Nym: Practical Pseudonymity for Anonymous Networks *

Internet Security Research Lab Technical Report 2006-4

Jason E. Holt and Kent E. Seamons

Internet Security Research Lab
Brigham Young University
seamons@cs.byu.edu

June 2006

Abstract

Nym is an extremely simple way to allow pseudonymous access to Internet services via anonymizing networks like Tor, without losing the ability to limit vandalism using popular techniques such as blocking owners of offending IP or email addresses. Nym uses a very straightforward application of blind signatures to create a pseudonymity system with extremely low barriers to adoption. Clients use an entirely browser-based application to pseudonymously obtain a blinded token which can be anonymously exchanged for an ordinary TLS client certificate. In the appendix, we give the complete Javascript application and the necessary patch to use client certificates in the popular web application MediaWiki, which powers the popular free encyclopedia Wikipedia. Thus, Nym is a complete solution, able to be deployed with a bare minimum of time and infrastructure support.

1 Introduction

Nym is a straightforward way to achieve cryptographically protected pseudonymity for privacy protecting systems like Tor [3]. While many strong, privacy protecting credential schemes have been proposed in the literature, few if any have ever achieved widespread adoption. Thus, in order to reduce barriers to entry for potential clients, we present a system designed to be Free and simple to implement, install, understand and use. We also present a simple yet complete infrastructure and show how existing individual services can easily adopt Nym to allow privacy protecting pseudonymity for their users.

Many existing services limit user abuses like posting spam or inappropriate comments by blocking IP addresses of abusers, or requiring users to prove ownership of a valid email address when creating a user account, which can then be disabled if the user misbehaves. While the academic literature does not consider such measures to be strong deterrents, they are nevertheless widely implemented as a tradeoff between the needs of servers to limit abuse, and the reluctance of users to maintain cryptographically strong digital identities or provide sensitive identity information (like bank or government identifiers) just to post a blog comment or edit a Wikipedia article.

Fortunately, the needs of the server and the needs of users are not mutually exclusive. Servers need a deterrent to users who misbehave, regardless of their actual identities. Thus, disabling access to a particular IP address or email address is usually sufficient, even though those resources correspond very loosely to individual people. Users, on the other hand, want to prevent identity theft, and may want their online interactions with a service to be judged on their own merits, rather than being collected into invasive dossiers and used for marketing, discrimination or harassment. Nym serves both of these needs, by allowing pseudonyms to be rationed on the same person-to-resource assumptions currently in use, but keeping a privacy protecting cryptographic barrier between the resource used to create a pseudonym (which can often be used to violate the user's privacy) and the pseudonym itself.

Since both servers and clients are often reluctant to dedicate extensive resources to data security, Nym includes a client-side implementation in Javascript, requiring no software installation on client computers beside their existing web browsers. This

*This research was supported by funding from the National Science Foundation under grant no. CCR-0325951 and prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

client fits in three pages of code+html, and we include it in its entirety. The client ends up with an ordinary TLS client certificate signed by a Nym-enabled Certificate Authority (CA).

On the server side, only a few lines of Apache configuration and less than a page of code in MediaWiki are required to create a web server that accepts client TLS certificates and allows existing MediaWiki tools to be used for blacklisting certificates of abusers. MediaWiki is one of the most popular WikiWikiWeb implementations on the Internet, developed in parallel with Wikipedia, the Free user-built encyclopedia.

1.1 Contribution

Nym grew out of a discussion on the Tor email list about Wikipedia's practice of blocking Tor users from making changes to articles. Wikipedia blocks most Tor exit nodes due to abusers who had used Tor in the past to avoid IP-address based bans. Privacy-protecting credential systems were mentioned, but it was pointed out that such systems tend to be patent-encumbered and difficult to implement. Another problem was the basis for pseudonymity; privacy protecting credential systems are generally described in terms of large, established agencies issuing digital credentials to the masses. Such systems would create a high-stakes game of cryptographically certified personal information which would naturally tend to intimidate users of an anonymity network like Tor. Furthermore, such scenarios cannot exist without extensive investment and cooperation between issuing agencies far removed from individual clients and servers. On the other hand, real, high-volume sites such as wikipedia.org and blogspot.com use enforcement mechanisms such as IP-based blocks and captchas, which can be circumvented relatively cheaply and don't cleanly map to individual people. In other words, there's a major gap between the high-stakes, high performance, high barrier to entry privacy systems described in the literature, and the real-world systems with modest requirements that serve millions of clients often with no formal security mechanisms at all.

As a concrete example, the paper "Negotiating Trust on the Web" [7] gives two example cases of identity and attribute based online transactions using digitial credentials. The first is of a merchant selling cigarettes online who requires proof of age via a digital driver's license, and the second is of an online nursery with a Better Business Bureau (BBB) credential which sells plants only to customers with a credit card credential and a reseller license credential. Before these scenarios could play out in real life, four organizations who are not direct parties to the transactions (the Department of Motor Vehicles, the BBB, a credit card company and a plant reseller certification authority) would all have to be in the business of issuing cryptographic credentials. The merchants would have to establish access policies for their services, maintain their own credentials and run trust negotiation software. Fortunately, such software exists [7] but clients would need to obtain, install and securely maintain various certificates, install client side trust negotiation software and configure their web browsers to use that software appropriately. Similarly, many of the credential systems which cryptographically derive the kinds of privacy Nym offers (and often in much more elegant and powerful ways than Nym does, e.g., [1]) use complicated credential formats which depend heavily on the underlying mathematics, with unique CAs, policy mechanisms and authentication protocols, and give sample scenarios which assume widespread adoption of their techniques. Implementations are patent-encumbered and nontrivial to install and integrate with an average user's existing practices.

