Tempering Kademlia with a Robust Identity Based System

Luca Maria Aiello, Marco Milanesio, Giancarlo Ruffo, Rossano Schifanella
Computer Science Department - Università degli Studi di Torino
Corso Svizzera 185, 10149 Turin, Italy
aiello.luca_maria@educ.di.unito.it
{milane,ruffo,schifane}@di.unito.it

Abstract

The lack of a trusted authority, responsible for peers' identity verification or for authentication purposes, makes actual P2P systems extremely vulnerable to a large spectrum of attacks. The main purpose of this paper is to present Likir (Layered Identity-based Kademlia-like InfRastructure), a framework that includes an identity based scheme and a secure communication protocol, built on top of Kademlia, that may provide an effective defense against well known attacks. This will be accomplished with the adoption of a certification service, with the use of an authentication protocol between nodes and with the introduction of credentials to make non-repudiable the ownership of the contents and messages inserted in the DHT. For sake of interoperability with other social networking services, Likir enables identity management under the Identity 2.0 framework. Under this perspective, the IBS (Identity-Based Signature) scheme is taken into consideration and analyzed as well.

1 Introduction

Structured Peer-to-Peer (P2P) Systems provide many services and competitive features, such as resilient distributed storage, high scalability and efficiency, and good resistance against random node failures, making the P2P based middleware largely suitable for a vast variety of distributed applications. Nevertheless, actual P2P systems are extremely vulnerable to a large spectrum of attacks [22], mainly due to the lack of a certification service responsible for peers' identity verification and for authentication purposes.

Identity verification is critical for two reasons. First, structured P2P systems are based on ID assignments to nodes and objects. A random nodeId assignment is subject to manipulation by an adversary, and current assignment techniques are trivially vulnerable. This problem is amplified due to a heterogeneous user community, that may want to access the application masking themselves behind different identities. Second, users are motivated to use a new application when they are allowed to log in with an owned identity, rather than inventing a new user name and a password to be kept in mind. We need a certified association between a verifiable identity (e.g., OpenID or Google's account) and a nodeId, in order to provide a trusted P2P service.

Literature on vulnerabilities of P2P networks have produced two main sets of solutions against different attacks: the first as *ad hoc* heuristics focused on well known threats (e.g., [10]); the latter as architectures based on a trusted certification authority [5] and complex public-key infrastructure (PKI), responsible for nodeId certification and distribution, that may result infeasible in practice. Our purpose is the definition of a simple and robust identity-based verification scheme, together with an authenticated communication protocol built on top of Kademlia, one of the most popular Distributed Hash Table (DHT).

An important aim of our proposal is non repudiation of contents managed by the structured P2P layer. In fact, *Likir* (Layered Identity-based Kademlia-like InfRastructure) forces an application with security constraints to manipulate objects after credentials verification, e.g., a node inserting a content is asked to prove the association between the presented ID and a verifiable identity. Other relevant threats are avoided or, at least, largely mitigated: authenticated message exchange protects the system against replay attacks, man in the middle, and content forgery. Furthermore, every nodeId is associated to a given identity during registration (or bootstrapping), allowing for Identity 2.0 Management, so that verification will need a human participation that would result in an important mitigation of the effectiveness of a Sybil attack.

Our proposal can be implemented using a traditional Public Key Cryptography as well as an Identity Based Cryptography scheme (see Section 4.6). Since it is extremely important to maintain scalability and to worsen as little as

possible efficiency (i.e., in terms of node state and operations' complexity), we complete our study with an empirical analysis that compares overheads introduced by both approaches.

The paper is organized as follows. Sections 2 and 3 present the state-of-art and the background wrt our proposal. Section 4 exposes the proposed architecture and protocol, and it is followed by a discussion about security and performance; additionally, a quantitative analysis of the feasibility of the protocol is given in Section 5. Conclusions and future works are given in Section 6.

2 Related Works

Recently, a lot of effort has been put on securing DHTs [23] and the applications built on them. The usual robustness and efficiency of a DHT-based system can be overwhelmed by the malicious behavior of groups of peers that do not follow properly the DHT protocol.

