Lightweight Practical Private One-Way Anonymous Messaging

Anirban Basu[1(✉)], Juan Camilo Corena[1], Jaideep Vaidya[2], Jon Crowcroft[3], Shinsaku Kiyomoto[1], Stephen Marsh[4], Yung Shin Van Der Sype[5], and Toru Nakamura[1]

[1] KDDI R&D Laboratories, Fujimino, Japan
{basu,corena,kiyomoto,tr-nakamura}@kddilabs.jp
[2] Rutgers, The State University of New Jersey, New Brunswick, USA
jsvaidya@rutgers.edu
[3] University of Cambridge, Cambridge, UK
jon.crowcroft@cl.cam.ac.uk
[4] University of Ontario Institute of Technology, Oshawa, Canada
stephen.marsh@uoit.ca
[5] KU Leuven, Leuven, Belgium
yungshin.vandersype@law.kuleuven.be

Abstract. Opinions from people, evident in surveys and microblogging, for instance, may have bias or low user participation due to legitimate concerns about privacy and anonymity. To provide sender (the participant) anonymity, the identity of the message sender must be hidden from the message recipient (the opinion collector) and the contents of the actual message hidden from any intermediate actors (such as, routers) that may be responsible for relaying the message. We propose a novel one-way message routing scheme based on probabilistic forwarding that guarantees message privacy and sender anonymity through cryptographic means; utilising an additively homomorphic public-key cryptosystem along with a symmetric cipher. Our scheme involves intermediate relays and can work with either a centralised or a decentralised registry that helps with connecting the relays to each other. In addition to theoretical analysis, we demonstrate a real-world prototype built with HTML5 technologies and deployed on a public cloud environment. The prototype allows anonymous messaging over HTTP(S), and has been run inside HTML5 browsers on mobile application environments with no configurations at the network level. While we leave constructing the reverse path as future work, the proposal contained in this paper complete and has practical applications in anonymous surveys and microblogging.

Keywords: Privacy · Anonymity · Message · Routing · Http · Cloud

J. C. Corena—The contributions of Juan Camilo Corena to this paper represent his work during his time at KDDI R&D; and are not related in any way to his work with his current employer – Google Inc.

© IFIP International Federation for Information Processing 2015
C.D. Jensen et al. (Eds.): IFIPTM 2015, IFIP AICT 454, pp. 76–91, 2015.
DOI: 10.1007/978-3-319-18491-3_6

1 Introduction

The art of providing online anonymity has been popular with the use of proxies to avoid revealing the actual message sender to the recipient. Through the principles of Chaum mixes [1] (and other mix networks based on that), the identities of senders are delinked from the actual messages through a mixing process. Secret sharing through Dining Cryptographers networks [2] also provides sender and recipient anonymity. Onion routing [3] ensures that encrypted messages are relayed through a pre-determined path of intermediate hops such that the actual senders cannot be identified. Probabilistic forwarding (e.g., Crowds [4]) also delinks messages from their actual senders without a preset forwarding path.

The Pfitzmann and Hansen terminology [5] defines three types of anonymity in terms of unlinkability as: (a) sender anonymity, (b) recipient anonymity and (c) relationship anonymity. Sender anonymity means that a particular message is not linkable to a sender, and a particular sender cannot be linked to any message. Likewise, recipient anonymity means that a recipient and one or more messages are unlinkable. Relationship anonymity, a weaker construct, underscores that it is impossible to trace who communicates with whom. This is implied by either sender anonymity or recipient anonymity or both. The authors also define unobservability in terms of undetectability as: (a) sender unobservability, (b) recipient unobservability and (c) relationship unobservability. Unobservability means that it is impossible to detect who sends (sender unobservability) or who recieves (recipient unobservability) or that a certain set of senders and recipients are at all communicating (relationship unobservability). Unobservability is expressed in terms of undetectability, which is established through indistinguishability of messages. This means that the messages sent out or received are indiscernible from random noise. The Pfitzmann and Hansen terminology states that unobservability implies anonymity. Furthermore, similar to relationship anonymity, sender and recipient unobservability both imply relationship unobservability.

In this paper, we propose a novel one-way anonymous message routing protocol, which guarantees the privacy of the messages and the anonymity of the sender through fundamentals of public key cryptography with homomorphic properties. Our proposed scheme is usable in scenarios such as anonymous microblogging and anonymous surveys. We present a practical HTML5 cloud-deployed prototype, which piggybacks messages over HTTP(S), thus requiring no configurations at the network level and making it easy to run on mobile devices. We provide a theoretical analysis of the anonymity guarantees in our protocol as well as performance results of our prototype. The proposed protocol is pertinent to systems where the sender's identity and the message itself require protection, not only from external attackers but also from the recipient.

From a legal perspective, the one-way anonymous routing system will fall under the substantive protection of the national applicable law. As the system may typically route personal data globally, the question arises whether the strict EU legal framework might hinder the practical usability of the system for initially non-EU related processing operations. Article 25 of Directive 95/46/EC provides that the transfer of personal data outside the EU is limited to third countries

with an adequate level of protection. Thus, if the routing is considered as a data transfer, EU data protection law could potentially bring an additional set of complex data protection requirements to ensure this adequate level of protection. Yet, it is pointed out that the transit or simple routing of data is not the same as the transfer of the data. Consequently the mere routing of data through the one-way anonymous routing system does not establish a data transfer under EU law[1]. Nevertheless, it is still possible that EU law is applicable in cases when, e.g., the personal data of EU citizens is processed, the controller is established in a EU Member States.

