# Reclaiming the Blogosphere, TalkBack: A Secure LinkBack Protocol for Weblogs

Elie Bursztein[*] Baptiste Gourdin[*‡] John C. Mitchell[*]
{elie|bgourdin|jcm}@cs.stanford.edu

Stanford University[*] LSV,ENS-Cachan[‡]

**Abstract.** A LinkBack is a mechanism for bloggers to obtain automatic notifications when other bloggers link to their posts. LinkBacks are an important pillar of the blogosphere because they allows blog posts to cross-reference each other. Over the last few years, spammers have consistently tried to abuse LinkBack mechanisms as they provide an automated way to inject spam into blogs. A recent study shows that a single blog may receive tens of thousands of spam LinkBack notifications per day. Therefore, there is a great need to develop defenses to protect the blogosphere from spammer abuses. To address this issue, we introduce TalkBack, a secure LinkBack mechanism. While previous methods attempt to detecting LinkBack spam using content analysis, TalkBack uses distributed authentication and rate limiting to prevents spammers from posting LinkBack notifications.

## 1 Introduction

Since their emergence over a decade ago [15], blogs have become a major form of communication, with more than 184 million blogs read by more than 346 million readers in 2008 [14]. Along with widespread legitimate use for communicating information and opinions, blogs have naturally attracted two forms of spam (unwanted postings): *comment spam* and *LinkBack spam*. Comment spam is an extension of traditional email spam and can be mitigated by requiring users to authenticate before commenting, or to solve CAPTCHAs [4]. On the other hand, *LinkBack spam* is specific to blogs. LinkBack mechanisms [23] are used to automatically insert cross-references between blogs. A new blog post citing an older one on a different blog can send a **LinkBack notification** to insert a link in the older post **automatically**. LinkBack notifications are an intrinsic part of the blogosphere, and a key ingredient used in blog ranking [21]. Because LinkBack notifications are automated, CAPTCHAs [22] and registration requirements cannot be used to defend against spam. So far, very little research has been conducted on LinkBack spam specifically, and only general anti-spam techniques based on content analysis are currently used.

A recent study [5] reveals that LinkBack spam is a huge issue as a single blog may receive tens of thousands of spam LinkBack notifications per day. Further, it found that LinkBack spammers skillfully use anti-spam-analysis techniques that foil content analysis by inserting random words to escape Bayesian filters. Because LinkBacks are concise, with very little content to analyze, it is difficult to filter LinkBack content by applying content-based filtering techniques.

To combat notification spam, in this paper, we introduce *TalkBack* a secure LinkBack mechanism based on public-key cryptography. TalkbBack departs from the previous approaches as it tackles the LinkBack spam problem at its root: instead of detecting spam via content analysis, TalkBack is designed to prevent spammers from posting LinkBack notifications. TalkBack creates two lines of defense. The first one is a lightweight PKI [1] (Public Key Infrastructure) that ensures the identity of blogs by using public-key cryptography, and makes it hard for spammers to register a fake blog. As a second line of defense, TalkBack enforces a global rate limiting system that ensures that with a single blog identity, a spammer cannot massively spam any collection of participating blogs. An additional benefit of adopting TalkBack is that bloggers can leverage the TalkBack PKI infrastructure to build secure and reliable whitelists and blacklists of blogs. We believe that by combining TalkBack with the other anti-spam mechanisms based on content analysis already in place, the blogosphere will have efficient defense in depth against the assault of spammers.

We have developed, tested and evaluated all the components needed for blog sites and bloggers to adopt TalkBack quickly, with additional attention directed toward Wordpress, the leading blog platform. To make TalkBack readily available to bloggers, we have developed a Wordpress plugin that is available from the standard Wordpress plugin directory `http://ly.tl/tb`. A full implementation of the TalkBack protocol is also available as an open-souce PHP library, maintained and freely available from google code (`http://ly.tl/tbs`), and an operating talkback authority is implemented and maintained at `https://talkback.stanford.edu`.

## 2 Background

In this section we summarize the way LinkBack mechanisms work, how they are abused by spammers, and some reasons why spammers do so.

The term **Blog** is a contraction of the term **web-log**. Blogs can be used for any topic but are often used by **bloggers** to share and exchange information and personal opinions on subjects that range from personal life to video games, politics, and wine. The **Blogosphere** is a collective term referring to all blogs and their interconnections, coined in 1999 by Brad L. Graham as a joke [6]. The blogosphere can be viewed as a graph, with blogs as nodes and edges corresponding to **LinkBacks** between blogs.

