# Design and Analysis of an Improved BitMessage Anti-spam Mechanism

Alexander Schaub*
*École Polytechnique, 91128 Palaiseau, France
alexander.schaub+INF570@polytechnique.edu

Dario Rossi*†
†Telecom ParisTech, 75013 Paris, France
dario.rossi@enst.fr

***Abstract*—The BitMessage protocol offers privacy to its anonymous users. It is a completely decentralized messaging system, enabling users to exchange messages preventing accidental eavesdropping – a nice features in the Post-Snowden Internet Era. Not only messages are sent to every node on the network (making it impossible to understand the intended recipient), but their content is encrypted with the intended recipient public key (so that s/he only can decipher it). As these two properties combined might facilitate spamming, a proof-of-work (PoW) mechanism has been designed to mitigate this threat: only messages exhibiting properties of the PoW are forwarded on the network: since PoW is based on computationally heavy cryptographic functions, this slows down the rate at which spammers can introduce unsolicited messages in the network on the one hand, but also makes it harder to send legitimate messages for regular users on the other hand. In this paper, we (i) carry on an analysis of the current PoW mechanism, (ii) propose a very simple, yet very effective, generalization of the formula that decouples spammers vs legitimate users penalty showing that (iii) at the optimum, our proposal halves the harm spammers can do, avoiding by definition any impact for legitimate users.**

## I. Introduction

The use of cryptography to protect private communication can be traced back to ancient Egypt [12]. Yet, it is not until the Internet era that protecting information and private communications has become harder, but for the very same reason even more vital. Recent news under the spotlight have indeed shown that privacy is not the norm, with a consequent uprise of HTTP over TLS communication. Yet, HTTPS is only securing the communication channel with some provider (e.g., Google, Facebook, etc.), which ultimately requires trust and does not prevent (even if the provider is loyal to user privacy) accidental leakage of information toward third parties.

For personal communication, peer-to-peer (P2P) has a long history of successful applications enforcing *anonymity* and preserving communication *privacy*, of which the best known examples are respectively Freenet [9] (often traced among the first P2P systems, providing anonymous filesharing), Skype [1] (among the most successful P2P applications ever, providing private VoIP/video/chat communications) and TOR [11] (providing a service-independent multi-hop encrypted VPN). Jointly achieving anonymity and privacy is knowingly hard. For instance, it has been shown that significant leaks can be exploited so that Skype users can be traced, geographically located, and even tied to ongoing BitTorrent downloads with very minimally intrusive techniques[15], which then could lead to deanonymization with usual techniques (see [10] and references therein). Conversely, TOR is harder to setup for (any) user with respect to installing applications such as Skype, and not only TOR exploits have been known since quite a while, but new leakage are still found nowadays [2].

A very recent addition to the P2P application spectrum is represented by BitMessage [17], that combines both anonymity and privacy on the same communication-oriented service. In BitMessage, anonymity is achieved by flooding private but encrypted information at large (or over all) the network, so that is very hard (or impossible) to identifying the communication parties. Privacy is instead achieved since only the intended recipient can decipher the message, which has been encrypted with her public key. Yet, since nodes in the BitMessage network receives many (or all) messages, they need to verify if they are intended recipient at any new message reception: to reduce the risk of message pollution and DDoS, a Proof-or-Work (PoW) mechanism *à la* BitCoin is used, which slows down the rate of successful attacks.

While BitMessage fills an important yet missing gap in the P2P application spectrum, a thorough investigation of the BitMessage spam prevention mechanism is missing so far. Moreover, the formula defining the minimum PoW in BitMessage has been changed recently, and without any technically sound analysis backing this change. In this paper, we focus on the trade-off of having a PoW that can successfully protect the system from attacks, while at the same time being low enough to allow legitimate users to send messages without incurring an exaggerate penalty. With this respect, our contributions in this paper can be summarized as:

- we are the first to propose a scientifically sound and comprehensive study of the BitMessage PoW;
- we propose and evaluate a simple yet effective generalization of the current PoW, that we optimally tune;
- we show that our proposal can halve the attacker impact w.r.t. to the current implementation.