2 Related Work

Research in the last three decades has made considerable advancements in facilitating private and anonymous communication over the Internet. Anonymous communication systems can be broadly classified into two categories. (1) *High-latency systems* provide strong anonymity but are only suitable for applications that are tolerant of the rather long delays in end-to-end message delivery. (2) *Low-latency systems* offer better performance and can be used with interactive bi-directional communication, such as Web browsing. Analogous to the nature of packet transport over UDP and TCP, high-latency and low-latency systems are sometimes referred to as *message-based* and *connection-based* systems respectively. Apart from these two systems, anonymous communication have been modelled as multiparty computation such as the Dining Cryptographer Networks [2] and secret sharing schemes [6].

High-latency message-based systems: David Chaum introduced the idea of a *mix* [1] – a building block in several high-latency anonymous communication systems. The mix is characterised by the grouping of several encrypted input messages into a batch and then forwarding, after decryption, some or all messages in the batch. This necessitates that the public key of the mixing node be known by the senders. The fact that the mix node decrypts messages does not jeopardise privacy or secrecy of messages because each message could be encrypted further with the public key of its intended recipient. The original Chaum mix outputted messages in lexicographical order while more recent schemes output messages in random order [7,8]. Mix networks address the problem of a single point-of-failure of the original Chaum mix by making messages pass through an ordered sequence of mixes that constitute a path. The paths are typically either *free routes* or *mix cascades*.

Low-latency connection-based systems: Work on low-latency anonymisation dates back to traffic redirection through a centralised and trusted proxy, which hides the IP address of the actual source host from the destination. This concept is used

[1] UK Information Commissioner Office, "The eight data protection principle and international data transfers", 2010, http://ico.org.uk/for_organisations/guidance_index/~/media/documents/library/Data_Protection/Detailed_specialist_guides/international_transfers_legal_guidance_v4_290410.ashx, 1.3.2.

by the commercial Anonymizer service. Unlike mixes, a proxy forwards all incoming traffic immediately without any packet reordering. The proxy is trusted not to reveal information about original senders. Onion routing [3,9,10] utilises traffic redirection through a static set of pre-determined onion routers, each of which maintains cryptographic keys of one upstream and one downstream router. A message sent through onion routing nodes has its layers of encryption are peeled off until it is sent off to the final recipient. To cater for performance, public key cryptography is only used to establish an encrypted circuit, while faster symmetric encryption is used to transfer the actual data. Churn in onion routing nodes limits the scalability of the protocol. Onion routing has been shown [11] to be detrimental to sender or recipient anonymity if either the first or the last router is compromised. Tor [12] represents the current state-of-the-art in the evolution of onion routing. Crowds [4] and AP3 [13] make use of probabilistic forwarding through a randomly established path while the response from the recipient is relayed back to the sender using the same already established path. Denoting the total number of relay nodes by N, the number of collaborating malicious nodes by C and the forwarding probability at each node as p_f, the Crowds protocol defined a property called *probable innocence*, which means that the first collaborating malicious node's predecessor is no more likely to be the true sender of that message than not if $N \geq \frac{p_f(C+1)}{p_f - 1/2}$. Tarzan [14] sends messages through a dynamically created encrypted tunnel through multiple hops in a peer-to-peer network, adding overhead costs of creating the tunnel. Similar to Tarzan, MorphMix [15] is a peer-to-peer low-latency anonymity system where any node in the network relay traffic for any other node.

3 Anonymous Message Routing

The crux of protecting the identity of any node from being traced as the sender is to ensure that an egress encrypted message is always different from an ingress encrypted message and yet both are of the same size, see Fig. 1. This implies that genuinely different messages are indistinguishable from those that look different only in the encrypted domain. Neither the sender nor any intermediate node should be able to decrypt the messages in any identifiable form. Thus, even the sender Alice, herself, is unable to identify her own message if it is forwarded back to her because there is no characteristic of the message (e.g., message signature) that remains unchanged in the multi-hop forwarding process. This *unlinkability* provides *deniability* to the sender: if Alice is to deny sending a message then there is no way to prove otherwise.

Throughout the remainder of this paper, it is assumed that all messages sent out in the network are of the same size. Messages of varying sizes should be split, padded and sequenced if necessary for re-construction. It is also assumed that nodes participating in the message relay network intermittently send out random messages containing just noise of the same size as the actual messages. The recipient will be able to filter the noise from the actual messages upon decrypting. We now briefly introduce the notion of *homomorphic encryption* before describing how we use it to devise unlinkability.

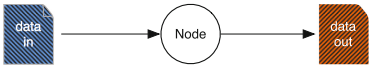


Fig. 1. The incoming and the outgoing encrypted messages to and from a node look completely different, offering no linkability.

