Cent. Eur. J. Comp. Sci. • 2(4) • 2012 • 412-425
DOI: 10.2478/s13537-012-0034-4

Central European Journal of **Computer Science**

Introducing OGR – a new approach to anonymous IP datagram transmission based on the Chaumian 'onion' encryption and the KISS principle

Research Article

Marek Kozłowski*

Warsaw University of Technology
Faculty of Mathematics and Information Sciences,
Pl. Politechniki 1, 00-661 Warsaw, Poland

Received 31 October 2011; accepted 15 October 2012

Abstract: The goal of this paper is to introduce a technique, called 'OGR', that allows anonymous transmission of IP datagrams of any kind. During the last decade numerous anonymizing solutions implementing the Chaumian 'onion' encryption have been proposed. Many of them are dedicated to specific services: either anonymous mail delivery or anonymous storage sharing, while the other ones, referred as *low-latency systems*, work real-time and are able to anonymize any TCP streams (with preference to HTTP), but poorly deal with or even don't support other transports. 'OGR' is strongly inspired by the original Chaumian *Mix-net* model and aims at universality (the ability to work at the level of IP datagrams rather than upper-layers protocols) and maximal simplicity (the KISS principle). It discards the idea of fixed *circuits* (or *cascades*) present in most *low-latency* solutions in favor of independent, intelligent packets. Although this approach may occur less efficient for huge data streams it's able to handle any request-reply protocols including those operating over UDP as well as all applications where other transports or raw IP packets are to be used.

Keywords: anonymous communication • onion routing • IP datagram • TCP/IP network

1. Introduction

Since the late 90's several systems for anonymizing network traffic have been proposed. Most of them utilize the idea of 'onion encryption' introduced by Chaum [2]. Fixed-size messages are encrypted several times with different keys and delivered via a sequence of anonymity servers (mixes). Each of them decrypts one layer and reveals a successive anonymity server on route of the message: it knows only its direct predecessor and successor, but doesn't know the origin, destination nor the message contents. Thanks to uniform size and decryption it's difficult for traffic analyzers to match packets that enter and leave a mix.
For a particular system several elements have to be defined, including: key management, specifying paths for reply messages, preventing attacks, mix management etc.

* *E-mail: m.kozlowski@mini.pw.edu.pl*

Therefore two main approaches to constructing anonymizing systems can be distinguished: a message-based and a circuit-based. The first of them applied to anonymous remailers is more flexible and doesn't require any extra maintenance. Unfortunately, it involves asymmetric cryptography which introduces excessive delay to packet processing at each mix. Moreover a set of disposable symmetric keys has to be distributed embedded within packets. As a result, the size of data efficiently transmitted in them is reduced. The latter one, preferred by low-latency systems, uses fixed routes and preset ephemeral symmetric keys. It benefits in decreased packet decryption time and reduced supplementary header fields. However, solutions proposed for key management introduce some level of inflexibility and require additional traffic. They are often capable of handling only connection-oriented protocols.
In this article I propose a communication model designed for anonymous delivery of IP datagrams. Only symmetric cryptography is used for packet processing, but the idea of fixed paths mentioned above is discarded in favor of independent, intelligent packets. While it may occur less efficient for huge data streams this approach can handle any request-reply protocols including those which operate over UDP as well as all applications where other transports or raw IP packets are to be used. It aims at applying numerous extensions or modifications to the original Chaumian model present in related work while keeping it lightweight, simple ('the KISS principle') and flexible.
The article is organized as follows.
Section 2 covers the original Chaumian concept of 'onion encryption'. It starts with the idea of a mix, then discusses the benefits of arranging mixes into cascades and the key distribution technique used in this model. The Chaumian concept of an untraceable anonymous return path ends that section.
Section 3 presents some recent algorithms that extend the original model. The first part discusses anonymous remailers. The stress is put on the problem of packet integrity check (protection against packet tagging) while providing onion blocks for replies. Then a few low-latency anonymous systems are presented and the idea of preset circuits is introduced.
Section 4 outlines the algorithm called OGR[1]. Emphasis is put on the new approach to key distribution, route specifying and packet processing. Note that the aim of this paper is to present a general concept rather than a detailed implementation nor protocol specification.
The article is ended with section 5 that discusses benefits of the approach that has been introduced and open problems for further research.

2. Chaumian 'onion' encryption

In the article *Untraceable electronic mail, return addresses, and digital pseudo-nyms* [2] Chaum presents a model for anonymous mail delivery – *Mix-net*. In this section I shortly outline its main idea called *onion encryption* which is the cornerstone for most modern anonymizing solutions with special emphasis on those elements that are particularly important for further discussion.

