Wheel of Trust: A Secure Framework for Overlay-based Services

Guor-Huar Lu and Zhi-Li Zhang
University of Minnesota
luxx0137@umn.edu, zhzhang@cs.umn.edu

Abstract— **The recent advances of distributed hash tables (DHTs) facilitate the development of highly scalable and robust network applications and services. However, with applications and services each employing their own DHTs that perform essentially the same tasks, an open infrastructure providing the core DHT functionalities for these applications and services would represent a cost-effective solution. In this paper we present a generic secure framework for deploying secure overlay-based applications/services. We combine DHTs and identity-based encryption (IBE) to develop a novel architecture that is scalable and robust against man-in-the-middle attacks. We also develop an innovative mechanism called "*Wheel of Trust*" that secures our framework against insider attacks. Based on the proposed architecture, we present some preliminary evaluation results from a prototype implementation.**

I. Introduction

The recent advances of distributed hash tables (DHTs [9], [4], [6]) facilitate the development of highly scalable and robust applications and services. However, most of these applications/services deploy their own DHT that essentially perform the same tasks. We believe that a common platform offering the core DHTs functionalities can greatly reduce the deployment cost and effort for these application/services. In addition, such a platform should offer the following security guarantees for these applications/services: 1) the framework needs to be robust against "*man-in-the-middle*" attacks where attackers lie in between clients and the framework (and services deployed on top of the framework). The framework should prevent attackers from injecting bogus information to the client or into the service, e.g., binding poisoning; and 2) the framework needs to be robust against insider attacks, in which one or more infrastructure nodes may be compromised by attackers. These malicious insiders aim to alter the information in their possession in order to gain most benefit from the service, e.g., phishing-like attacks. Unfortunately, such security guarantees cannot be easily achieved in a DHT-based system due to the completely distributed and de-centralized nature, even when combined with more traditional security mechanism such as RSA and PKI. In particular, the extraordinary amount of cooperation required between peers makes DHT-based system especially vulnerable against insider attacks, where some malicious nodes are part of the system. The combination of security, scalability and robustness makes designing such a framework a challenging task.

In this paper we develop a novel framework that offers a common infrastructure for the deployment of secure overlay-based applications and services. The key novelty of our framework lies in combining identity-based encryption (IBE) with DHT techniques. DHT affords our framework with scalability and robustness, while the combination of DHT and IBE makes our framework robust against man-in-the-middle attack. We also design an innovative mechanism that further enhanced our framework against insider attacks. Our framework uses a two-level architecture: at the core of the system are a set of special nodes that are highly fault tolerant, and play a critical role in ensuring the security of our system, which includes *private* (decryption) key generation [1]. These core nodes do not interact with users or hosting any services. These functions are performed by typical infrastructure nodes, which form a ring structure using Chord [9]. A user wants to store a key-value pair (k, v) to the system (and the service deployed on top of the framework) encrypts its request using a hashed id derived from the key k, and sends the encrypted request to the system; only the corresponding node in the system that owns the hashed id can decrypt the request, perform the appropriate actions and reply to the user in a secure manner. The key challenge in designing such a framework is to ensure that the core nodes only issues private keys to the correct node owning the hashed id, i.e., a node has to prove to the core nodes that the id space it owns indeed contains the hashed id. For this, we develop an innovative mechanism called "*Wheel of Trust*" (WoT), that maintains the node to id space mapping, and makes such information externally verifiable. In the next section we present what is IBE and motivate why IBE is uniquely suitable in building such a framework. We discuss the overall architecture and operations of our framework in Section III. In Section IV we present some preliminary results from a prototype implementation of our framework. Section V concludes the paper.

II. Background and Motivation

In this section we briefly introduce identity-based encryption (IBE), and motivate why it is particularly suitable in designing such a secure framework.