The remainder of this paper first introduces BitMessage and PoW (Sec.II), then introduces our proposal contrasting to the current one (Sec.III) and carrying on some numerical investigation (Sec.IV), before illustrating open points and summarizing our finding (Sec.V).

## II. A Crash Course on the BitMessage Protocol

The BitMessage protocol has first been proposed by Jonathan Warren [17]. Its aim is to replace e-mail/text messages, while providing as much security as PGP without having

**Algorithm 1** Compute PoW

```
1: $H_0 \leftarrow \text{hash}(message)$                ▷ BitMessage uses SHA512
2: $nonce \leftarrow 0$
3: $pow \leftarrow H_0 \ \ \& \ \ (2^{64}-1)$            ▷ First 8 bytes of $H_0$
4: while $pow > target$ do
5:     $nonce \leftarrow nonce + 1$
6:     $H \leftarrow \text{hash}(\text{hash}(nonce + H_0))$
7:     $pow \leftarrow H \ \ \& \ \ (2^{64}-1)$          ▷ First 8 bytes of $H$
   return $nonce$
```

to rely on third-party authorities. In this section we introduce the protocol (Sec.II-A), with special emphasis on the PoW mechanism (Sec.II-B), surveying the current implementation as well as alternative proposals under discussion (Sec.II-C).

## A. A secure and decentralized messaging protocol

For lack of space, in this paper we can only provide a terse introduction to the BitMessage protocol, and we refer the reader to [17] for a longer description. Briefly, BitMessage is a P2P protocol, where each peer is assigned a $\simeq 34$ character long address. This address contains a pair of Elliptic Curve Cryptography (ECC) keys and is used to sign and encrypt user messages. When Alice wants to send a message to Bob, she uses Bob's public key, comprised in its address, to encrypt the message. This operation preserves *privacy,* as only Bob is able to decipher the message, as he is the only one possessing the corresponding private key. In order to enforce *anonymity,* all messages are in fact sent to every other node in the network. To avoid known scalability pitfalls [16], the network is split up into several streams, so that the intended message recipient should be listening to the same stream the sender is in. Therefore, any malicious BitMessage node overhearing a message is neither able to pinpoint the communication entities (as any peer in the stream can be the intended recipient), nor to decipher its content (as the peer private key is needed for decryption). Similarly, one can not spy on Bob acquaintances by simply monitoring the ingoing and outgoing encrypted messages: some were meant for him, some are just relayed. Moreover, since the BitMessage address generation is done independently for each client, there is no need for any central entity, which makes this system highly resilient.

We point out that whereas BitCoin has attracted significant attention from several research communities, the BitMessage protocol has instead limitedly attracted attention of the security community, and especially for what concerns the anonymity and privacy aspects [7, 8, 13], but has otherwise been mostly ignored by both the peer-to-peer as well as the networking community at large.

## B. The Proof-of-Work (PoW) mechanism

Since every message is to be shared by every node on the same stream, this protocol may be prone to abusive flooding and/or spamming: BitMessage copes with this problem by employing a Proof-of-Work (PoW) mechanism.

The high-level idea behind PoW is to *slow down* any potential abuse: in order to have the right to send a message, every user is required to perform a certain amount of work, i.e., applying cryptographic functions, on the message content.

**Algorithm 2** Verify PoW

```
1: $H_0 \leftarrow \text{hash}(message)$
2: $pow \leftarrow \text{hash}(\text{hash}(nonce + H_0))$
3: if $pow \leq target$ then
       return True
4: else
       return False
```

Specifically, the algorithm described in Algorithm 1, consists in a partial collision on a message. In order to verify the PoW, any node receiving the message has merely to hash the whole message once, and perform one subsequent (double-) hash using the given nonce: if the value of the hash is small enough (i.e., the verified value starts with a number of zeros), then the message is valid and is either read (if the node is the intended recipient) or relayed (otherwise). Instead, a message exhibiting insufficient PoW will not be relayed by the nodes in the network, limiting the flood.

