Nymble: Anonymous IP-Address Blocking*

Peter C. Johnson[1], Apu Kapadia[1,2], Patrick P. Tsang[1], and Sean W. Smith[1]

[1] Department of Computer Science
Dartmouth College
Hanover, NH 03755, USA
[2] Institute for Security Technology Studies
Dartmouth College
Hanover, NH 03755, USA
{pete,akapadia,patrick,sws}@cs.dartmouth.edu

Abstract. Anonymizing networks such as Tor allow users to access Internet services privately using a series of routers to hide the client's IP address from the server. Tor's success, however, has been limited by users employing this anonymity for abusive purposes, such as defacing Wikipedia. Website administrators rely on IP-address blocking for disabling access to misbehaving users, but this is not practical if the abuser routes through Tor. As a result, administrators block *all* Tor exit nodes, denying anonymous access to honest and dishonest users alike. To address this problem, we present a system in which (1) honest users remain anonymous and their requests unlinkable; (2) a server can complain about a particular anonymous user and gain the ability to blacklist the user for future connections; (3) this blacklisted user's accesses before the complaint remain anonymous; and (4) users are aware of their blacklist status before accessing a service. As a result of these properties, our system is agnostic to different servers' definitions of misbehavior.

1 Introduction

Anonymizing networks such as Crowds [25] and Tor [15] route traffic through independent nodes in separate administrative domains to hide the originating IP address. Unfortunately, misuse has limited the acceptance of deployed anonymizing networks. The anonymity provided by such networks prevents website administrators from blacklisting individual malicious users' IP addresses; to thwart further abuse, they blacklist the *entire* anonymizing network. Such measures eliminate malicious activity through anonymizing networks at the cost of denying anonymous access to honest users. In other words, a few "bad apples" can spoil the fun for all. (This has happened repeatedly with Tor.[3])

Some approaches for blacklisting abusive users are based on pseudonyms [11,13,14,19]. In these systems, of which Nym [17] seems most relevant, users are required to log into

* This research was supported in part by the NSF, under grant CNS-0524695, and the Bureau of Justice Assistance, under grant 2005-DD-BX-1091. The views and conclusions do not necessarily reflect the views of the sponsors.

[3] The *Abuse FAQ for Tor Server Operators* lists several such examples at http://tor.eff.org/faq-abuse.html.en.

websites using an assigned pseudonym, thus assuring a level of accountability. Unfortunately, this approach results in *pseudonymity* for *all* users—ideally, honest users should enjoy full anonymity, and misbehaving users should be blocked.

To this end, we present a secure system in which users acquire an ordered collection of *nymbles*, a special type of pseudonym, to connect to websites. Without additional data, these nymbles are computationally hard to link, and hence using the stream of nymbles simulates anonymous access to services. Websites, however, can blacklist users by obtaining a *trapdoor* for a particular nymble, allowing them to link future nymbles from the same user—those used before the complaint remain unlinkable. Servers can therefore blacklist anonymous users without knowledge of their IP addresses while allowing honest users to connect anonymously. Our system ensures that users are aware of their blacklist status before they present a nymble, and disconnect immediately if they are blacklisted. Furthermore, websites avoid the problem of having to prove misbehavior: they are free to establish their own independent blacklisting policies. Although our work applies to anonymizing networks in general, we consider Tor for purposes of exposition. In fact, any number of anonymizing networks can rely on the same nymble system, blacklisting anonymous users regardless of their anonymizing network(s) of choice.

Our research makes the following contributions:

- **Blacklisting anonymous users.** We provide a means by which servers can blacklist users of an anonymizing network without deanonymizing them. Honest users enjoy anonymous access and are unaffected by the misbehavior of other users.
- **Practical performance.** A system such as ours, relying on a server to issue nymbles, will be adopted only if performance is acceptable. Our protocol minimizes storage requirements and the use of expensive asymmetric cryptographic operations.
- **Prototype implementation.** With the goal of contributing a workable system, we have built a prototype implementation. We provide performance statistics to show that our system is indeed a viable approach for selectively blocking users of large-scale anonymizing networks such as Tor.

Many in the community worry that "deanonymization" will become a vehicle for suppressing individuals' rights. This project moves in the other direction, by allowing websites to block users without knowing their identities, hopefully increasing mainstream acceptance of anonymizing technologies such as Tor.

2 Related Work

Anonymous credential systems such as Camenisch and Lysyanskaya's [7,8] use group signatures for anonymous authentication, wherein individual users are anonymous among a group of registered users. Non-revocable group signatures such as Ring signatures [26] provide no accountability and thus do not satisfy our needs to protect servers from misbehaving users. Basic group signatures [1,2,3,12] allow revocation of anonymity by no one except the *group manager*. As only the group manager can revoke a user's anonymity, servers have no way of linking signatures to previous ones and must query the group manager for every signature; this lack of scalability makes it unsuitable for our goals. *Traceable signatures* [18,30] allow the group manager to release a trapdoor that allows *all* signatures generated by a particular user to be traced; such an approach does not provide the

backward anonymity that we desire, where a user's accesses before the complaint remain anonymous. Specifically, if the server is interested in blocking only future accesses of bad users, then such reduction of user anonymity is unnecessarily drastic. When a user makes an anonymous connection the connection should *remain* anonymous. And misbehaving users should be blocked from making further connections after a complaint.

In some systems, misbehavior can be defined precisely. For instance, double-spending of an "e-coin" is considered misbehavior in anonymous electronic cash systems [4,10]. Likewise, compact e-cash [6], k-times anonymous authentication [28] and periodic n-times anonymous authentication [5] deem a user to be misbehaving if she authenticates "too many" times. In these cases, convincing evidence of misbehavior is easily collected and fair judgment of misbehavior can be ensured. While such approaches can encourage certain kinds of fair behavior in anonymizing networks (e.g., e-coins can be used to control bandwidth consumption of anonymous users), it is difficult to map more complex notions of misbehavior onto "double spending" or related approaches. It may be difficult to precisely define what it means to "deface a webpage" and for Wikipedia to *prove* to a trusted party that a particular webpage was defaced. How can the user be sure these "proofs" are accurate and fairly judged? Can we avoid the problem of judging misbehavior entirely? In this paper we answer affirmatively by proposing a system that does not require proof of misbehavior. Websites may complain about users for any reason; our system ensures users are informed of complaints against them, thus "making everybody happy"—except, of course, the misbehaving users, who remain anonymous but are denied access.

Syverson et al. [27] provide a solution to a closely-related problem. To facilitate anonymous and unlinkable transactions, users are issued a blind signature for access to a service. This blind signature can be renewed with another blind signature (for the subsequent connection) each time the user has been served. If a user misbehaves, a server can terminate its service to that user by not renewing that user's blind signature. As a result, misbehavior must be detected *during* the user's connection. In contrast, our system targets scenarios in which misbehavior is detected *after* the user has disconnected.

A preliminary *work-in-progress* version of this paper (suggesting the use of trusted hardware) was presented at the Second Workshop on Advances in Trusted Computing [29].

3 System overview

Resource-based blocking. Our system provides servers with a means to block misbehaving users of an anonymizing network. Blocking a particular *user*, however, is a formidable task since that user can acquire several identities—the *Sybil* attack is well known [16] in this regard. Our system, therefore, focuses on blocking *resources* that are (usually) controlled by a single user. In this paper, we focus on IP addresses as the resource, but our scheme generalizes to other resources such as identity certificates, trusted hardware, and so on. Our system ensures that *nymbles* are bound to a particular resource, and servers can block nymbles for that resource. We note that if two users can prove access to the same resource (e.g., if an IP address is reassigned to another user), they will obtain the same stream of nymbles. Since we focus on IP address blocking, in the remainder of the paper, the reader should be aware that blocking "a user" really means blocking that user's IP address (although, as mentioned before, other resources may be used). We will address the practical issues related with IP-address blocking in Section 7.