A. *Identity-Based Encryption*

Originally proposed as a means to simplify certificate management in email systems [1], IBE allows any arbitrary string (e.g., email address or other user identifiers) to be used as the public key. The corresponding private key is generated by a central authority (called the *private key generator*, or PKG in short). For example, if Bob wants to send a secret email to Alice, he encrypts the email using Alice's email address (i.e., her identity) as the public key. For decryption, Alice retrieves her private key from the PKG and subsequently uses it to decrypt the email. Hence in IBE, as long as Bob knows the identity (here email address) of Alice, he can send an encrypted email to Alice. Whereas, in the conventional public-key cryptosystems such as RSA, Bob first needs to obtain the public key of Alice, and has a way to ensure that the public key does indeed belong to Alice – hence a certificate system (e.g., a PKI – public key infrastructure) for validating public keys is needed. IBE obliviates the need for such a certificate system. In addition, IBE enables what we call *asynchronous* secret communication that is crucial to the construction of our framework: it allows one to establish a *forward* secret communication channel (using IBE for encryption) from a sender to a target identity, where the sender does not need to have prior knowledge or contact with the corresponding receiver; the private key corresponding to the target only needs to be generated by the PKG on-demand when the receiver requests for it. In contrast, in the conventional public key systems such as RSA, a public-private key pair must be pre-generated for each receiver before communications can proceed. In addition, IBE also allows the generation of digital signatures using identity-based signature schemes (IBS [2]). In IBS, when Alice wants to send a message to Bob, she simply generates a digital signature for the message using her private key. When Bob receives the message, he can ensure the integrity and authenticity of the message by verifying the signature using Alice's identity, i.e, Alice's email address. Note that Bob does not need to retrieve Alice's public key since Alice's identity (her email address) *is* the public key. For a detailed description of IBE and IBS, we refer interested readers to [1], [2].

B. Challenges and Problem Statement

We aim to build a generic secure infrastructure that acts as a common framework for deploying various applications and services. The goal is for the framework to offer a common set of functionalities to these applications and services while at the same time providing a certain level of security. We use DHT to build our framework for scalability and robustness, so that large scale applications and services can be deployed. As an infrastructure framework, our system is not built using typical "*peer-to-peer*" (P2P) end-hosts. In fact, only authorized and trusted nodes can join the infrastructure. In addition, being an infrastructure-based framework we assume node dynamics (nodes join/leave) occur far less frequently than typical P2P networks. In this paper we use Chord as our choice of DHT but our framework can be easily extended to other DHTs. In order to allow applications and services deployed on top of the framework to be secure, our framework needs to offer the following security guarantees: 1) the infrastructure must be robust against *"man-in-the-middle" (MITM)* attacks, in which attackers lie in between users and the framework (and applications and services deployed on top of the framework); and 2) the framework must be able to withstand insider attacks, in which a minority of nodes constituting the system have been compromised by attackers.

So why is designing a DHT-based framework with the postulated security requirements difficult? To understand where the problem lies, we need to examine the most basic DHT operations: `put` and `get`. In a nutshell, what DHT offers is a look-up service: a user `puts` (stores) a key-value pair (k, v)[1] into the system by hashing the key k into a hashed id id_k, where v is the data the user wishes to store and k is an identifier associated with v. This (k, v) pair is then stored at a node n that is responsible for id_k, i.e., n *owns* id_k. Similarly, if a user wants to `get` this data v, it generates id_k from k and the node n will then serve the request. We make two observations for these basic `put/get` operations. First, there is no *verifiable associations* between id_k and the node n serving the request. Second, for DHT operations to be generic, there cannot be restrictions on the choice of ks and vs. This couples with the fact that the system usually does not have prior knowledge of the user before communication makes verifying the association between the key-value pair difficult. These observations lead to two implications: 1) an attacker can easily impersonate the node n and serve users' requests on behave of n and vice versa; and 2) an attacker can easily replace the value in a key-value pair from the user to the system with a bogus one and vice versa. This makes it easy to perform MITM attacks as well as insider attacks, as the user has no way of knowing if its request is served by the right node in the system, and neither the user nor the system can be sure if the received key-value pair is the correct one.