The target value depends on the length of the message payload as well as the Time To Live (TTL), which is the time the message is kept in the network. The current version of the protocol uses the following formula:

$$target = \frac{2^{64}}{C\left(|message| + C'\right)\left(1 + \frac{TTL}{2^{16}}\right)} \tag{1}$$

where $|message|$ represents the length of the message, $C$ and $C'$ are some constants to respectively impose a minimum PoW on short messages, and to modulate the mean number of tries to perform (the current BitMessage implementation sets $C = 1000$ and $C' = 1000$). Since $pow$ in Algorithm 1 is an 8 byte integer taking values in $[0, 2^{64}-1]$, and since $pow$ is uniformly distributed in this interval due to the properties of cryptographic hash functions such as SHA512, the average duration of a Proof-or-Work $\mathbb{E}[T_{PoW}]$ is:

$$\mathbb{E}[T_{PoW}] = \frac{1}{hashRate} C\left(|message| + C'\right)\left(1 + \frac{TTL}{2^{16}}\right) \tag{2}$$

where $hashRate$ is the number of SHA512 operations that the user can perform in a second. Clearly, (2) holds for both legitimate users and spammers.

## C. PoW alternatives

Today's most popular PoW mechanism is the so-called *hashcash* [6] algorithm, used for instance by BitCoin, that can also be used by e-mail services which want to avoid spam: in order to send an e-mail, the sender has to find some nonce so that the hash of the e-mail content and the nonce starts with a determined number of zeros. This algorithm is similar to the one used by BitMessage: specifically, only the way to determine by how many zeros the hash should start with changes. For *hashcash*, it is decided by the server for all the messages it accepts to relay, while for BitMessage it mostly depends on the *length* of the message that is to be sent and its *lifetime* in the network (i.e. its TTL).

It has to be pointed out that an analysis [14] of the economics of spamming assert that PoW methods are inefficient so as to avoid spam: authors in [14] estimate that the PoW that makes sending spam so hard to become uneconomic, also makes it impractical for normal users to send legitimate messages. At the same time, this analysis dates back of over a decade [14], so it would need to be at least quantitatively updated. In the meanwhile, BitMessage is used by both legitimate users as well as spammers [5], so that an improved PoW would certainly be beneficial to the former.

Yet, because of the differences in hardware capabilities, between, for example, a smartphone and a desktop PC, the BitMessage PoW mechanism is unfair towards users with less powerful hardware. One possibility to overcome this problem would be to force users to wait a certain time between two messages, i.e., to impose a maximum rate for sending messages. However, it seems complicated to enforce this policy as the sender of the messages is encrypted and therefore only known to the recipient, so that intermediate nodes are unable to detect any malicious user. This solution has been actively discussed on the BitMessage forum [3], and could be used as a complementary solution to the PoW.

Yet, as it appears from discussion in the forum, alternative suggestions are still based on heuristics that, albeit intuitively sound, are subject to yet other trade-offs that have not been fully elucidated either. As mentioned earlier, it is not clear that a hard limit on the rate at which the messages are sent, can be enforced at all. Also, choosing a value for the maximal rate is a non-trivial task, as shown by [14]. The PoW method has the benefit that it allows to enforce a message rate limitation in a decentralized fashion. The main drawback is the fact that this limitation depends on the hardware capabilities of the users, which seems unfair. However, as we will show, a careful design of the PoW formula can help to reduce the risk of malicious exploitation of the network.

## III. PoW Design

To quantify the resilience of the system and analyze the current PoW formula, we first define a realistic scenario with three different kinds of users, subject to distinct behaviors and constraints (Sec.III-A). We then observe that, to let the PoW trade-off be more adverse to malicious users, it is desirable to break the TTL linearity in (1), and propose a generalized PoW model, of which the current one is a special case (Sec.III-B).