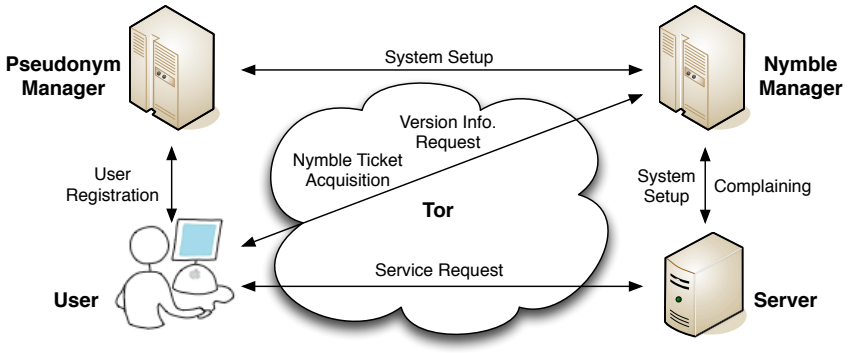

Fig. 1. System Architecture

Pseudonym Manager. The user must first contact the *Pseudonym Manager (PM)* and demonstrate control over a resource; for IP-address blocking, a user is required to connect to the PM directly (i.e., not through a known anonymizing network), as shown in Figure 1. We assume the PM has knowledge about Tor routers, for example, and can ensure that users are communicating with it directly.[4] Pseudonyms are deterministically chosen based on the controlled resource, ensuring that the same pseudonym is always issued for the same resource.

Note that the user *does not* disclose what server he or she intends to connect to, and therefore the user's connections are anonymous to the PM. The PM's duties are limited to mapping IP addresses (or other resources) to pseudonyms.

Nymble Manager. After obtaining a pseudonym from the PM, the user connects to the *Nymble Manager (NM)* through the anonymizing network, and requests nymbles for access to a particular server (such as Wikipedia). Nymbles are generated using the user's pseudonym and the server's identity. The user's connections, therefore, are pseudonymous to the NM (as long as the PM and the NM do not collude) since the NM knows only the pseudonym-server pair, and the PM knows only the IP address-pseudonym pair. Note that due to the pseudonym assignment by the PM, nymbles are bound to the user's IP address and the server's identity. To improve the collusion resistance of our system, the PM's duties can be split between n different PMs, which behave like Mixes [9]. As long as at least one of the Mix nodes is honest, the user's connections will be pseudonymous to the NM and anonymous to the PM (or PMs). For the purposes of clarity, we assume a single PM.

To provide the requisite cryptographic protection and security properties, the NM encapsulates nymbles within *nymble tickets*, and trapdoors within *linking tokens*. Therefore, we will speak of linking tokens being used to link future nymble tickets. The importance of these constructs will become apparent as we proceed.

[4] Note that if a user connects through an unknown anonymizing network or proxy, the security of our system is no worse than that provided by real IP-address blocking, where the user could have used an anonymizing network unknown to the server.

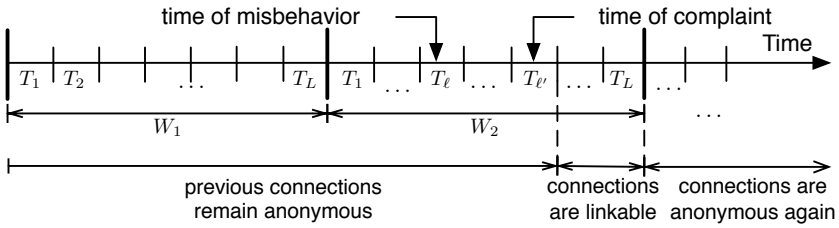

Fig. 2. The life cycle of a misbehaving user in our system

As illustrated in Figure 2, in our system, time is divided into linkability windows of duration $\mathcal{W}$, each of which is split into smaller time periods of duration $\mathcal{T}$, where the number of time periods in a linkability window $L = \frac{\mathcal{W}}{\mathcal{T}}$ is an integer. We will refer to time periods and linkability windows chronologically as $T_1, T_2, \ldots, T_L$ and W_1, W_2, ... respectively. While a user's access *within* a time period is tied to a single nymble ticket, the use of different nymble tickets *across* time periods grants the user anonymity between time periods—smaller time periods provide users with enough nymble tickets to simulate anonymous access. For example, $\mathcal{T}$ could be set to 5 minutes, and $\mathcal{W}$ to 1 day. The linkability window serves two purposes—it allows for *dynamism* since IP addresses can get reassigned to different well-behaved users, making it undesirable to blacklist an IP address indefinitely, and it ensures *forgiveness* of misbehavior after a certain period of time. We will discuss the choice of these parameters in Section 7.

Blacklisting a user. If a user misbehaves, the website may link any future connection from this user within the current linkability window (e.g., the same day). Consider Figure 2 as an example: A user misbehaves in a connection to a website during time period T_ℓ within linkability window W_2. The website detects the misbehavior and complains in time period $T_{\ell'}$ by presenting to the NM the nymble ticket associated with the misbehaving user and obtaining a linking token therefrom. The website is then able to link future connections by the user in time periods $T_{\ell'+1}, T_{\ell'+2}, \ldots, T_L$ of linkability window W_2. Therefore, users are blacklisted for the rest of the day (the linkability window) once the website has complained about that user. Note that the user's connections in $T_{\ell+1}, \ldots, T_{\ell'}$ remain unlinkable. This property ensures that a user's previous accesses remain anonymous, and allows our system to avoid judging misbehavior. We now describe how users are notified of their blacklisting status.

Notifying the user of blacklist status. Users who make use of Tor expect their connections to be anonymous. If a server obtains a linking token for that user, however, it can link that user's subsequent connections (we emphasize that the user's previous connections remain anonymous). It is of utmost importance, then, that users be notified of their blacklisting status before they present a nymble ticket to a server. In our system, the user can download the server's blacklist and verify whether she is on the blacklist. If so, the user disconnects immediately (the server learns that "some blacklisted user" attempted a connection). Since the blacklist is cryptographically signed by the NM, the authenticity of the blacklist is easily verified. Furthermore, the NM provides users with "blacklist version numbers" so that

the user can also verify the freshness of the blacklists. We ensure that race conditions are not possible in verifying the freshness of a blacklist. Our system therefore makes "everybody happy"—honest users can enjoy anonymity through Tor, servers can blacklist the anonymous users of their choice, and users can check whether they have been blacklisted before presenting their nymble ticket to the server. If blacklisted, the user does not present the nymble ticket, and disconnects.

4 The Nymble Authentication Module

In this section we present the *Nymble Authentication Module* (Nymble-Auth), our cryptographic construction that centers on the services provided by the *Nymble Manager (NM)*. Nymble-Auth allows users to authenticate to servers in a manner that both preserves the privacy of honest users and protects servers from misbehaving users. Nymble-Auth thus serves as the fundamental building block in our NYMBLE system. For simplicity, in this section we assume that users contact the NM directly. In the next section, we describe the entire NYMBLE system, which also includes the *Pseudonym Manager (PM)*.

4.1 The Model

Syntax. Nymble-Auth uses a semi-trusted third party, the *Nymble Manager (NM)*, to issue *nymble tickets* to users to authenticate themselves to servers. More specifically, Nymble-Auth consists of three entities: the NM, a set of *users*, and a set of *servers*, and a tuple of (possibly probabilistic) polynomial-time algorithms: Setup, NymbleTktGen, LinkingTknExt, ServerVerify, NMVerify and Link.

To initialize Nymble-Auth, the NM invokes Setup to initialize the system. Upon receiving a request from a user, the NM executes NymbleTktGen to generate a nymble ticket for that user. The user can obtain service at a server by presenting a valid nymble ticket for that server. Upon request from a server, the NM executes LinkingTknExt to extract a *linking token* from a valid nymble ticket of some user for linking future nymble tickets of the same user. The NM and the servers may run NMVerify and ServerVerify respectively to check the validity of a nymble ticket. Finally, servers may run Link to test if a user's nymble ticket is linked to a linking token.