One may argue that such issues can be easily addressed with traditional public key crypto-system such as RSA (and in some cases a public key infrastructure (PKI)). However, even when combined with a PKI, a completely distributed DHT-based infrastructure cannot offer the necessary security guarantees we postulated earlier. A case in point is the authenticate `put/get` operations[2] in OpenDHT [5]. These authenticated `put/get` operations are not robust against the MITM attacks, as the attacker may intercept the `put` message sent by a user, replace the public key with its own, and re-sign the message, and send it to the service, which has no way to verify the bogus `put` message. In addition, the user has no way of verifying that its request is indeed served by the correct node in the system. We note that even if PKI is used and a certificate is submitted along with the `put` message, an attacker can still replace the public key and the certificate with its own in the `put` message unless the user's identity is part of the key k. Moreover, for a user to lookup (k, v), it must know *a priori* the

[1]Note that the format and content of k and v depend on the application and service deployed.

[2]Under the authenticated put/get mode, each owner has a public/private key pair, denoted K_P and K_S, respectively. To insert a binding of key-value (k, v), an owner sends the following to the service: k, v, K_P, a nonce n , an expiration time t, and $\sigma = \{H(k, v, n, t)\}_{K_S}$, where $\{X\}_{K_S}$ denotes the digital signing of X with K_S and H is a secure hash function (e.g., SHA-1). To retrieve the binding, a querier sends the following $\{k, H(K_P)\}$ to the service and the service returns $\{(v, n, t, \sigma)\}$.

public key associated with k, which in itself involves another lookup step that needs to be secured.

We now illustrate how IBE can be used to address MITM attacks. The basic ideas are as follows. For a user u who wants to `put` a key-value pair (k, v) to the system, it employs IBE to encrypt (k, v), together with a *secret symmetric key* (for a pre-specified *symmetric* encryption scheme such as AES) and a *nonce* (e.g., a random number), using the hashed id id_k, and sends the encrypted `put` message to the service with id_k as the target. The "root node" of id_k (denoted as $root(id_k)$) retrieves the private key corresponding to id_k from the PKG and decrypts the message. If the `put` operation is successful, $root(id_k)$ returns a confirmation message containing the nonce encrypted with the secret symmetric key. Otherwise, a failure message is generated. From the confirmation message, the user can verify that the `put` operation is successfully completed. Likewise, for a user to `get` this key-value pair, it simply generates a `get` message together with a secret symmetric key and a nonce encrypted using id_k, and sends the encrypted `get` message to the service with id_k as the target. The service returns a reply message containing (k, v) and the nonce encrypted using the symmetric secret key. By successfully decrypting the reply message and verifying the nonce, the user can be assured that the returned value is indeed authentic. Any imposter interposed between the users and the service will not be able to tamper or inject bogus information into the service, nor return such to users. However, even though IBE is particularly suited in building a framework that is robust against MITM attacks, we still need an effective mechanism for authenticating whether or not a given node owns a particular id_k before issuing the corresponding private key. This authentication mechanism is crucial to the overall security of the system, as it directly determines whether or not secure channels can be established between users and the correct system node. We observe that in a DHT system such as Chord, a node n's id space is directly determined by its id and its neighbors' id, i.e., n's *neighboring relationship* dictates the id space it owns. Based on this, we design an innovative mechanism called "*Wheel of Trust*" that securely *bonds* this neighboring relationship and makes it externally verifiable, even if there are malicious insiders. This mechanism combines with secure channels created using IBE makes our framework robust against MITM attacks and insider attacks.

C. Related Work

Internet Indirection Infrastructure (i3 [8]) is another example of open infrastructure that supports multiple services on a common platform. i3 offers a rendezvous-based communication that decouples the act of sending from the act of receiving, and efficiently supports a wide variety of fundamental communication services. Compare to both OpenDHT and i3, we pay special attention on security issues and specifically incorporate necessary mechanisms into our design. In addition to IBE-based email systems, IBE has also been used in several networked applications and systems such as secure opportunistic communications in disconnected networks [7].