**LinkBack mechanisms** are used to produce a link from one blog post to another that references it. For instance if *Blog B* discussing French wine cites a post on *Blog A* about Bordeaux wine, a LinkBack mechanism allows *Blog B* to notify *Blog A* about this citation. As a result of this LinkBack notification, *Blog A* may then display a link back to *Blog B* (hence the name LinkBack). There are three main LinkBack mechanisms, which differ in their implementation details, that are currently used. These three mechanism are *TrackBack* [3], *PingBack* [12] and *RefBack* [23]. Note that for the largest blog platform *Blogger*, Google provides a specific LinkBack implementation based on Google Infrastructure. These LinkBack mechanisms were designed to help blog readers navigate from one post to other relevant posts. Every LinkBack mechanism is implemented into two parts: the **auto-discovery mechanism** and the **notification page**:

The **auto-discovery** mechanism embeds a `<link>` tag or a small Resource Description Framework (RDF) fragment in each blog post to tell other blogs to which page they should submit their LinkBack notification. RDF is a family of World Wide Web Consortium (W3C) specifications designed as a metadata data model, that are used as a general method for conceptual description or modeling of information in web resources.

The **notification page** is the web page dedicated to collecting LinkBacks notification and processing them. For example, the TrackBack notification is an HTTP POST request sent to the notification page which contains four post values: the *post title*, its *URL*, an *excerpt*, and the *blog name*. An example of a TrackBack [3] post request is:

```
POST http://www.example.com/TrackBack/5
Content-Type: application/x-www-form-urlencoded
title=Foo&url=http://www.bar.com/&
excerpt=My+Excerpt&blog_name=Foo
```

**Blog Spam.** Because LinkBack notifications provide an automated way to insert links into other bloggers' blogs, it is not surprising that malicious users began using it soon after it appeared. There are two main motivations for abusing LinkBack mechanisms: search engine optimization, and spam to lure users to malicious sites. One of the major spam-blocker providers, Akismet [18], reported blocking around 15 million LinkBack spams a day in April 2009, in comparison with 1.8 million legitimate LinkBacks. Hence it seems that the percentage of blog spam is slightly lower (90%) than the 98% spam reported for email [8]. However blog spam is more pernicious because it is asymmetric: one spam LinkBack notification might lure thousands of blog readers to a malicious site, whereas delivered email spam usually reaches at most one user.

## 3 Threat Analysis

In this section we introduce the attacker and threat model that TalkBack needs to address. These models are based on the conclusions of our blog spam study [5]. For this longitudinal study of TrackBack spam, 10 million samples from a massive spam campaign over a one-year period where collected and analyzed. Based on the analysis of blog spammers' behavior and resources, we believe that TalkBack needs to block the efforts of a sophisticated adversary that will not be fooled by simple defenses. In particular, we believe that in order to be effective, TalkBack must be able to thwart an attacker that is ressourceful, knowledgable and adaptative and accordingly should assume an adversary with perfect knowledge of the protocol and a lot of IPs, Domains and CPU power at his disposal.

While important to the operation of blogs, our threat model does not address system, services or web attacks because they cannot be addressed directly by a LinkBack mechanism. Therefore, TalkBack focuses on LinkBack threats, which are:

- **Blog Spoofing**: The attacker should not be able to spoof a blog identity. In particular, the attacker should not be able to impersonate a real blog because otherwise it is not possible to use whitelisting or blacklisting mechanisms. TalkBack must ensure the identity of a notification sender.
- **Cried Wolf attack**: The attacker should not be able to report legitimate notifications as a spam otherwise he will be able to prevent legitimate users to use TalkBack by reporting them as spammer. To address this attack when a notification is reported as spam, TalkBack must verify the sender and receiver identities.
- **LinkBack modification**: The attacker should not be able to mount a person-in-the-middle attack that alters the content of a LinkBack in order to spam a blog or abusively report a legitimate user. Accordingly, TalkBack needs to ensure LinkBack integrity.
- **LinkBack replay**: It should not be possible to resend or replay a notification. This is central to enforcing a rate limiting system.
- **Accumulation attack**: It should not be possible for an attacker to accumulate posting authorizations, over an extended period of time, in order to use them all at once to perform a massive "Blitzkrieg" spam.

- **Spamming in breadth**: A Spammer can send a few or even a single spam to many blogs, in a way that each blog will only see a negligible amount of the spam and cannot rate limit it. Therefore the rate limiting and spam detection systems need to be global.
- **Spamming in depth**: Spammer can use many IP addresses and URLs to send spam, which makes blacklisting very hard. Therefore, TalkBack must restrict a sender to a single blog identity so that we can leverage identity-based blacklisting and whitelisting.

| | RefBack | PingBack | TrackBack | TalkBack |
|---|---|---|---|---|
| Trigger Mechanism | Visit from the sender site | Code executed at posting time | Code executed at posting time | Code executed at posting time |
| Notification via | HTTP referer | XML-RPC call | HTTP POST | HTTP POST |
| Information sent | none | - $S$ post URL<br>- $R$ post URL | - $S$ post URL<br>- $S$ site name<br>- $S$ post title<br>- $S$ post excerpt | - $S$ post URL<br>- $S$ site name<br>- $S$ post title<br>- $S$ post excerpt<br>- Seed Token<br>- $S$ Public key<br>- $R$ Public key<br>- Signature |
| Auto-discovery mechanism | none | LINK Tag | Tag in the body | LINK Tag |
| $S$ Authenticity | - | - | - | ✓ |
| $R$ Authenticity | - | - | - | ✓ |
| Integrity | - | - | - | ✓ |
| Confidentiality | - | - | - | ✓ |

**Table 1.** LinkBack mechanisms comparaison

## 4 Overview

In this section, we give an overview of how TalkBack works. In particular, we describe how TalkBack addresses the threat of spammer notifications, the key steps required to post a TalkBack notification, how multiple authorities are handled and how TalkBack compares to the other LinkBack mechanisms.