### A. Modeling legitimate and malicious users

We consider the three types of users, whose main characteristic are summarized in Tab I. Namely, *Desktop users* employ BitMessage to safely send email-like messages on a desktop or laptop PC. They have a medium computing power, and are not that much concerned about storage space. However, they may not be on-line all day long, but are supposed to read their messages at least once a day. *Mobile users* employ BitMessage to safely send SMS-like messages. They have much more limited computing power, storage space and bandwidth, but are connected to the Internet almost all the time (therefore, TTL can be shorter as the recipient is also supposed to be on-line all the time). Finally, *Spammers* attempt to render BitMessage unusable by maximizing the annoyance of their actions, but have otherwise no preferred TTL or message length. They are supposed to have more computing power than desktop users, can be on-line 24/7 hours a day, and are not bandwidth limited.

TABLE I. BitMessage Legitimate and Spam Users Scenario

| | **PC user** | **Mobile user** | **Spam bot** |
|---|---|---|---|
| hashRate | $\simeq$ 5 MH/s | $\simeq$ 0.5 MH/s | $\simeq$ 30 MH/s |
| Aim | Send e-mails | Send short text messages | Maximize annoyance |
| TTL | 1 day | 4 hours | n.a. |

We use hash rates for BitCoin as a proxy for BitMessage rates, as the rates for SHA256 (used by BitCoin) and SHA512 (BitMessage) double hashes should be similar. Two points are worth making. First, albeit current implementation are limited to running software on the CPU, we assume that the PoW algorithm is able to uses the computational capabilities of the Graphic Processing Unit (GPUs), which is advantageous for spammers. Second, we also assume that while it is easy to find specialized FPGA and even ASIC to mine BitCoins (at rates exceeding 10GH/s) there will be no incentive nor market to sell BitMessage "spam bot" (and it would be too hard for spammers to divert a BitCoin mining rim into a BitMessage spam bot, since the hash functions that are used are not the same). Thus, we model spammers hash rates as being higher, though not incommensurately higher, than that of regular users. These assumptions have been validated by the harvesting attack, led by Robert White [5]. He achieved a typical hash rate of 5 Mh/s using an optimized CPU-only C implementation (which should be available to any regular user), and 25 Mh/s using a GPU implementation (that could be used by spammers). The code is freely available on the Internet [4].

While legitimate users have a typical access time and message lengths (that we consider of an SMS or 500 bytes including protocol overhead for mobile users, and email of 5000 bytes average size for PC users) spammers aim at maximizing the annoyance of their actions, but have otherwise no preferred TTL or message length. With this perspective, two metrics are worth considering: the (i) *number of persistent spam messages* on the stream and the (ii) *overall spam memory footprint*. Indeed, since each received message has to be hashed (in order to verify the PoW), and decrypted (to check if we are the recipient), the *number* of messages received correlates to the CPU pollution (of legitimate users) and to battery depletion (of mobile users). Additionally, the *size* of the waste can be critical for mobile users with limited storage capabilities, or with a bandwidth cap – though in reason of current smartphones specs, battery is likely a harsher constraint.

### B. Design of an Improved PoW

A rationale spammer would thus focus on *maximizing the number of persistent spam messages*. To examine this objective, let $A' = C\left(|message| + C'\right)/hashRate$ and $B' = 2^{16}$, so that we can rewrite (1) as:

$$\mathbb{E}[T_{PoW}] = A'\left(1 + \frac{\text{TTL}}{B'}\right) \tag{3}$$

From (4) it is easy to gather that PoW duration grows linearly with TTL: if the objective of the spammer is to maximize the number of messages kept in the system, he will try to maximize $f(x) = \text{TTL}/\mathbb{E}[T_{PoW}]$, which is the number of messages sent before the first one expires. To make the PoW more resilient