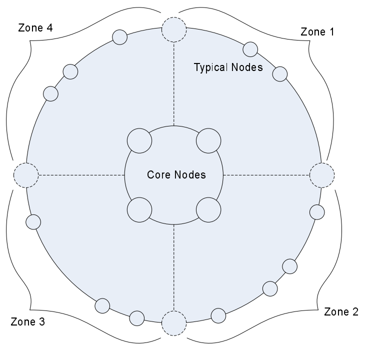


Fig. 1. Two-level architecture of our framework. At the center of the system are a set of core nodes act as PKGs each managing a portion of the id space.

III. Architecture and Operations

In this section we describe the overall architecture and operations of our framework. Our framework uses a two-level architecture as shown in Fig. 1. At the center of the system are a set of trusted core nodes act as private key generators (PKGs) in IBE. We assume they are highly fault tolerant and always available. Each core node[3] manages an equal portion of the id space in the system (called a *zone*) and is responsible for generating and issuing private keys to other nodes in the zone. Core nodes do not directly interact with users of the system or host any services. Instead, a set of general infrastructure nodes (refer to as *nodes* from now on) form an outer ring using Chord are used to host actual services and applications and handle user requests. We assume these nodes are trusted but are not as reliable as core nodes. In addition, malicious attackers can potentially compromise some of these nodes to perform insider attacks. We assume each node n has a pre-assigned unique id id_n and a private key corresponding to its id obtained through a secure offline channel. This private key is used to generate signatures using an *identity-based signature* (IBS) scheme [2] and allows other nodes in the system to authenticate n and confirm n's ownership of id_n. The security of our system relies on two main components: a) *Secure Channel Layer (SCL)*. SCL is responsible for creating secure channels between users and the system using IBE, and protects our system against MITM attacks; b) *Wheel of Trust (WoT)*. WoT is responsible for maintaining the association between a node and the id space the node is responsible. In other words, WoT acts as an authentication mechanism between nodes and PKGs and is effective even if some of the nodes in the system are compromised.

A. Secure Channel Layer

The key idea behind SCL is to use IBE to establish secure channels between users and the system, thereby protecting subsequent message exchanges against MITM attacks. When a user u wants to `put` a key-value pair (k, v) to the system, u first obtains a hashed id $id_k = H[k]$ where H is a secure hash function such as SHA-1. The SCL `put` operation (`scl_put`) generates a request by encrypting the following with id_k using

[3]We use the term core node and PKG interchangeably.

IBE: (k, v), a secret symmetric key s_k, and a random nonce n_u. This encrypted request is delivered to the system with id_k as the target. When a node n that is responsible for id_k, i.e., n is the root of id_k, receives the request, n retrieves a private (decryption) key for id_k from one of the PKG by providing some authentication information to prove it indeed *owns* id_k[4]. When n obtained the private key from the PKG, it decrypts and processes the user request. A `scl_response` message is then generated by encrypting the appropriate response and the nonce n_u with the secret symmetric key s_k using a pre-specified symmetric encryption algorithm such as AES. When the user receives the response, it can decrypt the response message using s_k and verifies the nonce to ensure that the operation is successfully completed. The `scl_get` operation works in a similar manner except no value v is encrypted. The combination of IBE and symmetric crypto-mechanism allows us to establish two-way secure channels between users and the system. This way, all communications are protected and attackers cannot tamper or inject bogus information.

