Blocking Misbehaving Users in Anonymizing Networks: Nymbles

Ms.S.Abinaya,
IT Department, III-year, Anna University
Vivekanandha college of Engineering for Women,
Tiruchengode,
Tamilnadu, India.
612911205002@vcew.ac.in

Ms.T.E.Bavisha,
IT Department, III-year, Anna University
Vivekanandha college of Engineering for Women,
Tiruchengode,
Tamilnadu, India.
612911205010@vcew.ac.in

Abstract— **Anonymizing networks such as allow users to access Internet services privately by using a series of routers to hide the client's IP address from the server. The success of such networks, however, has been limited by users employing this anonymity for abusive purposes such as defacing popular websites. Website administrators routinely rely on IP-address blocking for disabling access to misbehaving users, but blocking IP addresses is not practical if the abuser routes through an anonymizing network. As a result, administrators block all known exit nodes of anonymizing networks, denying anonymous access to misbehaving and behaving users alike. To address this problem, we present Nymble, a system in which servers can "blacklist" misbehaving users, thereby blocking users without compromising their anonymity. Our system is thus agnostic to different servers' definitions of misbehavior — servers can blacklist users for whatever reason, and the privacy of blacklisted users is maintained.**

Index Terms—Verifier Local Revocation(VLR), Trusted Third Party(TTP), Transport Control Protocol/Internet Protocol(TCP/IP)

I. INTRODUCTION (*Heading 1*)

Anonymizing networks such as Tor [18] route traffic through independent nodes in separate administrative domains to hide a client's IP address. Unfortunately, some users have misused such networks — under the cover of anonymity, users have repeatedly defaced popular websites such as Wikipedia. Since web-site administrators cannot blacklist individual malicious users' IP addresses, they blacklist the entire anonymizing network. Such measures eliminate malicious activity through anonymizing networks at the cost of denying anonymous access to behaving users. In other words, a few ''bad apples'' can spoil the fun for all. (This has happened repeatedly with Tor.[1])

There are several solutions to this problem, each providing some degree of accountability. In pseudonymous credential systems [14], [17], [23], [28], users log into websites using pseudonyms, which can be added to a blacklist if a user misbehaves. Unfortunately, this approach results in pseudonymity for all users, and weakens the anonymity provided by the anonymizing network..Anonymous credential systems [10], [12] employ group signatures. Basic group signatures [1], [6], [15] allow servers to revoke a misbehaving user's anonymity by complaining to a group manager. Servers must query the group manager for every authentication, and thus lacks scalability. Traceable signatures [26] allow the group man-ager to release a trapdoor that allows all signatures generated by a particular user to be traced; such an approach does not provide the backward unlink ability [30] that we desire, where a user's accesses before the complaint remain anonymous. Backward unlink ability allows for what we call subjective blacklisting, where servers can blacklist users for whatever reason since the privacy of the blacklisted user is not at risk. In contrast, approaches without backward unlink ability need to pay careful attention to when and why a user must have all their connections linked, and users must worry about whether their behaviors will be judged fairly.

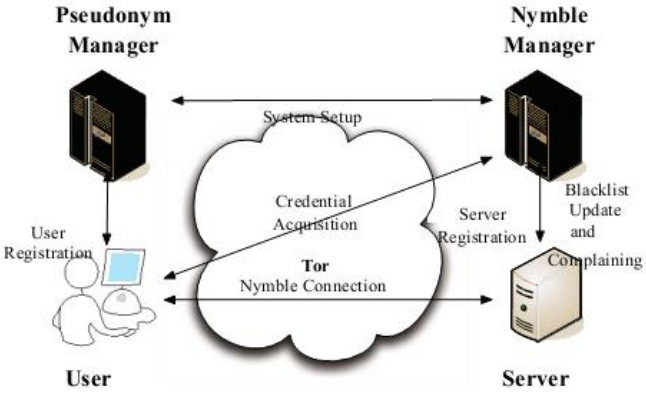


Fig 1 The Nymble system architecture

Subjective blacklisting is also better suited to servers

such as Wikipedia, where misbehaviors such as questionable edits to a webpage, are hard to define in mathematical terms. In some systems, misbehavior can indeed be defined precisely. For instance, double-spending of an ‘‘e-coin’’ is considered misbehavior in anonymous e-cash systems [8], [13], following which the offending user is deanonymized. Unfortunately, such systems work for only narrow definitions of misbehavior — it is difficult to map more complex notions of misbehavior onto ‘‘double spending’’ or related approaches [32].With dynamic accumulators [11], [31], a revocation operation results in a new accumulator and public parameters for the group, and all other existing users’ credentials must be updated, making it impractical. Verifier local revocation (VLR) [2], [7], [9] fixes this shortcoming by requiring the server (‘‘verifier’’) to perform only local updates during revocation. Unfortunately, VLR requires heavy computation at the server that is linear in the size of the blacklist. For example, for a blacklist with 1,000 entries each authentication would take tens of seconds,[2] a prohibitive cost in practice.