Security Notions. Roughly speaking, a secure Nymble-Auth must satisfy the following security properties (we will formalize these properties in Appendix A.1):

1. *Nymble tickets are Unforgeable.* As a result, they can be obtained only from the NM. Valid nymble tickets serve as legitimate authenticators issued by the NM for users to authenticate to servers.
2. *Nymble tickets are Uncircumventably Forward-Linkable.* Once a linking token is issued for a user/server/linkability-window tuple, all future nymble tickets for that tuple are linkable. This property allows for blacklisting misbehaving users.
3. *Nymble tickets are Backward Anonymous.* Without a linking token, nymble tickets are all anonymous and thus unlinkable. Given a linking token for a user/server/linkability-window tuple, previous nymble tickets are still anonymous, and so are nymble tickets of the same user for other servers and other linkability windows. This property ensures that all accesses by a user before the time of complaint remain anonymous.

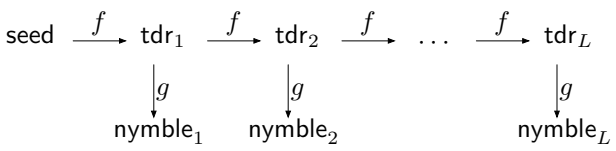

Fig. 3. Evolution of trapdoors and nymbles

Communication Channels and Time Synchronization. There are different security requirements for the communication channels between the entities in various protocols. A channel may be confidential and/or authenticated at one or both ends. Such a channel may be realized using SSL/TLS over HTTP under a PKI such as X.509. We call a channel secure if it is both confidential and mutually-authenticated. A channel may also be anonymous, in which case the communication happens through Tor. We emphasize that while the NM, the PM, and the servers must set up PKI key-pairs, our system does *not* require users to have PKI key-pairs.

All entities are assumed to share a synchronized clock. The requirement of the granularity of the clock depends on the application in question. We will have more discussion on how we ensure time synchronization and its consequences in Section 6.

4.2 Our Construction

Overview. At its core, Nymble-Auth leverages a hash-chain-like structure for establishing the relationship between nymbles and trapdoors. The same structure was used by Ohkubo et al. [24] for securing RFID tags by ensuring both indistinguishability and forward security of the tags. Although the primitive we use in this paper shares similarities with that in [24], our construction possesses different security requirements that must be satisfied to secure our system as a whole. In particular, the hash structure in Ohkubo et al. [24] satisfies *Indistinguishability* and *Forward Security*, both of which are captured by our security notion of *Backward Anonymity*. We formalize the *Unforgeability* requirement for Nymble-Auth, which assumes a different trust model from that in [24]. Finally, we also introduce a unique security requirement called *Uncircumventable Forward Linkability*.

As shown in Figure 3, trapdoors evolve throughout a linkability window using a *trapdoor-evolution function* f. Specifically, the trapdoor for the next time period can be computed by applying f to the trapdoor for the current time period. A nymble is evaluated by applying the *nymble-evaluation function* g to its corresponding trapdoor. We will instantiate both f and g with collision-resistant cryptographic hash functions in our construction. In essence, it is easy to compute future nymbles starting from a particular trapdoor by applying f and g appropriately, but infeasible to compute nymbles otherwise. Without a trapdoor, the sequence of nymbles appears unlinkable, and honest users can enjoy anonymity. Even when a trapdoor for a particular time period is obtained, all the nymbles prior to that time period remain unlinkable because it is infeasible to invert f and g. The NM *seeds* the sequence of trapdoors (and hence the nymbles) with its secret, the user's ID, the server's ID and the linkability window's ID of the requested connection. Seeds are therefore specific to source-destination-window combinations. As a consequence, a trapdoor is useful only for a particular website to link a particular user (or more specifically an IP address) during a particular linkability window.

Parameters. Let $\lambda \in \mathbb{N}$ be a sufficiently large security parameter. Let f and g be secure cryptographic hash functions, H be a secure keyed hash, HMAC be a secure keyed-hash message authentication code (HMAC), and Enc be a secure symmetric encryption, such that their security parameters are polynomial in λ. Let $|S|$ denote the number of servers.

Protocol Details. Now we detail each protocol in Nymble-Auth.

- $(nmsk, (hmk_{NS_1}, \ldots, hmk_{NS_{|S|}})) \leftarrow \mathsf{Setup}(1^\lambda)$.
  To set up the system, the NM picks, all uniformly at random from their respective key-spaces,
  1. a key khk_N for keyed hash function H,
  2. a key sek_N for secure symmetric encryption Enc, and
  3. $|S|+1$ keys hmk_N and $hmk_{NS_1}, hmk_{NS_2}, \ldots, hmk_{NS_{|S|}}$ for HMAC,

  and sets its secret key $nmsk$ as $(khk_N, sek_N, hmk_N, hmk_{NS_1}, \ldots, hmk_{NS_{|S|}})$. The NM stores $nmsk$ privately and, for each server S_j, sends hmk_{NS_j} to S_j through a secure channel. Each S_j then stores its secret key ssk_j as (hmk_{NS_j}) privately.
- $\mathtt{nymbleTKT} \leftarrow \mathsf{NymbleTktGen}_{nmsk}(\mathtt{id}, j, k, \ell)$.
  To generate a nymble ticket that allows a user with identity id to authenticate to server S_j during time period T_ℓ of linkability window W_k, the NM computes the following using its secret key $nmsk$:
  1. $\mathtt{seed} \leftarrow H_{khk_N}(\mathtt{id}, j, k)$, the seed for trapdoor evolution,
  2. $\mathtt{tdr} \leftarrow f^{(\ell)}(\mathtt{seed})$, the trapdoor for T_ℓ,
  3. $\underline{\mathtt{nymble}} \leftarrow g(\mathtt{tdr})$, the nymble for the same time period,
  4. $\mathtt{tdr}||\mathtt{id} \leftarrow \mathsf{Enc.encrypt}_{sek_N}(\mathtt{tdr}||\mathtt{id})$, a ciphertext that only the NM can decrypt,
  5. $\mathtt{mac}^N \leftarrow \mathsf{HMAC}_{hmk_N}(j||k||\ell||\mathtt{nymble}||\overline{\mathtt{tdr}||\mathtt{id}})$, the HMAC for the NM,
  6. $\mathtt{mac}^{NS} \leftarrow \mathsf{HMAC}_{hmk_{NS_j}}(j||k||\ell||\mathtt{nymble}||\overline{\mathtt{tdr}||\mathtt{id}}||\mathtt{mac}^N)$, the HMAC for S_j.