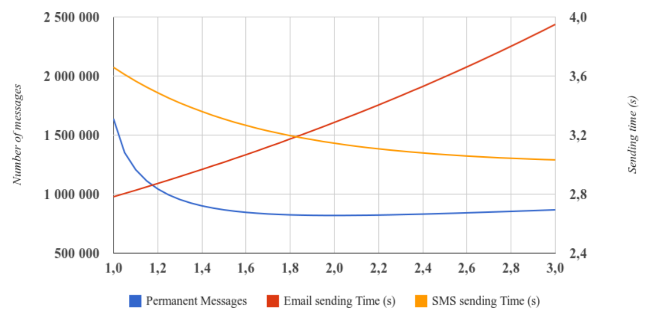

Fig. 1. Number of messages, e-mail sending time and SMS sending time as a function of $\alpha$

to attacks, we propose to break the linearity in TTL by raising TTL$^{\alpha}$ to an exponent $\alpha > 1$ in (4).

$$\mathbb{E}[T_{PoW}] = A' \left[ 1 + \left( \frac{\text{TTL}}{B'} \right)^{\alpha} \right] \tag{4}$$

By doing so, the function the spammer needs to maximize would become:

$$f(x) = \frac{1}{A} \frac{x}{1 + x^{\boldsymbol{\alpha}}}, \qquad A = A'/B', x = \text{TTL}/B' \tag{5}$$

that degenerates in the current PoW for $\alpha{=}1$ (another knob that can be used to make PoW more difficult is the normalization factor $B' = 2^{16}$, which could be reduced to make PoW harder for everybody, depending on the target TTL). The maximum of this function can be found using basic algebra as:

$$x_{max} = \frac{1}{(\alpha - 1)^{\frac{1}{\alpha}}} \qquad \text{if } \alpha > 1 \tag{6}$$

The TTL which will be chosen by the spammer will be quantized to seconds (as TTL is an integer value), however the quantization effects have a negligible impact on the total number of messages and we neglect it in the following. Notice also that the computational power of the spammer (is included in $A'$), does *not* affect the maximum (6).

## IV. Numerical Results

We now optimize, for the scenario under consideration, our generalized formula (6) to maximize resilience against spammers without hurting legitimate users, showing numerical results contrasting it with the current PoW. We elucidate the PoW trade-off by expressing legitimate user performance in terms of the time needed by PC users to send a single email-like message[1], and attackers performance in terms of the number of persistent spam messages.

### A. Results at a glance

In particular, we use the values of the PoW constants $B'{=}2^{16}$, $C{=}1000, C'{=}1000$ as those are the current default values for BitMessage, and assume attackers to send 200-bytes messages which yields to $A' = C(200 + C')/hashRate{=}1,200,000/hashRate$. We gather results in Fig. 1 using discrete optimization, but we point out that results

[1]Note that the time needed to send a short-lived SMS decreases when $\alpha$ increases, as long as the TTL of the SMS is below $B' = 2^{16}\text{s} \simeq 18\text{h}$

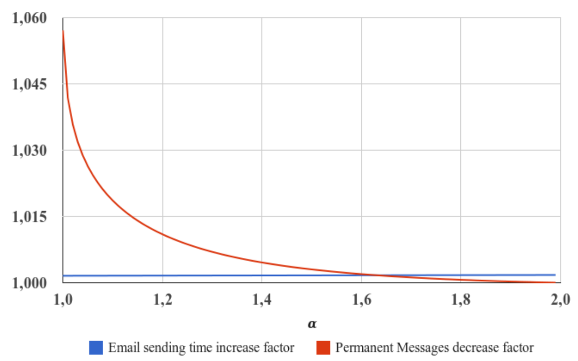

Fig. 2. Logarithmic derivative of the number of permanent messages and email-sending time as a function of $\alpha$

are very close to those obtained directly using (6) which is thus a good approximation. The picture clearly shows that when $\alpha \in [1, 2)$ the PoW becomes increasingly difficult, which in turn (drastically) decreases the number of permanent messages in the stream at a price of a (slight) increases the time needed to send an email-like message for legitimate users. Conversely, for $\alpha{\geq}2$ both the e-mail sending time as well as the maximum number of possible permanent messages increase (hence, $\alpha$ should be always strictly smaller than $\alpha{<}2$).