In [13] an exhaustive overview of the different behaviors of peers in the KAD network is given, pointing out that node identifiers are not necessarily persistent as was assumed before. In [14], authors consider the vulnerability of KAD against Sybil Attack and show that a solution is to prevent a peer from choosing its own ID and avoiding a peer to obtain a large number of IDs. Thus, they sketch out a centralized solution that makes it impossible for an attacker to obtain arbitrary KAD IDs: a central agent binds the ID to a cellular phone number.

Sybil Attack is also the core of the work in [1] and [2]. In the first work, a resistant routing strategy is introduced on a variant of Chord, assuring that lookups are performed using a diverse set of nodes, and thus that at least a subset of the nodes involved in the lookup process is not malicious. As a consequence, the lookup process makes forward progress, not only converging fast to the destination, but also minimizing the number of trusted bottlenecks: when choosing the next node in the path, the variant will take into account the sources of information about the previous hops, and strive to avoid relying on a single trusted bottleneck. In [2] an admission control system for structured P2P systems is given. The system constructs a tree-like hierarchy of cooperative admission control nodes, from which a joining node has to gain admission via client puzzles. As the burden of self-organization and admission control is placed on the P2P nodes themselves, the computational load of these activities must be low. Analysis shows that these costs are vanishingly small for all nodes in the network. Admission Control System (ACS) defends against Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. A node wishing to join the network is serially challenged by the nodes from a leaf to the root of the hierarchy. Nodes completing the puzzles of all nodes in the chain are provided a cryptographic proof of the examined identity.

S/Kademlia [3] is a secure key-based routing protocol based on Kademlia [19] that has a high resilience against common attacks by using parallel lookups over multiple disjoint paths, limiting free nodeId generation by using crypto puzzles in combination with public key cryptography, extending the Kademlia routing table by a sibling list, reducing the complexity of the bucket splitting algorithm and allowing a DHT to store data in a safe replicated way, and finally a lookup algorithm which uses multiple disjoint paths to increase the lookup success ratio.
In [17] periodic routing table resets, unpredictable identifier changes and a rate limit on routing table updates are given. This is presented as a solution for making attackers unable to entrench themselves in any position that they acquire in the network; moreover, attackers are unable to fix an appropriate strategy for targeting some specific nodes. Authors propose also a practical defense against the eclipse attack, extending the Bamboo DHT[1].
A distributed node ID generation scheme would limit the rate in which an attacker can obtain IDs. The authors of Pastry [20] require prospective nodes to generate a private/public key pair such that the hash of the public key has the first p bits equal to zero [12]. They also suggest to bind the IP address of the node with its ID and, to overcome the possibility of an attacker to accumulate node IDs, to invalidate node IDs periodically and using different setting for the hash initialization. However, this would require legitimate nodes to obtain new IDs every time this happens. Authors show how the use of secure routing can be reduced by using self-certifying application data.
Finally, an admission control framework suitable for different flavors of peer groups and match them with appropriate cryptographic techniques and protocols is presented in [18].

As explained in Section 4.7, in this paper we limit the risk of a Sybil Attack, and, consequently, an Eclipse Attack, binding each node to a specific identity, and requiring a user interaction in the authentication procedure during bootstrapping. Moreover, each nodeId will be certified by a Certification Service granting the traceability and the non-repudiability of the messages.

A somehow similar approach to the presented work is the one by Ryu et al. [16], in which an ID assignment protocol based on identity-based cryptography is presented, showing that the id-based cryptography is a suitable and affordable technique that preserves scalability by introducing a negligible overhead. The described procedure has to be executed for each node at each bootstrap and shows a weak authentication method (i.e., based on a callback to the presented IP address). On the contrary, we introduce a proper renewal mechanism for authenticating users' identity and

[1] http://bamboo-dht.org/

nodes, in order to perform it only once during the first bootstrap. Moreover Likir enables Identity 2.0 verification, redirecting the user to authenticate himself at an off-line identity provider (e.g., *myopenid.com*). Finally, our protocol makes nodes interaction and message exchanges subject to the verification of users' credentials, facing a wider spectrum of attacks.

3 Background

The adversaries that we consider are participants in a DHT system (with reference to Kademlia) that do not follow the protocol correctly. We assume that a malicious node is able to generate packets with arbitrary contents (including forged source IP addresses) and, furthermore, to overhear or modify communications between other nodes. Malicious nodes can conspire together, and a single user can easily run several nodes on the same computer, carrying out a large scale attack even without disposing of a huge quantity of network bandwidth or without a considerable computational power.