  Finally the NM returns nymbleTKT as $\langle j, k, \ell, \mathtt{nymble}, \overline{\mathtt{tdr}||\mathtt{id}}, \mathtt{mac}^N, \mathtt{mac}^{NS}\rangle$.
- $\mathtt{valid}/\mathtt{invalid} \leftarrow \mathsf{ServerVerify}_{ssk_j}(k, \ell, \mathtt{nymbleTKT})$.
  To verify if a nymble ticket $\mathtt{nymbleTKT} = \langle j', k', \ell', \mathtt{nymble}, \overline{\mathtt{tdr}||\mathtt{id}}, \mathtt{mac}^N, \mathtt{mac}^{NS}\rangle$ is valid for authenticating to server S_j at time period T_ℓ during linkability window W_k, S_j does the following using its key ssk_j:
  1. return invalid if $(j, k, \ell) \neq (j', k', \ell')$, or $\mathsf{HMAC}_{hmk_{NS_j}}(j'||k'||\ell'||\mathtt{nymble}||\overline{\mathtt{tdr}||\mathtt{id}}||\mathtt{mac}^N) \neq \mathtt{mac}^{NS}$,
  2. return valid otherwise.
- $\mathtt{valid}/\mathtt{invalid} \leftarrow \mathsf{NMVerify}_{nmsk}(j, k, \ell, \mathtt{nymbleTKT})$.
  To verify if a nymble ticket $\mathtt{nymbleTKT} = \langle j', k', \ell', \mathtt{nymble}, \overline{\mathtt{tdr}||\mathtt{id}}, \mathtt{mac}^N, \cdot\rangle$ is valid for authenticating to server S_j at time period T_ℓ during linkability window W_k, the NM does the following using its key $nmsk$:
  1. return invalid if $(j, k, \ell) \neq (j', k', \ell')$, or $\mathsf{HMAC}_{hmk_N}(j'||k'||\ell'||\mathtt{nymble}||\overline{\mathtt{tdr}||\mathtt{id}}) \neq \mathtt{mac}^N$,
  2. return valid otherwise.
- $\mathtt{linkingTKN}/\perp \leftarrow \mathsf{LinkingTknExt}_{nmsk}(j, k, \ell^*, \mathtt{nymbleTKT})$.
  To extract the linking token from a nymble ticket $\mathtt{nymbleTKT} = \langle \cdot, \cdot, \ell, \cdot, \overline{\mathtt{tdr}||\mathtt{id}}, \cdot, \cdot\rangle$ for server S_j's use at time period T_{ℓ^*} during linkability window W_k, the NM does the following using his secret key $nmsk$:
  1. return $\perp$ if $\ell^* < \ell$ or $\mathsf{NMVerify}_{nmsk}(j, k, \ell, \mathtt{nymbleTKT}) = \mathtt{invalid}$,

2. compute $\texttt{tdr}||\texttt{id} \leftarrow \mathsf{Enc.decrypt}_{sek_N}(\overline{\texttt{tdr}||\texttt{id}})$,
3. pick $\texttt{tdr}^*$ uniformly at random from the range of f if a linking token has already been issued for the $(\texttt{id}, j, k)$-tuple, otherwise compute $\texttt{tdr}^*$ as $f^{(\ell^*-\ell)}(\texttt{tdr})$ and record that a linking token has been issued for the $(\texttt{id}, j, k)$-tuple,[5]
4. return $\texttt{linkingTKN}$ as $\langle j, k, \ell^*, \texttt{tdr}^* \rangle$.

- $\texttt{linked}/\texttt{not-linked} \leftarrow \mathsf{Link}(\texttt{nymbleTKT}, \texttt{linkingTKN})$.
  To test if a nymble ticket $\texttt{nymbleTKT} = \langle j, k, \ell, \texttt{nymble}, \cdot, \cdot, \cdot \rangle$ is linked by the linking token $\texttt{linkingTKN} = \langle j', k', \ell', \texttt{tdr}' \rangle$, anyone can do the following:
  1. return $\texttt{not-linked}$ if $(j, k) \neq (j', k')$ or $\ell < \ell'$, or if $g(f^{(\ell-\ell')}(\texttt{tdr})) \neq \texttt{nymble}$,
  2. return $\texttt{linked}$ otherwise.

Security Analysis. We formalize the notions of *Correctness*, *Unforgeability*, *Backward Anonymity* and *Uncircumventable Forward Linkability* in Appendix A.1. We now state the following theorem about the security of Nymble-Auth, and sketch its proof in Appendix A.2.

Theorem 1. *Our* Nymble-Auth *construction is secure in the Random Oracle Model.*

5 The NYMBLE System

We now describe the full construction of our system, focusing on the various interactions between the Pseudonym Manager (PM), the Nymble Manager (NM), the servers and the users.

Parameters. In addition to those in the Nymble-Auth module, parameters in the NYMBLE system include the description of an additional secure cryptographic hash function h and a secure signature scheme Sig with security parameters polynomial in λ. Also, $\mathcal{T}$ denotes the duration of one time period and L denotes the number of time periods in one linkability window. Let t_0 be the system start time.

5.1 System Setup

In this procedure, the NM and the PM set up the NYMBLE system together. The PM picks a key khk_P for keyed hash function H uniformly at random from the key-space. The NM, on the other hand, does the following:

1. execute Setup of Nymble-Auth on some sufficiently large security parameter λ, after which the NM gets its secret key $nmsk$ and each server S_j gets its own secret key ssk_j as described in the previous section,
2. generate a private/public-key-pair (x, y) for Sig using its key generation algorithm,
3. pick an HMAC key hmk_{NP} for HMAC uniformly at random from the key-space and share it with the PM over a secure channel, and
4. give each server S_j an empty blacklist $\texttt{BL}_j = \langle j, 1, \langle \perp \rangle, \perp, \sigma \rangle$ through a secure channel, where σ is the signature generated as $\mathsf{Sig.sign}_x(j||1||1)$. Why the blacklist is formatted this way will become clear soon.

At the end of this procedure, the PM stores (khk_P, hmk_{PN}) privately, while the NM stores $(nmsk, x, hmk_{PN})$ privately and publishes the signature public key y. Also, each server S_j stores ssk_j privately.

[5] In Section 5.5, we will show how state about issued trapdoors is offloaded to servers.

5.2 User Registration

In this procedure, user Alice interacts with the PM in order to register herself to the NYMBLE system for linkability window k. Alice obtains a pseudonym from the PM upon a successful termination of such an interaction. The communication channel between them is confidential and PM-authenticated.

To register, Alice authenticates herself as a user with identity id to the PM by demonstrating her control over some resource(s) as discussed, after which the PM computes $\mathtt{pnym} \leftarrow H_{khk_P}(\mathtt{id}, k)$ and $\mathtt{mac}^{PN} \leftarrow \mathsf{HMAC}_{hmk_{NP}}(\mathtt{pnym}, k)$, and returns $\langle \mathtt{pnym}, \mathtt{mac}^{PN} \rangle$ to Alice, who stores it privately.

5.3 Acquisition of Nymble Tickets

In order for Alice to authenticate to any server S_j during any linkability window W_k, she must present a nymble ticket to the server. The following describes how she can obtain a credential from the NM containing such tickets. The communication channel is anonymous (e.g., through Tor), confidential and NM-authenticated.

Alice sends her $\langle \mathtt{pnym}, \mathtt{mac}^{PN} \rangle$ to the NM, after which the NM:

1. asserts that $\mathtt{mac}^{PN} = \mathsf{HMAC}_{hmk_{NP}}(\mathtt{pnym}, k)$,
2. computes $\mathtt{nymbleTKT}_\ell \leftarrow \mathsf{NymbleTktGen}_{nmsk}(\mathtt{pnym}, j, k, \ell)$, for $\ell = 1$ to L, and
3. returns cred as $\langle \mathtt{seed}, \mathtt{nymbleTKT}_1, \mathtt{nymbleTKT}_2, \ldots, \mathtt{nymbleTKT}_L \rangle$, where $\mathtt{seed} = H_{khk_N}(\mathtt{pnym}, j, k)$ is the seed used within NymbleTktGen.

Alice may acquire credentials for different servers and different linkability windows at any time. She stores these credentials locally before she needs them.

Efficiency. This protocol has a timing complexity of $O(L)$.[6] All the computations are quick symmetric operations—there are two cryptographic hashes, two HMACs and one symmetric encryption per loop-iteration A credential is of size $O(L)$.

5.4 Request for Services

At a high level, a user Alice presents to server Bob the nymble ticket for the current time period. As nymble tickets are unlinkable until servers complain against them (and thereby blacklisting the corresponding user or IP address), Alice must check whether she is on Bob's blacklist, and verify its integrity and freshness. If Alice decides to proceed, she presents her nymble ticket to Bob, and Bob verifies that the nymble ticket is not on his blacklist. Bob also retains the ticket in case he wants to later complain against the current access. For example, Wikipedia might detect a fraudulent posting several hours after it has been made. The nymble ticket associated with that request can be used to blacklist future accesses by that user.

