1 := \alpha_1/\beta_1^{\alpha_1} \), and the \( m_0 \) is accepted as a valid plaintext, iff \( m_1 = 1 \).

For re-encryption of \((\alpha_0, \beta_0; \alpha_1, \beta_1)\) random values \( k'_0 \) and \( k'_1 \) are chosen. New, re-encrypted ciphertext takes the following form:

\[
(\alpha_0 \cdot \alpha_1^{k'_0}, \beta_0 \cdot \beta_1^{k'_0}; \alpha_1^{k'_1}, \beta_1^{k'_1}).
\]

2.3. Enforcing partial decryption

It is easy to enforce decryption of a message by some set of nodes [14]. Namely, the ciphertext has to have the following form:

\[
E_{x_1, x_2, \ldots, x_\lambda}(m) = (m \cdot (y_1 y_2 \cdots y_k)^{k_0}, g^{k_0}, (y_1 y_2 \cdots y_k)^{k_1}, g^{k_1})
\]

where \( y_1, y_2, \ldots, y_k \) are public keys of nodes which have to process the ciphertext, and \( x_1, x_2, \ldots, x_k \) are their corresponding private keys. Hence,

\[
E_{x_1, x_2, \ldots, x_\lambda}(m) = (m \cdot g^{\sum_{i=1}^{x_1} k_0}, g^{k_0}; g^{\sum_{i=1}^{x_1} k_1}, g^{k_1})
\]

is a ciphertext with decryption key \( \sum_{i=1}^{\lambda} x_i \) and it can be re-encrypted. Moreover, it can also be partially decrypted with key \( x_1 \).
\[ E_{x_2, \ldots, x_\lambda}(m) = \left( \frac{\alpha_0}{\beta_0^{x_j}}, \beta_0; \frac{\alpha_1}{\beta_1^{x_j}}, \beta_1 \right). \]

After partial decryption we get a URE-ciphertext for decryption key \( \sum_{i=2}^\lambda x_i \). So in order to retrieve \( m \) the ciphertext must be decrypted with private keys \( x_1, \ldots, x_k \) (however, not necessarily in this order).

2.4. URE-Onions

Unfortunately, the encryption method used for construction of onions as presented above, is cumbersome in some situations. For instance, it is impossible to manipulate internal layers without peeling off the onion. This makes it harder to prevent attacks such as repetitive attack. The problem can be alleviated with URE-ciphertexts [14]. Namely, let \( E_x(m) \) denote an URE-ciphertext of \( m \) with decryption key \( x \). As for the regular onion protocol, a path of nodes \( J_1, J_2, \ldots, J_\lambda \) is chosen at random. Then a modified onion is built from \( \lambda \) ciphertexts, called blocks. The \( i \)th block, for \( 1 \leq i \leq \lambda - 1 \) has the following form:

\[ E_{x_{J_1} + \ldots + x_{J_i}}(J_{i+1}) . \]

and an additional, last block is computed as:

\[ E_{x_{J_1} + \ldots + x_{J_\lambda}}(m) . \]

Alternatively, instead of \( J_{i+1} \) a block may contain any (random) identifier that can be understood as the address of \( J_{i+1} \) by the server \( J_i \) – in fact we apply this scheme in this way. The most important new feature of this encoding scheme is that we deviate from the encapsulation idea – messages for different routing steps are included in separate ciphertexts. Additionally, these ciphertexts are supposed to be permuted at random when the onion is processed.

Routing the onions First, a modified onion is sent to node \( J_1 \). When \( J_1 \) receives an onion, it partially decrypts and re-encrypts all its blocks. During decryption phase each block \( (\alpha_0, \beta_0; \alpha_1, \beta_1) \) is replaced by

\[ \left( \frac{\alpha_0}{(\beta_0)^{x_j}}, \beta_0; \frac{\alpha_1}{(\beta_1)^{x_j}}, \beta_1 \right) . \]

and during subsequent re-encryption phase a block of the form:

\[ \left( \frac{\alpha_0}{(\beta_0)^{x_j}}, \left( \frac{\alpha_1}{(\beta_1)^{x_j}} \right)^{k_1}, \beta_0(\beta_1)^{k_1}; \left( \frac{\alpha_1}{(\beta_1)^{x_j}} \right)^{k_2}, (\beta_1)^{k_2} \right) \]

is computed for some random \( k_1, k_2 \). After decryption phase, \( J_1 \) can read the next destination from one of the blocks. In order to hide how far the onion is from its destination, the fully decrypted block is not removed (shortening the package), but instead they are replaced by random contents. Then all blocks are shuffled (randomly permuted or sorted, etc.). Notice that the encoding scheme ensures that two onions encapsulating the same message will look completely different. Hence, the scheme is robust against repetitive attack.
2.5. **Navigators**