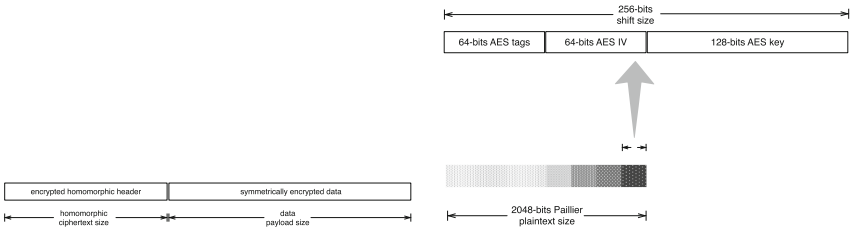
Homomorphic encryption – a brief primer: Homomorphic encryption allows computing over encrypted data without requiring the knowledge of either the actual data or any results produced through the computation. Depending on the type of computational operations supported, homomorphic cryptosystems are classified as: (1) additive, (2) multiplicative, (3) somewhat homomorphic (e.g., allowing a number of additions and one multiplication), and (4) fully homomorphic. Denoting the encryption of a plaintext message m as $\mathcal{E}(m)$, in the generalised case of fully-homomorphic encryption, for any function f in the plaintext domain there exists a function g in the ciphertext domain, such that $\mathcal{E}(f(m_1, \ldots, m_n)) \equiv g(\mathcal{E}(m_1), \ldots, \mathcal{E}(m_n))$. Fully homomorphic schemes [16] are mathematically sound in terms of security but their computational requirements often make them unfeasible for practical applications. The Paillier public-key cryptosystem [17], and its variant, the Damgård-Jurik cryptosystem [18], have practical implementations and both exhibit only additively homomorphic properties: the encryption of the sum of two plaintext messages m_1 and m_2 is the modular product of their individual ciphertexts, i.e., $\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \cdot \mathcal{E}(m_2)$); and the encryption of the product of one plaintext message m_1 and another plaintext multiplicand π is the modular exponentiation of the ciphertext of m_1 with π as the exponent, i.e., $\mathcal{E}(m_1 \cdot \pi) = \mathcal{E}(m_1)^{\pi}$.

3.1 Unlinkable Message Forwarding

Any relay node n_i, connected to a centralised or distributed node inter-connectivity registry, can forward messages on behalf of another node. We describe how a relay node can perform unlinkable message forwarding of short messages as well as arbitrary length messages.

Note that *short* messages are those that fit within the plaintext key space of the homomorphic cryptosystem that is used while messages that do not fit are termed as *long* messages. It is possible to break long messages into multiple short messages, each fitting into the plaintext key space of the homomorphic cryptosystem. This will necessitate that multiple homomorphic operations be run over every part of the long message. To reduce the number of computationally expensive homomorphic operations, symmetric encryption can be used to encrypt longer messages.

The length of the messages plays a vital role in their indistinguishability. The implementation of our proposed protocol should use the message length for all messages, breaking down the large messages if necessary and allowing for re-construction at the recipient's end. If the implementation chooses a message length that fits within the plaintext space of the homomorphic cryptosystem then the message forwarding protocol does not need to use symmetric encryption, thus avoiding the limitation on the number of forwarding hops that we describe below.



(a) The encrypted header and actual data part of a message that is forwarded on by nodes.

(b) Inside the 2048-bits Paillier encrypted header associated with a 128-bit AES encrypted message.

Fig. 2. Homomorphic headers and parameters of the symmetric cipher

Unlinkability in Short Messages Through Homomorphic Encryption: Given a plaintext message m to be sent to a recipient with an additively homomorphic cryptosystem whose encryption function is denoted by $\mathcal{E}()$, we have an identity equation $\mathcal{E}(m)_{n_{i+1}} = \mathcal{E}(m)_{n_i} \cdot \mathcal{E}(0)$. Thus, any ingress ciphertext $\mathcal{E}(m)_{n_i}$ is transformed by the node n_i into a completely different egress ciphertext $\mathcal{E}(m)_{n_{i+1}}$ while the underlying contents of the message m remains unchanged. The final recipient can decrypt the message m from the ciphertext sent by a relay node because it possesses the equivalent private key for the public key used in the homomorphic encryption. The main limitation of this approach is that m must be contained within the plaintext domain of the additively homomorphic cryptosystem. Thus, with a 2048-bits Paillier cryptosystem, the maximum size of the message is 256 bytes. Even if it is possible to encrypt an arbitrary length message by breaking it up into small blocks (where each block fits into the plaintext domain of the cryptosystem), the encryption function of an additively homomorphic cipher is relatively slow in comparison with an equivalent strength symmetric cipher.

Unlinkability in Long Messages with Symmetric and Homomorphic Encryption: Arbitrary length messages can be encrypted by any symmetric block cipher or stream cipher. The general idea is to (re-)encrypt the ciphertext at each relay node, so that the ingress message and the egress message look different but are of the same size. Since the ingress message at any forwarding node is already encrypted, the node simply needs to encrypt it again with a random symmetric cipher key. This key and any other relevant parameters are then added to a homomorphically encrypted header, where the size of the header also remains the same. The generalised representation of the message and its headers is shown in Fig. 2(a). The size of the header, but not the actual message, is limited by the plaintext size of the homomorphic cryptosystem.

Let the size of the symmetric key be denoted as $|k|$ and that of the other relevant parameters (e.g., initialisation vector in a block cipher) as $|p|$ while $k||p$ represents their bitwise concatenation, which is of size $|k| + |p|$. All relay nodes use same length keys and parameters. Assume that the encrypted header

Algorithm 1. Forwarding algorithm with homomorphic obfuscation and symmetric message encryption at relay node n_i.