2.1. Anonymity servers (mixes)

Clients remain anonymous to their parties and their communication stays secret for any traffic analyzing hosts thanks to anonymity servers called *mixes*. All the traffic to be anonimized is carried in fixed-size packets (derivative works use the terms *chunks* or *cells*). If uniqueness is requisite the data within packets is padded with random bits or time stamps. All packets passing through a mix are modified to make matching incoming and outgoing ones difficult. Chaumian model uses asymmetric cryptography (RSA algorithm) for this transformation: clients encrypt messages (containing data and their destination addresses) with a mix's public key and the mix decrypts it with its private key. Decryption can be denoted as:

$$pub_M(rand + addr_B + data) \rightarrow addr_B + data, \qquad (1)$$

[1] *The name is an acronym for 'Onion General-purpose Routing'. The name alludes to the quote from the 'Shrek' movie: 'Ogres are like onion. [...] Ogres have layers. Onions have layers, you get it?'*

where:

- B denotes the recipient, M – the mix,
- $rand$ is a random string or a time stamp added for uniqueness,
- $pub_M()$ denotes encryption with a public key of M; '+' is a concatenation operator.

2.2. Cascades and onion encryption

Any single mix can be intercepted by attackers. Negative impact of it can be overcome if messages pass through several mixes (so-called *cascade*). Before entering a cascade combined of mixes: $(M_1, M_2, ..., M_N)$ a message is encrypted N times with the keys: $(pub_{M_N}, pub_{M_{N-1}}, ..., pub_{M_1})$ and transformed to the form:

$$pub_{M_1}(rand_1 + addr_{M_2} + pub_{M_2}(rand_2 + addr_{M_3} + pub_{M_3}(rand_3 + addr_{M_4} + ... \\ ... + pub_{M_N}(rand_N + addr_B + data)...))). \tag{2}$$

Every mix decrypts a packet with its own private key according to (1) and obtains the address of the successive mix. Any mix of a cascade knows only addresses of its direct predecessor and successor and is unable to read the plain message. Such layered preparation of a packet is commonly referred as *onion encryption*.
Each mix removes first $BSIZE = length(rand + addr)$ bytes of a packet after it's decrypted, so $BSIZE$ random bytes (*junk*) has to be appended for keeping it fixed-size. Blocks of junk must be encrypted/decrypted independently of the rest of a packet. For this reason each packed is formed of blocks of the size $BSIZE$ encrypted separately. The same fragments of data sent via previously used cascade should form unique blocks. This may be achieved by using the CBC (Cipher Block Chaining) mode for encryption, although Chaum proposed simpler and more efficient solution: public key cryptography is used for the first block only; random bytes stored in the first block act as a (symmetric) key used to encrypt next blocks. Instead of the form (2) we obtain:

$$pub_{M_1}(sym_1 + addr_{M_2}) + sym_1(pub_{M_2}(sym_2 + addr_{M_3})) + sym_1(sym_2(pub_{M_3}(sym_3 + addr_{M_4}))) + ... \\ ... + sym_1(sym_2(...(sym_N(data_1))...)) + ... + sym_1(sym_2(...(sym_N(data_k))...)). \tag{3}$$

Each intermediate mix processes messages as illustrated by Figure 1; the steps denoted 1-5 are as follows:

1. decrypt the first block ($BSIZE$ bytes) of the message with the mix's private key,
2. decrypted first block contains the *next hop IP address* and a disposable symmetric key,
3. decrypt each of successive blocks with this symmetric key,
4. remove the first block from the message,
5. append a block of random data at the end of the message to keep its size fixed.

Processed message is then sent to the host specified by the *next hop IP address* revealed in the step 2.

2.3. Untraceable return address

The algorithm just described assures anonymity and secrecy. Note that it works unidirectionally. Senders know addresses of message recipients but recipients of onion-encrypted messages don't know addresses of their senders. Some mechanism for replying to anonymously delivered messages must be provided. For this purpose Mix-net utilizes so-called untraceable return addresses. A sender selects a return path cascade: $(M_1^*, M_2^*, ..., M_N^*)$, obtains mix addresses and public keys, prepares an onion:

$$pub_{M_1^*}(sym_1^* + addr_{M_2^*}) + sym_1^*(pub_{M_2^*}(sym_2^* + addr_{M_3^*})) + sym_1^*(sym_2^*(pub_{M_3^*}(sym_3^* + addr_{M_4^*}))) + ... \\ ... + sym_1(sym_2(...pub_{M_N^*}(sym_N^* + addr_B))...)). \tag{4}$$