Each server in the system maintains two data structures, the blacklist BL and the linking-list LL, to handle blacklisting-related mechanisms to be described below. BL is

[6] A naïve implementation would involve a two-level for-loop with $O(L^2)$ complexity at the NM. However, such a loop can be trivially collapsed into single-level, with $O(L)$ complexity instead.

in the form of $\langle j, k, \langle \texttt{entry}_1, \ldots, \texttt{entry}_v \rangle, \texttt{digest}, \sigma \rangle$, where each entry $\texttt{entry}_m = \langle m, \texttt{nymbleTKT}_m, T_m \rangle$. LL is a list of $\langle t_m, \texttt{tdr}_m, \texttt{nymble}_m \rangle$ entries.

The following describes in detail the protocol, during which Alice wants to access the service provided by server Bob (S_j) at time period T_ℓ during linkability window W_k. She will need to make use of seed and $\texttt{nymbleTKT}_\ell$ in cred, which is the credential she obtained from the NM earlier for accessing Bob's service within window W_k. The communication channel between Alice and the NM is NM-authenticated and anonymous (through Tor), while that between Alice and Bob is secure, server-authenticated and anonymous (through Tor).

1. *(Blacklist Request.)* Upon receiving Alice's request, Bob returns to Alice his current blacklist BL, where $\texttt{BL} = \langle \cdot, \cdot, \langle \texttt{entry}_1, \ldots, \texttt{entry}_v \rangle, \cdot, \sigma \rangle$, each $\texttt{entry}_m = \langle m, \texttt{nymbleTKT}_m, T_m \rangle$ and each $\texttt{nymbleTKT}_m = \langle \cdot, \cdot, \cdot, \texttt{nymble}_m, \cdot, \cdot, \cdot \rangle$.[7] As described earlier, each entry corresponds to a blacklisted user, where $\texttt{nymbleTKT}_m$ was the nymble ticket used by that user in time period T_m.
2. *(Version-number Request.)* Upon receiving Alice's request, the NM returns v_j, the current version number of Bob's blacklist recorded by the NM.
3. *(Blacklist Inspection.)* Alice terminates immediately as failure if:
   - $\mathsf{Sig.Verify}_y(j||k||v_j||h(\ldots h(h(\texttt{entry}_1)||\texttt{entry}_2)\ldots||\texttt{entry}_v), \sigma) = \texttt{invalid}$,[8] i.e., the blacklist is not authentic or not intact, or
   - $g(f^{(T_m)}(\texttt{seed})) = \texttt{nymble}_m$ for some $m \in \{1, \ldots, v\}$,[9] i.e., Alice has been blacklisted.

   Otherwise she sends $\texttt{nymbleTKT}_\ell = \langle \cdot, \cdot, \cdot, \texttt{nymble}, \cdot, \cdot, \cdot \rangle$ to Bob.
4. *(Ticket Inspection.)* Bob returns failure if:
   - $\mathsf{ServerVerify}_{ssk_j}(k, \ell, \texttt{nymbleTKT}) = \texttt{invalid}$, i.e., the ticket is invalid, or
   - nymble appears in his linking-list LL, i.e. the connecting user has already been blacklisted.

   Otherwise Bob grants Alice's request for service and starts serving Alice. Bob records $\texttt{nymbleTKT}_\ell$ along with appropriate access log for potentially complaining about that connection in the future.

Efficiency. Recall that v is the size of Bob's blacklist. Blacklist integrity checking is of $O(v)$ in time. Again, all cryptographic operations in each loop-iteration are symmetric and there is only one digital verification at the end, independent of v. Checking if being linked has a time complexity of $O(vL)$ at the user in the worst case, but all computations involved are simple hashes. Also, one could trade off space to make it $O(v)$ instead. Time complexity of nymble matching at the server is linear in the size of the Linking-list using linear search. But efficient data structures exist which can make these steps logarithmic or even constant (e.g., using hash tables).

[7] If Bob doesn't want to use the NYMBLE system for this request, he may skip the rest of the protocol and start serving (or denying) Alice immediately.

[8] By remembering an older digest if Alice has accessed Bob earlier within the same window, Alice can instead compute only part of the recursive hash above.

[9] This step may be sped up by trading off space by storing the original nymble tickets issued in the user's credential, making this step a simple lookup.

5.5 Complaining

By presenting to the NM the nymble ticket associated with an access in which Bob thinks the user misbehaved, Bob obtains a linking token that will allow him to link all future nymble tickets for that user.[10] The following enumerates the protocol, during which server Bob (S_j) complains about $\texttt{nymbleTKT}$. The communication between Bob and the NM is conducted over a secure channel. Let the time of complaint be at time period T_ℓ during linkability window W_k, where $\ell < L$, i.e. the complaint is not during the last period of a linkability-window.

1. *(Complaining.)* Bob sends to the NM the nymble ticket $\texttt{nymbleTKT} = \langle \cdot, \cdot, \ell', \cdot, \cdot, \cdot, \cdot \rangle$ he wants to complain about and $\langle \texttt{digest}, \sigma \rangle$ from his current blacklist $\texttt{BL}$.
2. *(Complaint Validation.)* The NM rejects a complaint if
   - $\mathsf{NMVerify}_{nmsk}(j, k, \ell', \texttt{nymbleTKT}) = \texttt{invalid}$, i.e., the ticket is invalid, or
   - $\mathsf{Sig.verify}_y(j||k||v_j||\texttt{digest}, \sigma) = \texttt{invalid}$, where v_j is the version number of Bob's blacklist recorded by NM, i.e. the (digest of) Bob's blacklist is not authentic, intact or fresh.

   The NM proceeds otherwise.
3. *(Linking-token Issuing.)* The NM computes the following:
   - $\texttt{linkingTKN} \leftarrow \mathsf{LinkingTknExt}_{nmsk}(j, k, \ell + 1, \texttt{nymbleTKT})$,
   - $\texttt{entry}' \leftarrow (v_j + 1, \texttt{nymbleTKT}, \ell)$, $\texttt{digest}' \leftarrow h(\texttt{digest}||\texttt{entry}')$ and then $\sigma' \leftarrow \mathsf{Sig.sign}_x(j||k||v_j + 1||\texttt{digest}')$.

   The NM increments v_j by 1 and returns $\langle \texttt{linkingTKN}, \texttt{entry}', \texttt{digest}', \sigma' \rangle$ to Bob.
4. *(List Update.)* Bob updates his blacklist $\texttt{BL}$ and linking-list $\texttt{LL}$ as follows:
   - In $\texttt{BL}$, Bob increments v by 1, inserts $\texttt{entry}'$ as the last entry, and updates σ to σ' and $\texttt{digest}$ to $\texttt{digest}'$.
   - In $\texttt{LL}$, Bob appends a new entry $\langle \ell + 1, \texttt{tdr}, \texttt{nymble} \rangle$, where $\texttt{tdr}$ is the trapdoor in $\texttt{linkingTKN}$ and $\texttt{nymble} \leftarrow g(\texttt{tdr})$.

Efficiency. The NM's timing complexity is $O(L + v)$. The $O(L)$ is due to a call to $\mathsf{LinkingTknExt}$, which involves only hashing or HMAC operations. Verifying if a linking token has already been issued for the user involves $O(v)$ symmetric decryption operations. Signing one digital signature is the only asymmetric cryptographic operation.

5.6 Update

Misbehavior is forgiven every time the system enters a new linkability window. Users who misbehaved previously can then connect to websites anonymously until they misbehave and are complained against again. The nymble tickets, linking tokens, and pseudonyms that are specific to one linkability window become useless when the system enters a new window. Consequently, servers empty their blacklists and linking-lists while NM resets all version numbers to zero at the end of every linkability window. Moreover, the NM also issues empty blacklists to the servers in the same way as in the System Setup procedure.