In contrast, our scheme takes the server about one millisecond per authentication, which is several thousand times faster than VLR. We believe these low overheads will incentivize servers to adopt such a solution when weighed against the potential benefit of anonymous publishing (e.g., whistle-blowing, reporting, anonymous tip lines, activism, and so on.).

II. AN OVERVIEW OF NYMBLE

We now present a high-level overview of the Nymble system, and defer the entire protocol description and security analysis to subsequent sections.

A. Resource based Blocking

To limit the number of identities a user can obtain (called the Sybil attack [19]), the Nymble system binds nymbles to resources that are sufficiently difficult to obtain in great numbers. For example, we have used IP addresses as the resource in our implementation, but our scheme generalizes to other resources such as email addresses, identity certificates, and trusted hardware. We address the practical issues related with resource-based blocking in Section 8, and suggest other alternatives for resources. We do not claim to solve the Sybil attack. This problem is faced by any credential system [19], [27], and we suggest some promising approaches based on resource-based blocking since we aim to create a real-world deployment

B. The Pseudonym Manager

The user must first contact the Pseudonym Manager (PM) and demonstrate control over a resource; for IP-address blocking, the user must 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.[6] 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 the PM’s duties are limited to mapping IP addresses (or other resources) to pseudonyms. As we will explain, the user contacts the PM only once per link ability window (e.g., once a day).

C. The 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). A user’s requests to the NM are therefore pseudonymous, and nymbles are generated using the user’s pseudonym and the server’s identity. These nymbles are thus specific to a particular user-server pair. Nevertheless, as long as the PM and the NM do not collude, the Nymble system cannot identify which user is connecting to what server; the NM knows only the pseudonym-server pair, and the PM knows only the user identity-pseudonym pair.

To provide the requisite cryptographic protection and security properties, the NM encapsulates nymbles within nymble tickets. Servers wrap seeds into linking tokens and 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.

D. Time

Nymble tickets are bound to specific time periods. As illustrated in Figure 2, time is divided into link ability windows of duration W, each of which is split into L time periods of duration T (i.e., $W = L * T$). 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 higher rates of anonymous authentication, while longer time periods allow servers to rate-limit the number of misbehaviors from a particular user before he or she is blocked. For example, T could be set to 5 minutes, and W to 1 day (and thus $L = 288$).

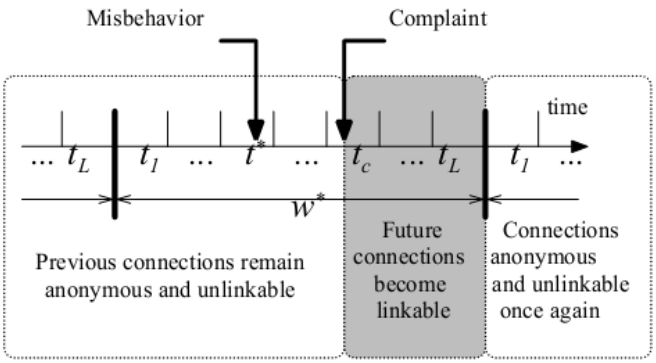


Fig 2 Misbehaving user's life cycle.

The linkability window allows for dynamism since resources such as IP addresses can get re-assigned and it is undesirable to blacklist such resources indefinitely, and it ensures forgiveness of misbehavior after a certain period of time. We assume all entities are time synchronized and can thus calculate the current linkability window and time period. An excellent style manual for science writers is [7] if the server complains in time period t_C about a user's connection in t^*, the user becomes linkable starting in t_C. The complaint in t_C can include nymble tickets from only t_{C-1} and earlier.

E. Blacklisting a User

If a user misbehaves, the server 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 connects and misbehaves at a server during time period t^* within linkability window w^*. The server later detects this misbehavior and complains to the NM in time period t_C ($t^* < t_C \leq t_L$) of the same linkability window w^*. As part of the complaint, the server presents the nymble ticket of the misbehaving user and obtains the corresponding seed from the NM. The server is then able to link future connections by the user in time periods $t_C, t_C + 1, \ldots, t_L$ of the same linkability window w^* to the complaint. Therefore, once the server has complained about a user, that user is blacklisted for the rest of the day, for example (the linkability window). Note that the user's connections in $t_1, t_2, \ldots, t^*, t^* + 1, \ldots, t_C$ remain unlinkable (i.e., including those since the misbehavior and until the time of complaint). Even though misbehaving users can be blocked from making connections in the future, the users' past connections remain unlinkable, thus providing backward unlinkability and subjective blacklisting.