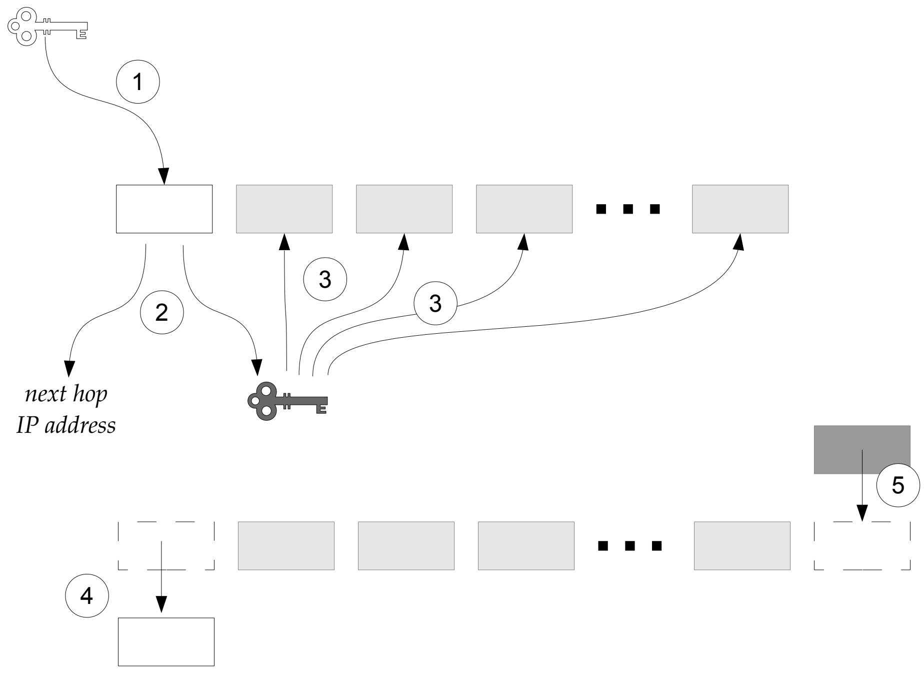

Figure 1. Packet processing by a mix.

and includes it as first N blocks of the payload (the maximal length of efficient data must be then reduced by $N*BSIZE$)[2] – see Figure 2. The recipient encrypts their replay with the key attached to the message, precedes it with above onion and sends to M_1^*. Packets containing messages and replays are undistinguishable for mixes – in both cases mixes operate in the same way.

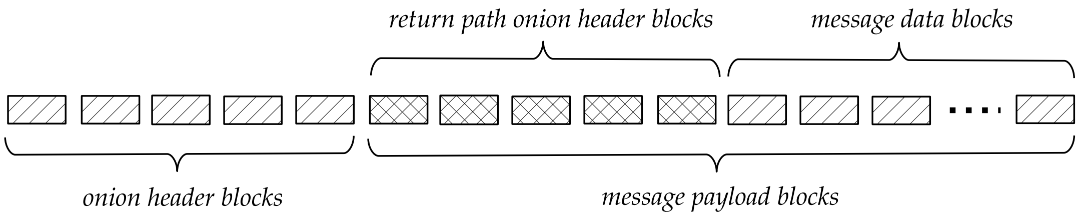

Figure 2. Message containing untraceable return address.

[2] *I assume both cascades of the same length N. In fact their lengths can differ.*

3. Related work

Interest in research in the area of anonymizing systems started to raise dramatically in the late 90's[3]. Three classes of algorithms / protocols can be distinguished:

- anonymous remailers: cypherpunk, Mixmaster[4] and Mixminion[5] [3],
- anonymous (P2P) storage sharing, for example: Freenet[6], GNUnet [1], MUTE/Calypso[7],
- general purpose low-latency anonymization networks, for example: TOR [4], JonDonym (formerly: JAP, Java Anon Proxy)[8], Crowds [6], MorphMix[9] [7], Tarzan[10] [5].

This article describes anonymous datagram transmission. It is not intended as a comprehensive listing nor detailed comparison of modern anonymity-related solutions. My interest in this section is limited to algorithms for similar models of transmission, that is those in which:

- host A sends packets (messages or fragments of data streams) to host B,
- A may receive replies from B,
- packets are exchanged in secure and untraceable way,
- host A stays anonymous for other hosts (including the host B),
- only limited delay is allowed.

Therefore any P2P or F2F community networks for anonymous resource sharing stay out of scope of this paper. For anonymous remailers I skip techniques which introduce too much latency or excessive traffic (for example: *batches* and *dummy traffic* present in [2] and [3]) and omit concepts that strictly concern the ESMTP protocol as well as mix discovery techniques and mix performance/capabilities.

3.1. Anonymous remailers

The simplest remailer called *cypherpunk* (type I remailer) uses the transformation (1) (see 2.1). A template: an ASCII file containing recipient's address, subject, cypherpunk markers and a message body is PGP-encrypted with remailer's public key and sent to the remailer of choice as an e-mail attachment. The remailer decrypts the file, then generates RFC 5322 compliant e-mail based on the template and sends it via ESMTP (RFC 5321) to the recipient. For higher level of security remailers can be chained (cascaded); in such case a file must be onion-style PGP-encrypted.
Mixmaster[4] (type II remailer) is a simple and relatively exact implementation of the Chaumian algorithm described above. Two important extensions to Mix-net are introduced:

[3] *For a comprehensive listing of related papers ordered chronologically see the bibliography of the Free Haven Project:* http://freehaven.net/anonbib
[4] *Mixmaster Project* http://mixmaster.sourceforge.net,
Mixmaster Protocol Version 2 http://tools.ietf.org/html/draft-sassaman-mixmaster-03,
Mixmaster Protocol Version 3 (draft) http://www.eskimo.com/~rowdenw/crypt/Mix/draft-moeller-v3-01.txt
[5] *Mixminion: Design of a Type III Anonymous Remailer* http://mixminion.net
[6] *Freenet Project* http://freenetproject.org
[7] *Calypso Project* http://calypso.sourceforge.net
[8] *JonDonym Project* http://anonymous-proxy-servers.net/en/,
JAP Protocol http://anon.inf.tu-dresden.de/develop/doc/mix˙short/
[9] *Morphmix Project* https://home.zhaw.ch/~rema/projects/morphmix
[10] *Tarzan Project* http://pdos.csail.mit.edu/tarzan