The key idea of TalkBack is to prevent spammers from posting LinkBacks, or make it prohibitively costly to do so. As mentioned previously, based on our blog spam study [5], we do not consider content analysis effective when used alone. In a nutshell, TalkBack can be viewed as a lightweight PKI (*Public Key Infrastructure*) [1] that authenticates bloggers, blogs and LinkBacks. The blogger and blog identities are verified by a registration mechanism (Sec. 5) that ties a Blogger and his blog to a public-key. This public key is used in every notification to ensure that a spammer can't spoofs a blog identity, modifies its content, or replays it.

The blog registration process uses various security checks to ensure that the blogger is the blog's real owner. It also implements security mechanisms to make sure that it is not possible to register a blog automatically. As a defense in depth mechanism, TalkBack uses a rate limiting system that ensures that even if a spammer is able to successfully register a blog, the amount of spam he can send with it is limited. To ensure that the authority is not a contention point, TalkBack was designed to work with multiples authorities.

TalkBack is also designed to accommodate any blogger' additional privacy needs with a *confidentiality mode* that encrypts notification content and makes sure that no information is disclosed to anyone (including the authority) except the receiver.

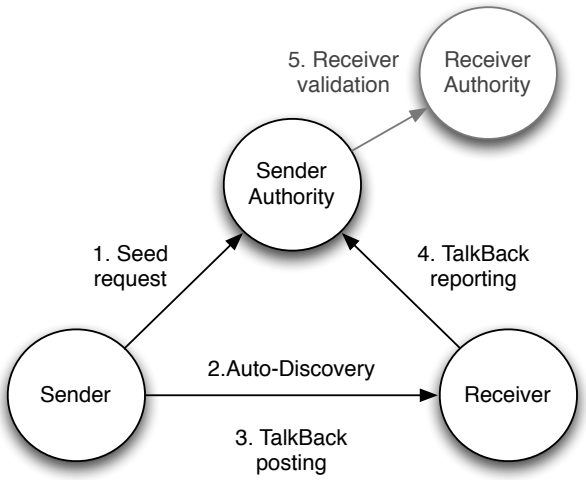


**Fig. 1.** TalkBack protocol overview

**Posting a LinkBack.** Once blogs are successfully registered with an authority, posting a LinkBack is achieved in at most four steps (Diagram 1). In the TalkBack protocol there are three participants (three if the sender and the receiver refer to the same authority) :

1. $A$: The **sender authority**, which is used to authenticate the sender and enforce rate limiting.
2. $S$ : The **sender**, which is the blog that wants to send the LinkBack notification
3. $R$: The **receiver**, which is the blog that receives and processes the LinkBack notification.
4. $A'$: The **receiver authority**, which might be different from the sender authority and is used to authenticate the receiver.

The five steps used to send a LinkBack are depicted in diagram 1. Here is what happens during these steps:

1. The sender ($S$) requests a seed from the authority. This seed is used to prevent accumulation attacks, replay attacks, and to enforce rate limiting (Sec 7).

2. The sender ($S$) crawls the receiver ($R$) blog as usual, discovers the notification URL (Sec 6) and in addition fetches the receiver ($R$) public key used to authenticate the receiver in the LinkBack and to encrypt data (Sec 7).
3. The sender ($S$) uses the seed fetched at step 1 along with the notification URL and public key fetched at step 2 to build and send the secure LinkBack to the receiver ($R$).
4. The receiver ($R$) performs security verification on the received LinkBack and eventually forwards it to the authority $A$ to ensure that the LinkBack and sender $S$ are still valid.
5. If the receiver ($R$) authority ($A'$) is different from the sender ($S$) authority ($A$), then the sender authority contacts the receiver authority to fetch and validate the receiver ($R$) identity. To improve performance, the receiver public key is cached. Note that there is a RESTful API in place that allows authorities to communicate. For clarity and because communication between authorities is straightforward, we assume for the rest of the paper that authorities ($A$) and ($A'$) are the same.

As we will see in the sections 6 and 8, TalkBack provides numerous additional features, such as caching and whitelisting, designed to reduce the workload. Accordingly, in some cases posting a LinkBack only requires performing the Step 3.