Of course, these systems and scenarios were created to illustrate how a powerful attribute-based access control system could be used to manage complicated transactions in a world with many certifying authorities and transactions needing cryptographic certainty. But popular Internet services do not seem to be converging on such a scenario. The authors are aware of no websites serving the general public which use TLS client certificates or maintain attribute certificates (beyond traditionally issued X.509 TLS server certificates, which are generally only used for establishing the owner of a domain name) for establishing trust with their users, and no jurisdictions which issue driver's licenses as cryptographically signed certificates. However, there are many extremely popular sites which use ad-hoc, low security techniques to keep automated abusers and repeat offenders at bay.

We offer Nym as a pseudonymity system for open-access, low-liability services such as wikipedia.org and blogspot.com that would provide reasonable privacy for their users, discourage abuse approximately as well as such sites currently do, and would be simple enough for users and admins to use without major inconvenience or restructuring. The context of the problem was quite liberating; while traditional academic scenarios involve merchandise purchases and truly sensitive personal information, most of the potential Nym users will use Nym for only casual pseudonymity at first, and in the worst case scenario can only vandalize community-edited Wiki pages (which can easily be restored) or post abusive blog comments until Nym access is disabled for that site.

Our research contributions include:

- A complete pseudonymous credential system, designed to be used without an infrastructure provided by external government or industry authorities
- A system with extraordinarily low barriers to adoption for both clients and service providers, providing adequate privacy for clients and adequate abuse resistance for servers
- A system straightforward to understand and analyze, with an implementation short enough to be rapidly audited by a single person

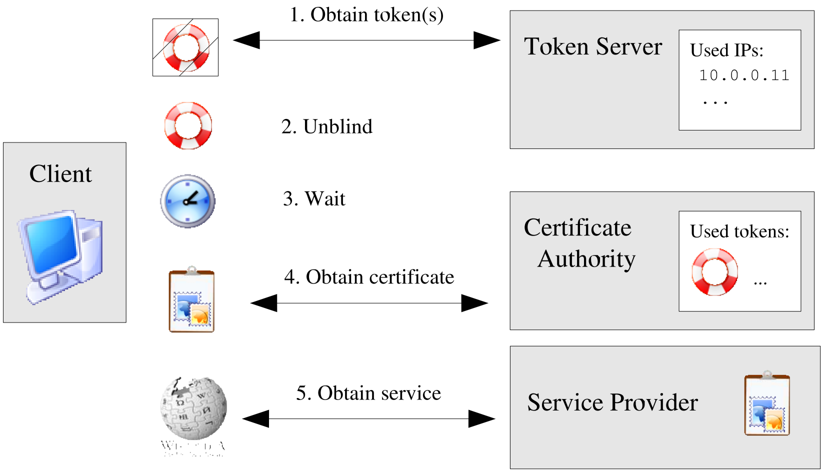


Figure 1: Overview of Nym architecture. Only the first step involves revealing identifying information; the remainder of the process can take place via an anonymizing network. LGPL icons courtesy Wikimedia Commons.

- A report of the barriers we encountered while trying to build such a system

2 System Description

We found that users of a large anonymity network were being denied access to popular Internet services as a result of the abuse made possible by strong anonymity. Solutions to related problems described in the literature seemed unsuitable due to their high complexity, cost of adoption and assumptions about pre-existing infrastructure. Consequently, we built a system that:

- Allows clients to obtain and use pseudonym credentials with a minimum of effort, without even installing additional software
- Allows service providers to accept these credentials with a minimum of effort
- Allows service providers to limit abuse using familiar methods of isolating abusers, such as blocking access to holders of a particular IP or email address, by ensuring that pseudonym certificates are only issued according to these heuristics

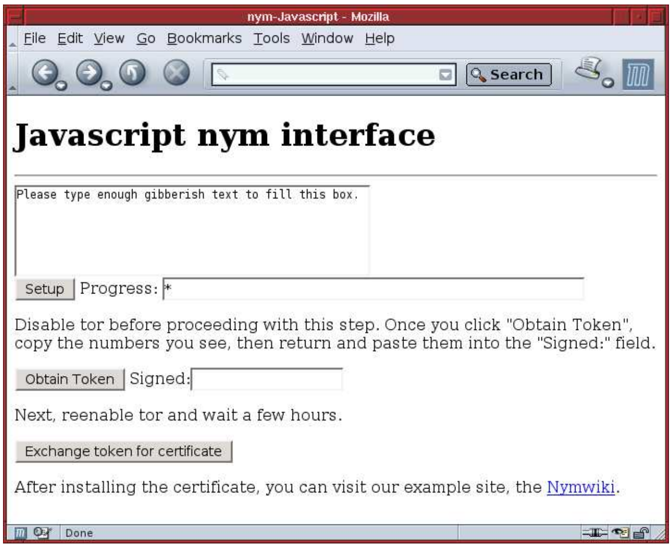
Nym consists of four components. A token server implements blind signatures as described in Chaum, Fiat and Naor's 1990 paper "Untraceable Electronic Cash" [2], issuing tokens based on a simple rule such as "one token per IP address". We implemented the token server as a CGI script in 87 lines of Perl, including IP-rationing code. The second component is a CA which exchanges a token for a signature on an X.509 client certificate. Our software distribution includes scripts that automatically set up a new CA and prepare it for signing client certificates. The third is the client-side software which talks to the token server and CA, and the fourth is the interface between client certificates and Internet services.