Routing Poisoning. Routing tables of active nodes are maintained over time and renewed through a push-based approach: unsolicited messages, such as the publication of route tables of neighboring nodes or lookup messages sent from unknown nodes, supply an information that is used to update table's entries. In this scenario it is possible for a malicious peer to inject random routing data or to route entries favorable to the attacker into victim nodes, and this becomes more critical during bootstrapping: if the selected bootstrap node is malicious, it can easily provide corrupted or fake routing data to the joining node, without any risk of being discovered.

Eclipse Attack. The Eclipse Attack is a form of routing poisoning which aims to separate a set of victim nodes from the rest of the overlay network, mediating most overlay traffic and effectively eclipsing correct nodes from each other's view. Against this, the anonymous auditing technique is proposed in [10]. When the Eclipse Attack is targeted against the stored contents on DHT, making them inaccessible to lookups, then it is known as *node insertion attack*: a vast number of nodes marked with identifiers numerically close to the key *k* of the target content are initiated, receiving the most lookup requests for *k* and answering with fake contents or not replying at all, effectively hiding the content. Node insertion attack cannot be fought with anonymous auditing. It is important to notice that the eclipse attacks can effectively take place only when attacker nodes are able to assign their own nodeId without restrictions. It is possible to prevent this attack ensuring that nodeIds are randomly generated, or assigned by a trusted third party.

Sybil Attack. Since typically there exists no verifiable link between the participating entity (human user or machine) and its identity (the nodeId) it is possible for any entity to show multiple identities to the system. The generation of multiple identities under a single entity is called Sybil attack and it undermines the redundancy property of a P2P system, because it enables the gathering of a large number of nodes on few machines, centralizing unsafely many keys' responsibilities and content replicas. The Sybil entities are usually exploited to increase the effectiveness of other attacks (e.g., Eclipse, DDoS) without the need for huge computational resources or without the help of other colluding entities. A possible approach to locate Sybil nodes is periodically sending a different challenge to each node. This challenge requires a high computational effort to be solved, so that one machine cannot solve a challenge for each Sybil node it hosts within a specified short time interval, even if this approach is difficult in practice in an heterogeneous domain [8]. A central authority that assigns certified node id only after a user registration process might limit this phenomenon, because the time required to the creation of a new node would be considerably longer.

Index poisoning and content pollution. Those peers responsible for key k are asked to store all the pairs $< k, v >$, and return them when requested. The value v can be either a content or a set of meta-data and references to the sources of the requested content. An index poisoning based attack [11] consists in inserting corrupted contents among the storages of a group of index nodes. A corrupted content might be something not related to the key for which it was stored, or even a fake information, like a reference to the wrong source. An attacker can make a bogus content highly visible by flooding fictitious records under 'strategic' indexes (e.g., among nodes responsible for hot keys), flushing legitimately stored content. Credentials can be an effective countermeasure against pollution: if the content is bound to the identity of an owner, when a fake resource is found, it is possible to trace back to content creator. If the application implements a reputation system, it could be possible to penalize or even ban a malicious node.

DDoS attack. A distributed denial of service attach consists in inducing a large number of nodes of the overlay to generate a huge amount of messages to be sent to a target entity located internally or externally the P2P network. It can be achieved with a redirect technique [9], carried out through an index poisoning attack. In file-sharing systems, the attacker can insert meta-data related to a very popular content, pointing to the target IP address as a source of such a file: the victim will be overflowed by connection requests until the 'polluted' content will be kept in index nodes' storage. As it would be too costly to oblige replica nodes to verify the authenticity of each inserted content, it is necessary to adopt a reputation system so that peers who have made incorrect insertions are recognized as soon as possible and banned from the network.

Man In The Middle. If a node can intercept and modify the content of response messages, it may alter the data providing wrong information to the requester. Overlay networks as eMule Kad use a buddy system to manage the nodes behind the NAT: each of these nodes establishes a TCP connection with a chosen buddy node, which acts as an application gateway. Some studies [15] show that in the Kad network at least half of the network is prone to a MITM attack. To avoid this, communicating must be sure about the integrity of messages and about the identity of the sender. An authenticated channel between endpoints can instantly exclude a third malicious entity.