F. Notifying the user of Blacklist Status

If a user misbehaves, the server 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 connects and misbehaves at a server during time period t^* within linkability window w^*. The server later detects this misbehavior and complains to the NM in time period t_C ($t^* < t_C \leq t_L$) of the same linkability window w^*. Since the blacklist is cryptographically signed by the NM, the authenticity of the blacklist is easily verified if the blacklist was updated in the current time period (only one update to the blacklist per time period is allowed). If the blacklist has not been updated in the current time period, the NM provides servers with ''daisies'' every time period so that users can verify the freshness of the blacklist (''blacklist from time period t_{old} is fresh as of time period t_{now}''). As discussed in Section 4.3.4, these daisies are elements of a hash chain, and provide a lightweight alternative to digital signatures. Using digital signatures and daisies, we thus ensure that race conditions are not possible in verifying the freshness of a blacklist. A user is guaranteed that he or she will not be linked if the user verifies the integrity and freshness of the blacklist before sending his or her nymble ticket.

III. Security Model

Nymble aims for four security goals. We provide informal definitions here; a detailed formalism can be found in our technical report [16], which explains how these goals must also resist coalition attacks.

A. Goals and Threats

An entity is honest when its operations abide by the system's specification. An honest entity can be curious: it attempts to infer knowledge from its own information (e.g., its secrets, state, and protocol communications). An honest entity becomes corrupt when it is compromised by an attacker, and hence reveals its information at the time of compromise, and operates under the attacker's full control, possibly deviating from the specification.

Blacklistability assures that any honest server can indeed block misbehaving users. Specifically, if an honest server complains about a user that misbehaved in the current linkability window, the complaint will be successful and the user will not be able to ''nymble-connect,'' i.e., establish a Nymble-authenticated connection, to the server successfully in subsequent time periods (following the time of complaint) of that linkability window.

Rate-limiting assures any honest server that no user can successfully nymble-connect to it more than once within any single time period.

Non-frameability guarantees that any honest user who is legitimate according to an honest server can nymble-connect to that server. This prevents an attacker from framing a legitimate honest user, e.g., by getting the user blacklisted for someone else's misbehavior. This

property assumes each user has a single unique identity. When IP addresses are used as the identity, it is possible for a user to ‘‘frame’’ an honest user who later obtains the same IP address. Non-frameability holds true only against attackers with different identities (IP addresses).

A user is legitimate according to a server if she has not been blacklisted by the server, and has not exceeded the rate limit of establishing Nymble-connections. Honest servers must be able to differentiate between legitimate and illegitimate users.

Anonymity protects the anonymity of honest users, regardless of their legitimacy according to the (possibly corrupt) server; the server cannot learn any more information beyond whether the user behind (an attempt to make) a nymble-connection is legitimate or illegitimate.

B. Trust Assumptions

We allow the servers and the users to be corrupt and controlled by an attacker. Not trusting these entities is important because encountering a corrupt server and/or user is a realistic threat. Nymble must still attain its goals under such circumstances.

Who	Whom	How	What
Servers	PM & NM	honest	Blacklistability & Rate-limiting
Users	PM & NM	honest	Non-frameability
Users	PM	honest	Anonymity
Users	NM	honest & not curious	Anonymity
Users	PM or NM	honest	Non-identification

With regard to the PM and NM, Nymble makes several assumptions on who trusts whom to be how for what guarantee. We summarize these trust assumptions as a matrix in Figure 3. Should a trust assumption become invalid, Nymble will not be able to provide the corresponding guarantee. For example, a corrupt PM or NM can violate Black-list ability by issuing different pseudonyms or dentials to blacklisted users.

A dishonest PM (resp. NM) can frame a user by issuing her the pseudonym (resp. credential) of another user who has already been blacklisted. To undermine the Anonymity of a user, a dishonest PM (resp. NM) can first impersonate the user by cloning her pseudonym (resp. credential) and then attempt to authenticate to a server—a successful attempt reveals that the user has already made a connection to the server during the time period. Moreover, by studying the complaint log, a curious NM can deduce that a user has connected more than once if she has been complained about two or more times. As already described in Section 2.3, the user must trust that at least the NM or PM is honest to keep the user and server identity pair private.

IV. PRELIMINARIES

A. Notation

The notation $a \in_R S$ represents an element drawn uniformly at random from non-empty set S. N_0 is the set of non-negative integers, and N is the set $N_0 \setminus \{0\}$. s[i] is the ith element of list s. s||t is the concatenation of (the unambiguous encoding of) lists s and t. The empty list is denoted by ∅. We sometimes treat lists of tuples as dictionaries. For example, if L is the list ((Alice, 1234), (Bob, 5678)), then L [Bob] denotes the tuple (Bob, 5678). If A is a (possibly probabilistic) algorithm, then A(x) denotes the output when A is executed given the input x. a: = b means that b is assigned to a.