Figure 1 illustrates this process. Figure 2 shows a screenshot of a simplified client interface through which users can complete the entire process of obtaining a token and exchanging it for a client certificate. To ensure a good random seed, users type random text into a box. Then they click a button to generate token parameters, and a second button to submit the blinded token to the token server. Due to Javascript limitations, users then must copy and paste the signed, blinded token returned by the token server into the "Signed:" field, wait a random interval, and then click another button to exchange the token for a client certificate. Once they install the certificate, users are ready to access Nym-enabled services.

The following subsections give details on each of Nym's subsystems. Note that there's nothing Nym-specific about the fourth component, Service Providers, since by the time clients connect to servers they're using perfectly normal client TLS certificates. But since our work emphasizes reducing barriers to entry, we show how services such as MediaWiki can easily be adapted to use client certificates with their existing authentication mechanisms.

Once the client has a certificate installed in her web browser, she can use it with compatible Internet services to establish a pseudonymous reputation. Web services accustomed to blocking IP addresses or email addresses of misbehaving users can instead treat client certificates, issued one per unique IP or email address, as unique identifiers, blocking certificates of users that misbehave without ever learning the actual identifying information used to obtain the certificate.

Each network message in our implementation is sent via a web interface, and we assume that all connections use TLS for integrity and confidentiality. To use Nym, clients do the following:


nym-Javascript - Mozilla
File Edit View Go Bookmarks Tools Window Help
Search
Setup
Progress:
Obtain Token
Signed:
Exchange token for certificate
Done

Javascript nym interface

Please type enough gibberish text to fill this box.

Disable tor before proceeding with this step. Once you click "Obtain Token", copy the numbers you see, then return and paste them into the "Signed:" field.

Next, reenable tor and wait a few hours.

After installing the certificate, you can visit our example site, the Nymwiki.

Figure 2: Our Javascript client allows users to complete the entire process of obtaining a token and exchanging it for a client certificate. This screenshot shows a minimal interface; our more detailed client adds explanatory text to help users better understand how to properly use Nym.

1. Obtain a token in exchange for proving possession of a resource such as an IP address or email address to the token server. This is the only step in which the client might need to expose identifying information such as an IP address; the rest can take place via an anonymizing network
2. Unblind the token
3. Wait a random interval sufficient to foil transaction time correlations
4. Exchange one or more tokens for a CA's signature on an X.509 client certificate
5. Load the certificate into a web browser and use it to gain access to a TLS-based web service

2.1 Token Server

The token server is quite simple. Users connect and submit a blinded token for signing along with proof of possession of a resource. In the case of IP addresses, this proof is implicit in making a connection to the server from that IP. Other resources might require more explicit proofs, such as following a unique, randomized link sent to the user's email address, in just the same way popular services currently verify possession of email accounts [4]. The token server checks to see whether a token has already been issued for the user's resource, and if not, signs and returns the token. Thus, the token server represents the bank in the system described in [2], without the need for elaborate means of preventing multiple spending (since, in our initial implementation, users of a particular token server only have one place to "spend" their tokens.)

Blind signatures allow a client to transform a value for signing in such a way that the client can later invert the transformation, leaving the signature on the intended value intact. The signer cannot correlate the value it signs with the signed value the client ends up with. This makes blind signatures nontrivial to use with rich documents such as attribute certificates, since the signer has no way to examine the value to be signed. However, if the verifier enforces a suitable constraint on the client, for instance that the client knows the hash function preimage of the signed value, the hash's preimage resistance ensures that the client can be limited to obtaining a single valid signed token per transaction with the signer. This is all we need for Nym; the token server is certifying only a single bit of information, namely that the client has "spent" a quasi-identifying resource in exchange for the token.

Of course, the vague correspondence between IP addresses and human beings is not always the best way to isolate abusers for blacklisting. Proof of possession of any other relatively scarce resource can also be a basis for discouraging abuse. Captchas are a common technique for preventing automated abuse. Computational puzzles can be used for rate-limiting abuse. Some sites require users to prove ownership of an email address, often excluding free, more-or-less unlimited email services like Hotmail and Yahoo! mail, before allowing access to services. Any subset of these methods can be made the basis for "earning" a token.

Formally:

1. The token server chooses a security parameter k and generates an RSA key with modulus n such that n is k bits long, a public exponent e and a private exponent d, then publishes $\{n, e\}$. It also chooses a cryptographic hash function:

$$h : \{0, 1\}^* \to \{1..n - 1\}$$

2. The client generates two random integers, a token $r \in \{0, 1\}^*$ and blinding factor $b \in \{1..n - 1\}$. It sends the server:

$$b^e h(r)\ (mod\, n)$$

3. The server decides whether to issue a token to the client. In our implementation, this is accomplished by checking a list of IP addresses to see whether a token has already been issued to the client's IP address. The server terminates the protocol if it decides not to issue a token.

4. Otherwise, the server signs the token and returns this value to the client:

$$(b^e h(r))^d \equiv bh(r)^d\ (mod\, n)$$

5. The client unblinds the signed, blinded token to obtain the signed token $t = h(r)^d\ (mod\ n)$ using the multiplicative inverse of b:

$$t = h(r)^d \equiv b^{-1}bh(r)^d\ (mod\, n)$$

   Since the values $h(r)$ and $b^e h(r)\ (mod\ n)$ cannot be correlated due to the random choice of b, the transcript of the session cannot be used to identify the client when it later "spends" the token by revealing r and $h(r)$. To ensure that the server hasn't tried to "tag" the user with an invalid signature, the client verifies the signature to ensure that t really equals $h(r)^d\ (mod\, n)$:

$$t^e \equiv ?\ h(r)\ (mod\, n)$$

2.2 Certificate Authority

The CA's job is to accept an unblinded, signed token in exchange for signing a client certificate. The client can then load the client certificate into her browser and use it to access Nym-compatible services. Since in the simple case, the CA certifies only that it has received a valid token from the user, the traditional fields in a client certificate are unnecessary, and indeed pose a privacy risk to end users. Consequently, we provide a script which generates client certificate requests with a standard set of default answers. Thus, only serial numbers, key identifiers and validity dates distinguish client certificates. Ideally, validity dates would be quantized to common intervals to help avoid leaking details about the user's behavior, but as we point out below, making unusual changes to an X.509 certificate can be quite awkward in a system designed for simplicity.