Kademlia vulnerabilities. Kademlia [19] is a structured P2P system featured by the use of a XOR metric for computing distance between points in the identifier space. In Kademlia every node has a random 160-bit *nodeId* and maintains a routing table consisting of up to 160 k-buckets. Every k-buckets contains at most *k* entries with <IP address, UDP port, NodeId> triples of other nodes, with *k* as a redundancy factor for robustness purposes. Buckets are arranged as a binary tree and nodes get assigned to buckets according to the shortest unique prefix of their nodeIds. Kademlia combines provable consistency and performance, latency minimizing routing, and a symmetric, unidirectional topology.

The Kademlia protocol is vulnerable to all the discussed attacks, even if it can mitigate the harmfulness of some of them. Nodes' identifiers are not certified and they can be generated at will on the local node, so it's possible to quickly instantiate a large number of Sybil nodes with arbitrary Ids in order to complete a node insertion attack. There is no credential associated with contents maintained in storages and no control is performed by replica nodes over the information stored in the DHT thus allowing the index poisoning and derivative attacks. There is no authentication protocol between nodes. Nevertheless, k-buckets provide resistance to certain DoS and index pollution attacks; in fact, one cannot flush nodes routing state by flooding the system with new nodes. Kademlia nodes will only insert the new nodes in the k-buckets when old nodes leave the system. Unfortunately, it is very easy to inject into a route table information relating to contacts whose identifier is very close to the victim node Id, because of the bucket splitting procedure.

Finally, it is possible to affect the lookup procedure to lead the searching node to contact a set of replica peers controlled by the attacker. The Kademlia lookup procedure for a key χ starts selecting α nodes whose ids are the nearest to the local id and sending to each of them a FIND-NODE(χ) RPC. If a malicious node receives a FIND-NODE RPC, it responds with k triples that identify colluding nodes whose id is claimed to be close to the lookup key. The searching peer has no way for verifying messages and it will trust every response.

Identity Based Signature (IBS). The IBS is a cryptographic technique that allows to compute a key pair whose public counterpart could be easily obtained from an ASCII string. This new paradigm of cryptography allows a user to verify a signature of another user from his identifier, that could be his email address, his node's IP address, his OpenId[2]. URL, and so on.

This scheme has developed from the initial idea of Shamir [21], and subsequently revisited by Boneh and Franklin [4] and Cocks [6]. It assumes that a generic user A wants to sign a message, and that to another user B needs to check it. The process is divided into four phases:

1) Setup: a trusted third party, the Private key Generator (PKG) creates a pair of "master" keys; the public master key MK^+ and its private counterpart MK^-. MK^+ is made available to all the users of the system.

2) Private key extraction: A presents his identity (Id_A) to the PKG, who produces a private key K_A^- from MK^- and Id_A; the new key is then sent to A.

3) Signature generation: using its private key K_A^-, A creates a signature s on message m, sending (m, s) to B.

4) Signature verification: B checks whether s is a genuine signature on m using Id_A and MK^+. If so it returns "true", "false" otherwise.

The ideas behind this process are based on pairing functions and elliptic curves, whose presentation goes beyond the goals of this work. The concept relevant to the proposed framework is that every user can verify the signatures of all the users whose identifier is known, but no one can compute another user's private key without the private master key. This positive feature represents the biggest drawback of the scheme, too, and it is knowns as the *key escrow* property: if an attacker takes possession of the master key, he could easily generate all the private keys: the PKG is a genuine single point of failure.

4 Protocol

Likir is layered on Kademlia and its architecture is based on the presence of a Certification Service (CS). The CS can be a centralized or decentralized authority whose task is to generate random nodeIds and to certify the link between nodeIds and users' identities by signing peculiar tokens. To accomplish this, we suppose that a classic public key cryptography scheme is used: in this section we assume that the CS is a centralized authority owner of a public key known to every Kademlia node, and holder of its private counterpart. Similarly, we assume that each user who intends to take advantage of the network services should be in possession of a key pair. In section 4.6 are presented the modi-

[2] http://openid.net/

fications to this scheme due to the introduction of the IBS system support. The following notation is used throughout:

A, B	:	*Likir* nodes
$NodeId_A$	:	node A's Kademlia identifier
$UserId_A$	:	node A's user identifier
K_A^+, K_A^-	:	node A's public and private key
K_{CS}^+, K_{CS}^-	:	CS public and private key
$Sign(m, k)$	:	message m signed with the key k
$H(o)$	:	hash code of the object o
$AuthId_A$	:	node A's authenticated id
$Auth_{AB}$	:	authentication produced by A for B
ts, TTL	:	timestamp, time to live
$a\|\|b$	:	concatenation of strings a and b

Likir enhances the join procedure, the node interaction protocol and the content storage procedure defined by Kademlia. In a preliminary initialization phase a node applies to the Certification Service for a certified NodeId and for bootstrap information; since the certified NodeId has an extensive temporal validity, initialization is not executed at every bootstrap but only periodically. After the initialization, the node performs the network join procedure to take part to the overlay. In order to correctly interact with other nodes, the newly joined one must follow a communication protocol for incoming and outcoming messages; especially, the node must produce special credentials related to every content to be inserted in the DHT.

4.1 Initialization

Node A must obtain its own certified id, in order to interact with other peers. To this aim the node sends a request to the CS containing an identifier and its public key:

$$NodeIdReq = UserId_A, K_A^+$$

The $UserId_A$ is the identity by which user A presents himself to the network community. It is an identifier of a generic account of user A and whose validity must be verifiable by the same CS. It may be assumed that the $UserId$ is an existing and verifiable identity, (e.g. an OpenID URL or an email address), in which case the CS should initiate an interaction with an external authority (e.g. an Identity Provider, a mail server) to verify its effectiveness. Otherwise the same CS could be able to maintain user accounts and verifying the identity with a password request.

The CS makes the $UserId$ verification procedure (whose steps depend on the nature of the $UserId$ itself), and then binds the user identity with his public key and with a $NodeId$ by producing the following token:

$$AuthId_A = Sign(NodeId_A||UserId_A||K_A^+||exp_A, K_{CS}^-)$$

The $NodeId$ is randomly chosen; exp_A is a timestamp that establishes the expiration date of the signed $NodeId_A$. The CS keeps track of the association between $UserId$ and $AuthId$, so that all subsequent $NodeIdReq$ received by the same users receive in response the same $AuthId$ passed earlier, unless it is expired or close to expiration. This is a precaution to avoid the CS producing useless signatures. Then, the CS sends to the client a response message structured as follows:

$$CS \rightarrow A : AuthId_A, Sign(bootstrapList, K_{CS}^-)$$

The $bootstrapList$ is a list of triple $< NodeId, IP, port >$ that points to a set of nodes that the CS assumes active; by contacting at least one of these nodes, the peer can join the network. The way in which the CS obtains the entry of bootstrap list is described in Section 4.5.

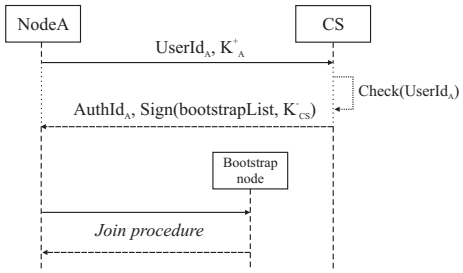

Figure 1. Initialization and join procedures

4.2 Join

Once initialization step is completed, the node may initiate the network join procedure as described by the Kademlia protocol, namely sending a lookup request for its own $NodeId$ to one of the bootstrap contacts. However, once obtained an $AuthId$, it is important that the nodes avoid contacting the certification service, unless if necessary. After making the first join using information obtained from the $bootstrapList$, each node should get in a different way a list of nodes to be contacted for subsequent join operations. For example a node can maintain its own list of trusted bootstrap nodes, or the same CS could periodically insert a signed $bootstrapList$ in the DHT, so that every active node could download it before disconnection and use it for its next join. Only if all the known nodes are off-line the CS will be contacted again to request a new $bootstrapList$. The node Initialization and the subsequent network join are shown in figure 1.

4.3 Nodes interaction

A node A can successfully send a RPC (join primitive included) to a node B and obtain a proper response only if both A and B observe the following communication protocol:

I $A \rightarrow B : NodeId_A, N1$

II $B \rightarrow A : NodeId_B, N2$

III $A \rightarrow B : AuthId_A, Auth_{AB}, \text{RPC-REQ}$

IV $B \rightarrow A : AuthId_B, Auth_{BA}, \text{RPC-RES}$

We call this four way exchange a $session$ between A and B. RPC-REQ and RPC-RES fields are respectively the request and response RPC defined in Kademlia; $N1$ and $N2$ are randomly generated nonces. Messages sent at steps I and II must be somehow marked differently (e.g. different opcode), to distinguish the request from the response.