Let us note that the URE-ciphertext of 1 can be treated as some kind of container into which message $m$ can be inserted, by multiplying the first element of quadruple 

$$(\alpha_0, \beta_0; \alpha_1, \beta_1) = (y_{k_0}, g_{k_0}; y_{k_1}, g_{k_1})$$

by $m$. So we can “inject” $m$ into such “empty” ciphertext, and this yields a valid URE-ciphertext of $m$. A similar situation occurs in case of the URE-onions. If some message block encodes 1, one can insert any message into it by simply multiplying the first component of the block by $m$. Since the URE-onion is used as a “container” encoding some anonymous path in this context, it will be called *navigator* (in [15] more general forms of navigators are used). Navigators have many applications. For instance, they can be used as a kind of anonymous return channel. Namely, a node receiving a navigator which encodes some path to the sender, can insert its own message into the navigator and send it as an URE-onion.

3. **Hiding Data Sources**

In this section we present mechanisms for hiding identities of P2P nodes holding certain data. The protocol combines two seemingly contradictory features: accessibility of data and anonymity of the party holding it. The main idea behind our protocol is that instead of direct requests, data $x$ can now be fetched only by sending a request to one of the *access points* for $x$. These access points do not store $x$, but instead are able to contact the node holding $x$ via some *access path* consisting of $\lambda$ intermediate nodes. As a result, access points do not learn identity of the node storing $x$, and the data sources remain hidden from network participants. In addition to this, we introduce a mechanism allowing dynamic changes of the access paths. We show that this modification decreases adversary’s chance to detect the node holding $x$.

3.1. **Access paths**

**Access points** For data with identifier $x$ we define access points – the nodes where requests for $x$ can be sent. The access points do not store data $x$, but only know how to forward a request for $x$ which will eventually reach the node holding this information. Since they do not store values, their number can be relatively large, which is important to achieve protection against adversaries trying to block access to particular contents.

Network addresses of $k$ access points for $x$ are derived from values of $H(x, i)$ for $i = 1, \ldots, k$, where $H$ is some cryptographic hash function. Since function $H$ is public, every network user can find each access point for $x$.

**Access Structure** Let us consider data $x$, node $P$ holding $x$, and respective access points $A_1, \ldots, A_k$ of $x$. For each $A_i$ there is an *access path* between $A_i$ and $P$, consisting of $\lambda$ intermediate nodes $A_{i,j}$ for $1 \leq j \leq \lambda$ (see Figure 3.1). Each $A_{i,j}$ keeps a secret key $d_{i,j}$ and a navigator designed to communicate anonymously with the access point $A_i$. The access point has also a navigator for communication with $P$. 
**Access Initialization** In order to initialize access points for $x$:

- Node $P$ chooses $k$ random paths leading to access points $A_1, \ldots, A_k$. For access point $A_i$, let the access path be $A_i = A_{i,0}, A_{i,1}, \ldots, A_{i,\lambda}, A_{i,\lambda+1} = P$. For each $0 < j < \lambda$ a private key $d_{i,j}$ is generated for $A_{i,j}$; let $y_{i,j} = g^{d_{i,j}}$ be the corresponding public key. The connection between $A_{i,j}$ and $A_{i,j+1}$ is labeled by some unique random identifier $r_{i,j}$ also generated by $P$.
- For each path $A_{i,1}, \ldots, A_{i,\lambda}$, node $P$ creates a raw navigator $N_i$ using the public keys $y_{i,j}$ for $i = 1, \ldots, \lambda$. This navigator has a special form – the block which is to be read by $A_{i,j}$ is computed as:

$$ (r_{i,j} \cdot (y_{i,1} \cdot \ldots \cdot y_{i,j})^{k_{i,j}}, g^{k_{i,j}}; (y_{i,1} \cdot \ldots \cdot y_{i,j})^{k'_{i,j}}, g^{k'_{i,j}}) , $$

where $k_{i,j}$ and $k'_{i,j}$ are random numbers stored by $P$. This navigator will further get modified (according to path changes) but always allows routing towards $P$. The message block has the following form:

$$ ((y_{i,1} \cdot \ldots \cdot y_{i,\lambda})^{k_{i,\lambda}}, g^{k_{i,\lambda}}; (y_{i,1} \cdot \ldots \cdot y_{i,\lambda})^{k'_{i,\lambda}}, g^{k'_{i,\lambda}}) . $$

$P$ also prepares an initialization onion $IN_i$. It is a regular onion as described in Section 2.1. It contains an identifier of $A_{i,j+1}$ as well as $r_{i,j}$ in the layer that will be decoded by $A_{i,j}$.
- $P$ sends $N_i$ and $IN_i$ to $A_i$ via an anonymous channel (any protocol of this kind may be used).
- $A_i$ sends $IN_i$ to $A_{i,1}$. Node $A_{i,1}$ peels off $IN_i$ and reads connection identifier $r_{i,1}$ and label of the next server $A_{i,2}$. It stores the pair $(r_{i,1}, A_{i,2})$ for the later use. Then it sends the sub-onion obtained from $IN_i$ to $A_{i,2}$. Nodes $A_{i,2}, \ldots, A_{i,\lambda}$ behave in the same way, which eventually creates a connection from $A_i$ to $P$: each node $A_{i,j}$ learns its own pair $(r_{i,j}, A_{i,j+1})$ which determines the successor on the path to $P$. For technical reasons (due to the update procedure) $A_{i,j}$ informs $A_{i,j+1}$ via some secure channel about the identifier $r_{i,j}$; node $A_{i,j+1}$ stores a pair $(r_{i,j}, A_{i,j})$ as information on the connection with $A_{i,j}$.

Since initialization onion $IN_i$ is used only once, its encoding need not to be protected against repetitive attack.

**Requesting data** A user $U$ who wants to get data $x$ sends a request to one of the access points of $x$, say $A_i$. After receiving the request, $A_i$ forwards the request through the
Hiding Data Sources in P2P Networks

After reaching the request from \( U \), server \( P \) sends \( x \) to \( U \) through some anonymous channel.

**Processing a Request** A request for \( x \) arriving at access point \( A_i \) is processed as follows:

**Forwarding the request to \( P \):** access point \( A_i \) receiving a request for \( x \) uses navigator \( N_i \) and embeds the request into it. Before it is sent by \( A_i \), all fields of the navigator are re-encrypted with some random parameters. Then the navigator is sent to \( A_{i,1} \). After decryption phase one of navigator blocks should contain the identifier \( r_{i,1} \), which indicates the next node on the path (\( A_{i,2} \) for \( A_{i,1} \)). The navigator \( N_i \) is then re-encrypted and sent to \( A_{i,2} \). The procedure is repeated until the navigator reaches node \( P \), which decodes the ciphertext containing the request from \( U \).

**Delivering \( x \):** When \( P \) receives a request from \( U \) it creates some anonymous path from \( P \) to the user node and sends \( x \) through this path.

### 3.2. Paths evolution

Unfortunately, the above scenario does not guarantee anonymity of data sources in case of traffic analysis. It is so, because an adversary may tap nodes (starting with \( A_{i,0} \)) and listen to the communication. Based on traffic characteristics he may detect identity of subsequent path nodes \( A_{i,1}, A_{i,2}, \ldots \), and finally reach \( A_{i,\lambda} = P \). If no countermeasures are implemented, then it is only the matter of time when the adversary can detect \( P \).

For the reasons mentioned above we implement evolution of access path. We enhance the basic protocol with periodical changes of access paths which are triggered locally. Thus replacement nodes remain unknown to all network nodes except the nodes involved in the change. In this way we make tracing an access path harder. The idea is that a node may leave the path before the adversary observing this node has enough information in order to find the next hop on the path. Should this happen, then the adversary has to move back to the closest node which is still on the path. Further implications of path evolution are discussed in Section 4.