[10] Here "future" means starting from the next time period, rather than the same period immediately after the complaint. This prevents race conditions such as when a user has inspected that she is not blacklisted by Bob and is about to present her nymble ticket, but in the meantime Bob obtains a linking-token for the current time period.

At the end of each time period $T_{\ell'}$ that is not the last one in a linkability window, each server updates its linking-list LL by replacing every entry $\langle \ell, \mathtt{tdr}_\ell, \mathtt{nymble}_\ell \rangle$ such that $\ell = \ell'$ with the entry $\langle \ell+1, f(\mathtt{tdr}_\ell), g(f(\mathtt{tdr}_\ell)) \rangle$. Only hashing is required to accomplish this and the number of hash operations involved is two times the size of LL.

6 Evaluation

We chose to implement our system using PHP because of its popularity for interactive web sites (including MediaWiki, the software behind Wikipedia). PHP contains both built-in cryptographic primitives (e.g., SHA-1 hashing) as well as an interface to the OpenSSL cryptographic library; we used both as appropriate to maximize performance. Additionally, we chose to use a relational database to store blacklists and blacklist versions because the interface is convenient in PHP and database servers are generally available in the environments in which we envision the Nymble system being used.

We picked SHA-256 [21] for collision-resistant hash functions f, g and h; HMAC-SHA-1 [23] with 160-bit keys for both keyed hash function H and keyed-hash message authentication code HMAC; AES-256 in OFB mode with block size of 32 bytes [22] for symmetric encryption Enc; and 1024-bit RSA [20] for digital signature Sig. We chose RSA over DSA for digital signatures because of its higher signature verification speed—in our system, signature verification occurs more often than signing.

Our implementation consists of separate modules to be deployed to the Pseudonym Manager, the Nymble Manager, and servers, as well as common modules for database access and cryptographic operations. Orthogonal to the correctness of the system, we felt deployability was also an important goal, and to that end we attempted to minimize modifications required to "Nymble-protect" existing applications.

Applications wishing to use our system include a single PHP file defining two functions: `nymbletkt_is_required(ip)` and `nymbletkt_is_valid(nymbletkt)`. The former determines whether an operation from a particular IP address requires a nymble ticket; the second determines whether the supplied nymble ticket is valid for the current time period. Hence the only modifications necessary are to supply a nymble ticket input field if it is required and to verify the nymble ticket when it is submitted.

To test the system, a single machine acted as Pseudonym Manager, Nymble Manager, and server, running PHP 5.1.6, PostgreSQL 8.1, and Apache 2.0.55, atop a default install of Ubuntu 6.10 and Linux 2.6.17 with SMP enabled. The machine itself was an Intel Core 2 Duo 6300 (2 cores at 1.86 GHz each) and 1 GB memory. Clients of various hardware configurations accessed the server via the local network.

Table 1 shows the speed of operations important to both end-users and server administrators evaluating the Nymble system. Experiments were run 50 times and the results averaged. The *all-ticket credentials* generated were for 1-day linkability windows with 5-minute time periods, thus consisting of 288 nymble tickets. We also measured performance with *single-ticket credentials* containing the ticket for only the time period in which access to a service was requested. Single-ticket credentials are much more efficient to generate, but the NM learns all the time periods (as opposed to only the first time period) when connections are made by the pseudonymous users. The nymble ticket verification step is the added overhead required during a single user action, visible to both the server

and the client. The linking token generation is the operation carried out by NM when a server complains. Every time period, each server must iterate every entry in its blacklist; our measurement reflects the time required for a single server to iterate 100 entries.

Table 1. Timing measurements

Operation	Executed by	Time
All-ticket credential generation (288 nymbleTKTs)	NM	224 ms
Single-ticket credential generation (1 nymbleTKT)	NM	1.1 ms
Nymble ticket verification (Verify)	Server	1 ms
Linking token generation (LinkingTknExt)	NM	15 ms
Blacklist update	Server	8 ms

We believe the all-ticket credential generation time of 224ms is reasonable for a network like Tor, especially since nymble tickets will be needed only for restricted actions such as edits on Wikipedia—latest data indicate that about two edits per second to Wikipedia's English pages.[11] We expect much fewer edits to be made via Tor. If all-ticket credential generation proves to be a bottleneck, the NM can issue single-ticket credentials to drastically reduce the load. We expect the measured time of 1.1ms for generating single-ticket credentials to be more than sufficient.

We have implemented a Firefox extension that automatically obtains a pseudonym from the PM, obtains a credential for a server when needed, verifies the server's blacklist, chooses the correct nymble ticket for the current time period, and inserts it into an HTML form every time the user wishes to traverse a protected page. We assume that the NM is the arbiter of time, and the user and servers can obtain the current time period and linkability window from the NM. For example, when a user obtains the current version number for a particular website's blacklist, the user also learns the current time period.

Finally, we emphasize the NYMBLE system also scales well in terms of space complexities. Credentials that users store are of size $20 + 148L$ bytes each, or 42KB when $L = 288$. The blacklist has a size of $168 + 152v$ bytes, where v is the number of users blacklisted by the server during the current linkability window. Most importantly, the amount of data the NM has to maintain is minimal, namely only one 32-bit integer per registered server for storing the server's current version number.

7 Discussion

IP-address blocking. As described in Section 3, users demonstrate control over an IP address by connecting to the PM directly. Since this connection is made without Tor, some users may object to the temporary loss of anonymity. It is important to provide users with an accurate portrayal of the associated risks (and benefits) before using our system.

There are some inherent limitations to using IP addresses as the scarce resource. If a user can obtain multiple IP addresses she can circumvent nymble-based blocking and continue to misbehave. We point out, however, that this problem exists in the absence

[11] http://stats.wikimedia.org/EN/PlotsPngDatabaseEdits.htm

of anonymizing networks as well, and the user would be able to circumvent regular IP-address based blocking using multiple IP addresses. Some servers alleviate this problem with subnet-based IP blocking, and while it is possible to modify our system to support subnet-blocking, new privacy challenges emerge; a more thorough description of subnet-blocking is left for future work. Another limitation is that a user Alice may acquire nymbles for a particular IP address and then use them at a later time (to misbehave) within the linkability window even after she has relinquished control over that IP address. This type of attack allows Alice a little more flexibility—with regular IP-based blocking, Alice would have to perform misbehaviors while in control of the various IP addresses.

Some other resource may be used to acquire pseudonyms, but we believe IP-address blocking is still the most pragmatic approach in today's Internet. Our approach closely mimics IP-address blocking that many servers on the Internet rely on routinely. Some may be concerned that users can misbehave through Tor *after* their IP address has been blocked, effectively allowing them to misbehave twice before being blocked (once using regular IP-address blocking, and once using Nymble). We argue, however, that servers concerned about this problem could require Nymble-based authentication from *all* users, whether or not they connect through an anonymizing network.

Time periods and linkability windows. Since nymbles are associated with time periods, it is desirable to keep the duration $\mathcal{T}$ of time periods small. On the other hand, larger values of $\mathcal{T}$ can be used to limit the rate of anonymous connections by a user. Since users remain blacklisted for the remainder of the linkability window after a complaint, it is desirable to keep the duration of the linkability window $\mathcal{L}$ long enough to curtail the malicious user's activities, but not so long as to punish that user (or honest users to whom the IP address may get reassigned) indefinitely. In our example we suggested $\mathcal{T} = 5$ min and $\mathcal{L} = 1$ day, but further experimentation is needed to determine reasonable values for these parameters.

Server-specific linkability windows. An enhancement would be to provide support to vary $\mathcal{T}$ and $\mathcal{L}$ for different servers. As described, our system does not support varying linkability windows, but does support varying time periods. This is because the PM is not aware of the server to which the user wishes to connect, yet it must issue pseudonyms specific to a linkability window. In our system, therefore, the linkability window must be fixed across all servers. Supporting varying time periods is easy, and the NM can be modified to issue the appropriate set of nymble tickets based on the servers' parameters.