**Comparison with Other Mechanisms.** As shown in the table 1, TalkBack is the only LinkBack mechanism that provides security features. None of the other mechanisms provide a way to ensure sender and receiver authenticity, or LinkBack integrity and confidentiality. To ensure that TalkBack will be a widely adopted standard, it has been design to be robust, lightweight, easy to implement and compatible with web standards. This is why we choose, like PingBack, to use the standard HTML tag `<link>` to embedded the discovery mechanism and the blog public key (Sec 6). This choice ensures that adding this information does not interfere with the page validity and makes the required information easily retrievable by a crawler. Similarly we choose to use, like TrackBack, the HTTP POST method to post notifications because the success of the RESTful API support the fact that using the HTTP POST method is the easiest way to send data from one web service to another. Finally, one can observe that TalkBack is backwards-compatible with the TrackBack mechanism which is currently the most popular LinkBack method. This is meant to ensure that the transition to the TalkBack method can be done smoothly without breaking existing systems.

## 5 Blog Registration

In this section, we describe the registration process that a blogger needs to complete before she is allowed to send or receive TalkBack notifications. The goal of this registration is two-fold, it aims at both linking the user identity to a blog URL and at linking the blog URL to a public key. We also describe the update process that the blogger can use to tell the authority that the blog's public key has changed.

The registration process is accomplished in four steps:

1. **Authority selection**: First the user has to choose from a list which authority he wish to enroll with. Currently the only available option is to use our authority (`http://...` ) but both our open-source library and Wordpress plugin are already able to handle multiples authorities. Adding an authority is as simple as adding its public key and URL in the plugin configuration file and push the update to the user via the Wordpress update system.

2. **Turing test**: The second part of the registration aims at preventing automated registration.To do so, we ask the user to solve a CAPTCHA [22]. During this phase we also ask for a name, blog URL, a password, and an email address. Note that our Wordpress plugin reuses the information supplied by the user when setting up Wordpress. Accordingly the registration process is almost completely automated the user should only have to verify the information and supply a password. At the end of this stage the authority asks other authorities is they have already the blog enrolled. If it is the case, then the registration process is aborted and the user is redirected to the authority she is already enrolled with.
3. **Identity Verification**: The third part of the process involves verifying the user by sending an email to the supplied email address. The user is then required to click on an URL that embeds a secret token in its parameters. Because this is a crude and only partly effective identity verification, we give an incentive to the user to provide a stronger form of identity verification, as explained below.
4. **Blog Ownership verification**: The last part of the registration process ensures that the user registers a blog that he owns. This step is very close to the blog reclaiming process used by Technorati [21] and various analytic tools such as Google Analytics: we ask the user to add a random string to his blog index page headers. The random string is embedded into the header by adding the following meta tag:

   ```
   <meta name=TalkBack-Id"  content="random-string" />
   ```

   Once the users says he is ready, we crawl the blog URL, verify that the string is present and fetch the public key and link it to the user identity. Our Wordpress plugin takes care of generating the blog private/public key pair and add the necessary header. The public key fetched at this step is the one that will be used to identify the blog and verify TalkBack notification signature validity.

Note here that the authority never sees the user's private key. As a matter of fact this key is intentionally not part of the generation process for two reasons: First, this limits the incentive to compromise the authority database: even with its content, the attacker will not be able to forge notifications.

Secondly, it limits the trust that the user needs to have in the authority – the authority is not able to post notifications on the user's behalf because they must be signed by her private key.

TalkBack's rate limiting system is reputation-based: the better the user reputation, the more TalkBack notifications she is allowed to send per day. The user's reputation can increases in two ways: first, every time the user sends a talkback notification and this notification is not reported by the receiver as spam, the user's reputation increases toward a maximum. Secondly, the user can increase her reputation by providing stronger proof of her identity. For instance, we envision that the user will be able to link her Facebook or Twitter account to her blog account to have a higher limit[1]. In case of spam reports the reputation of the user decreases until the account is locked. The rate limiting enforcement and the lockout system are designed to mitigate the harm that a spammer can do by stealing the user's private key or registering a fake blog.

Additionally, recall here that taking over the user's account does not allow the attacker to post TalkBack notifications on user's behalf because the authority does not know the user's private key. Similarly attackers can't perform a "*cry wolf attack*" and abusively report legitimate notifications as spam, because a blogger can only report notifications actually received and proves her identity by signing the report with her private key.

## 6 Auto-Discovery

In this section we show how the auto-discovery and notification mechanism for TalkBack is implemented.

Similarly to the PingBack method, the URL for the notification mechanism is embedded in each blog post using a `<link>` tag. For example, if the blog's URL is *"myblog.com"* then in each post, there will be an auto-discovery link which look like this:

```
<link  rel="alternate" type="talkback-notification/plain|encrypted|both"
href="http://myblog.com/notify.htm?id=%postid" />
```

The *rel* parameter is used to tell the sender which kind of TalkBacks the blog accepts. The three acceptable policies are:

1. **Plain**: This policy indicates that the blog only accepts TalkBack notifications that are signed for authenticity and integrity but not encrypted (confidentiality). Confidentiality requires extra computation and is not useful if the blog is public because the content of the TalkBack will ultimately be displayed on a public blog post.