Require: Additively homomorphic encryption function, $\mathcal{E}$, for the final message recipient R.
Require: Symmetric encryption function $\mathbb{E}$.
Require: Ingress encrypted message $\epsilon(m)_{n_i}$ from node n_{i-1}.
if $|\epsilon(m)_{n_i}| \leq$ the the ciphertext space of $\mathcal{E}$ **then**
  $\mathcal{E}(m)_{n_i} \leftarrow \epsilon(m)$.
  Compute ciphertext $\mathcal{E}(m)_{n_{i+1}} \leftarrow \mathcal{E}(m)_{n_i} \cdot \mathcal{E}(0)$.
  Egress message $\epsilon(m)_{n_{i+1}} \leftarrow \mathcal{E}(m)_{n_{i+1}}$.
else
  Split $\epsilon(m)_{n_i}$ into header $\mathcal{E}(h_{n_i})$ and symmetric ciphertext c_{n_i}.
  Generate random symmetric encryption key k_{n_i} and parameters p_{n_i}.
  Compute header $\mathcal{E}(h_{n_{i+1}}) \leftarrow \mathcal{E}(h_{n_i})^{2^{|k_{n_i}|+|p_{n_i}|}} \cdot \mathcal{E}(k_{n_i}||p_{n_i})$.
  Compute ciphertext $c_{n_{i+1}} \leftarrow \mathbb{E}(c_{n_i})$ with the key and parameters k_{n_i} and p_{n_i} respectively such that $|c_{n_{i+1}}| = |c_{n_i}|$.
  Egress message $\epsilon(m)_{n_{i+1}} \leftarrow \mathcal{E}(h_{n_{i+1}})||c_{n_{i+1}}$.
end if
With probability $p_{f_{th}}$, send $\epsilon(m)_{n_{i+1}}$ to node $n_{i+1} \in L_n$ through S.
With probability $1 - p_{f_{th}}$, send $\epsilon(m)_{n_{i+1}}$ to final message recipient R.

at the i-th relay is $\mathcal{E}(h_{n_i})$ where initially, $h_{n_1} = k_{n_1}||p_{n_1}$ (here, n_1 is the actual sender). Each relay node n_i adds the symmetric cipher information as $\mathcal{E}(h_{n_{i+1}}) = \mathcal{E}(h_{n_i})^{2^{|k|+|p|}} \cdot \mathcal{E}(k_{n_i}||p_{n_i})$. The operation in the encrypted domain is equivalent, in the plaintext domain, to a left shift of h_{n_i} by $|k| + |p|$ bits and a placement of $k_{n_i}||p_{n_i}$ in the rightmost $|k| + |p|$ bits of h_{n_i} to produce $h_{n_{i+1}}$. Since the additively homomorphic encryption guarantees semantic security, $\mathcal{E}(h_{n_{i+1}})$ and $\mathcal{E}(h_{n_i})$ are indistinguishable. Similarly, the symmetrically encrypted egress and ingress ciphertexts at any relay $c_{n_{i+1}}$ and c_{n_i} are also indistinguishable. The egress encrypted header of $\mathcal{E}(h_{n_{i+1}})$ and the egress ciphertext $c_{n_{i+1}}$ at any relay node n_i together achieve the desired unlinkability. The recipient, possessing the equivalent private key for the public key used in the homomorphic encryption, can decrypt the encrypted header, right shift by the size of $|k| + |p|$ and recover the actual message m in rounds by recovering the symmetric cipher key and other parameters for each round.

A limitation of this approach compared to the one with purely homomorphic encryption is that the forwarding operation can be continued so long as the size of $h_{n_{i+1}}$ is less than the plaintext space of the additively homomorphic cipher. However, it is not possible for any node n_i to know, in the ciphertext domain, if h_{n_i} or $h_{n_{i+1}}$ are within the maximum size of the plaintext domain of the homomorphic encryption. Once the size is exceeded, the left shift operation will lose information about symmetric keys making it impossible for the recipient to decrypt the message. The number of hops that a message goes through can be controlled by the forwarding probability. Figure 2(b) shows how the encrypted header looks like for a 2048-bit Paillier homomorphic cryptosystem and with an

Algorithm 2. Algorithm for dispatching a message at the sender node n_1.

Require: Additively homomorphic encryption function, $\mathcal{E}$, for the final message recipient R.
Require: Symmetric encryption function $\mathbb{E}$.
Require: Plaintext message m.
1: **if** $|m| \leq$ the plaintext space of $\mathcal{E}$ **then**
2: Compute ciphertext $\mathcal{E}(m)_{n_1}$.
3: Egress message $\epsilon(m)_{n_1} \leftarrow \mathcal{E}(m)_{n_1}$.
4: **else**
5: Generate random symmetric encryption key k_{n_i} and parameters p_{n_i}.
6: Compute header $\mathcal{E}(h_{n_1}) \leftarrow \mathcal{E}(k_{n_1}||p_{n_1})$.
7: Compute ciphertext c_{n_1} from the symmetric encryption with the key and parameters k_{n_1} and p_{n_1}, i.e., $c_{n_1} \leftarrow \mathbb{E}(m)$.
8: Egress message $\epsilon(m)_{n_1} \leftarrow \mathcal{E}(h_{n_1})||c_{n_1}$.
9: **end if**
10: Obtain, from the intermediary relay service S, a list L_n of nodes that the message can be forwarded to.
11: Send $\epsilon(m)_{n_1}$ to random node $n_i \in L_n$ through S.

instance of a 128-bit AES cipher at each hop. The tags and the initialisation vector (IV) of the AES cipher are both set to 64-bits, which are valid parameters for the CCM or the Galois/counter mode implemented by the Stanford Javascript Crypto Library.