It is not immediately obvious why a separate CA and certificate issuing step are worthwhile. Indeed, early versions of Nym assumed that the token server would sign a client certificate directly rather than issuing a token which later must be exchanged for a certificate. An anonymous contributor also pointed out that client certificates can be avoided entirely if web services accept tokens as an alternative to usernames and passwords.

We feel that using a token directly to login to a service is problematic for these reasons:

- Clients need a way to enter the token into login pages. This may require inconvenient frequent cutting and pasting
- If services neglect to use TLS for their login pages, attackers can sniff user tokens and impersonate their owners
- If multiple services make use of a single token server, service administrators can use their users' tokens to impersonate the users on other services

Having the token server sign a client certificate directly is better than using plain tokens, but has the following drawbacks:

- OpenSSL (and, we assume, other libraries) assumes that certificate requests will be examined and modified by the CA before signing. This is incompatible with our simple blind signature scheme, since the token server receives only a blinded value to sign. We found that duplicating the signing process to allow extracting the value to be signed, sending it to a token server, unblinding and reinserting it would require extensive use of the OpenSSL library and would have required more code and development time than the entire remainder of the Nym system.

- Since the token server cannot learn any details of the unblinded value without complicated zero knowledge proofs, clients could not easily be limited to obtaining signatures on well-formed client certificates. Malicious clients could trivially obtain signatures on all manner of values, including CA certificates, server certificates, and arbitrary other documents. Of course this is still true of our token-only server, however it is much more intuitive for users to understand that its signatures are only valid on Nym tokens to be used with our software, whereas a token server whose key is intended for use in signing client certificates is much more likely to be misunderstood by humans or software when malicious users present variations on that theme.
- When a separate CA is used via an anonymizing network, the CA can examine the client's certificate before signing, and ensure that only properly formed certificates are signed. CAs could thus accept multiple tokens in exchange for signatures, and indicate via X.509v3 extensions in the client certificate which tokens were used. For example, multiple users behind the single external IP of a NAT firewall could instead obtain certificates using tokens issued to unique email addresses. Or, users wishing to access a frequently vandalized site could be required to use an entire set of tokens issued on IP address, email address, one or more captchas plus a computational puzzle.

Formally, the CA operates as follows:

1. The client submits a token r and signature

   $$\sigma \equiv h(r)^d \; (mod\, n)$$

   along with an X.509 client certificate request.

2. The server checks that the token is valid by verifying that

   $$\sigma^e \equiv h(r) \; (mod\, n)$$

   and terminates the protocol otherwise.

3. The server also terminates the protocol if the token has already been used to obtain a signed certificate from the CA, or if the certificate to be signed is not of the correct form.

4. The server returns the signed certificate to the client.

2.3 Client-side Software

The client's behavior in obtaining a token and exchanging the token for a certificate is formally described in the previous two subsections. Once the client installs the certificate in her browser, TLS handles authentication and secrecy needs as she interacts with web services. Our implementation of the client-side requirements began as a set of Perl scripts and Unix shell scripts making heavy use of the command-line utility *openssl* included with the OpenSSL package. That implementation was quite straightforward to complete once we settled on using a separate CA, as described in the last subsection. However, since it required several uncommon Perl libraries and used Unix utilities, it was unlikely that most Internet users would be successful in using it.

With the help of Free pure-Javascript big integer and SHA-1 implementations, we succeeded in building a client which runs entirely in Javascript-enabled web browsers. A compact form of that client (omitting detailed user instructions) is given in its entirety in section A.1.

The primary drawback of the Javascript client is that we have found no easy way to generate an X.509 client certificate request in Javascript. Consequently, we modified the CA so that it generates a certificate if the client doesn't provide one. This is obviously a drawback to users who wish to ensure that their pseudonyms can never be impersonated, since the CA must generate the user's private key as part of the process. However, some users who may eventually use Nym, such as political dissidents, will be more concerned with deniability than authenticity, and will benefit from the ability to claim that the CA might have impersonated them. Note that encoding requirements are the difficulty, not key generation, and future work may include a certificate request template into which the Javascript code can insert public key data.

Recently we became aware that some browsers include browser-specific HTML extensions for generating and encoding keypairs and storing private keys until a certificate has been generated and signed by a CA. Future versions of our Javascript client may support these features, allowing users of that client to obtain certificates without exposing their private keys to the CA.

2.4 Internet services

It was surprisingly easy to make our first target application, the MediaWiki software used by the Wikipedia and many other websites, compatible with TLS client certificates. First, a TLS-enabled web server for use by Nym users must be configured to demand client certificates signed by the CA. In the case of Apache 1.3, this can be accomplished by adding just 3 lines to the httpd.conf configuration file (given in the appendix). When these lines are present, the web server rejects clients without valid certificates signed by the specified CA. When a valid user connects, the web server sets environment variables corresponding to information from the certificate. These variables can be used with an existing application's authentication mechanism to uniquely identify each user's certificate. If a user misbehaves, the application can place the user's certificate on a blacklist, and refuse further access to that user.