### B. Optimizing the proposed PoW

The question is now to determine which $\alpha$ value (i) best counters the spammer's possibility to harm the system, while (ii) allowing legit users to unperturbedly send messages at the same time. We notice that the current formula (4) allows to linearly increase the difficulty for both spammers and legit users: for example, changing $A$ to $\tilde{A}{>}A$, then the number of permanent messages decreases by $A/\tilde{A}$ and the email-sending time increases by $\tilde{A}/A$. Therefore, our formulation makes sense only for those $\alpha$ for which the email-sending time increases *slower* than the permanent number of messages increases. More formally, let's denote by $n(\alpha)$ the number of permanent messages for a given value of $\alpha$, and $m(\alpha)$ the corresponding mail sending time. Increasing $\alpha$ by some small value $\varepsilon > 0$ makes sense only if:

$$\frac{m(\alpha + \varepsilon)}{m(\alpha)} \leq \frac{n(\alpha)}{n(\alpha + \varepsilon)} \tag{7}$$

which is equivalent to

$$\frac{m(\alpha + \varepsilon) - m(\alpha)}{\varepsilon m(\alpha)} \leq \frac{n(\alpha) - n(\alpha + \varepsilon)}{\varepsilon n(\alpha + \varepsilon)} \tag{8}$$

which yields the following condition, as $\varepsilon \to 0$ :

$$\frac{m'(\alpha)}{m(\alpha)} \leq -\frac{n'(\alpha)}{n(\alpha)} \tag{9}$$

where $m'(\alpha)$ denotes the derivate of function $m$. In other words, increasing $\alpha$ is advantageous as long as the logarithmic derivative of $m$ is smaller or equal than the absolute value of the logarithmic derivative of $n$: the optimal value of $\alpha$ is precisely the one of which the equality holds. For the scenario under consideration, Fig. 2 shows that absolute values of the logarithmic derivatives intercept for $\alpha \in [1.6, 1.7]$, and more precisely at $\alpha = 1.64$ (according to a gradient method with a precision $\varepsilon = 0.01$) which is thus our optimal value.

Having determined the optimal $\alpha$ value, we can assess the improvement over the current PoW. Since the legacy formula involves two constants that can be chosen in order make the PoW harder by a constant factor, we can compare the formulas with respect to a fixed email-sending time, or a fixed number of permanent messages. As this doesn't matter much, we will compare them with a *fixed email sending time of one second*. As shown in Tab. II, the number of *permanent spam messages are roughly halved* by simply letting $\alpha = 1.64$ compared to the default $\alpha = 1$.

### C. Impact of network delay

We have so far neglected the impact of network delay. However, if there is a *constant* delay for sending each message, the optimal choice of the TTL changes (and depends on the actual hash rates, unlike in our analysis), and the effect of varying $\alpha$ also changes. The considered formulas are not linear anymore, but it is still possible to compute the constants that will yield an e-mail sending time of one second, and therefore the improvement ratios for different values of $\alpha$. Table V shows the effect of an additional fixed delay for each message – cumulatively accounting for network effects, the encryption process, etc – and the effect of choosing a different $\alpha$. While, for example, a 100ms delay penalty may seem excessive, the spammer itself would probably attempt at disguising his/her actions by e.g., tunneling over the TOR network, which induces a higher latency. In Table V, we show the improvement factor over $\alpha = 1$ for two values of $\alpha$: $\alpha = 1.64$ which is the optimal value of $\alpha$ if we suppose there is no delay, and the optimal value once the delay is taken into account In short, if there is a constant delay for each message (which seems natural), then there is even a *stronger* incentive at choosing $\alpha > 1$, and the longer the delay, the greater the effect of $\alpha > 1$. Also, choosing $\alpha = 1.64$ (as in the case with no latency) will *always* result in a significant improvement over the legacy formula.