- Messages are sent within fixed-size packets. Should the need arise messages are fragmented – the last mix is responsible for message reassembling. Therefore packet processing is nonuniform: different at intermediate mixes, different at final mixes for single-packet messages (message delivery) and different at last mixes for fragmented messages (message reassembling and delivery). Thus headers at each layer contain an additional field: packet type identifier, which determines packet format and processing.
- Packet blocks are encrypted separately. Any change in a further block (a payload block) can stay undetected by subsequent mixes. This may result in marking packets (so called *tagging attack*). To prevent it, each layer's header contains another additional field: a whole packet digest. Using message digests implies:
  - junk added by mixes cannot be random, but it must be constant or pseudo-random (obtained by a pseudo-random generator started with a given sequence),
  - no replays are possible, because packet digests cannot be determined at the time of preparation of return address blocks.

Mixminion[5] [3] (type III remailer) is the most advanced and actively developed specification. From our perspective the most important changes include:

- ciphering mix-to-mix communication with TLS,
- frequent mix public *key rotation* along with keeping logs of messages encrypted for the current key for replay prevention. Validity periods (individual for each mix) are to be published together with keys,
- diversifying mixes by capabilities and performance; introducing global, public mix information databases (*directory servers*),
- using immune to tagging attack *Single Use Reply Blocks* (SURBs) for anonymous replays.

Contrary to Mixmaster, Mixminion doesn't use message digests for each layer. Instead, a message header contains two onion-encrypted sub-header blocks of a predefined size. The first one encodes the initial path. At some selected mix (a *crossover point*) the *swap* operation is performed: the second header is decrypted with the payload's hash and both headers are swapped. In other words: the further path is encoded in the second sub-header and the payload. In case of tagging the packet gets unreadable/undeliverable or the modification gets hidden and can be observed only by its recipient (after the final decryption).

3.2. Anonymous general purpose low-latency networks

Anonymous remailers were being developed evolutionary. For low-latency communication several competitive projects raised simultaneously. Therefore they are presented more parallelly than in the previous subsection.
Although low-latency systems are more flexible than remailers, most of them support only a very limited range of protocols. Usually they assume connection-oriented proxied or proxyable communication over the TCP transport; HTTP is a preferred protocol (*Crowds* – only HTTP, *TOR* – TCP/SOCKS with HTTP preference, *JonDonym* – TCP, HTTP assumed, *MorphMix* – TCP 'for internet usage'). Only *Tarzan* operates on the IP layer.
Assumed model of communication reflects in the idea of *curcuits*. A circuit is a cascade preset for a new session. Before opening a session the host selects the mixes $(M_1, M_2, ..., M_N)$ and anonymously negotiates with each mix M_i symmetric keys (Diffie-Hellman's key exchange) used for onion encryption. Circuits eliminate the need of public key cryptography for packets containing data (TCP stream segments). Instead of next mix addresses circuit IDs may be used for path discovery. Circuit management requires additional circuit control packets to be introduced aside from normal data packets. (M_i, M_{i+1}) links may be TLS encrypted for higher level of security.
In JonDonym[8] clients don't build cascades by selecting individual mixes for a new TCP connection. Instead they choose one of preset 3-mix cascades. All cascades are separate, that is no cascades share common mixes. The first mix of each cascade represents it to a client (provides information on the cascade, public keys of the mixes, symmetric keys for host-to-mix encryption etc) and to the *InfoService* – a database that stores information on the whole system. Fixed-size packets are onion-encrypted for each mix. Only the part of each packet that stores the circuit ID is additionally (AES-)encrypted between mixes. A client must trust to the organization that manages the mixes and offers a circuit. Most circuits that offer low-latency and high bandwidth are commercial and payable.

TOR, MorphMix and Tarzan build circuits iteratively hop-by-hop. An existing circuit is used to anonymously negotiate with a mix that's being added. Mixes selection algorithms differ:

- in TOR mixes are selected by a client from all active ones,
- in Tarzan mixes are selected randomly from some pool available to the client,
- in MorphMix mixes are selected by the last mix assisted by some trusted mix called a *witness*.

TOR [4] is the most popular anonymity software. Developed by the same researches as Mixminion it shares numerous similar concepts (for example: TLS link encryption, key rotation, exit policies, directory services etc). Fixed-size cells are onion-encrypted. Circuit IDs are used for path selection. TCP session should periodically (and transparently) migrate to other circuits. Thus new circuits are built in advance.
MorphMix[9] [7] and Tarzan[10] [5] are both abandoned projects. Morphmix has many in common with TOR but aims at elimination of central mix database in favor of decentralized P2P model. Tarzan is interesting as a solution that anonymizes IP packets rather than TCP streams – it aims at the goals we'd like to achieve. Original IP datagrams (without sender's information) are encapsulated in UDP packets by the first mix. The last mix delivers it with its own sender's data in a manner similar to NAT (RFC 3022). The authors put focus on P2P mix management and hiding packets with *traffic mimic*. Unfortunately the transmission protocol seems to embed numerous weaknesses:

- the transmission algorithm and packet format are not clearly specified,
- it appears that variable-size packets are allowed,
- the sense of circuit management with UDP packets seems to be questionable,
- public keys are permanent (no key rotation),
- there are satisfactory solutions for the TCP transport. Datagram anonymization seems to be reasonable for UDP, request-reply protocols and dedicated raw IP packets. This kind of traffic may be quite sparse. The cost of building circuits may be disproportionate for series of just a few packets.