**Node replacement** During every period of time (of a fixed length) each path node starts a procedure of leaving the path with probability \( \beta \). We assume that the decision to leave the path is independent from other path nodes. We also assume that the procedures of leaving the path are not executed exactly at the same time at neighbor nodes - standard means for avoiding such a situation may be used. During the replacement procedure other path nodes learn about the replacement itself, but they never learn the identity of a new path member.

Let us assume that a node \( A_{i,j}, 0 < i \leq \lambda \), decides to leave the path. Then the followings steps are executed:

- \( A_{i,j} \) chooses at random a node \( A' \) that will take over duties of \( A_{i,j} \) on the access path. The connections \( (A_{i,j-1}, A_{i,j}) \) and \( (A_{i,j}, A_{i,j+1}) \) are to be replaced by connections \( (A_{i,j-1}, A') \) and \( (A', A_{i,j+1}) \), respectively. During the replacement, identifiers \( r_{i,j-1} \) and \( r_{i,j} \) need to be updated in order to correspond to the new connections.
• $A_{i,j}$ informs $A'$ about the key $d_{i,j}$. Some random key offset $\delta$ is chosen locally by $A'$. Then key $d_{i,j}$ is replaced by $d' = d_{i,j} + \delta$ by node $A'$. The update $y' = g^{\delta}$ of the public key is shown to $A_{i,j}$.

• The problem now is that the changes of the public key must be reflected in the navigator $N_i$. For this purpose, $A_{i,j}$ sends $y'$ together with identifier $r_{i,j}$ to $P$. Of course, $A_{i,j}$ does not know $P$. However, it can open an anonymous channel to $P$ using onions with requests for $x$ processed by $A_{i,j}$ towards $P$. Namely, assume that a request is sent by $A_{i,j}$ using an onion $N$. Right afterwards $A_{i,j}$ may re-send re-encrypted $N$ with the first message block element multiplied by some fixed number $\pi$. Then it re-sends $N$ twice – after re-encryption and multiplying the same element by $y'$ and $r_{i,j}$, respectively.

• If $P$ receives some request $z$ and then $z \cdot \pi$, then it means that some $A_{i,j}$ wants to open an anonymous channel. $P$ expects next messages to contain the key update. One of the following messages should therefore contain $z \cdot y'$ and another one $z \cdot r_{i,j}$. So $P$ can retrieve $y'$ and $r_{i,j}$. From the value of $r_{i,j}$ node $P$ recognizes which node on the path has transferred its duties and knows that the public key update of the $j$th node is $y'$.

• $P$ informs $A_i$ about necessary changes in $N_i$. Namely, through some anonymous secure channel it sends all necessary data to $A_i$: the number $j$, and for each block of $N_i$ of the form

\[
\left( u \cdot (y_1, y_{t+1} \ldots y_{t})^k_{i,t}, g^{k_{i,t}}; (y_1', y_{t+1}' \ldots y_{t}')^k_{i,t}', g^{k_{i,t}'} \right),
\]

for $t \geq j$, node $P$ sends

\[
\left( (y')^k_{i,t} \right) \quad \text{and} \quad \left( (y')^{k_{i,t}'} \right).
\]

Node $A_i$ multiplies the first and third component of this block by the numbers obtained.

Obviously, the same mechanism may be applied in order to update identifiers $r_{i,j-1}$ and $r_{i,j}$. We skip the details here.

4. Resistance to Dynamic Adversary

In order to protect against adversaries trying to track down data sources we have introduced access paths evolution mechanism, where decisions how to change are performed independently and randomly by the nodes on the paths – with no central control. This should provide high protection against adversaries who tap intermediate nodes and per-
form traffic analysis. Let $S_0, S_1, \ldots, S_\lambda$ denote an access path from the access point $S_0$ to the node $P = S_\lambda$. The adversary will tap $S_i$ (starting with $i = 0$) and analyze incoming and outgoing traffic trying to determine $S_{i+1}$. For simplicity we assume that during each round the adversary may discover identity of $S_{i+1}$ with probability $\alpha$ independently of the attack history. The procedure is repeated until the adversary reaches $P$. We assume that the adversary makes no mistakes – either he guesses $S_{i+1}$ or remains observing $S_i$. Obviously, without path updates the adversary would reach $P$ in expected time $\lambda/\alpha$.