Algorithm 1 describes the forwarding protocol at a relay node n_i. Step 3 for short messages and Steps 8 and 9 for long messages achieve the desired unlinkability of the ingress and egress messages.

3.2 Unlinkable Message Sending

The actual sender always sends the message in encrypted form to a random relay node. Since the messages sent out are of the same size, and some of the messages sent out are just random noise, it is impossible to tell by looking at an egress message that the node sending out that message is the actual sender, or simply a relay node forwarding a message from someone else. The protocol for sending out a message is similar to that for forwarding messages in terms of the cryptographic operations. Algorithm 2 describes the protocol that takes place on the sender node.

4 Adversarial Analysis

Traffic analysis [19] is regarded as the de-facto yardstick to determine resilience of anonymous communications systems against attackers. Ignoring the contents of the message, traffic analysis aims to derive information regarding senders and recipients from the traffic metadata, such as packet arrival times and message lengths. Below we discuss several of the key attacks along with the resilience of our system to these attacks.

One of the key attacks in traffic analysis is the *timing attack*. In this attack, a passive global adversary who can observe the ingress and egress connections from the anonymous relay network may be able to link the inputs and outputs based on their patterns of packet inter-arrival times. Although there are delays in message delivery in our protocol, such delays are not as large or consistent as those in high-latency systems. In our system, the order in which messages are sent can get disrupted due to the forwarding probability, thus contributing to a different order at the arrival point. Nodes can also inject noisy traffic with messages of the same length as the original messages to obfuscate the message arrival pattern at the recipient's end.

Another important attack is the *replay or tagging attack* [20] in which the adversary controls the entry and the exit points of the anonymiser network and is able to tag a message at the entry point only to be able to detect it at the exit point. The attack is able to break the Tor network[2] because of the presence of pre-established paths. In our model, the node at which the message enters or exits the anonymiser network is not fixed. Therefore our system is more robust to this attack, since (when a large number of nodes exist in the network), the attacker will have to control a large proportion of the nodes in order to ensure that most messages will enter and exit the network through a node under the control of the adversary.

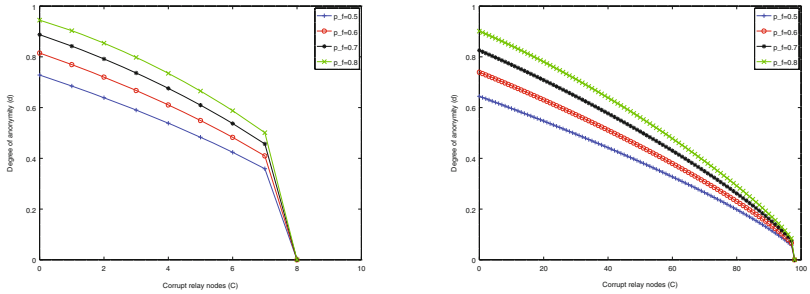
In the *predecessor attack*, if the paths for message routing are observed over time, the attacker can guess with reasonable accuracy the true senders of the messages. In our protocol, the paths are not constant due to the probablistic routing. Furthermore, depending of the frequency and origins of the noisy messages, the attacker will find it hard to differentiate the senders of actual messages from senders of noise, given that the senders of actual messages may also send out noise, and vice-versa. However, our scheme is not secure against predecessor attacks by design. Given a large number of collaborating Sybil [21] nodes under control of a single attacker, the anonymity of the sender will fall back to k-anonymity whereby the attacker can be confident that the sender is one out-of-the k remaining non-malicious nodes. This can be dangerous if k is small.

In the *fingerprinting attack*, the attacker utilises the variation of the size and the rate in which the messages are sent. In the disclosure attack [22] or the refined statistical disclosure attack [23], the attacker uses set intersections over time to determine the recipient in an anonymous communication. Through our protocol, adding noisy messages in the network at certain unpredictable rates can reduce the effectiveness of such attacks. In addition, our system is not concerned with recipient anonymity.

4.1 Analysis of Anonymity

We now present a theoretical analysis of anonymity. Diaz et al. [24] formulated a measure of anonymity in terms of Shannon entropy. Our anonymous message routing protocol resembles a Crowds-style probabilistic forwarding. With the probability of any node forwarding a message set to p_f (thus, the probability of

[2] See: https://blog.torproject.org/blog/one-cell-enough.



(a) The total number of nodes $N = 10$. (b) The total number of nodes N =100.