B. Cryptographic Primitives

Nymble uses the following building blocks (concrete instantiations are suggested in Section 6):

- Secure cryptographic hash functions. These are one-way and collision-resistant functions that resemble random oracles [5]. Denote the range of the hash functions by H.
- Secure message authentication (MA) [3]. These consist of the key generation (MA.KeyGen) and the message authentication code (MAC) computation (MA.Mac) algorithms. Denote the domain of MACs by $\mathbf{M}$.
- Secure symmetric-key encryption (Enc) [4]. These consist of the key generation (Enc.KeyGen), encryption (Enc.Encrypt), and decryption (Enc.Decrypt) algorithms. Denote the domain of ciphertexts by Γ.
- Secure digital signatures (Sig) [22]. These consist of the key generation (Sig.KeyGen), signing (Sig.Sign), and verification (Sig.Verify) algorithms. Denote the domain of signatures by Σ.

C. Data Structures

Nymble uses several important data structures:

seeds evolve throughout a linkability window using a seed-evolution function $\mathbf{f}$; the seed for the next time period ($\text{seed}_{\mathbf{next}}$) is computed from the seed for the current time period ($\text{seed}_{\mathbf{cur}}$) as

$$\text{Seed}_{\text{next}} = \mathbf{f}\ (\text{seed}_{\mathbf{cur}}).$$

The nymble ($\text{nymble}_{\mathbf{t}}$) for a time period t is evaluated by applying the nymble-evaluation function g to its corresponding

$$\text{Seed}\ (\text{seed}_{\mathbf{t}}),\ \text{i.e.,}$$

$$\text{nymble}_{\mathbf{t}} = g\ (\text{seed}_{\mathbf{t}}).$$

The NM sets seed_0 to a pseudo-random mapping of the user’s pseudonym pnym, the (encoded) identity

Sid of the server (e.g., domain name), the linkability window w for which the seed is valid, and the NM's Secret key seedKey$_N$. Seeds are therefore specific to user-server-window combinations. As a consequence, a seed is useful only for a particular server to link a particular user during a particular linkability window.

Algorithm 1 PMCreatePseudonym

Input: (uid, w) ∈ H × N
Persistent state: pmState ∈ S_P
Output: pnym ∈ P
1: Extract nymKey$_{P,\text{macKeyNP}}$ from pmState 2: nym: = MA.Mac (uid||w, nymKeyP)
3: mac: = MA.Mac (nym||w, macKeyNP) 4:
return pnym: = (nym, mac)

Algorithm 2 NMVerifyPseudonym

Input: (pnym, w) ∈ P × N
Persistent state: nmState ∈ S_N
Output: b ∈ {**true**, **false**}
1: Extract macKey$_{NP}$ from nmState 2:
(nym, mac):= pnym
3: **return** mac $\stackrel{?}{=}$ MA.Mac (nym||w, macKeyNP)

D. Communication Channels

Nymble utilizes three types of communication channels, namely type-Basic, -Auth and -Anon (Figure 6). We assume that a public-key infrastructure (PKI) such as X.509 is in place, and that the NM, the PM and all the servers in Nymble have obtained a PKI credential from a well-established and trustworthy CA. (We stress that the users in Nymble, however, need not possess a PKI credential.) These entities can thus realize type-Basic and type-Auth channels to one another by setting up a TLS[8] connection using their PKI credentials. All users can realize type-Basic channels to the NM, the PM and any server, again by setting up a TLS connection. Additionally, by setting up a TLS connection over the Tor anonymizing network,[9] users can realize a type-Anon channel to the NM and any server.

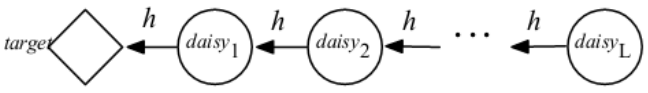


Fig.5. Given daisy$_i$ it is easy to verify the freshness of the blacklist by applying h i times to obtain target. Only the NM can compute the next daisy$_{i+1}$ in the chain.

Algorithm 7 NMSignBL

Input: (sid, t, w, target, blist) ∈ H × N^2 × H × B_n, n ∈ N_0
Persistent state: nmState ∈ S_N
Output: cert ∈ C
1: Extract macKey$_N$, signKey$_N$ from keys in nmState 2:
content := sid||t||w||target||blist
3: mac:= MA.Mac(content, macKey$_N$) 4:
sig:= Sig.Sign(content, signKey$_N$) 5:
daisy := target
6: **return** cert := (t, daisy, t, mac, sig)

V. Our Nymble Construction

A. System steup

During setup, the NM and the PM interact as follows.

1) The NM executes NMInitState() (see Algorithm 10) and initializes its state nmState to the algorithm's output.
2) The NM extracts macKey$_{NP}$ from nmState and sends it to the PM over a type-Auth channel. macKey$_{NP}$ is a shared secret between the NM and the PM, so that the NM can verify the authenticity of pseudonyms issued by the PM.
3) The PM generates nymKey$_P$ by running Mac.KeyGen() and initializes its state pmState to the pair (nymKey$_P$, macKey$_{NP}$).
4) The NM publishes verKey$_N$ in nmState in a way that the users in Nymble can obtain it and verify its integrity at any time(eg during registration).