3 Threats

Like widely-used spam reduction approaches, Nym does not guarantee infallibility in preventing abuse, but rather offers a tradeoff between implementation cost and failure rate. Here we examine several credible risks to Nym's security from both the client and server perspectives.

3.1 Client Privacy and Reputation

Pseudonymity is a privacy tradeoff clients can choose in order to gain access to a service which is unwilling or unable to deal with fully anonymous abuse. Since servers will often want to limit availability of pseudonyms based on unique (or at least somewhat inconvenient to obtain) resources which can often be linked to users' real identities, users must choose to trust that Nym will keep those details unlinkable to the actions performed under their pseudonyms.

Nym is of course susceptible to the traditional risks of implementation bugs and cryptanalysis. Like Tor and other anonymity services, it is also susceptible to attacks which correlate agent behaviors over time. For example, consider a token server which has issued n tokens that can only be redeemed at a single CA. If the CA has issued n certificates, then clearly there must be no unredeemed tokens. If user $n+1$ obtains and redeems her token before user $n+2$ obtains a token, the token server and CA can collude to correlate that user's pseudonym with the information she used to obtain her token. Small anonymity sets are always a problem for privacy systems, and Nym is no exception.

Pseudonymity also allows users to build a reputation tied to their pseudonym. Malware, theft, use of untrustworthy client machines or carelessness could all result in a user's pseudonym being compromised, allowing the impostor to exploit or tarnish the user's reputation under that pseudonym, including the possibility of causing the pseudonym to be blacklisted. If the attacker knows the identity of the user, this also causes privacy problems for the user.

3.2 Abuse Limiting and Pseudonym Scarcity

Since Nym focuses on low barriers to entry and limited liability for end users, we presently offer very weak correspondence between people and pseudonyms. An attacker who can obtain multiple pseudonyms correspondingly multiplies the effort that service admins must exert to block his abuse. Given that many ISPs issue a different IP address each time a user connects, even unskilled abusers could potentially accumulate a significant number of pseudonyms in a system which limits only by IP. Likewise, as spammers have shown, email addresses from free services like hotmail.com are not hard to accumulate. This, ironically, makes free email providers themselves resort to mechanisms for limiting account creation, even as other services use them as just such a mechanism.

Why, then, does Nym bother with such flawed heuristics? For the same reason that sites like wikipedia.org and blogspot.com use them today: because they work well enough to keep abuse to a manageable level, without imposing enough on their clients' time and resources to make them want to go elsewhere. Of course, Nym would work just as well with notarized documents and government issued digital ID cards. But the demand, at least for the Internet services people use on a daily basis, seems to be for mechanisms which adequately suppress abuse without causing major inconvenience or risk for clients.

It should be noted that Nym can increase the ability of an abuser to cause damage by obtaining a large number of resources. For instance, Wikipedia vandals sometimes circumvent IP-based blocks by reconnecting to their ISP under a different IP address. If this goes on long enough, Wikipedia admins eventually block the ISP's entire range of IP addresses. If Wikipedia adopted Nym, the vandal might then try to obtain a Nym certificate for pseudonymous vandalism. In this case, he would fail if Wikipedia's token server was augmented to recognize that his IP is on Wikipedia's blacklist. However, a premeditating vandal could obtain pseudonyms for all the IPs available to him before starting his vandalism spree. If he obtained the certificates in quick succession, CA admins could notice the pattern and block the range of certificates just as Wikipedia admins currently block a range of offending IPs. But if he was patient enough to obtain his pseudonyms over time, he could conceivably then vandalize until he ran out of pseudonyms, without giving away any telltale signs that would allow his pseudonyms to be blocked en masse.

While we do consider this a drawback of Nym, remember that our target market is already making tradeoffs. Wikipedia wouldn't function at all, regardless of Nym, given enough sufficiently powerful and dedicated vandals. Fortunately, the vast majority of vandals on Wikipedia are deterred by a single block[1]. In such a landscape, Nym's disadvantage against dedicated vandals may well be worthwhile in exchange for its significant privacy offerings.

4 Barriers to Adoption

We observed four major barriers to adoption and/or implementation of security applications like Nym, and describe them in the following subsections. Our choice of a browser-based Javascript client aims to address the first point, while targeting low-liability sites like MediaWiki installations and blogs is an effort to make barriers to adoption (in particular, the liabilities involved with adopting new and unproven security systems) low enough for real users to participate. The two last items are places where existing standards and applications made it difficult to build and use new, simple privacy systems like Nym.

4.1 Software Installation

While this risks being too obvious, software is still difficult and risky to install. Users of proprietary software must trust the developers to avoid malicious code and inadvertent security flaws, and have to deal with install scripts which may not be tightly integrated with the rest of the OS. Free software OSes provide tighter install integration and source code, but there is far too much source code for users to examine, and installers generally work with precompiled binaries anyway. While Javascript, and to some extent Java, provide ubiquitous cross-platform environments with protected execution environments, quirky implementations and lack of library and language versatility make these platforms unattractive for many applications. We spent a significant portion of our development effort working around CPU-usage limits in our browser, resulting in a poorly kludged big number library.