Fig. 3. The degree of anonymity d [24] with respect to the number of corrupt nodes C.

it sending to the final recipient as $1 - p_f$) and the number of forwarding nodes under the control of the attacker set to C (e.g., Sybil nodes), the entropy of the system, $H(X)$, after the attack has happened is formulated as shown as

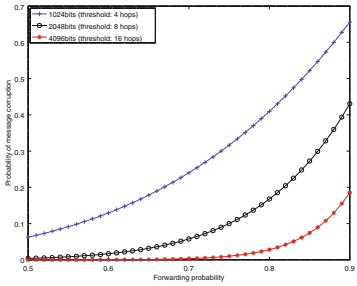
$$H(X) = \frac{N - p_f(N - C - 1)}{N} log_2 \left[\frac{N}{N - p_f(N - C - 1)}\right] + p_f \frac{N - C - 1}{N} log_2 \left[\frac{N}{p_f}\right]$$

The maximum entropy is given as $H_M = log_2(N - C)$. The *degree of anonymity* d of the relay network is given by $d = \frac{H(X)}{H_M}$. The condition $d = 0$ applies to the special cases: (a) where there is only one user, i.e., $N = 1$; and (b) when $C = N - 1$. In all other cases, it is evident that $0 \leq d \leq 1$ Diaz et al. [24]. Setting $N = 10$ and $N = 100$, Fig. 3 illustrates the degree of anonymity, d, with respect to the number of corrupt nodes, C, under the control of the attacker. The higher the forwarding probability p_f, the better the degree of anonymity.

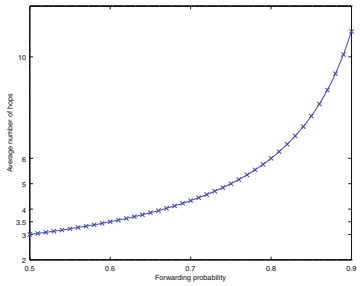
However, we have already seen that increasing the forwarding probability is not ideal for messages sent using symmetric encryption because: (a) a high forwarding probability implies a lower chance of the message reaching the final recipient; and (b) with symmetric encryption, a message passing through more than a certain number of hops is rendered undecryptable when it reaches the final recipient.

Denoting the message forwarding probability by p_f, the probability of the message reaching the k-th hop (instead of reaching the final recipient) is p_f^{k-1}. This is because the sender will never send the message directly to the recipient, so the probability of the message passing through the first hop is 1, followed by the forwarding probability at each hop. Assume that the maximum allowed value of k is some threshold k_{th} for some Paillier cryptosystem. Thus, any $k > k_{th}$ will corrupt a message. Therefore, the probability that a message is corrupted is synonymous with the probability of the message passing through the $(k_{th}+1)$-th hop; and is given as $p_{mc} = p_f^{(k_{th}+1)-1} = p_f^{k_{th}}$.

Similarly, the average number of hops that a message passes through is given as $\bar{n} = 1 + \frac{1}{1-p_f}$ and is illustrated in Fig. 4(b). The extra unity in the equation



(a) The probability of message corruption versus the forwarding probability. Three Paillier plaintext sizes are shown: 1024, 2048 and 4096 bits. It is assumed that 128-bits AES ciphers are used with 64-bits initialisation vectors and 64-bits tags.



(b) The average number of hops a message passes through versus the forwarding probability.

Fig. 4. Hops and message corruption

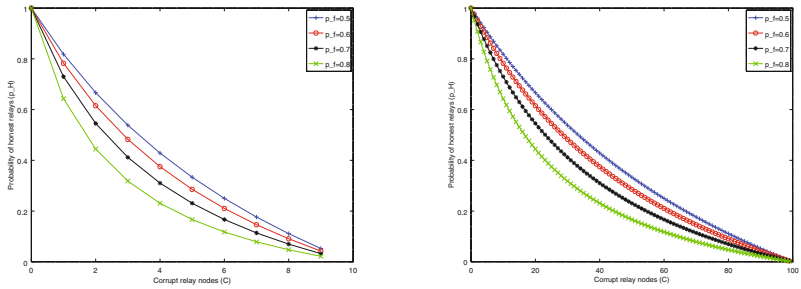
exists because the message will pass through at least one hop, i.e., the sender will never send it directly to the recipient.

We have seen that with a 128-bits AES password, 64-bit tags and 64-bit initialisation vector, every message requires, at the time of forwarding, 256-bits of additional space in the 2048-bits Paillier plaintext space. Thus, with a 2048-bits Paillier cryptosystem, the message can go through a total of 8 hops (including the original sender) while the total number of hops is 4 and 16 for a 1024-bits Paillier cryptosystem and a 4096-bits Paillier cryptosystem respectively. Thus, $k_{th} = 4$ (1024-bits), $k_{th} = 8$ (2048-bits) and $k_{th} = 16$ (4096-bits). For the message to be corrupt, it should pass through at least a $k_{th} + 1$ relay. Figure 4(a) shows the probability of message corruption versus the forwarding probability considering 128-bits AES password, 64-bits tags and 64-bits initialisation vectors.

Diaz et al. [24] also proposed the degree of anonymity from the perspective of the sender as the probability p_H of a message passing only through honest nodes, given as $p_H = 1 - \frac{C}{N - p_f(N-C)}$. For $N = 10$ and $N = 100$, Fig. 5 illustrates the probability that a message passes through honest nodes, p_H, with respect to the number of corrupt nodes, C, under the control of the attacker. The higher the forwarding probability p_f, the worse the value of p_H.

5 A HTML5 and Google App Engine Implementation

To illustrate the practical viability of our proposal, we have implemented it as two mobile web applications and deployed on the Google App Engine for Java. One application (`WSA-CRouter`) is responsible for forwarding the messages while the other (`WSA-MB`) is a generic message recipient. The front-end of the former,



(a) The total number of nodes $N = 10$. (b) The total number of nodes $N = 100$.