B. Server Registration

Algorithm 8 VerifyBL

Input: (sid, t, w, blist, cert) ∈ H × N^2 × B_n × C, n ∈ N_0
Output: b ∈ {**true**, **false**}
1: (t_d, daisy, t_s, mac, sig) := cert 2:
if $t_d = t \vee t_d < t_s$ **then**
3: **return** **false**
4: target := $h^{(t_d - t_s)}$(daisy)
5: content := sid||t_s||w||target||blist
6: **return** Sig.Verify(content, sig, verKey$_N$)

Algorithm 9 NMVerifyBL

Input: (sid, t, w, blist, cert) ∈ H × N^2 × B_n × C, n ∈ N_0
Persistent state: nmState ∈ S_N
Output: b ∈ {**true**, **false**}
1-6: Same as lines 1–6 in VerifyBL
7: Extract macKey$_N$ from keys in nmState
8: **return** mac = MA.Mac(content, macKey$_N$)

To participate in the Nymble system, a server with identity sid initiates a type-Auth channel to the NM, and registers with the NM according to the Server Registration protocol below. Each server may register at most once in any linkability window.

1) The NM makes sure that the server has not already registered: If $(Sid, \cdot, \cdot) \in$ nmEntries in its nmState, it terminates with failure; it proceeds otherwise.
2) The NM reads the current time period and linkability window as t_{now} and w_{now} respectively, and then obtains a svrState by running (see Algorithm 11)

$\text{NMRegisterServer}_{\text{nmState}}(sid, t_{now}, w_{now})$.

3) The NM appends svrState to its nmState, sends it to the Server, and terminates with success.
4) The server, on receiving svrState, records it as its state, and terminates with success.

In svrState, macKey_{NS} is a key shared between the NM and the server for verifying the authenticity of nymble tickets; $\text{time}_{\text{lastUpd}}$ indicates the time period when the blacklist was last updated, which is initialized to t_{now}, the current time period at registration.

C. User Registration

A user with identity uid must register with the PM once each linkability window. To do so, the user initiates a type-Basic channel to the PM, followed by the User Registration protocol described below.

1) The PM checks if the user is allowed to register. In our current implementation the PM infers the registering user's IP address from the communication channel, and makes sure that the IP address does not belong to a known Tor exit node. If this is not
2) Otherwise, the PM reads the current link ability window as w_{now}, and runs with success.

$$\text{pnym} := \text{PMCreatePseudonym}_{\text{pmState}}(\text{uid}, w_{now}).$$

The PM then gives pnym to the user, and terminates
3) The user, on receiving pnym, sets her state usrState to (pnym, $\emptyset$), and terminates with success.

D. Credential acquisition

To establish a Nymble-connection to a server, a user must provide a valid ticket, which is acquired as part of a credential from the NM. To acquire a credential for server sid during the current linkability window, a registered user initiates a type-Anon channel to the NM, followed by the Credential Acquisition protocol below.

1) The user extracts pnym from usrState and sends the pair (pnym, sid) to the NM.
2) The NM reads the current linkability window as w_{now}. It makes sure the user's pnym is valid: If

$$\text{NMVerifyPseudonym}_{\text{nmState}}(\text{pnym}, w_{now})$$

returns **false**, the NM terminates with failure; it proceeds otherwise.
3) The NM runs which returns a credential cred. $\text{NMCreateCredential}_{\text{nmState}}(\text{pnym}, \text{sid}, w_{now})$,The NM sends cred to the user and terminates with success.
4) The user, on receiving cred, creates usrEntry: = (sid, cred, **false**), appends it to its state usrState, and terminates with success.

Type	Initiator	Responder	Link
Basic	–	Authenticated	Confidential
Auth	Authenticated	Authenticated	Confidential
Anon	Anonymous	Authenticated	Confidential

Fig. 6. Different types of channels utilized in Nymble.

Algorithm 10 NMInitState

Output: nmState $\in S_N$

1: $\text{macKey}_{NP} := \text{Mac.KeyGen}()$ 2:
$\text{macKey}_N := \text{Mac.KeyGen}()$ 3:
$\text{seedKey}_N := \text{Mac.KeyGen}()$
4: $(\text{encKey}_N, \text{decKey}_N) := \text{Enc.KeyGen}()$ 5:
$(\text{signKey}_N, \text{verKey}_N) := \text{Sig.KeyGen}()$
6: $\text{keys} := (\text{macKey}_{NP}, \text{macKey}_N, \text{seedKey}_N,$
7: $\text{encKey}_N, \text{decKey}_N, \text{signKey}_N, \text{verKey}_N)$ 8:
nmEntries := $\emptyset$
9: **return** nmState := (keys, nmEntries)

VI PERFORMANCE EVALUATIONS

We implemented Nymble and collected various empirical performance numbers, which verify the linear (in the number of ‘‘entries’’ as described below) time and space costs of the various operations and data structures.