Crowds [6] doesn't use onion addressing nor circuits for specifying a path. The path 'forward' is selected randomly: each Crowds member (so-called *jondo*) with some probability delivers a packet to the destination or forwards it to any other (randomly selected) one on behalf of itself and records the 'transaction'. Those 'transactions' allow determining return paths. No onion encryption nor mixing techniques are applied. The reason Crowds is mentioned here is an interesting insight: if each mix registers recent traffic passing through (it seems to be requisite for replay prevention) there is no need for specifying onion-encrypted return paths.

4. OGR

4.1. A new approach

Anonymizing solutions presented in the previous section are based on repeated packet encryption. For computational purposes symmetrical cryptography should be applied. The essential problem concerns informing a mix which symmetrical key should be used for packet decryption. Two main approaches can be distinguished:

1. Keys are embedded into packets. Each mix uses its private key to reveal a symmetric key for packet processing.
2. Inflexible packet paths (circuits) are preset. Each mix-to-path binding is assigned a key.

In the first one packets traverse the network independently and store enough information to reach their destination. This solution may seem natural for IP datagram transmission which bases on the packet switching paradigm. Unfortunately this solution involves three inconveniences:

- Time-consuming asymmetric cryptography must be applied *n* times by the sender and once at each mix.

- Symmetric keys for each mix must be embedded into packets. For algorithms with longer keys a few hundred bytes have to be additionally occupied by packet headers.
- When a mix rotates its public key a new one has to be propagated to clients before subsequent packets may be sent.

Introduction of built hop-by-hop circuits eliminates asymmetric cryptography for packet processing and reduces the number of bytes occupied by packet headers. Public key rotation doesn't influence packet processing. Those problems are overcome for the cost of adding some inflexibility and additional traffic before an anonymous packet transmission starts:

- Setting up new circuits on demand involves setting up *n* TLS sessions and carrying *n* Diffie-Hellman or similar key negotiations. For longer data streams the cost is negligible but the overhead is questionable if only a few anonymous datagrams are to be send (as for a simple request-reply protocol or protocol with sparse keep-alives).
- Circuits are bound to clients' IP addresses: if a client changes its IP address all circuits have to be re-created; a few network interfaces cannot share circuits.
- Circuits have no redundant fragments.
- If one of intermediate mixes gets compromised, overloaded or unavailable there is no immediate reconfiguration procedure.
- The cost of building a new circuit of the same length is constant (existing circuits don't accelerate it).

Therefore I propose a new, simple and flexible solution that combines advantages of both: message oriented and circuit-oriented solutions. A client anonymously negotiates symmetric keys with a flexible set of mixes, each of them stays ready for use for some amount of time. This set can be modified anytime. Should the need arise a client can choose any sequence of those mixes for datagram delivery and sent data without any additional cascade preparation. If some reply datagram is expected then the same or any other sequence of mixes can be specified for its delivery. For each datagram a different route can be used. This approach contrasted to previously discussed ones is shown on Figure 3.

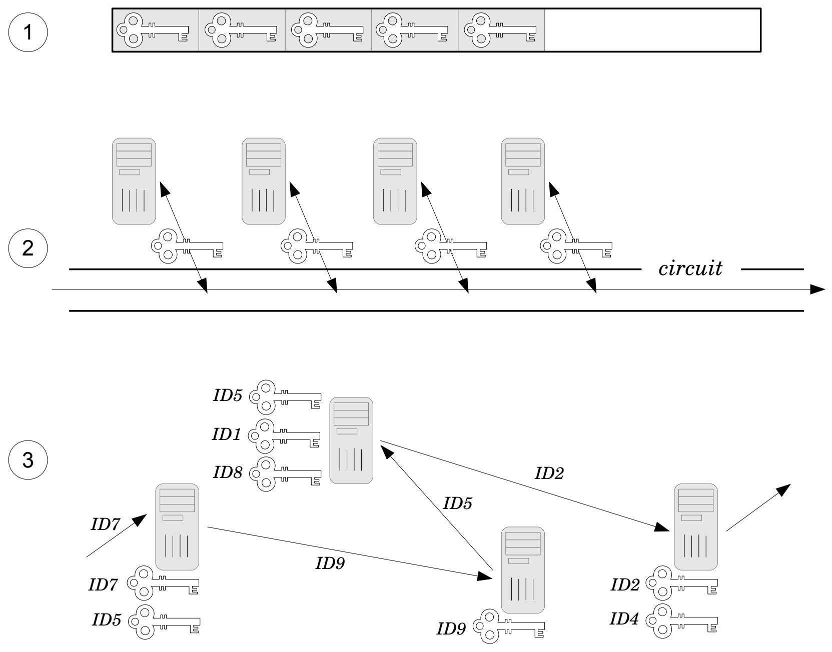

Figure 3. Three approaches to key selection.