Fig. 5. The probability of a message passing through honest nodes p_H ([24]) with respect to the number of corrupt nodes C.

which is where the anonymisation algorithms run, is implemented in HTML5 technologies: Javascript and JQuery mobile. The backends of both applications are based on simple Java servlets and the no-SQL key-value datastore of the App Engine. The role of the cloud is that of an untrusted registry providing interconnectivity between the relay nodes. We have tested our application on different HTML5 compliant browsers on various devices: multiple desktop computers, tablet computers and mobile phones. We have used two Javascript libraries from Stanford University: the JSBN (and JSBN2) and the SJCL for supporting cryptographic operations.

Due to a bug in SJCL's conversion of bit arrays to strings (base64 or hex encoded), we had to pad, on the left, the concatenation of AES parameters with 4-bits set to 1 each. Thus, with a 128-bit AES as shown in the example in Fig. 2(b), we actually had a 260-bits shift size instead of the 256-bits as illustrated. This constrains our maximum hop limit to 7 instead of 8 as one would expect from a 2048-bit plaintext space of Paillier.

5.1 Anonymous Messaging over HTTP(S)

The two applications `WSA-CRouter` (the message forwarding registry and front-end)[3] and `WSA-MB` (the public message recipient)[4] are both hosted on the Google App Engine for Java. The front-end of `WSA-CRouter` is run on each node in the diagram. For simplicity, the private keys corresponding to various public keys used in the messaging are stored by the `WSA-MB` application on the App Engine. In reality, this should not happen because a cloud insider can get hold of these keys and decrypt messages while these are being forwarded.

The application `WSA-CRouter` is a centralised registry service that routes messages between the different relay nodes. In essence, this type of registry can be decentralised over a structured P2P networks, such as Chord [25] or Pastry [26].

[3] See: https://wsa-crouter.appspot.com.
[4] See: https://wsa-mb.appspot.com.

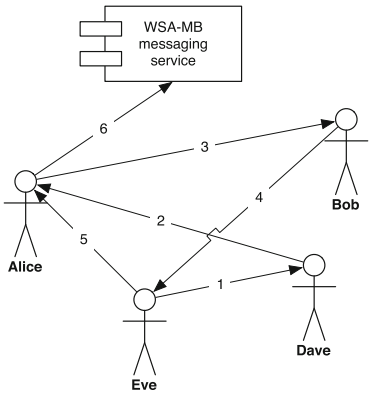


Fig. 6. The flow of a message from Eve to the final recipient.

This is especially useful if the connections to a centralised registry are blocked by a network admininstrator. In our prototype, `WSA-CRouter` implements the Google Channel API service to notify connected relay nodes of messages bound for them. In terms of functionality, the Channel API is close to the proposed Web Sockets standard (RFC 6455). The backend of `WSA-CRouter` running the cloud receives messages sent to it by a node; looks up the next hop address on that message and sends it on to the next hop if it is connected. The Algorithms 1 and 2 run entirely on the relay node client-side. The front-end of `WSA-CRouter` receives a message from another node sent through the cloud. Either it randomly forwards the message on to another connected and active relay node; or it sends the message to the final recipient. Figure 6 shows how a message, originally sent by Eve, passes through Dave, Bob and Alice to the recipient `WSA-MB` service. To the recipient, the message appears to have originated from Alice, when in reality it is from Eve.

If the message has been encrypted with a purely homomorphic obfuscation then it is impossible for the recipient to tell who the actual sender of the message is. However, if the message is encrypted with homomorphic and symmetric obfuscation and the message has not been corrupted because of passing through many hops then the recipient, upon decrypting, can tell that the number of hops the message has passed through. This will reveal that in the most likely case, the message did not originate at the node from which the final recipient received the message.

The backend of the `WSA-MB` service, upon receiving the encrypted message, looks up the public key sent with it and finds the corresponding private key. Depending on the type of obfuscation used, it decrypts either the message or its homomorphically encrypted headers and stores it in the datastore. A client Javascript application – the front-end of `WSA-MB` – can then retrieve the decrypted messages or headers, decrypting in rounds, the headers if required to retrieve the underlying plaintext messages.

5.2 Performance Evaluation

We describe the performance of our prototype in terms of the time taken for message forwarding on the client-side front-end and that of various functions on the cloud-side back-end.

Client-Side Performance: Table 1 demonstrates the performance of message forwarding on different browsers and platforms. In our set of test devices, we had:

(a) (mobile) Chrome 35.0.1916.38/iOS 7.1.1 running on a iPhone 5S;
(b) (desktop) IE 10.0.9200.16899S/Windows 8.0 running on a hardware consisting of a 3 GHz Intel Core i7 processor and 16 GB RAM;
(c) (desktop) Firefox 29.0/Ubuntu Linux 14.10 running on a hardware consisting of a 3 GHz Intel Core i7 Extreme processor and 16 GB RAM;
(d) (laptop) Chrome 35.0.1916.114/Mac OS X 10.9 running on a Macbook Air hardware consisting of a 1.8 GHz Intel Core i7 processor and 8 GB RAM.