A. Implementation and experimental setup

We implemented Nymble as a C++ library along with Ruby and JavaScript bindings. One could, however, easily compile bindings for any of the languages (such as Python, PHP, and Perl) supported by the Simplified Wrapper and Interface Generator (SWIG) for example. We utilize Open SSL for all the cryptographic primitives. We use SHA-256 for the cryptographic hash functions; HMAC-SHA-256 for the message authentication MA; AES-256 in CBC-mode for the symmetric encryption Enc; and 2048-bit RSASSA -PSA for the digital signatures

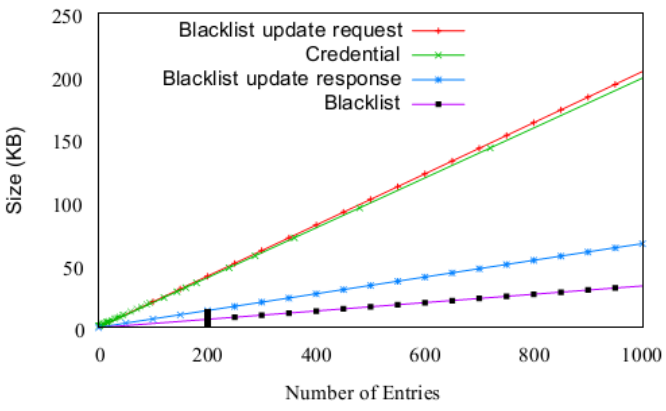


Fig.7. The marshaled size of various Nymble data structures. The X-axis refers to the number of entries—complaints in the blacklist update request, tickets in the credential, tokens and seeds in the blacklist update response, and nymbles in the blacklist.

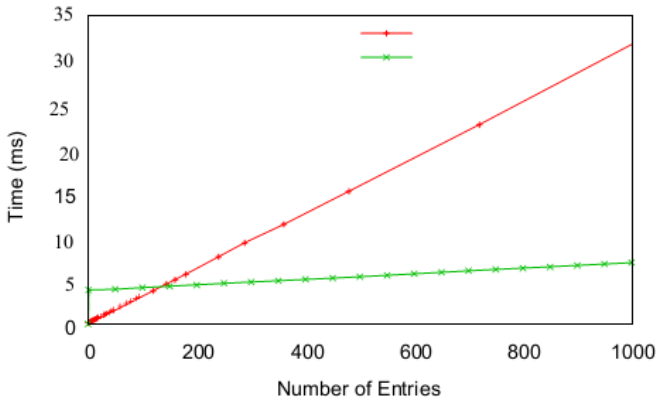
We chose RSA over DSA for digital signatures because of its faster verification speed—in our system, verification occurs more often than signing.

We evaluated our system on a 2.2 GHz Intel Core 2 Duo Macbook Pro with 4 GB of RAM. The PM, the NM, and the server were implemented as Mongrel (Ruby's version of Apache) servers. The user portion was implemented as a Firefox 3 extension in JavaScript with XPCOM bindings to the Nymble C++ library. For each experiment relating to protocol performance, we report the average of 10 runs. The evaluation of data-structure sizes is the byte count of the marshalled data structures that would be sent over the network.

B. Experimental Result

The X-axis represents the number of entries in each data structure—complaints in the blacklist update request, tickets in the credential (equal to L, the number of time periods in a linkability window), nymbles in the black-list, tokens and seeds in the blacklist update response, and nymbles in the blacklist. For example, a linkability window of 1 day with 5 minute time periods equates to L = 288.[11] The size of a credential in this case is about 59 KB. The size of a blacklist update request with 50 complaints is roughly 11 KB, whereas the size of a blacklist update response for 50 complaints is only about 4KB. The size of a blacklist with 500 nymbles is 17 KB.

In general, each structure grows linearly as the number of entries increases. Credentials and blacklist update requests grow at the same rate because a credential is a collection of tickets which is more or less what is sent.



Blacklist updates take several milliseconds and credentials can be generated in 9 ms for the suggested parameter of L=288.