Any special class of traffic for circuit management (except packets for Diffie-Hellman's symmetric key negotiation) is not distinguished as for circuit-based systems. Requests to release or reconfigure some of mixes that share keys with a client are undistinguishable from normal traffic.
The algorithm encrypts headers as a whole and adds pseudo-random alternation of the payload for tagging attack protection. Applying additional *rotation* procedure on mixes (described in detail in *Secret sequence life cycle*, Subsection 4.2.1) prevents replay attacks.
Note that IP datagram transmission is connectionless and unreliable by default. IP packets can be lost, duplicated, reordered etc. The algorithm doesn't correct such transmission errors nor provide any kind of extra reliability. Any datagram transmission problems must be handled by upper-layer protocols.
A few notes on the scope and detail level of presented algorithm should be mentioned here.
It is intended to be as universal as possible. No solution for mix management is specified. It's only assumed that each host can anytime obtain comprehensive up-to-date list of all mixes, their properties and capabilities. While TOR directory servers seem to perfectly provide this service, any alternative solution can be used.
No algorithm parameters (denoted with monospace capitals), details on symmetric ciphers nor implementation of common tasks (Diffie-Hellman's key exchange, sorting, generating a pseudo random stream, etc.) are forced.

4.2. Algorithm

4.2.1. Secret sequence

Each mix stores a set of entries: (id1, id2, key, stream, validity) called *secret sequences*.

- id1 is a 4-byte entry identifier selected by the mix.
- id2 is a 4-byte value obtained by encrypting the id1 with the key. id2 can be used as an entry identifier alternatively to id1.
- key is a symmetric key (for example: AES-256) used for onion encryption/decryption.
- stream is a pseudo-random string of the length PSIZE bytes. A mix that performs standard forwarding (see 4.2.4) XORs it with a packet payload for tagging attack prevention. RC4 stream cipher is to be considered for generating this value.
- validity is a parameter that specifies the POSIX time at which the sequence must be deleted. This value should be chosen by the mix based on its load, performance and expected uptime.

Notes:

- During packet preparation the sending host must be able to determine stream values of any intermediate mix. For this reason a stream should deterministically dependent on an id.
- Packets are encrypted by symmetric key cipher with a key. A stream is used only for introducing some distortion for tagging attack prevention, therefore it doesn't have to provide high level of security. OGR may possibly occur resistant to such attacks even without it thanks to its design (each packet can traverse a different path, packet processing by a mix, packet forwarding policy, secret sequence rotation etc), so this low-cost mechanism is added just in case and can be redefined or omitted in protocol specification.
- For stability reasons the time difference between UTC and a mix clock should be sent to clients during secret sequence negotiation or propagated with mix availability information or, alternatively, mixes should be obliged to periodically synchronize their clocks to one of predefined public NTP stratum 1 servers. The exact solution should be chosen at the implementation level. Moreover for some very short time period after the validity expires mixes are expected to accept and process 'delayed' packets.

Secret sequence life cycle

Clients may negotiate secret sequences with any mixes they wish to use. Most alternative solutions recommend a Diffie-Hellman's key exchange for such negotiation as it provides perfect forward secrecy (secrecy even if an attacker captures the traffic to a mix and further obtains its former private key). DH packets should be exchanged via OGR so that clients stay anonymous. The negotiation doesn't have to be reliable, that is if there is no response from the mix the client is free to resend the request or negotiate with any other mix.
There are no *keep alive* packets. The secret sequence is stored till the validity expires, afterwards it must be deleted. The validity period cannot be extended. A client can force a mix to delete a secret sequence before the time specified by the validity with a special datagram.
Upon reception of a special packet (an 'id2 packet' – see 4.2.4) a mix runs the *rotation* procedure – it modifies the secret sequence as follows[11]:

- id1 ← id2,
- id2 ← id2 encrypted with the key,
- stream ← a new pseudo-random string of PSIZE bytes.

4.2.2. *Path selection*

OGR uses the term *path* on cascades. A client (host A) can use any tuple of mixes it shares secret sequences with as a path. Any single mix can occur several times in a path. Each datagram may traverse a different path. For the recipient to respond to a datagram, paths are specified as circles (they start and end with the host A). One of intermediate mixes must be selected as a proxy: a host responsible for direct datagram delivery and reception of the reply.