B. Wheel of Trust

Although SCL enables easy establishment of secure channels between users and the system, it is evident that we need an authentication mechanism to ensure PKGs only issue private keys to the correct node. In other words, for a node n requesting a private key for some id id_k, the PKG must be able to determine if n indeed owns the id space containing id_k. We observe that a node n's id space is determined by its id and its neighbors' ids, i.e., n's *neighboring relationship* dictates the id space range it is responsible. Our mechanism, called *Wheel of Trust (WoT)*, maintains and secures node-to-id space mapping and makes such information externally verifiable, thus acting as an authentication mechanism between nodes and PKGs. For each node n, WoT verifies and stores n's neighboring information (we call this n's neighbor record $NR(n)$[5]) at a random node in the system (we call this node the *shadow* of n). Once $NR(n)$ is stored, we issue n a *neighbor certificate NC(n)* that allows n to prove its id space range to an external entity. $NC(n)$ contains $NR(n)$ and a time stamp, and is signed using an *authorization key* possessed by n's shadow so that n cannot forge this certificate on its own. When a node m lookups some id id_k in which n is responsible, in addition to the typical replies, n also returns $NC(n)$ back to m so m can verify the certificate to ensure that $NR(n)$ contained in $NC(n)$ is authentic, thus ensuring n is indeed responsible for id_k.

WoT consists of three components: *selection*, *rotation*, and *update*. The selection algorithm determines which node in the system is responsible for storing a given node n's neighboring information. To prevent node n collude with its shadow, the rotation algorithm periodically changes n's shadow. The update algorithm is used to ensure neighboring information is correctly updated whenever node dynamics (join/leave) occur. In addition, to prevent shadows from tampering with the NRs they store, each NR is signed by its original owner. We now describe each component in detail.

1) Selection: To select a shadow for a node n during a period t, we compute a hashed id $s^t(n) = H[n||t||rv]$, where rv is a public random value announced by PKGs at the beginning of each period t. We assume all nodes in the system will receive rv within a small time window δ[6]. The shadow of n is then $root(s^t(n))$, i.e., the node that is responsible for the hashed id $s^t(n)$. This construction has the following property: 1) shadows are randomly selected among all nodes in the system, and each node can easily determine each other's shadow based on public information (a node's id and rv); and 2) a node cannot predict its shadow for the next period, as rv is not announced until the beginning of the next period $t + 1$.

2) Rotation: To prevent a node n collude with its shadow (thus preventing n to forge a bogus $NC(n)$), the rotation algorithm periodically selects a new shadow in the system for n[7]. In addition, the rotation algorithm also transfers and verifies n's neighboring information from n's old shadow $\alpha = root(s^{t-1}(n))$ in period $t - 1$ to n's new shadow $\beta = root(s^t(n))$ in period t. Fig. 2(a) depicts the message exchange for rotation. To initiate the rotation, α computes $s^t(n)$ and sends a rotate message containing the following to β: $s^t(n)$, n, $NR(n)$, and the entire message is signed using α's authorization key $K_{s^{t-1}(n)}$ obtained from the PKG in the previous period. Note that the signature generated using $K_{s^{t-1}(n)}$ proves that α is the shadow of n in period $t-1$. Once β receives the message from α, β verifies the received $NR(n)$ by retrieving $NC(p)$ and $NC(s)$ from p and s respectively. Once $NR(n)$ is verified, β requests a new authorization key corresponds to $s^t(n)$ from the PKG by sending the PKG its neighbor certificate. When the PKG receives the request, it verifies that β indeed owns $s^t(n)$ during the period t, the PKG then generates the authorization key $K_{s^t(n)}$ to β. When β receives $K_{s^t(n)}$, β generates $NC(n) = \{NR(n), ts\}_{K_{s^t(n)}}$[8], where ts is a timestamp, and sends this certificate to n.

We note that if node dynamics occur before a rotation process is initiated, an update process is performed first to avoid inconsistency of the neighboring information. However, in some rare cases nodes may fail *during* the rotation process. In the case where n fails, α should simply abort the process. If p and/or s fail, β can retrieve their neighboring information from their shadows for verification purpose. Similarly, if β fails during the process, α should deliver the information to β's successor β', β' will then updates its information (to obtain a new $NC(\beta')$) before carrying out the rotation process.