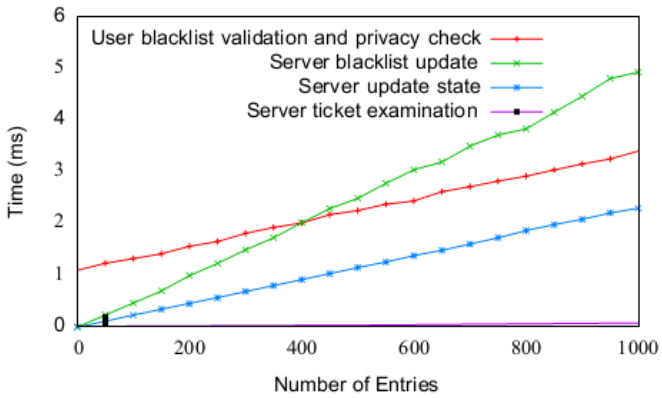


Fig.8. Nymble's performance at (a) the NM and (b) the user and the server when performing various protocols.

as a complaint list when the server wishes to update its blacklist. In our implementation we use Google's Protocol Buffers to (un)marshal these structures because it is cross-platform friendly and language-agnostic.

Figure 8(a) shows the amount of time it takes the NM to perform various protocols. It takes about 9 ms to create a credential when L = 288. Note that this protocol occurs only once every linkability window for each user wanting to connect to a particular server. For blacklist updates, the initial jump in the graph corresponds to the fixed overhead associated with signing a blacklist. To execute the update blacklist protocol with 500 complaints it takes the NM about 54 ms. However, when there are no complaints, it takes the NM on average less than a millisecond to update the daisy.

Figure 8(b) shows the amount of time it takes the server and user to perform various protocols. These protocols are relatively inexpensive by design, i.e., the amount of computation performed by the users and servers should be minimal. For example, it takes less than 3 ms for a user to execute a security check on a blacklist with 500 nymbles. Note that this figure includes signature verification as well, and hence the fixed-cost overhead exhibited in the graph. It takes less than a millisecond for a server to perform authentication of a ticket against a blacklist with 500 nymbles. Every time period (e.g., every 5 minutes), a server must update its state and blacklist. Given a linking list with 500 entries, the server will spend less than 2 ms updating the linking
list. If the server were to issue a blacklist update request with 500 complaints, it would take less than 3 ms for the server to update its blacklist.

VII SECURITY ANALYSIS

Theorem 1: Our Nymble construction has Blacklistability, Rate-limiting, Non-frameability and anonymity provided that the trust assumption in session 3.2 hold true. and the cryptographic primitives used are secure. We summarize the proof of Theorem 1. Please refer to our technical

report [16] for a detailed version.

A. Blacklistability

An honest PM and NM will issue a coalition of c unique users at most c valid credentials for a given server. Because of the security of HMAC, only the NM can issue valid tickets, and for any time period the coalition has at most c valid tickets, and can thus make at most c connections to the server in any time period regardless of the server's blacklisting. It suffices to show that if each of the c users has been blacklisted in some previous time period of the current linkability window, the coalition cannot authenticate in the current time period k^*.

Assume the contrary that connection establishment k^* using one of the coalition members' $ticket^*$ was successful even though the user was blacklisted in a previous time period $k-$. Since connection establishments $k-$ and k^* were successful, the corresponding tickets $ticket^-$ and $ticket^*$ must be valid. Assuming the security of digital signatures and HMAC, an honest server can always contact an honest NM with a valid ticket and the NM will successfully terminate during the blacklist update. Since the server blacklisted the valid $ticket^-$ and updates its linking list honestly, the ServerLinkTicket will return fail on input $ticket^*$, and thus the connection k^* must fail, which is a contradiction.

B. Non-Frameability

Assume the contrary that the adversary successfully framed honest user i^* with respect to an honest server in time period t^*, and thus user i^* was unable to connect in time period t^* using $ticket^*$ even though none of his tickets were previously blacklisted. Because of the security of HMAC, and since the PM and NM are honest, the adversary cannot forge tickets for user i^*, and the server cannot already have seen $ticket^*$; it must be that $ticket^*$ was linked to an entry in the linking list. Thus there exists an entry ($seed^*$, $nymble^*$) in the server's linking list, such that the nymble in $ticket^*$ equals $nymble^*$. The server must have obtained this entry in a successful blacklist update for some valid $ticket_b$, implying the NM had created this ticket for some user i.

If $i \neq i^*$, then user i's $seed_0$ is different from user i^*'s $seed_0$ so long as the PM is honest, and yet the two $seed_0$'s evolve to the same $seed^*$, which contradicts the collision-resistance property of the evolution function. Thus we have $i = i^*$. But as already argued, the adversary cannot forge i^*'s $ticket_b$, and it must be the case that i^*'s $ticket_b$ was blacklisted before t^*, which contradicts our assumption that i^* was a legitimate user in time t^*.

C. Anonymity

We show that an adversary learns only that some legitimate user connected or that some illegitimate user's connection failed, i.e., there are two anonymity sets of legitimate and illegitimate users.