[1] This feature is not yet implemented

2. **Encrypted**: This policy indicates that the blog only accepts TalkBack notifications that are signed for authenticity and integrity and encrypted to ensure confidentiality. This is useful when one wants to preserve notification confidentiality. In this case even the authority (Sec 7) is unable to read it!
3. **Both**: This policy indicates that the blog is accepting both plain and encrypted TalkBack notifications.

This URL contains a variable: **%postid** which is an internal ID used by the blog to know to which post the notification is referring to. To be able to send a notification the sender also needs the receiver's public key, if he doesn't have it, he can fetch it by looking to the link tag:

```
<link rel="alternate" type="talkback-crypto/publicList-hashList"
href="http://myblog.com/talkback-key">
```

The *rel* parameter is used to indicate to the sender which cryptographic algorithms the blog supports. There are two lists of algorithms separated by a **-** (dash):

1. **PublicList**: This is the list used to tell which public key cryptographic algorithms the blog supports. If the blog supports multiple algorithms, they are separated by a **,** (comma). Currently TalkBack uses RSA, but we hope in the future to use elliptic curves (EC) when they become more widely available as this will decrease the size of the public keys and thus reduce the network load.
2. **HashList**: The hash list specifies which hash functions can be used. Currently, TalkBack uses SHA1. In the future TalkBack will support the upcoming SHA3 standard.

Note that, as explained in the threat model section, we assume that the attacker will be able to fetch any public information so we don't even try to prevent him from doing so. Instead we rely on the Kerckhoffs' principle and base the security of TalkBack on the security of the keys used.

## 7 Protocol

In this section, we describe how the TalkBack core protocol works step by step (Diag 1). To do so, we use a formal representation that abstracts away some implementation details for the purpose of clarity. Note that, like every other LinkBack protocol, TalkBack is fully automated and does not require any blogger intervention. The blog engine takes care of sending notifications automatically when a blogger writes a blog post. Our Wordpress plugin hook into the Wordpress notification system to do so.

**Notation.** We take the following conventions: $Ar$ denotes the receiver TalkBack authority, $As$ the sender TalkBack authority, $S$ the sender of the TalkBack notification, and $R$ the receiver of the TalkBack notification.

For message direction, we write $R \rightarrow A$ to say that the TalkBack receiver ($R$) sends a message to the TalkBack authority ($A$). For encryption, we use $\{n\}_A$ to denote that the nonce $n$ is encrypted with the public key of $A$. For the sake of clarity, we take the convention that the signature is applied to all the nonces located on the left-hand of the signature symbols. For example $n1, n2, n3, Sig_A$ is equivalent to $n1, n2, n3, Sig(n1, n2, n3)_A$. Finally the letter on the left is used to indicate whether the message is used for the plain version $\mathcal{P}$, the encrypted version $\mathcal{E}$, or both versions $\mathcal{B}$ of TalkBack.

**Step 1: Seed Request.** In the first step the sender $S$ requests a seed from the authority $As$ that will be used to generate the tokens used in the next step. The seed request goes as follows:

$\mathcal{B}$: $S \rightarrow As$ $\{ts,\ \mathsf{H}(TB)\}_{As}, Pk_S, Sig_S$
$\mathcal{B}$: $As \leftarrow S$ $\{R_s, R_n, R_t\}_S, Sig_{As}$

First, $S$ sends a seed request that contains its public key $Pk_S$ used to be identified and the *hash* ($\mathsf{H}(TB)$) of the four TalkBack content variables which are *title, excerpt, URL, blog_name*. The sender public key ($Pk_S$) is used to identify the sender blog and the timestamp to introduce randomness. In return, the sender ($S$) receives from our authority $As$ a *random seed* ($R_s$), the *number of TalkBack notifications* ($R_n$) he is allowed to use this seed for and the seed expiration time ($R_t$). Note here that the number of notifications allowed for a given seed depends on the user reputation and the version used. The rationale behind this decision is that if the user chooses to use the secure version to ensure confidentiality, it is unlikely that he will notify a lot of blogs.

The seed is used to enforce the rate limiting system: since the blogger can only use the seed to generate a limited number of tokens, he will not be able to spam the blogosphere massively. Using a seed decreases the network load as only one request/response to the authority is sufficient to acquire all the tokens needed to post the notification for a given post. In case the user exceeds his quota of notifications, which should not happen often for an honest user as it is relatively large, he has the ability to visit the authority to reset his quota by solving a CAPTCHA. Of course, we rate limit the number of times the user can reset his quota. The security rationale behind allowing users to reset their quotas is that if we make it as expensive for the spammer to reset the quota than creating a blog, he will have no incentive to use this functionality.

**Step 2: Discovery and Token Generation.** Once the sender has acquired the *random seed* ($R_s$) and the *number of TalkBack notifications* ($R_n$) he is allowed to send, he crawls all the links embedded in his blog post and discover those who point to blogs that use TalkBack thanks to the discovery mechanism. For each notification that needs to be sent, the following standard S/Key generation algorithm [7] is used to compute the required unique token from the seed.