4.2 Risks of Early Adoption

The users and administrators who most need cryptographic protection for their transactions are the least likely to become early adopters of new security applications. For example, while Nym could eventually allow political dissidents in highly repressive regimes to contribute pseudonymously to services like Wikipedia, and perhaps might be the greatest beneficiaries of such a system, we would hardly expect them to be the first in line for the beta test. Fortunately, since such services have many casual users, it is reasonable to assume that some will be willing to try out novel applications like Nym, especially considering its installation-free client and simple architecture, and that those users won't be overly upset if privacy-compromising bugs are later discovered. On the server side, by targeting generally fault-tolerant, low-sensitivity applications like personal blogs and community edited encyclopedia articles (where hundreds of vandalisms are already removed on a daily basis), Nym offers a low-risk proposition to service administrators, while potentially exposing its security offering to many thousands of real-world users.

Even when security is easy to adopt, though, as is the case for Nym with MediaWiki, service providers often have little motivation to offer increased privacy or even increased security to their users. Despite the ease of installing our 17-line patch, we considered writing a proxy which would accept Nym's TLS client certificates and then create a corresponding username like 'nymuser42' on the target MediaWiki site, so that any actions performed by that Nym user via the proxy would happen under the corresponding user account. Such a configuration would then require no effort at all on the part of MediaWiki admins; a helpful privacy enthusiast could simply set up the proxy, and could even monitor the target site and proactively block abusing Nym users to discourage the site admins from blocking the entire proxy when users misbehave. Our analysis indicates that at least in the case of MediaWiki, such a proxy would be complicated and error-prone to implement. However, it remains a possible avenue for users who benefit from privacy to themselves bear the costs of implementing it.

4.3 Standards Complexity

We found that while tools like OpenSSL are indispensible when dealing with cryptography standards, moving beyond or modifying traditional tasks like certificate request generation and signing almost immediately bogs down in complicated library calls. Several times we spent hours trying to work with X.509 internals, only to end up making tradeoffs in other areas to stay as far away from the internals as possible. This contrasts sharply with the mathematical details, which could often be implemented or modified with only a few lines of code.

[1] We make this observation as Wikipedia editors who have spent many hours patrolling the "recent changes" list and identifying vandalism

4.4 Application Support

While we found that modern browsers support client certificates remarkably well, they lack good privacy protection mechanisms with respect to those certificates. Users must be sure to use pseudonym certificates only when necessary, since doing so creates a cryptographically signed record which can be linked to other uses of the pseudonym. Since it is trivial to configure Apache, for instance, to request but not require client certificates from all users, Mozilla users bear the awkward burden of either remembering to disable client certificates when they're not needed, or authorizing use of the certificate for every page viewed on a target site. Browsers need to help users remember when certificates are in use, and make it easier to toggle their use on and off or select certificates for use only with particular sites.

5 Future Work

The most obvious features to add to Nym are more refined mechanisms for limiting pseudonym abuse in meaningful ways. IP-based rationing has a number of obvious problems, and clearly there is not a close correspondence between IP addresses and people. Email address rationing would be quite simple to implement, and applications such as Hashcash and captcha generators may also be easy to integrate with Nym. Token servers could even be configured to require monetary payments in exchange for particular tokens. Of course, a CA could issue certificates with no tokens at all, in which case it would be a simple pseudonymity system with no limiting. Token servers and CAs could also accept certificates issued earlier as a basis for new token or certificate issuing, allowing more nuanced ways of limiting issued pseudonyms.

If Nym becomes popular, with multiple token servers and CAs issuing a varied range of certifications, web services may eventually want to express allowable combinations of tokens using a complex policy. For instance, a blogging site might require the client to have either a certificate from a CA which limits one certificate per IP address, or, in the case that an another user of a shared IP address has earlier obtained a certificate for the user's address, allow the user to present two certificates, one obtained in exchange for a Hashcash token, the other obtained based on proof of ownership of an email address. In such cases, existing work in trust negotiation and advanced credential systems could be quite useful.

The careful reader will have observed by now that we have described no centralized online "double spending" database to prevent users from using a token with multiple CAs to obtain more than one certificate. Since our initial implementation was aimed at users of a single anonymizing network (Tor) using a single popular application (MediaWiki), we assumed that administration of the token server, CA and web server would be coordinated. In retrospect, it seems likely that other services may wish to do the same, each running their own independent and unrelated servers, in which case a central database will be unnecessary. While this reduces the number of users in each anonymity set, it fits better with the assumption that low barrier to entry is a higher de facto priority than security and privacy. Rather than writing large amounts of code to address possible scenarios, our intent is to wait and see where (and if) the demand actually occurs. It may turn out that services and users will want separate CAs for each service, with a centralized token server, in which case features like token database expiration (in which users can obtain new tokens after a certain amount of time) may be desirable. Or, a single CA may emerge which accepts tokens from multiple independent token servers and issues certificates which indicate what tokens were used to obtain a certificate.

If users of our Javascript client trust CAs to generate their private keys, they may also be interested in a convenience feature wherein they maintain a username and password and a copy of the public and private parts of their certificate with the CA, allowing them to recover their certificate should they lose it. On the other hand, we may also be able to support built-in browser functionality supporting keypair generation, so that future versions of our Javascript client won't allow the CA to see client private keys.

Unlinkability is another interesting issue to consider. In our current implementation, users are issued a client certificate which they use for all their pseudonymous transactions, yet there exist anonymous credential systems that can make transactions unlinkable while still allowing credentials to be revoked if users misbehave [5]. A low-tech alternative might be to make the CA issue a set of certificates to each user, valid for successive time intervals in the future. The CA records the sequence of certificates, deleting certificates from the sequence when they expire. If a user misbehaves, service admins can request that the CA revoke all future certificates for that user. If the CA is compromised, attackers will not be able to link previous user transactions, since expired certificates will have been deleted.