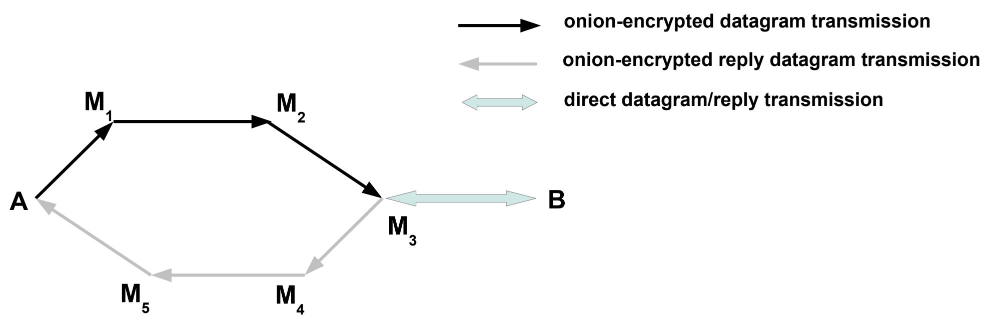
onion-encrypted datagram transmission
onion-encrypted reply datagram transmission
direct datagram/reply transmission
A
M1
M2
M3
M4
M5
B

Figure 4. Path: A sends an anonymous datagram to B and receives a reply datagram. M_3 acts as a proxy.

4.2.3. *Packet format*

OGR software intercepts datagrams to be anonymized at a client workstation and processes each of them as follows:

1. source IP and source port addresses are set to 0,
2. the datagram is padded to PSIZE bytes with random bytes,
3. the datagram is onion encrypted, preceded with an onion-encrypted header and an id (see beneath).

Such onion-prepared packets are transmitted between mixes over the UDP transport.
Each mix receives the packet in the form:

IP — UDP — [id — header — payload]

[11] *Should the need arise some* key *transformation may also be further introduced.*

4.2.4. Packet processing

Id

id (4 bytes) identifies (as an id1 or id2) the secret sequence to be used for packet processing[12]. If the id is incorrect (doesn't match any secret sequence) the packet should be dropped. With slight probability a random packet to a random mix may be further generated for error detection camouflage.
If a secret sequence is determined by the id2 (an 'id2 packet'), then the rotation operation on the secret sequence is performed, afterwards the processing is the same as for id1.

Header

The header is processed independently of the payload. Only the key is used.
A header contains up to MAX_ONIONS (proposed: 16) header onions of the size 8 bytes each. The processing at each mix is fairy simple:

- decrypt the entire header with the key,
- collect the first header onion data:
  - next_mix_IP_address (4B) – IP address of the next mix on the path or an action identifier,
  - next_mix_sequence_id (4B) – id to be sent to the next mix,
- generate a *junk* (8B) and append it at the end of the header to keep it fixed-size[13].

Processing of both the id field as well as a header is illustrated by Figure 5.

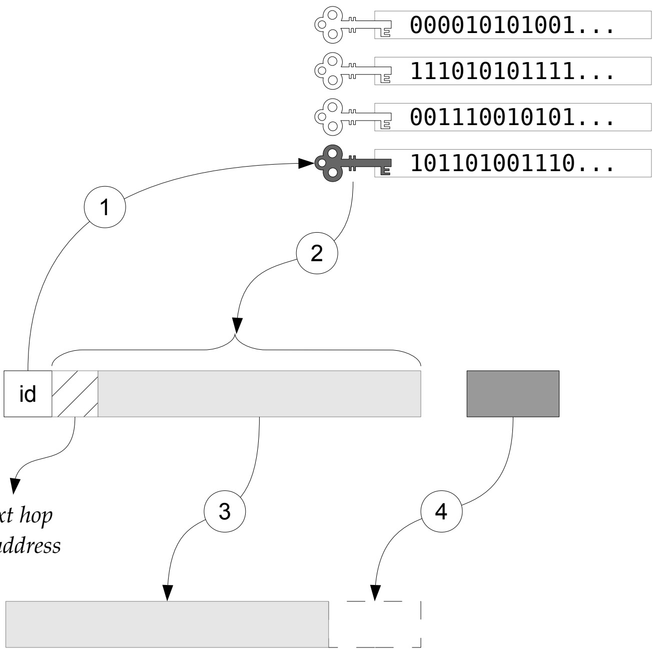

Figure 5. Id and header processing by a mix.

[12] *The problem may occur if* id2 *of one sequence is equal to* id1 *of some other one. It is believed extremely unlike to happen although some solution to this problem should be applied at the implementation level.*
[13] *For headers to be processed as a whole by next mixes junk blocks must be prepared very precisely. For computational purposes a sequence of junk blocks for subsequent mixes might be created by a sender and placed as an additional field of a packet. A mix copies one such block to the header and rotates the junk block sequence:* $j_i = j_{i+1}, j_N = j_1$.

A mix should drop any subsequent packets with repetitious headers after time T for reply attack prevention. If a client sends two packets through the same path, then the id2 instead of id1 number should be used for at least one mix.

Payload processing

Payload processing depends on the value of next_mix_IP_address:

- an address of a known mix – standard mix forwarding,
- 0.0.0.1 – direct datagram delivery (acting as a proxy),
- 0.0.0.2 – secret sequence deletion,
- 0.0.0.x – reserved for future use,
- other address – error notification.

Standard mix forwarding

1. decrypt the payload with the key (onion decryption),
2. XOR the payload with the stream (this step intended as a simple workaround for the tagging attack problem),
3. transfer [next_mix_sequence_id — processed_header — processed_payload] to the mix determined by next_mix_IP_address in a new UDP packet.