**Step 3–4: Posting Notifications.** Now that the sender has the list of URL he needs to send notifications to and the associated tokens, he will start posting these notifications. Each TalkBack notification is posted using the following protocol:

$\mathcal{P}$: $S \rightarrow R$ $\mathsf{H}(TB)$, $T_x$, $ts$, $Pk_{As}$, $Pk_S$, $Pk_R$, $Sig_S$ TB
$\mathcal{E}$: $S \rightarrow R$ $\{\mathsf{H}(TB)\}_{As}$, $T_x$, $ts$, $Pk_{As}$, $Pk_S$, $Pk_R$, $Sig_S$, $\{TB\}_R$

$\mathcal{P}$: $R \rightarrow Ar$ $TB$, $T_x$, $ts$, $Pk_{As}$, $Pk_S$, $Pk_R$, $Sig_S$, $Sig_R$
$\mathcal{E}$: $R \rightarrow Ar$ $\{\mathsf{H}(TB)\}_{As}$, $T_x$, $ts$, $Pk_{As}$, $Pk_S$, $Pk_R$, $Sig_S$, $Sig_R$

$\mathcal{B}$: $R \leftarrow Ar$ $D, ts2, Sig_R, Sig_{Ar}$

In the plain ($\mathcal{P}$) version, the sender ($S$) does a HTTP POST to the receiver ($R$) notification page discovered in the previous step. The sender sends the four content variables (TB) which are the *title, excerpt, URL and the blog_name*, the unique token $T_x$ generated during the previous step, $ts$ the notification creation time, the sender public key $Pk_S$, and the receiver public key $Pk_R$. Everything is signed with the sender private key ($Sig_S$). Sending the sender authority public key ($Pk_A$) is needed to support multiple authorities: the receiver uses it to know which authority to contact to validate the TalkBack. The encrypted ($\mathcal{E}$) version sends the four content variables encrypted $\{TB\}_R$ with the receiver's public key $R$ and the encrypted hash of the four content variables $\{\mathsf{H}(TB)\}_A$ with the sender authority's public key. The receiver can verify the talkback content integrity by hashing and encrypting it with the authority public key and finally compare these two encrypted hashes. This hash is needed to ensure that the attacker does not tamper with the encrypted content and to prove to the authority that the encrypted content is really what the sender sent. Please note that we intentionally did not include the encrypted variables as part of the signature to ensure that the receiver does not have to forward them to the authority while validating the notification.This is not an integrity issue because the hash of encrypted variables is signed and acts therefore as the signature itself. Finally it is worthwhile to note that it is necessary to include the receiver's public key ($Pk_R$) in the notification and as part of the signature because otherwise the spammer will be able to perform a "cry wolf attack" and report an arbitrary sender even if this one never sent him a notification directly.

In the second step the receiver ($R$) verifies that the TalkBack is valid by verifying that that receiver public-key ($Pk_R$) is his own key and that the verifies the sender signature is valid. If the TalkBack is valid, he signs it and forwards it to the sender authority for validation.

Finally, in both ($\mathcal{B}$) cases the authority $A$ answers whether the notification is valid or not ($D$) and the reason for rejection if needed. To decide if a notification is valid the authority verifies the following elements:

– **Signatures** : The authority verifies that the signatures match the public keys present in the notification and the notification integrity with them.

- **Token**: The authority verifies three things about the random token $T_x$: First, that it is indeed associated with the sender's public key, secondly that it wasn't used before, and finally, that it is not expired.
- **Sender authenticity**: The authority is able to verify the sender's authenticity by correlating two things : first that the signature matches the public key present in the TalkBack, which means that the sender knows the private key, and secondly that the token matches the seed request made by this sender. No one is able to change the sender's public key embedded into the notification because it will invalidate the sender's signature.
- **Receiver authenticity**: The receiver's authenticity is ensured by the fact the the signature matches the public key present in the notification. The receiver's public key was put in the notification by the sender and therefore can't be swapped with another one without invalidating the sender's signature. Therefore, if the sender's and the receiver's signatures are valid it is a valid TalkBack.

## 8 Optimizations

In this section, we present some optimizations that have be implemented to improve TalkBack's performance. These optimizations aim at reducing blogs and authority workload.

The first optimization that is used to improve TalkBack performance is to use notification batch processing. Upon receiving a notification, the receiving blog instead of forwarding it immediately to the authority for validation, puts it in a queue and waits until the queue is full or a maximum age is reached to send to the authority. Queuing decreases both the workload and the network load as it only uses a single pipelined connection between the blog and our authority. Our Wordpress plugin allows bloggers to customize the queuing behavior in terms of queue size and maximum time before processing to accommodate their needs.