6 Conclusion

In 1999, Whitten and Tygar [6] pointed out that regardless of infrastructure and implementation quality, simple barriers to entry such as a steep learning curve or confusing graphical interface can make a system fail in practice. While Nym doesn't seek to push the boundaries of achievable privacy and offers no new cryptographic constructions, it does seek to make it very simple for web users and service providers to implement pseudonymity solutions with abuse-prevention mechanisms on par with current best practices.

It is too soon to tell whether Nym will be widely adopted (and due to the nature of Free software and Tor, we may never learn about many applications of Nym). Nevertheless, we hope that our experiences will encourage others to build low-barrier systems with real potential for adoption, and thus build stepping stones to the more powerful systems which so often go unused. In particular, we endorse the following principles:

1. Minimize the barriers to adoption for both users and service administrators

2. Design systems that address the needs of existing, popular services

3. As much as possible, keep designs small and simple to facilitate code/design audits and quickly establish the design's trustworthiness

We applaud the recent work which has addressed mitigating phishing attacks, helped users manage the multitude of passwords required to access popular web sites, and helped users better understand the nature of the protection offered by TLS-enabled web services. More work can still be done in these areas, as well as in protecting user privacy in the presence of embedded code such as Javascript and ActiveX controls, cookies and "web bugs", and federated identity management. Social networking and seller feedback tools are also becoming popular, yet often neglect their academic forebears in reputation systems and webs of trust.

To summarize, we presented a very simple system for abuse-resistant pseudonymity, designed to meet practical needs of a wide base of users while minimizing the barriers to adoption which so often seem to hinder the adoption of security solutions. We showed that our system integrates smoothly with very common Internet applications. We showed how the system can be expanded to meet future needs, and pointed out which aspects of our implementation were made the most difficult by the available software and standards. We hope that our work will play a part in bridging the gap between security theory and practice.

References

[1] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. The MIT Press, 2000.

[2] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 319–327, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[3] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[4] S. Garfinkel. Email-based identification and authentication: An alternative to pki? *IEEE Security & Privacy*, pages 20–26, November/December 2003.

[5] S. G. Stubblebine, P. F. Syverson, and D. M. Goldschlag. Unlinkable serial transactions: protocols and applications. *ACM Transactions on Information and System Security*, 2(4):354–389, 1999.

[6] A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *8th USENIX Security Symposium*, 1999.

[7] M. Winslett, T. Yu, K. Seamons, A. Hess, R. Jarvis, B. Smith, and L. Yu. Negotiating Trust on the Web. *IEEE Internet Computing Special Issue on Trust Management*, 6(6):30–37, November/December 2002.

A Appendix: Source Code

The following three lines can be added to an Apache 1.3 httpd.conf configuration file to require client certificates from incoming users:

```
SLVerifyClient 2
SSLCACertificateFile /path/to/cacert.pem
SSLVerifyDepth 1
```

The following addition to MediaWiki then allows administrators to blacklist certificates of clients who misbehave. MediaWiki has an existing interface for banning users based on their incoming IP addresses, and we thought it best to leverage this interface where possible. MediaWiki has a variable which stores client IP addresses, and we found that it was easy to replace the actual incoming IP address (which for a pseudonymous user would typically be that of a Tor exit node) with arbitrary strings such as “pseudonymous_user_05” for a user presenting a client certificate with serial number 5. This string would then be logged in place of the client’s IP address. This has the advantage that such identifiers aren’t easily confused with numeric IP addresses when administrators examine audit logs, but we found that such strings confused the IP address banning code. Consequently, we chose a tradeoff in which we assume that most MediaWiki installations will not use the reserved “10.0.0.0” network, and translate client certificate serial numbers to strings which look like IP addresses in that network, rather than risking obscure bugs in other parts of the system that could arise by using freeform strings. Administrators can then be informed that when a pseudonymous user misbehaves, the audit trail kept by MediaWiki will list a 10.* address instead of the user’s actual IP address, and that such users’ certificates can be blocked using that pseudo-IP, just as a non-pseudonymous user would be blocked by their actual IP address.

This tradeoff is clearly documented, and since our patch constitutes only 17 lines of PHP code, skilled administrators wishing to use our feature on a network in which clients actually come from the reserved “10.0.0.0” network can easily modify the code to map to other reserved networks instead. Such hacks are certainly debatable from a software development and maintainability standpoint, but we wanted to maintain our design goal of simplicity in this case.

```
global $wgMapClientCertToIP;

if ( $wgMapClientCertToIP && isset( $_SERVER['SSL_CLIENT_M_SERIAL'] ) ) {
  # This is a little classier, but requires more codebase changes
  # and might cause more problems
  # $ip = 'anonuser.' . $_SERVER['SSL_CLIENT_M_SERIAL'];

  # This, on the other hand, is almost guaranteed to work, but could
  # cause problems for people using the 10.*.*.* private IP range
  $s = $_SERVER['SSL_CLIENT_M_SERIAL'];

  if($s >= (2<<24)){ die('Client certificate ID too large(!)'); }
  $o1 = ($s>>16);
  $o2 = ($s>>8) & 255;
  $o3 = $s & 255;
  $ip = '10.' . $o1 .'.'. $o2 .'.'. $o3;
}
```

A.1 Javascript client

This code implements the Nym client code entirely in Javascript. Due to the vagaries of Javascript implementation, Mozilla browsers (and likely others) halt Javascript programs which run for more than a few seconds continuously, and require the user to click a button for each 5 seconds of further computation. Since BigInt.js implements modular inverses inefficiently, calculating the inverse of the blinding factor takes several minutes. So to maintain user sanity, we had to hack BigInt.js to use Javascript’s callback mechanism to return control to the browser every few seconds. Other than that, BigInt.js was used unchanged.