## V. Conclusions

In this paper we study the Proof-of-Work (PoW) used by BitMessage to fight the spam, we generalize the current formula and provide optimal values that could halve the spammer impact with no noticeable impact for legitimate users. In particular, PoW should avoid malicious users to flood the network while allowing legit users to exploit the BitMessage protocol without penalty at the same time. We argue that the current BitMessage PoW formula has not been thoroughly analyzed: our analysis in this paper shows that a minimal modification to the current PoW can have large impact against spam at no additional cost for legitimate users.

This work is of course preliminary and can be extended along a number of relevant directions, the first of which is to assess the impact of *variable* delay, which requires complementary techniques to ours. For instance, (i) measuring the propagation delay of the messages in the BitMessage network could be useful to choose the best TTL for mobile users; (ii) simulation can help investigating the effect of stochastically variable delay, and finally (iii) a proof-of-concept implementation might be needed in order to verify to what extent the assumptions we have made in this paper hold in the real world.

TABLE II. Comparison of legacy vs improved PoW

| | $\alpha$ = 1 | | $\alpha$ = 1.64 | |
|---|---|---|---|---|
| A'$\times$ hashRate (millions) | 1.20 | 0.43 | 1.20 | 0.39 |
| Sending time (s) | 2.78 | 1 | 3.08 | 1 |
| Permanent messages (millions) | 1.60 | 4.56 | 0.84 | 2.59 |

TABLE III. Improvement ratio for different values of $\alpha$ and delay ($\Delta$)

| | $\Delta$ = 0 | $\Delta$ = 0.01 | | $\Delta$ = 0.05 | | $\Delta$ = 0.1 | |
|---|---|---|---|---|---|---|---|
| Value of $\alpha$ | 1.64 | 1.64 | 2.04 | 1.64 | 3.55 | 1.64 | 5.33 |
| Improvement ratio | 1.71 | 2.11 | 2.19 | 2.94 | 4.02 | 3.43 | 6.16 |

We point out that our contributions are far from being an academic exercise. For instance, during the 2013 BitMessage harvesting-attack [5], attackers spent 3h to generate messages whose transmission clogged the BitMessage for about 18h: we plan to contact the BitMessage community to propose this simple and technically sound drop-in proposal.

## Remerciements

Result in this paper have been obtained during a IN570 course project at Ecole Polytechnique. `http://www.enseignement.polytechnique.fr/informatique/INF570/`

## References

[1] https://www.skype.com.

[2] https://trac.torproject.org/projects/tor/wiki/doc/TorifyHOWTO.

[3] https://bitmessage.org/forum/index.php/topic,2979.0.html.

[4] https://github.com/grant-olson/bitmessage-powfaster.

[5] The confessions of robert white. https://bitmessage.org/forum/index.php?topic=2975.0.

[6] A. Back. Hashcash – a denial of service counter-measure. Technical report, 2002.

[7] D. Bradbury. Can we make email secure? *Elsevier Network Security*, 2014(3):13–16, 2014.

[8] S. Chagani and A. Doll. Dead man's switch: Disseminating secrets in the face of a determined adversary.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[10] X. Ding, L. Zhang, Z. Wan, and M. Gu. A brief survey on de-anonymization attacks in online social networks. In *IEEE CASoN*, Sept 2010.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[12] D. Kahn. *The codebreakers: The comprehensive history of secret communication from ancient times to the Internet.* Simon and Schuster, 1996.

[13] A. Kovacs, I. Karakatsanis, and D. Svetinovic. Argumentation-based security requirements analysis: Bitmessage case study. In *IEEE International Conference on Internet of Things (iThings)*, 2014.

[14] B. Laurie and R. Clayton. "proof-of-work" proves not to work. In *Workshop on Economics and Information, Security*, 2004.

[15] S. Le Blond, C. Zhang, A. Legout, K. Ross, and W. Dabbous. I know where you are and what you are sharing: exploiting P2P communications to invade users' privacy. In *ACM SIGCOMM IMC*, 2011.

[16] J. Ritter. Why gnutella can't scale. no, really, 2001.

[17] J. Warren. Bitmessage: A peer-to-peer message authentication and delivery system. https://bitmessage.org/bitmessage.pdf.