As a second optimization, a blogger can also leverage the TalkBack PKI to build a whitelist of blogs that he trusts and for which he will accept TalkBack notifications without validating them with the authority. Using sender public-keys to build a whitelist is more simple and robust than using a password or a "secret URL" as it does not requires the use of a shared secret that needs to be exchanged and can be leaked. Then, on a regular basis the blog can check the authority revocation list to see which keys in the whitelist have been revoked and for what reasons. Whitelisting decreases the blog and authority workload. The main downside of whitelisting is that once a blog is whitelisted, the blogger will be able to flood this particular blog with notifications as the rate limiting is enforced by the authority, so the blogger should only whitelist trusted blogs. Our Wordpress plugin already support this form of whitelisting.

## 9 Performance

To evaluate the efficiency of the TalkBack approach and facilitate adoption, we have implemented the protocol in an open-source PHP library (`http://ly.tl/tbs` ), are maintaining an authority `https://talkback.stanford.edu` on a dedicated 16 core server and are providing a Wordpress plugins to make TalkBack readily available to bloggers (`http://ly.tl/tb` ). The TalkBack blog-side implementation in PHP is around 5000 lines of PHP and uses openSSL and mcrypt as crypto engines. Mcrypt is used to provide backward comparability to PHP version $< 5.3$.

In order to evaluate how TalkBack will scale, we have performed two evaluations: the first one to understand how well an authority can scale and the second evaluation to determine how well the receiving blog-side process will scale.

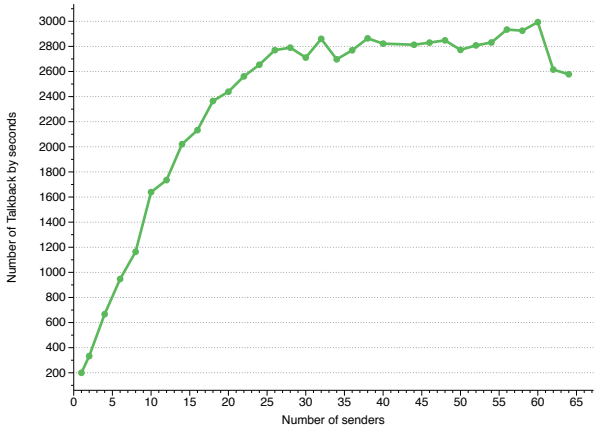


**Fig. 2.** Number of TalkBacks processed by second by Authority

To understand how well authorities scale, we have evaluated how many TalkBacks a single authority (ours) is able to process per second. To make sure that the bottleneck was on the authority, we generated ahead of time 100 000 TalkBacks and used several senders/machines to send them at once to the authority. The senders are not actual blogs but custom php scripts that use our library. As visible on figure 2, as the number of senders increases the number of TalkBacks processed increases until it reach a plateau around 2800 TalkBacks a second. This benchmark is properly evaluated on a 24-hour basis, because blogs notification is spread relatively evenly across a 24-hours period [20], due to timezone variations and blogger habits. Accordingly the fact that our authority using a single frontend is able to process around *242 Million* TalkBacks a day

makes us confident that even though TalkBack is designed to allow multiple authorities our authority alone with additional frontends will be able to sustain the entire blogosphere that currently consists of around 180 Million blogs [14].

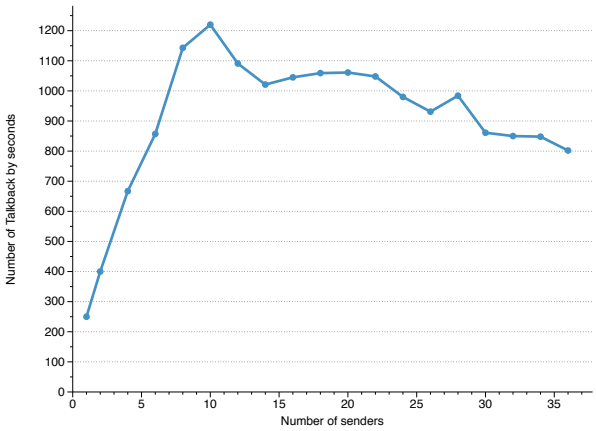


**Fig. 3.** Number of TalkBacks processed by second by the receiving blog

We conducted a similar experiment to see how fast a receiving blog is able to process TalkBack notifications. To make this test realistic, we used as a receiving blog Wordpress 3.0 (the latest version) equipped with our plugin. As in the previous test senders are custom scripts that send 1000 talkback notifications as fast as possible. We also used a more standard hardware platform as the blog was hosted on a 2.4GHz Intel quad core. As visible in figure 3 a single blog is able to process more than 1000 TalkBacks a second which is more than enough even for very high traffic blog. It is unlikely that a single blog will received more than *84 millions* notifications a day.

## 10 Additional Relevant work

In this section we present relevant work to our approach.

**TrackBack Validator.** The WordPress TrackBack Validator [19] looks at the sender URL to validate that the post contains the URL of the receiver. This approach increases the network load because each receiver will look at the sender's page leaving the blog vulnerable to a DDOS attack.