Distinguishing between two illegitimate users We argue that any two chosen illegitimate users out of the control of the adversary will react indistinguishably. Since all honest users execute the Nymble-connection Establishment protocol in exactly the same manner up until the end of the Blacklist validation stage (Section 5.5.1), it suffices to show that every illegitimate user will evaluate safe to **false**, and hence terminate the protocol with failure at the end of the Privacy check stage (Section 5.5.2). For an illegitimate user (attempting a new connection) who has already disclosed a ticket during a connection establishment earlier in the same time period, ticket. Disclosed for the server will have been set to true and safe is evaluated to **false** during establishment k^*. An illegitimate user who has not disclosed a ticket during the same time period must already be blacklisted. Thus the server complained about some previous $ticket^*$ of the user. Since the NM is honest, the user's $nymble^*$ appears in some previous blacklist of the server. Since an honest NM never deletes entries from a blacklist, it will appear in all subsequent blacklists, and safe is evaluated to **false** for the current blacklist. Servers cannot forge blacklists or present blacklists for earlier time periods (as otherwise the digital signature would be forgeable, or the hash in the daisy chain could be inverted).

Distinguishing between two legitimate users The authenticity of the channel implies that a legitimate user knows the correct identity of the server and thus boolean ticket. Disclosed for the server remains **false**. Furthermore, UserCheckIfBlacklisted returns **false** (assuming the security of digital signatures) and safe is evaluated to **true** for the legitimate user.

Now, in the ticket presented by the user, only nymble and ctxt are functions of the user's identity. Since the adversary does not know the decryption key, the CCA2 security of the encryption implies that ctxt reveals no information about the user's identity to the adversary. Finally, since the server has not obtained any seeds for the user, under the Random Oracle model the nymble presented by the user is indistinguishable from random and cannot be linked with other nymbles presented by the user. Furthermore, if and when the server complains about a user's tickets in the future, the NM ensures that only one real seed is issued (subsequent seeds corresponding to the same user are random values), and thus the server cannot distinguish between legitimate

users for a particular time period by issuing complaints in a future time period.

D.Across multiple linkability windows

With multiple linkability windows, our Nymble construction still has Accountability and Nonframeability because each ticket is valid for and only for a specific linkability window; it still has Anonymity because pseudonyms are an output of a collision-resistant function that takes the linkability window as input.

VIII CONCLUSIONS

We have proposed and built a comprehensive credential system called Nymble, which can be used to add a layer of accountability to any publicly known anonymizing network. Servers can blacklist misbehaving users while maintaining their privacy, and we show how these properties can be attained in a way that is practical, efficient, and sensitive to needs of both users and services.

We hope that our work will increase the mainstream acceptance of anonymizing networks such as Tor, which has thus far been completely blocked by several services because of users who abuse their anonymity.

REFERENCE

[1] E. Bresson and J. Stern. Efficient Revocation in Group Signatures. In Public Key Cryptography, LNCS 1992, pages 190–206. Springer, 2001.

[2] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In EUROCRYPT, LNCS 2045, pages 93–118. Springer, 2001.

[3] J. Camenisch and A. Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In CRYPTO, LNCS 2442, pages 61–76. Springer, 2002.

[4] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In CRYPTO, LNCS 3152, pages 56–72. Springer, 2004.

[5] D. Chaum. Blind Signatures for Untraceable Payments. In CRYPTO, pages 199–203, 1982.

[6] D. Chaum. Showing Credentials without Identification Transferring Signatures between Unconditionally Unlinkable Pseudonyms. In AUSCRYPT, LNCS 453, pages 246–264. Springer, 1990.

[7] D. Chaum and E. van Heyst. Group Signatures. In EUROCRYPT, pages 257–265, 1991.

[8] C. Cornelius, A. Kapadia, P. P. Tsang, and S. W. Smith. Nymble: Blocking Misbehaving Users in Anonymizing Networks. Technical Report TR2008-637, Dartmouth College, Computer Science, Hanover, NH, December 2008..

[9] L. Nguyen. Accumulators from Bilinear Pairings and Applications. In CT-RSA, LNCS 3376, pages 275–292. Springer, 2005.

[10] I. Teranishi, J. Furukawa, and K. Sako. k-Times Anonymous Authentication (Extended Abstract). In ASIACRYPT, LNCS 3329, pages 308–322. Springer, 2004.

[11] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable Anonymous Credentials: Blocking Misbehaving Users Without TTPs. In CCS '07: Proceedings of the 14th ACM conference on Computer and communications security, pages 72–81, New York, NY, USA, 2007. ACM.

[12] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. PEREA: Towards Practical TTP-Free Revocation in Anonymous Authentication. In ACM Conference on Computer and Communications Security, pages 333–344. ACM, 2008.

[13] C. Cornelius, A. Kapadia, P. P. Tsang, and S. W. Smith. Nymble: Blocking Misbehaving Users in Anonymizing Networks. Technical Report TR2008-637, Dartmouth College, Computer Science, Hanover, NH, December 2008.

[14] I. Damgard. Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals. In CRYPTO, LNCS 403, pages 328–335. Springer, 1988.

[15] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In Usenix Security Symposium, pages 303–320, Aug. 2004.