**Path Evolution and the Adversary**  If there was no evolution of the paths, an adversary who reached some server on the path, would know for sure all nodes between this server and the access point starting the path. Since path evolution takes place, nodes previously known to the adversary are supposed to leave the path randomly. As a result, only some subset of the path prefix remains known to the adversary. For simplicity of the analysis we assume that once the adversary guesses that some node is on a path, afterwards he can always verify whether it still belongs to this path.

The first observation is that the further the adversary advances (say up to $S_i$), it becomes more likely that the nodes left behind have already left the path. The probability that a given node has left the path grows with its distance to $S_i$. Additionally, there is also a fair chance that the adversary will “choke” at some point of the path – this happens when the node tapped leaves the path! In such case the adversary has to backtrack to the closest known path node – the one that is known to the adversary and has not left the path in the meantime. If there is no such node left, the attack must start from the very beginning.

The attack can be described by the following stochastic process:

- the adversary performs a random walk on a path of length $\lambda$, he starts from the left and the goal is to reach the opposite side of the path,
- at each step with probability $\alpha$ the adversary may move one step right,
- the nodes visited by the adversary may be marked – a node becomes marked, when the adversary enters this node from the left,
- during each step each marked node becomes unmarked with probability $\beta$; unmarking occurs independently for each node,
- if the node currently pointed by the adversary becomes unmarked, he has to backtrack to the rightmost marked node.

This process corresponds to the situation that the adversary monitors only the last known node on the path. We should also consider another type of adversary, called a strong adversary, who monitors all known nodes on path prefix and is always trying to fill path “gaps” (unknown intermediate nodes). In case of a strong adversary:

- if a node is unmarked and its predecessor is marked, then it becomes marked with probability $\alpha$;

The crucial question is how fast can the adversary advance to the right. Of course, the answer depends on $\alpha$ and $\beta$. Generally we may assume that $\alpha \leq \beta$, since performing traffic analysis is always time consuming, while exchanging nodes on the path can be performed fast and with little effort. The main problem is how large must be $\beta$ compared to $\alpha$ so that the adversary would have to perform a very large number of steps in order to reach the end of the access path.
The most important point for the adversary is to keep moving forwards, because if he stays longer at some position, then more and more nodes behind him get unmarked. As a result, backtracking may require a large number of positions. Additionally, in a long period of time there is a fair chance that a short series of failures appear (un-markings of the current node). When this happens and the number of marked nodes behind the adversary is smaller than the length of this series, the adversary has to start from the beginning. In the following subsections we shall see that such a situation occurs quite frequently.

4.1. Probability of Adversary’s Success - Analytical Approach

4.1.1. Number of Nodes behind the adversary

The adversary taps $S_l$ for some $l < \lambda$. Our aim is to estimate the number of nodes behind $S_l$ that remain marked. Let $X_i$ be a random variable indicating that node $S_i$ is still marked. Then $Y_j = \sum_{i=1}^{j} X_i$ is the number of marked nodes up to $S_j$. Let us investigate the distribution of $X_i$. Of course it depends on the number of steps that have passed after marking this node by the adversary:

$$\text{Pr}(X_i = 1) < (1 - \beta)^{l-i}.$$ 

Indeed, at least $l - i$ rounds has passed after the last visit of the adversary in $S_l$ - he needs at least $l - i$ moves to reach $S_l$. In fact, the actual values are usually much smaller, since the advancement of the adversary is significantly slower. It is clear that $E[X_i] = \text{Pr}(X_i = 1)$. So by linearity of expectation,

$$E[Y_{l-1}] < \sum_{i=1}^{l-1} (1 - \beta)^{l-i} = (1 - \beta) \frac{1 - (1 - \beta)^{l-1}}{\beta}.$$ 

Now let us estimate the variance of $Y_{l-1}$. We have $E[X_i] = E[X_i^2]$, since random variables $X_i^2$ and $X_i$ have exactly the same distribution. Since $E[Y_j] > 0$, we have $\text{Var}[Y_{l-1}] = E[Y_{l-1}^2] - (E[Y_{l-1}])^2 < E[Y_{l-1}^2]$. Now

$$E[Y_{l-1}^2] < E[(X_1 + \ldots + X_{l-1})^2] = \sum_{i=1}^{l-1} E[X_i^2] + \sum_{i \neq j} E[X_i X_j].$$