Please note that this implementation was not designed to be especially small, and we were pleasantly surprised to discover that it was in fact small enough to include in this paper. We also haven’t yet had time to refine it for readability; this is normal, workhorse code. If anything, it reinforces our experience that simple security applications need not be big or complicated to be useful. The strings __MODULUS__ and __PUBEXP__ should be replaced with the (base 10) modulus and public exponent of the desired token server before use.

```
<html><head><title>nym-Javascript</title></head> <body>

<script src="nymBigInt.js"></script> <script src="sha1.js"></script>
<script>
function randBigInt(b) {
    var ans = int2bigInt(0,b,0);
    randBigInt_(ans, b, 1); return ans; }

var r, hr, b, binv, be, behr, modulus;
var pubexpint  = __PUBEXP__;
var pubexp, keysize=2048;

function tokenmaker() {
    var randint = bigInt2str(randBigInt(160), 10);
    modulus = str2bigInt(document.a.modulus.value,10,1,1);
    if(document.a.randomtext.value.length < 200) {
        document.a.randomtext.value =
    "Please type at least 200 random characters then try again.";
        return; }
    r = sha1mod("0"+document.a.randomtext.value+randint,keysize,modulus);
    r = bigInt2str(r, 10);
    document.a.token.value=document.c.r.value=r; }

function sha1mod(m,len,modulus) {
    var expanded;
    var i=1;
    expanded = str2bigInt(sha1expand(m+i,len), 16, 1, 1);
    while(greater(expanded, modulus)) {
        expanded = str2bigInt(sha1expand(m+i,len), 16, 1, 1);
        i++;
    }
    return expanded;
}

function sha1expand(m,len) {
    var i=1, digest=’’;
    if(len % 8 != 0) {
    document.write("Error in sha1expand! Length must be a multiple of 8.");
        return 0; }

    len = Math.floor(len/8);
    while(digest.length < 2*len) { digest += hex_sha1(’’+i+m); i++; }
    return digest.substring(0,2*len); }

function hashtoken() {
    modulus = str2bigInt(document.a.modulus.value,10,1,1);
    hr = sha1mod(document.a.token.value, keysize, modulus);
    document.a.hashed.value=bigInt2str(hr, 10); }

function blindmaker() {
    var randint = bigInt2str(randBigInt(160), 10);
    b = sha1mod("1"+document.a.randomtext.value+randint,keysize, modulus);
    document.a.blind.value=bigInt2str(b,10); }

function blinder1() {
    noblockinverseMod(dup(b), modulus); }

function blinder2() {
    if(!inversemoddone || !inversemodret) {
        document.write("Mod inverse not done yet."); return 0; }
    binv                     = getInverseModResult();
    document.a.binverse.value= bigInt2str(binv,10);
    pubexp                   = int2bigInt(pubexpint, 1, 1);
    be                       = powMod(b, pubexp, modulus);
    behr                     = multMod(be,hr, modulus);
    document.a.be.value      = bigInt2str(be,10);
    document.a.pubexp.value  = bigInt2str(pubexp,10);
    document.a.blinded.value = bigInt2str(behr,10);
    document.b.tosign.value  = bigInt2str(behr,10); }

function unblind() {
    var sbstr     = document.b.signedblinded.value;
    var sb        = str2bigInt(sbstr,10,1,1);
    var binverse  = str2bigInt(document.a.binverse.value,10,1,1);
    modulus       = str2bigInt(document.a.modulus.value,10,1,1);
    var unblinded = multMod(binverse,sb, modulus);
    document.c.signed.value =bigInt2str(unblinded,10);
    pubexp        = int2bigInt(pubexpint, 1, 1);
    var check     = powMod(unblinded, pubexp, modulus);
    document.a.check.value  =bigInt2str(check,10);
        if(document.a.check.value != document.a.hashed.value) {
                document.write("Server signature check failed."); } }

function genparams() { tokenmaker(); hashtoken(); blindmaker(); }
</script>

<h1>Javascript nym interface</h1>
<form name="a"> <input name="token" type=hidden> <input name="hashed" type=hidden> <input name="blind" type=hidden>
<input name="be" type=hidden> <input name="binverse" type=hidden> <input name="blinded" type=hidden>
<input name="modulus" value="__MODULUS__" type=hidden>
<input name="pubexp" type=hidden> <input name="check" type=hidden>
<hr>

<textarea name="randomtext" cols=50 rows=5>
Please type enough gibberish text to fill this box.
</textarea> <br>

<input name="tokenmakerb" value="Setup" onclick="genparams(); blinder1();" type="button">
Progress: <input size=60 name="progress" value="*"> <p>
</form> <p>

Disable tor before proceeding with this step.  Once you click "Obtain
Token", copy the numbers you see, then return and paste them into the
"Signed:" field.

<form name="b"
action="http://erg.no-ip.org/cgi-bin/nyms/token_cgi/signer.cgi" method=POST>
<input name="tosign" type=hidden> <input type=button value="Obtain Token" onclick="blinder2(); document.b.submit()">
Signed:<input name="signedblinded"><br>
</form><p>

Next, reenable tor and wait a few hours.

<form name="c" method=POST
action="http://erg.no-ip.org/cgi-bin/nyms/ca_cgi/signcert.cgi">
<input type=hidden name="r"><input name="signed" type=hidden>
<input type=button value="Exchange token for certificate" onclick="unblind(); document.c.submit()"> <p>

After installing the certificate, you can visit our example site, the
<a href="https://erg.no-ip.org/wiki/index.php/Main_Page">Nymwiki</a>.

</form> </body> </html>
```