Table 1. The mean forwarding time for different platforms using the same Javascript implementations of 2048-bits Paillier and 128-bits AES encryptions

Platform	Mean forwarding time
(mobile) Chrome/iOS	70.9s
(desktop) IE/Windows	17.2s
(laptop) Chrome/OS X	4.06s
(desktop) Firefox/Linux	1.89s

Throughout the experiment, we have used 2048-bit Paillier keys and 128-bit AES keys with 64-bit initialisation vectors and 64-bit tags. The connections to the `WSA-CRouter` web application have been also encrypted over HTTPS. The iPhone 5 S has the worst forwarding performance and this can be attributed to its less powerful hardware as well as the Javascript library on Chrome for iOS 7 compared to the computers. It is to be noted that Internet Explorer performs fairly badly too on reasonably high-end hardware. Use-cases such as anonymous surveys are tolerant to the relatively long delay (due to the over-a-minute delay per hop) if all the hops en-route are similar low-powered devices. However, microblogging such as a commentary of live events may not be able to tolerate large delays.

Cloud-Side Performance: Both `WSA-CRouter` and `WSA-MB` applications have been deployed on the F2 instance class (1200MHz CPU, 256MB RAM) of the Google App Engine for Java. The message forwarding delay in the Table 2(a) is negligible compared to the delays on the client-side (Table 1).

Table 2. Performances of message forwarding and cryptographic functions

(a) The performance of the `WSA-CRouter` implementation on the Google App Engine.

Function	Performance
Message forwarding	25ms

(b) The performance of the `WSA-MB` implementation on the Google App Engine using 2048-bits Paillier encryption.

Function	Performance
Key generation	3.88s
Message receipt	0.13s
Message decryption	1.16s

Table 2(b) shows the performances of the different functions in the `WSA-MB` application. The functions *Message decryption* essentially has the same performance irrespective of the size of the message because it only decrypts the homomorphic header, leaving the header to be shifted and decrypted on the client-side; or it decrypts the homomorphic message. Thus, both cases involve just one 2048-bits Paillier decryption along with parsing the messaging and looking up the datastore for the private key corresponding to the public key.

6 Conclusions and Future Work

In this paper, we have proposed a private one-way anonymous messaging protocol and a real world prototype to demonstrate the applicability of our protocol in lightweight HTML5-enabled browsers on mobile devices. Our protocol is suitable for particular application scenarios where the sender's identity needs to be protected not only from external attackers but also from the recipient. Our protocol uses probabilistic forwarding through a random set of relay nodes and protects the anonymity of the sender and the privacy of the messages through principles in homomorphic cryptosystems. We are in the process of implementing a real-world anonymous survey system based on our prototype. In the future, we plan to look into the construction of the reverse path. We also plan to deal with churn that can cause message loss and intentional message loss.

References

1. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–90 (1981)
2. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptology **1**(1), 65–75 (1988)
3. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 137–150. Springer, Heidelberg (1996)
4. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for web transactions. ACM Trans. Inf. Syst. Secur. (TISSEC) **1**(1), 66–92 (1998)
5. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management v0.34, August 2010
6. Kikuchi, H.: Sender and recipient anonymous communication without public key cryptography. In: CSEC (1998)

7. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: design of a type iii anonymous remailer protocol. In: IEEE Security and Privacy, pp. 2–15. IEEE (2003)
8. Moeller, U., Cottrell, L.: Mixmaster protocol version 3 (2004)
9. Syverson, P.F., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: IEEE Symposium on Security and Privacy, pp. 44–54. IEEE (1997)
10. Reed, M.G., Syverson, P.F., Goldschlag, D.M.: Anonymous connections and onion routing. IEEE J. Sel. Areas Commun. **16**(4), 482–494 (1998)
11. Syverson, P.F., Tsudik, G., Reed, M., Landwehr, C.: Towards an analysis of onion routing security. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 96–114. Springer, Heidelberg (2001)
12. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. Technical report, DTIC Document (2004)
13. Mislove, A., Oberoi, G., Post, A., Reis, C., Druschel, P., Wallach, D.S.: Ap3: cooperative, decentralized anonymous communication. In: Proceedings of the 11th workshop on ACM SIGOPS European workshop, p. 30. ACM (2004)
14. Freedman, M.J., Morris, R.: Tarzan: a peer-to-peer anonymizing network layer. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 193–206. ACM (2002)
15. Rennhard, M., Plattner, B.: Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In: Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society, pp. 91–102. ACM (2002)
16. Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3), 97–105 (2010)
17. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
18. Damgård, I., Jurik, M.: A generalisation, a simplication and some applications of paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992. Springer, Heidelberg (2001)
19. Danezis, G., Clayton, R.: Introducing traffic analysis (2007)
20. Pries, R., Yu, W., Fu, X., Zhao, W.: A new replay attack against anonymous communication networks. In: IEEE International Conference on Communications, ICC 2008, pp. 1578–1582. IEEE (2008)
21. Douceur, J.R.: The Sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
22. Kedogan, D., Agrawal, D., Penz, S.: Limits of anonymity in open environments. In: Petitcolas, F.A.P. (ed.) IH 2002. LNCS, vol. 2578, pp. 53–69. Springer, Heidelberg (2003)
23. Danezis, G.: Statistical disclosure attacks. In: Gritzalis, D., Capitani di Vimercati, S., Samarati, P., Katsikas, S. (eds.) Security and Privacy in the Age of Uncertainty. IFIP, vol. 122, pp. 421–426. Springer, New York (2003)
24. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 54–68. Springer, Heidelberg (2003)
25. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Comput. Commun. Rev. **31**(4), 149–160 (2001)
26. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)