Since $X_i$ and $X_j$ are independent random variables we have

$$\text{Pr}(X_i X_j = 1) < (1 - \beta)^{2l-i-j}$$

and so

$$E[Y_{l-1}^2] < E[Y_{l-1}] + \sum_{i \neq j} (1 - \beta)^{2l-j-i}$$

$$< (1 - \beta) \frac{1 - (1 - \beta)^{l-1}}{\beta} + \left( \sum_{i=1}^{l-1} (1 - \beta)^{l-i} \right)^2 - \sum_{i=1}^{l-1} ((1 - \beta)^{l-i})^2$$

$$= (1 - \beta) \frac{1 - (1 - \beta)^{l-1}}{\beta} + \left( (1 - \beta) \frac{1 - (1 - \beta)^{l-1}}{\beta} \right)^2 - \frac{(1 - \beta)^2 (1 - (1 - \beta)^{2l-2})}{1 - (1 - \beta)^2}.$$ 

Now, using the approximation of variance and expected value of $Y_{l-1}$ one can see that the chance that many nodes except the current node $S_l$ of the adversary are still marked
is indeed slim. For this purpose we can use Tchebychev’s inequality. For instance, for $\beta = 1/2$ the chance that two or more previously visited nodes are still marked is smaller than $1/30$. Note, that this approximation does not depend on actual position on the path (parameter $l$) - the estimation holds for any position, even those close to the end of the path.

4.1.2. Probability of two Consecutive Successes during $k$ Bernoulli Trails

Let us consider a sequence of $k$ Bernoulli trials, where probability of success in each trial is $\gamma$. Then the probability $\Pr\{\gamma, k\}$ that at least once during the sequence we achieve two successes in a row for $k \geq 2$ can be computed as $1 - p_1(k) + p_2(k)$ where

$$p_1(k) = \frac{(1-\gamma - \sqrt{\Delta} - (1 - \gamma^2))(1-\gamma + \sqrt{\Delta})^{k-1}}{-\sqrt{\Delta}}$$

$$p_2(k) = \frac{(1 - \gamma^2 - \frac{1-\gamma + \sqrt{\Delta}}{2})(1-\gamma - \sqrt{\Delta})^{k-1}}{\sqrt{\Delta}}$$

and

$$\Delta = \sqrt{(1-\gamma)^2 + 4(1-\gamma)\gamma}.$$  

In particular for $\gamma = 1/2$ we get:

$$\Pr\{1/2, k\} = 1 - \left(\frac{5 + 3\sqrt{5}}{10}\right)\left(\frac{1 + \sqrt{5}}{4}\right)^k - \left(\frac{5 - 3\sqrt{5}}{10}\right)\left(\frac{1 - \sqrt{5}}{4}\right)^k.$$  

Using this formula we can estimate that in series of $k = 10$ trials two consecutive successes will occur with probability $\Pr\{1/2, k\} > 0.75$. For $k = 20$ trials, we have $\Pr\{1/2, k\} > 0.95$. One can easily see that if the adversary has to backtrack in two consecutive rounds, and no more than one marked node remains on the path, he will return to the access point and will have to start the attack from the beginning. Unfortunately for the attacker, probability of such an event is pretty high.

4.2. Experimental results

Although the analytical estimations presented in previous subsection are based on pessimistic assumptions, the phenomena discovered are confirmed by our experimental results. The following table presents results of experiments consisting of 100.000 trials performed for each choice of parameters. The probability that a given node is replaced by another node within a single step equaled $\frac{1}{2}$, and the probability that an adversary finds the next path node during a single step was $\frac{1}{K}$.
experiment 1: 0 0 -1
experiment 2: 1 1 0 1 0 -5
experiment 3: 1 1 1 1 0 -1 1 0 -7
experiment 4: 0 0 0 -1
experiment 5: 0 0 -1
experiment 6: 1 1 1 0 -5
experiment 7: 0 1 0 1 0 0 0 -3
experiment 8: 1 1 1 0 0 1 0 1 1 0 -2 1 1 0 1 0 -5
experiment 9: 0 1 1 0 1 0 1 0 0 -7
experiment 10: 0 1 0 1 1 0 -5
experiment 11: 0 1 1 1 1 0 1 1 0 -1 4
experiment 12: 0 1 1 1 1 0 -5
experiment 13: 0 1 1 1 0 1 1 0 -1 1 0 -7
experiment 14: 1 1 1 1 0 -5
experiment 15: 1 0 1 1 1 0 1 1 0 -3