3) Update: Whenever a node joins or leaves the system, we update any neighboring information that might be affected. We use the fact that whenever a node joins or leaves the network and needs to update its neighboring information, its

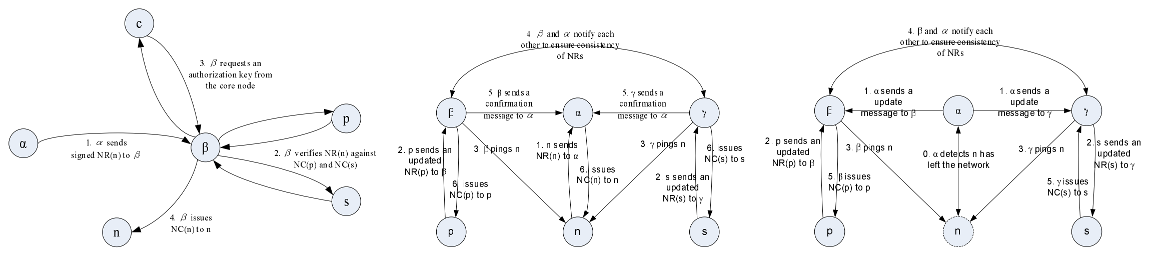
[4]We note that this key retrieval operation is carried out using SCL in a similar manner as user-server operations.

[5]For example, if n has a predecessor p and a successor s, then $NR(n) = \{p, n, s\}$.

[6]Typically δ should be less than 30 minutes for large networks

[7]In practice we set the period t between 24 to 48 hours to avoid frequent rotation.

[8]We denote $\{M\}_X$ as a message M being digitally signed with a private key X.


c
3. β requests an authorization key from the core node
α
1. α sends signed NR(n) to β
β
p
2. β verifies NR(n) against NC(p) and NC(s)
s
n
4. β issues NC(n) to n
4. β and α notify each other to ensure consistency of NRs
β
5. β sends a confirmation message to α
α
5. γ sends a confirmation message to α
γ
2. p sends an updated NR(p) to β
3. β pings n
1. n sends NR(n) to α
6. issues NC(n) to n
3. γ pings n
6. issues NC(s) to s
6. issues NC(p) to p
2. s sends an updated NR(s) to γ
p
n
s
4. β and α notify each other to ensure consistency of NRs
β
1. α sends a update message to β
α
1. α sends a update message to γ
γ
2. p sends an updated NR(p) to β
3. β pings n
0. α detects n has left the network
3. γ pings n
2. s sends an updated NR(s) to γ
5. β issues NC(p) to p
5. γ issues NC(s) to s
p
n
s

(a) Rotation from the old shadow α to the new shadow β for node n.

(b) Update process when a new node n joins between p and s.

(c) Update process when a node n leaves the system.

Fig. 2. Message exchange for rotation and update algorithms

neighbors' information also need to be updated. Fig. 2(b) shows the update process when a node n joins in between two existing nodes p and s, where s is p's successor. When n joins the system, n sends a signed $NR(n)$ to its shadow $\alpha = root(s^t(n))$. Similarly, p and s also update their NRs and send the updated $NR(p)$ and $NR(s)$ to their shadows $\beta = root(s^t(p))$ and $\gamma = root(s^t(s))$ respectively. When β and γ receive the updated NRs from p and s, they ping n directly to ensure n has indeed joined the network. β and γ then send confirmation messages containing old and updated NRs to α. When α receives confirmation messages from both β and γ and verified that $NR(n)$ received from n is correct, α retrieves an authorization key corresponding to $s^t(n)$ from the PKG. α then stores $NR(n)$ and sends $NC(n)$ to n.