**Reputation system.** Using a reputation system alone for TrackBack spam is ineffective because an attacker may change the blog URL for every posts. Therefore

any long-term classification based on TrackBack is bound to fail because there is no way to prevent spoofing (under the current TrackBack specification).

**IP Blacklisting.** While blacklisting based on IP might currently work as the spammers today seem to use only a small number of IPs, it is not a sustainable solution because in the long run, it is likely that spammers will use botnets and therefore have a huge pool of IPs.

**Rate limiting.** Rate limiting at the blog level is not effective because a blog does not have a global view of the situation and therefore cannot stop spammers that target a huge number of blogs and post only once to each of them with the same IP.

**More Relevant Work.** Previous studies of spam email report that around 120 billions spam emails are sent every day [8]. In [9] and [11], the authors study a spam campaign by infiltrating the Storm botnet, while [2] analyzes the revenue generated by Storm spam. A DOS defense study [13] notes that ideas spread more quickly in the blogosphere than by email. In previous work on linkback spam, [16] examines ways that the language appearing in a blog can be used as a blocking defense. Similarly, [17] studies how the language of web pages, including blogs, can be used to detect spam. In [10] the authors use Support Vector Machines (SVM) to classify blog spam.

## 11 Conclusion

We propose a secure LinkBack protocol called TalkBack. This protocol is designed to prevent unauthorized LinkBack notifications by verifying blogs' authenticity and by imposing a rate limiting system. Although TalkBack adds cryptographic operations to the main LinkBack actions, we believe this level of defensive effort is appropriate, given the result reported by previous study of blog spam activity [5]. We have implemented and are maintaining the required TalkBack authority. We also provide an open-source library for integrating TalkBack into blog engines and a Wordpress plugin that makes TalkBack readily available to bloggers. Our performance evaluation shows that a single authority can sustains the entire blogosphere which makes TalkBack a viable option to defend against spammer.

## References


1. Carlisle Adams and Steven Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations 2nd edition.* Addison-Wesley, 2002.
2. Clive Akass. Storm worm 'making millions a day'. http://www.pcw.co.uk/personal-computer-world/news/2209293/strom-worm-making-millions-day, Feb 2008.
3. Six Apart. Trackback technical specification. `http://www.sixapart.com/pronet/docs/trackback_spec`, 2004.
4. Six Apart. Six apart guide to comment spam. `http://www.sixapart.com/pronet/comment_spam`, 2006.

5. Elie Bursztein, Peifung Lam, and John C. Mitchell. Trackback spam: Abuse and prevention. In ACM, editor, *Cloud Computing Security Workshop (CCSW 2009)*, 2009.
6. Brad L. Graham. Bradland must see http comments. blog `http://www.bradlands.com/weblog/comments/september_10_1999/`, Sep. 1999.
7. Neil M. Haller. The s/key one-time password system. In Internet Society, editor, *Symposium on Network & Distributed Systems Security*, 1994.
8. Ironport. Internet security trends. `http://www.ironport.com/securitytrends`, 2008.
9. Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 3–14, New York, NY, USA, 2008. ACM.
10. Pranam Kolari, Akshay Java, Tim Finin, Tim Oates, and Anupam Joshi. Detecting spam blogs: A machine learning approach. In *2006. Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.
11. C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamcraft: An inside look at spam campaign orchestration. In USENIX, editor, *LEET*, 2009.
12. Stuart Langridge and Ian Hickson. Pingback 1.0. Technical report, Hixie, 2002.
13. Ashraf Matrawy, Anil Somayaji, and P. C. Oorschot. Mitigating network denial-of-service through diversity-based traffic management. In *ACNS'05*, pages 104–121. Springer Science+Business Media, 2005.
14. Universal McCann. Power to the people - social media tracker wave.3. `http://www.universalmccann.com/Assets/wave_3_20080403093750.pdf`, 2008.
15. Declan McCullagh and Anne Broache. Blogs turn 10 who is the father? `http://news.cnet.com/2100-1025_3-6168681.html`, 2010.
16. Gilad Mishne, David Carmel, and Ronny Lempel. Blocking blog spam with language model disagreement. In *In Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.
17. Alexandros Ntoulas and Mark Manasse. Detecting spam web pages through content analysis. In *In Proceedings of the World Wide Web conference*, pages 83–92. ACM Press, 2006.
18. Automattic Production. Askimet trackback statistics. `http://akismet.com/stats/`, 2010.
19. Dan Sandler and Andy Thomas. Trackback validator. http://seclab.cs.rice.edu/proj/trackback/, 2009.
20. K.C. Sia, J. Cho, and H.K. Cho. Efficient monitoring algorithm for fast news alerts. *IEEE Transactions on Knowledge and Data Engineering*, pages 950–961, 2007.
21. Technorati. Technorati top 100 blogs. http://technorati.com/pop/blogs/, 2011.
22. L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In Sringer, editor, *Eurocrypt*, 2003.
23. Wikipedia. Linkback. `http://en.wikipedia.org/wiki/Linkback`, 2011.