8 Conclusion

We present a system that allows websites to selectively block users of anonymizing networks such as Tor. Using our system, websites can blacklist users *without* knowing their IP addresses. Users not on the blacklist enjoy anonymity, while blacklisted users are blocked from making future accesses. Furthermore, blacklisted users' previous connections remain anonymous. Since websites are free to blacklist anonymous users of their choice, and since users are notified of their blacklisting status, our system avoids the complications associated with judging "misbehavior." We believe that these properties will enhance the acceptability of anonymizing networks such as Tor by enabling websites to selectively block certain users instead of blocking the entire network, all while allowing the remaining (honest) users to stay anonymous.

Acknowledgments

This paper greatly benefited from discussions with Sergey Bratus, Alexander Iliev, and Anna Shubina. We also thank Roger Dingledine, Paul Syverson, Parisa Tabriz, Seung Yi, and the anonymous reviewers for their helpful comments.

References

1. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *LNCS*, pages 255–270. Springer, 2000.
2. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *LNCS*, pages 614–629. Springer, 2003.
3. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 136–153. Springer, 2005.
4. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *LNCS*, pages 302–318. Springer, 1993.
5. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 201–210. ACM, 2006.
6. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 302–321. Springer, 2005.
7. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.
8. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.
9. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.
10. David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
11. David Chaum. Showing credentials without identification transfeering signatures between unconditionally unlinkable pseudonyms. In Jennifer Seberry and Josef Pieprzyk, editors, *AUSCRYPT*, volume 453 of *LNCS*, pages 246–264. Springer, 1990.
12. David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
13. Lidong Chen. Access with pseudonyms. In Ed Dawson and Jovan Dj. Golic, editors, *Cryptography: Policy and Algorithms*, volume 1029 of *LNCS*, pages 232–243. Springer, 1995.
14. Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *CRYPTO*, volume 403 of *LNCS*, pages 328–335. Springer, 1988.
15. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Usenix Security Symposium*, pages 303–320, August 2004.
16. John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *IPTPS*, volume 2429 of *LNCS*, pages 251–260. Springer, 2002.

17. Jason E. Holt and Kent E. Seamons. Nym: Practical pseudonymity for anonymous networks. Internet Security Research Lab Technical Report 2006-4, Brigham Young University, June 2006.
18. Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 571–589. Springer, 2004.
19. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *LNCS*, pages 184–199. Springer, 1999.
20. NIST. FIPS 186-2: Digital signature standard (DSS). Technical report, National Institute of Standards and Technology (NIST), 2000. `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`.
21. NIST. FIPS 180-2: Secure hash standard (SHS). Technical report, National Institute of Standards and Technology (NIST), 2001. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`.
22. NIST. FIPS 197: Announcing the advanced encryption standard (AES). Technical report, National Institute of Standards and Technology (NIST), 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
23. NIST. FIPS 198: The keyed-hash message authentication code (HMAC). Technical report, National Institute of Standards and Technology (NIST), 2002. `http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf`.
24. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.
25. Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.
26. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *LNCS*, pages 552–565. Springer, 2001.
27. Paul F. Syverson, Stuart G. Stubblebine, and David M. Goldschlag. Unlinkable serial transactions. In Rafael Hirschfeld, editor, *Financial Cryptography*, volume 1318 of *LNCS*, pages 39–56. Springer, 1997.
28. Isamu Teranishi, Jun Furukawa, and Kazue Sako. *k*-times anonymous authentication (extended abstract). In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 308–322. Springer, 2004.
29. Patrick P. Tsang, Apu Kapadia, and Sean W. Smith. Anonymous IP-address blocking in tor with trusted computing (work-in-progress). In The Second Workshop on Advances in Trusted Computing (WATC '06 Fall), November 2006.
30. Luis von Ahn, Andrew Bortz, Nicholas J. Hopper, and Kevin O'Neill. Selectively traceable anonymity. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *LNCS*, pages 208–222. Springer, 2006.

A Security Model, Proofs and Analysis

A.1 Security Model for Nymble-Auth

Correctness means the system functions as intended when all entities are honest. Unforgeability guarantees that valid nymble tickets can only be obtained from NM. Backward Anonymity makes sure nymble tickets are anonymous without an associated trapdoor and remain anonymous even with an associated trapdoor as long as that trapdoor is meant for a time period later than the nymble tickets. Finally, Uncircumventable Forward Linkability says that valid nymble tickets are always linked to an associated trapdoor meant for a time prior to those nymble tickets.

Definition 1 (Correctness). A Nymble-Auth construction is *correct* if it has *Verification Correctness* and *Linking Correctness*, defined as follows:

- *(Verification Correctness.)* If all entities in the system are honest (i.e. they execute the algorithms according to the system specification), then ServerVerify returns `1` (indicating `valid`) on any `nymbleTKT` output by NymbleTktGen, with overwhelming probability.
- *(Linking Correctness.)* If all entities in the system are honest, then Link returns `linked` on any `linkingTKN` generated by $\mathsf{LinkingTknExt}(i, j, k, \ell)$ and any `nymbleTKT` generated by $\mathsf{NymbleTktGen}(i', j', k', \ell')$ if and only if $(i, j, k) = (i', j', k')$ and $j \leq j'$, with overwhelming probability.

□

We describe three oracles before defining various security games. The existence of the oracles models the adversary's capability in the real world of learning as much information about the nymbles and trapdoors as possible by probing the system. They are:

- $\mathcal{O}_{TKT}(i, j, k, \ell)$, or the *Ticket Oracle*. It returns $\texttt{nymbleTKT}_{i,j,k,\ell}$, the nymble ticket as output by the NymbleTktGen algorithm on input (i, j, k, ℓ),
- $\mathcal{O}_{TKN}(\texttt{nymbleTKT}, \ell)$, or the *Token Oracle*. It returns the linking token $\texttt{linkingTKN}_{i,j,k,\ell}$ as output by the LinkingTknExt algorithm on input $(\texttt{nymbleTKT}, \ell)$, and
- $\mathcal{O}_K(j)$, or the *Server Corruption Oracle.* It returns k_{NS_j}, the symmetric key of server S_i as output by the Setup algorithm.

Definition 2 (Unforgeability). A Nymble-Auth construction is *Unforgeable* if no *Probabilistic Poly-Time (PPT)* adversary $\mathcal{A}$ can win the following game against the Challenger $\mathcal{C}$ with non-negligible probability:

1. *(Setup Phase.)* $\mathcal{C}$ executes $\mathsf{Setup}(1^\lambda)$ on a sufficiently large λ, keeps $nmsk$ secret and gives $nmpk$ to $\mathcal{A}$.
2. *(Probing Phase.)* $\mathcal{A}$ may arbitrarily and adaptively query three oracles $\mathcal{O}_{TKN}(\texttt{nymbleTKT}, \ell)$, $\mathcal{O}_{TKT}(i, j, k, \ell)$ and $\mathcal{O}_K(j)$.
3. *(End Game Phase.)* $\mathcal{A}$ returns $\langle j^*, k^*\ell^*, \texttt{nymbleTKT}^* \rangle$. $\mathcal{A}$ wins the game if $\texttt{nymbleTKT}^*$ is not an output of a previous $\mathcal{O}_{TKT}(\cdot, \cdot, \cdot, \cdot)$ query, $\mathcal{A}$ did not query $\mathcal{O}_K(j^*)$ and $\mathsf{Verify}_{k_{NS_{j^*}}}(j^*, k^*, \ell^*, \texttt{nymbleTKT}^*) = \texttt{1}$.

□

That is, $\mathcal{A}$ should not be able to forge a valid `nymbleTKT` without the NM's secret parameters.