When n leaves the system, its predecessor p and successor s both need to update their NRs. In addition, the neighboring information n holds for other nodes also needs to be reconstructed so the rotation process can be correctly carried out for the next period. To ensure that no attackers can remove/alter other nodes' information if they are still in the system, we use direct monitoring between a given node and its shadow and neighbors, e.g, through periodic keep-alive message using direct IP connections, and only update neighboring information if direct monitoring fails. Fig. 2(c) shows the message exchange when a node n leaves the system. When n leaves the network, α, p, and s would detect that n is no longer reachable through direct monitoring. α then sends update messages to p and s's shadows β and γ informing them n has left the system, α then removes $NR(n)$. Similarly, p and s would send updated NRs to β and γ, and β and γ then store the updated NRs when they receive the update message from α. The procedure for reconstructing lost NRs when a shadow node is down is actually simple. We note that each node has its neighbor certificate that they cannot alter for the period t. When a shadow α of a node n dies, a new shadow α' is elected (typically α's successor). In this case, the reconstruction is as follows: a) α' obtains a new $NC(\alpha')$ from its shadow; b) n sends its $NC(n)$ to α'; and c) α' obtains the authorization key for $s^t(n)$ from the PKG and issues a new $NC(n)$ to n.

C. Analysis

Here we give a brief analysis on the correctness of rotation and update (as selection is straightforward) to ensure neighboring information are correctly maintained after the execution of each algorithm. As mentioned earlier, every NR is signed by the original node to prevent its shadow from tampering, and every NC is signed by both the node and its shadow, and includes a time stamp. Thus, a shadow can only verify information and issue certificates based on verified information, and a node cannot forge its certificate unless it colludes with its shadow. We assume: a) all NRs are correct prior to every execution. This is a reasonable assumption as we start off from a set of trusted core nodes, thus all neighboring information should be correct to begin with; and b) a node cannot collude with its shadow. Again, this is reasonable as we rotate shadows from time to time so the probability of a node colluding with its shadow is very small. In fact, the probability is small even for nodes colluding with its neighbors shadows (or any other node's shadow). During the rotation process, the following nodes are involved: the node n, its old shadow α for the period $t-1$, its new shadow β for the period t, a core node (for issuing authorization keys), and n's predecessor and successor, p and s (for verification purpose). If all NRs are correct prior to rotation, the only way n can inject incorrect NR to β and get a bogus certificate back is during the verification phase. This means that p and s have to be able to fake their respective NRs so that β would override the NR delivered by α. But in order to do so, p and s need to collude with their shadows, which contradicts our assumption. Therefore it is not possible for n to inject bogus information to β and we can assure that $NR(n)$ remains correct after rotation.

For the update process, we first discuss the case when a new node n joins the system between p and s (see Fig. 2(b)). As stated in our assumption, n can collude with any other nodes except α. However, if n colludes with p and s, n does not gain any advantages as their neighboring relationship determines n's id space range. Thus, n needs to collude with nodes outside of p and s in order to gain advantage of the system. To do so, n needs to collude with some node $pp < p$ and $ss > s$ in the hope of gaining more id spaces. However, this is not possible as p and s both need to update their NRs, and this would trigger β and γ to send update message to α, causing a conflict. Thus, for n to successfully inject bogus NR to the system, n needs to collude with β and γ as well as pp and ss and their shadows so that pp and ss's shadow would send confirmation message with the bogus NRs to α instead of β and γ. However, this means that both pp and ss needs to collude with their shadows in order to generate the bogus NRs

TABLE I

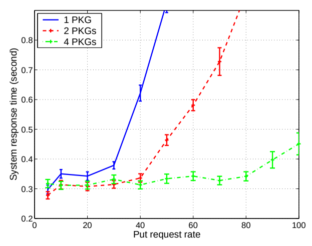
Computational overhead for crypto-primitives.