Table 2. trajectories of the adversary

<table>
<thead>
<tr>
<th>path length = 10</th>
<th>path length = 15</th>
<th>path length = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K )</td>
<td>2 3 4</td>
<td>2 3 4</td>
</tr>
<tr>
<td>weak adversary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 steps</td>
<td>23803 1806 230</td>
<td>3631 63 3</td>
</tr>
<tr>
<td>100 steps</td>
<td>45627 4271 531</td>
<td>9204 147 7</td>
</tr>
<tr>
<td>strong adversary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 steps</td>
<td>50651 6403 712</td>
<td>17787 480 15</td>
</tr>
<tr>
<td>100 steps</td>
<td>82442 16426 2036</td>
<td>47645 2193 62</td>
</tr>
</tbody>
</table>

Table 1. the number of successful attacks

The following phenomena can be observed:

- there is a big difference between a strong adversary and a weak adversary that monitors only the last known node. However, increasing frequency of path changes compensates for this problem,
- if path length is 20 and the rate of path change is 2 times bigger than the advancing adversary, then he succeeded for none of 100,000 trials to reach the end of the path within 50 steps– regardless of the adversary model.

Let us illustrate some typical “trajectories” of the adversary: the numbers illustrate how many steps forward are done during consecutive steps (the leftmost number denote the advance during the first step). Backtracking \( i \) positions is marked as \(-i\). A trajectory terminates, once the adversary returns to the access point. This occurs if the current position of the adversary gets unmarked and there are no other marked positions left.

These examples show that after traveling along the path for a longer time it happens very often that the adversary is returned to the beginning of the path in just one step.

4.3. Attack success chances – analytical approach

Determining probability of reaching the right end of the path, when various protocol parameters are used, helps to understand adversary’s chances. We have deter-
mined exact values of success probabilities, as a function of \( \alpha \) and number of attack rounds. The computation was performed for paths of length 8 with \( \beta = 0.5 \) and \( \alpha = 0.05, 0.10, \ldots, 0.45, 0.50 \). Figure 2 presents the results. Numerical values are given in the following table. The columns represent respective \( \alpha \)'s and rows are computed for different number of attack rounds (20, 24, 28 and 32).

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.003</td>
<td>0.008</td>
<td>0.017</td>
<td>0.031</td>
<td>0.051</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002</td>
<td>0.005</td>
<td>0.012</td>
<td>0.024</td>
<td>0.043</td>
<td>0.069</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.002</td>
<td>0.006</td>
<td>0.015</td>
<td>0.031</td>
<td>0.054</td>
<td>0.085</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.003</td>
<td>0.008</td>
<td>0.019</td>
<td>0.037</td>
<td>0.065</td>
<td>0.102</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3. Probabilities of attack success - numerical values**

As can be seen, even for short paths of length 8 an adversary needs many rounds to raise the chance of reaching path end up to 0.1. This happens even for \( \alpha = \beta = 0.5 \). Should \( \alpha \) fall below this value, the chances drop below 0.01 just for \( \alpha = 0.3 \). What can also be observed is that for \( \alpha = 0.5 \cdot \beta \) the probability of success drops to 0.003 which is 30 times smaller than in case of \( \alpha = 0.5 \cdot \beta \). This leads to the following conclusions:

- even for moderate access path lengths the adversary is expected to need many rounds,
- the protocol can be refined when the ratio \( \alpha/\beta \) can be estimated.
5. Final Remarks

The process of gaining knowledge about the access paths by the adversary can be modeled stochastically and analyzed. Certainly, our approach in this paper is not the only possible one. Our goal was just to support the protocol design and show its features, rather than fully explore the stochastic behavior of the attacks. Further research on this topic is possible.

References