Definition 3 (Backward Anonymity). A Nymble-Auth construction has *Backward Anonymity* if no PPT adversary $\mathcal{A}$ can win the following game against the Challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2:

1. *(Setup Phase.)* $\mathcal{C}$ executes Setup on a sufficiently large security parameter, keeps $nmsk$ secret, and gives $nmpk$ to $\mathcal{A}$.
2. *(Probing Phase I.)* $\mathcal{A}$ may arbitrarily and adaptively query the three oracles $\mathcal{O}_{TKT}(\cdot, \cdot, \cdot, \cdot)$, $\mathcal{O}_{TKN}(\cdot, \cdot)$ and $\mathcal{O}_K(\cdot)$.

3. *(Challenge Phase.)* $\mathcal{A}$ decides on positive integers $i_0^*, i_1^*, j^*, k^*, \ell^*$ such that the following conditions hold:
   - For all queries $\mathcal{O}_{TKT}(i,j,k,\ell)$, for each $b \in \{0,1\}$ we have that $(i,j,k,\ell) \neq (i_b^*, j^*, k^*, \ell^*)$, i.e., $\mathcal{A}$ has not already obtained any of the challenge `nymbleTKT`s from $\mathcal{O}_{TKT}$.
   - For all queries $\mathcal{O}_{TKN}(\cdot,\ell)$, $\ell > \ell^*$, i.e. all trapdoors obtained are of time periods greater than the challenge nymble tickets.
   - For each $b \in \{0,1\}$, for all queries $\mathcal{O}_{TKT}(i_1,j_1,k_1,\ell_1)$ where $(i_1,j_1,k_1) = (i_b^*, j^*, k^*)$ and queries $\mathcal{O}_{TKN}(\texttt{nymbleTKT},\ell)$ where `nymbleTKT` is the output of some query $\mathcal{O}_{TKT}(i_2,j_2,k_2,\ell_2)$ such that $(i_2,j_2,k_2) = (i_b^*, j^*, k^*)$, $\ell_1 < \ell$, i.e. the adversary is not allowed to query for a linking token and a nymble ticket such that the trapdoor within the linking token can be used to link the nymble ticket.[12]

   Then $\mathcal{C}$ flips a fair coin $b^* \in_R \{0,1\}$ and returns $\mathcal{A}$ with

$$\langle \texttt{nymbleTKT}_{i_{b^*},j^*,k^*\ell^*}, \texttt{nymbleTKT}_{i_{\widetilde{b^*}},j^*,k^*,\ell^*} \rangle,$$

   where $\widetilde{b^*}$ is the negation of b^*.
4. *(Probing Phase II.)* $\mathcal{A}$ may arbitrarily and adaptively query the three oracles, except that the conditions above must still hold.
5. *(End Game Phase.)* $\mathcal{A}$ returns guess $\hat{b} \in \{0,1\}$ on b^*. $\mathcal{A}$ wins if $\hat{b} = b^*$.

□

That is, $\mathcal{A}$ should not be able to link `nymbleTKT`s without the appropriate `linkingTKN`.

Definition 4 (Uncircumventable Forward Linkability). A Nymble-Auth construction has *Uncircumventable Forward Linkability* if no PPT adversary can win the following game she plays against the Challenger $\mathcal{C}$ with non-negligible probability:

1. *(Setup Phase.)* $\mathcal{C}$ executes Setup on a sufficiently large security parameter, keeps $nmsk$ secret and gives $nmpk$ to $\mathcal{A}$.
2. *(Probing Phase.)* $\mathcal{A}$ may arbitrarily and adaptively query the three oracles $\mathcal{O}_{TKN}(i,j,k,\ell)$, $\mathcal{O}_{TKT}(i,j,k,\ell)$ and $\mathcal{O}_K(j)$, where $i,j,k,\ell \geq 1$.
3. *(End Game Phase.)* $\mathcal{A}$ returns $\langle j^*, k^*, \ell_0^*, \ell_1^*, \texttt{nymbleTKT}_0^*, \texttt{nymbleTKT}_1^*, \ell^* \rangle$. $\mathcal{A}$ wins the game if $\mathsf{Verify}(j^*, k^*, \ell_b^*, \texttt{nymbleTKT}_b^*) = 1$ for $b \in \{0,1\}$, $\ell_0^* \leq \ell^* \leq \ell_1^*$, $\mathcal{A}$ did not query $\mathcal{O}_K(\cdot)$ on j^* and

$$\mathsf{Link}(\texttt{nymble}_1^*, \mathsf{TrapdoorExt}(\texttt{nymble}_0^*, \ell^*)) = 0.$$

□

That is, $\mathcal{A}$ cannot obtain two `nymbleTKT`s for any server such that they are unlinkable, but should have otherwise been linkable, with the appropriate `linkingTKN`.

A.2 Security Proofs for Nymble-Auth

Proof (Theorem 1). (Sketch.) We prove the theorem by showing our Nymble-Auth construction is correct, unforgeable, backward anonymous and uncircumventably forward linkable. Due to page limitation, we gives only proof sketches here.

[12] This restriction is justified because we will use Nymble-Auth in such a way that a user will never present a nymble ticket again to a server once the server has acquired a linking token for that user.

CORRECTNESS. Correctness of Nymble-Auth is straightforward. Namely, verification correctness is implied by the correctness of the HMAC HMAC. Linking correctness is implied by the determinism and collision-resistance of hash functions f, g.

UNFORGEABILITY. Our Nymble-Auth construction has unforgeability due to the security of HMAC, which guarantees that without the knowledge of the associated key, no PPT adversary can produce a valid MAC on any input string, even if the adversary learns arbitrary input-output HMAC pairs. As a result, any PPT in our construction is allowed to query the oracle for arbitrary nymble tickets and yet is unable to produce a new one with a correct MAC on it.

BACKWARD ANONYMITY. Our Nymble-Auth construction has backward anonymity due to the security of the symmetric encryption Enc and the collision-resistance of the hash functions f and g. The only pieces within a nymble ticket that are correlated to user identities are the nymble and the encrypted trapdoor. The security of Enc guarantees that ciphertexts leak no information about their underlying plaintexts to any PPT adversary, so that the encrypted trapdoor is computationally uncorrelated to the underlying trapdoor and thus the identity of the user to which the nymble ticket belongs. The nymble is the output of $g(\cdot)$ on its associated trapdoor, which is in turn the output after a series of application of $f(\cdot)$ on a seed dependent on the user identity. Under the Random Oracle Model, hash function outputs are random (but consistent), therefore the nymbles are distinguishable from random strings in the view of an adversary who does not have any trapdoor for the user-server-window tuple to which nymbles in the challenge nymble tickets are associated.

Knowing one or more associated trapdoors does not help the adversary in winning the game as the trapdoors are also indistinguishable from random values. This is the case because all except the first linking token returned by $\mathcal{O}_{TKN}$ contain a true random value as the trapdoor. The first linking token contains a genuine trapdoor, but it is indistinguishable from a random value because $f(\cdot)$ and $g(\cdot)$ are random oracles and the adversary is not allowed to query $\mathcal{O}_{TKT}$ for a nymble ticket and query $\mathcal{O}_{TKN}$ for a linking token such that the time period of the nymble ticket is greater than or equal to the time period of the linking token.

UNCIRCUMVENTABLE FORWARD LINKABILITY. Our Nymble-Auth construction has Uncircumventable Forward Linkability as a consequence of unforgeability and linking correctness. Specifically, assume there exists an PPT adversary who can break uncircumventable forward linkability; then the two nymble tickets she output at the End Game phase must be such that they are query outputs of the nymble ticket oracle, because otherwise the adversary would have broken the unforgeability of Nymble-Auth, which leads to a contradiction. Now since the two output nymble tickets are generated according to specification, the linking algorithm will return `linked` on any trapdoor extracted from the nymble ticket earlier in time period, which again leads to a contradiction. □