Direct datagram delivery

1. process the current header once again (as for a newly incoming packet) with the same secret sequence[14],
2. decrypt the payload with the key (onion decryption),
3. check the datagram stored in the payload: first four bytes of the payload should form a correct IP packet with both IP address and source port equal to 0. The length is specified in its header, so the padding to be dropped is clearly identifiable:
    - if the datagram is correct, then send it as SNAT routers do (RFC 3022). If any reply datagram is received in less than T seconds, then pad it to the size PSIZE with random bytes and process as a new, unprocessed payload (with the same secret sequence). After T seconds pad a no reply message to the size PSIZE with random bytes and process as a new, unprocessed payload (with the same secret sequence),
    - if the datagram is incorrect, then pad an error message to the size PSIZE with random bytes and process as a new, unprocessed payload (with the same secret sequence).

Secret sequence deletion

1. process the current header once again (as for a newly incoming packet) with the same secret sequence[14],
2. pad a delete confirmation message to the size PSIZE with random bytes and process as a new, unprocessed payload (with the same secret sequence),
3. delete the current secret sequence.

[14] *If the new* next_mix_IP_address *is not an address of a known mix, then stop processing and drop the packet. With slight probability a random packet to a random mix may be further generated for error detection camouflage.*

Error notification

1. process the current header once again (as for a newly incoming packet) with the same secret sequence[14],
2. pad an error message to the size PSIZE with random bytes and process as a new, unprocessed payload (with the same secret sequence).

5. Conclusion

In this paper a new approach for anonymous datagram transmission has been presented. While strongly inspired by related work, it introduces a completely new mechanism for key management and path specification. An emphasis is put in its simplicity and flexibility without compromising the level of security and anonymity. It combines advantages of numerous former solutions, both message-based and circuit-based.
The main strengths of the algorithm include:

- it is extremely lightweight, simple and flexible,
- any kind of datagram traffic can be carried,
- only symmetric cryptography is used for packet processing,
- packet headers are reduced and simplified,
- clients can send datagrams immediately without additional circuit preparation,
- mixes ready-to-use are not bound to client's IP addresses. It's important especially for clients on dial-up connections or with a few interfaces,
- there is no need of reconfiguration if one of ready-to-use mixes gets compromised, overloaded or turned off,
- clients can negotiate with any mix anytime (no 'telescope' hop-by-hop circuit building),
- replay and tagging attacks are prevented.

However there are some potential weaknesses and vulnerabilities that may require additional consideration. I discuss them below:

- Id's are transferred in open form. Fortunately, they carry a very limited information to an attacker and their usage is restricted in time thanks to rotation.
- Anonymous datagram transmission with OGR is unreliable. It's true but let's notice that the IP protocol itself provides no reliability either.
- If packets get lost there is no simple way of determining which mix doesn't operate correctly. However in the IP protocol there is also no embedded solution that allows quick determining where on the path to the destination packets are lost or get corrupted. A different path can be used for re-transmission.
- Only datagrams up to PSIZE bytes can be transferred. A simple workaround for this problem is packet fragmentation at client's host. Note that packet headers are relatively short, therefore PSIZE can be very close to maximal datagram length so this problem is relatively rare to occur.
- The algorithm can operate less efficient for large TCP streams than TOR. Presumably it's true but TOR is dedicated to TCP only while OGR can handle any kind of traffic.

The above comparison allows considering OGR a very promising general-purpose anonymizing solution.

References

[1] Bennett K., Grothoff C., GAP – practical anonymous networking, In: Proceedings of Privacy Enhancing Technologies workshop (PET 2003). Springer-Verlag, LNCS 2760, http://grothoff.org/christian/aff.pdf (last accessed: 01/06/2012)

[2] Chaum D., Untraceable electronic mail, return addresses, and digital pseudo-nyms, Commun. ACM, 4, February 1981, http://freehaven.net/anonbib/cache/chaum-mix.pdf (last accessed: 01/06/2012)

[3] Denezis G, Dingledine R., Mathewson N., Mixminion: Design of a Type III Anonymous Remailer Protocol, In: Proceedings of the 2003 IEEE Symposium on Security and Privacy, IEEE Computer Society, May 2003, http://www.mixminion.net/minion-design.pdf (last accessed: 01/06/2012)

[4] Dingledine R., Mathewson N., Syverson P., Tor: The Second-Generation Onion Router, In: Procedings of the 13th USENIX Security Symposium, August 2004, http://www.usenix.org/events/sec04/tech/dingledine.html (last accessed: 01/06/2012)

[5] Freedman M.J., Morris R., Tarzan: A Peer-to-Peer Anonymizing Network Layer, In: Proceedings of the ACM Conference on Computer and Communications Security (CCS 9), Washington, D.C. November 2002, http://pdos.csail.mit.edu/tarzan/docs/tarzan-ccs02.pdf (last accessed: 01/06/2012)

[6] Reiter M.K., Rubin A.D., Crowds: Anonymity for web transactions, ACM TISSEC, 1, June 1998, http://avirubin.com/crowds.pdf (last accessed: 01/06/2012)

[7] Rennhard M., Plattner B., Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection, In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)