Operations	Overhead
`symmetric_encrypt`	40 μs
`symmetric_decrypt`	25 μs
`IBE_key_gen`	23 ms
`IBE_encrypt`	33 ms
`IBE_decrypt`	24 ms

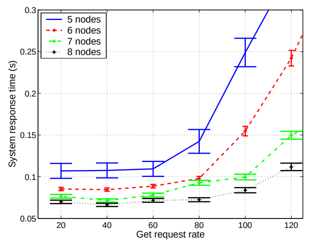
necessary for the update, which is not possible. When a node n leaves the network, its neighbor record would be removed from the system and all NRs it holds would be reconstructed at its successor. Here we are concerned that $NR(n)$ maybe removed prematurely due to framing, i.e., other nodes collude to remove $NR(n)$ from n's shadow. However, as n and its shadow monitors each other directly, the only possible way is for β and γ (refer to Fig. 2(b)) to perform denial-of-service attack against n (so that monitoring no longer works). If this is the case, n cannot perform any services anyway. And if the attack is stopped, n can reinsert $NR(n)$ back to α easily if its $NC(n)$ is still valid. Otherwise n can simply insert a new $NR(n)$ as in the join operation.

IV. Implementation and Evaluation

In this section we present some preliminary results from a prototype implementation of our framework. Our prototype is written in C and is based on the Boneh-Franklin IBE library and i3's Chord implementation. Our prototype consists of the two main components: SCL and WoT, and is evaluated in a local testbed. The testbed consists of several Pentium-4 2.4 GHz PCs each equipped with 512MB of RAM located in the same LAN. As most of our operations involves IBE, we first evaluate the computation overhead of IBE operations. Table I shows the computational overhead for each IBE operation as well as symmetric encryption and decryption. Clearly, IBE operations are more expensive than typical symmetric crypto-mechanisms. We next evaluate the system level performance for `scl_get` and `scl_put` operations. To investigate how the number of PKGs influences the put operation, we use four machines as typical nodes and vary the number of PKGs from one to four. We insert randomly generated unique (k, v) pairs into the system at various rates and measure the system response time. Fig. 3(a) shows the system response time versus the put request rate. The system performance increases as the number of PKGs increase, as for every unique key k the PKG needs to issue a corresponding private key, thus the more PKGs we have the faster the system can generate private keys.We next investigate the impact the number of nodes has on the performance of get operations. We set up one PKG and vary the number of nodes. We first put 5000 unique (k, v) pairs into the system, and let each node cache the private keys obtained from the PKG. We then perform get operations on these 5000 (k, v) pairs at various rates. Fig. 3(b) shows the system response time versus the get request rate. As we can see, the performance of the system improves as we add more nodes. In addition, the system response time is lower because no PKG and private key generation are involved in the get operation. Judging from these preliminary results, it is clear



(a) Put request with varying number of PKGs.



(b) Get request with varying number of typical nodes.

Fig. 3. System response time versus request rate

that the performance of the system scales as the system grows. However, the computational overhead of IBE operations has quite an impact on the system performance, a price we pay for added security. In addition to the local testbed evaluation, we also perform limited experiments on the PlanetLab, and the observation is that even though IBE operations are expensive, when deployed over a wide area network, the overlay network latency has larger impact on the system performance and can sometimes contribute more than 70% of the system response time. For more evaluation results and discussion, please refer to the technical report version of this paper [3].

V. Conclusion

In this paper we have developed a novel secure framework for overlay-based applications and services by combining IBE and DHT. In building such a framework, we have also developed several mechanisms that protect our framework against "man-in-the-middle" attacks and insider attacks. We believe our approach explores a new dimension in constructing secure next generation network infrastructures that is particularly suitable for many overlay-based services.

Acknowledgement

This work was supported in part by the NSF grants CNS-0435444 and CNS-0626812 and a University of Minnesota Digital Technology Center DTI grant.

References

[1] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, 2139:213–229, 2001.

[2] B. Libert and J. Quisquater. The exact security of an identity based signature and its applications. Cryptology ePrint Archive, Report 2004/102, 2004.

[3] G.-H. Lu and Z.-L. Zhang. Wheel of Trust: A Secure Framework for Overlay-based Services. Technical Report, Univ. of Minnesota.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01*, 2001.

[5] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05*, 2005.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, London, UK, 2001. Springer-Verlag.

[7] A. Seth and S. Keshav. Practical security for disconnected nodes. In *(NPSEC)*, November 2005.

[8] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM '02*, 2002.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.