Authentication tokens are structured as follows:

$$Auth_{AB} = Sign(NodeId_B||N2||H(\text{RPC-REQ}), K_A^-)$$
$$Auth_{BA} = Sign(NodeId_A||N1||H(\text{RPC-RES}), K_B^-)$$

Figure 2 shows the message flow between two nodes during a session. In step III (and IV), the receiving node checks signatures (in $AuthId$ and $Auth$), expiration times validity, equalities between nonces, and equalities between $NodeId$ in step I (and II), in $Auth$, and in $AuthId$.
Signature and expiration time validity checks on $AuthId$ demonstrate the existence of a valid and randomly generated $NodeId$, associated with an $UserId$ and with a public key; validity of signature in $AuthId$ and equality check on $NodeId$ assures that the sender is the same entity certified by $AuthId$ and that the present node is the correct recipient of the message. Equality checks on nonces in $Auth$ and the ones received previously protect against replay attacks. A's verification of $NodeId_B$ included in $AuthId_B$ assures that B is really the node that A wanted to contact; B's verification of $NodeId_A$ included in $AuthId_A$ proves that the RPC has been called by the same node that started the session. Finally, both peers execute an integrity check on the RPC hash to verify that no attacker has replaced the original RPC with a bogus one. The reader should observe that nonces are used against man in the middle attacks instead of exchanging timestamps because we cannot assume that hosts are synchronized to a common clock.

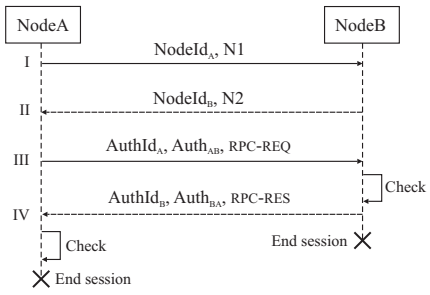

Figure 2. An example of node session

4.4 Content storage system

RPCs follow Kademlia's definitions, except for the store RPC. Let A be a node, owner of a content Obj. If A wants to store Obj in the DHT it locates via lookup the k nodes closest to the content key and then sends to them a store message structured as follows (suppose that B is a generic replica node):

$$A \rightarrow B : AuthId_A + Auth_{AB} + StoreRPC$$
$$StoreRPC = k||Obj||Cred$$
$$Cred = Sign(UserId_A||k||H(Obj)||ts||TTL, K_A^-)$$

$Cred$ binds the $UserId$ to the key for which the content was inserted and to the hash code of the content, so that is subsequently possible to prove that the owner had inserted the content Obj at the key k. $Cred$ includes also a timestamp and a time to live to specify the content submission time and its persistence period. During the periodic content spreading procedure, all replica nodes send store messages keeping the original credentials associated with each content. A node performing a lookup for contents related to a key χ receives all the objects marked with χ from replica nodes responsible for that key; before passing the content to the application, the node must verify the credentials signature and the object hash and must discard the object if the check fails.

If the application ascertain that the content is somehow polluted (e.g. the key that marks the content is not related with it), it can benefit from the information included in the credentials to penalize the owner of the content. This could be simply accomplished by instructing the underlying node to blacklist the cheater user in order to refuse all the incoming requests marked with the malicious node's $AuthId$; every $Likir$ node allows the application to insert a new identity to a local blacklist. The description of a reputation service that can manage feedbacks from the users and the details concerning a possible revocation policy for the identifiers of misbehaving users, are beyond the goal of this paper. However, it is important to say that an effective reputation manager, that can be external to the network, as well as integrated in the application, can help to exclude more rapidly the polluter from the whole network. Nevertheless, the propagation of polluted content is largely limited due to credentials' verification.

4.5 Bootstrap list construction

The bootstrap node selection is a problem inherent to the fully distributed nature of P2P networks. The bootstrap information acquisition process must prevent an attacker to manipulate bootstrapping information to let a victim join a malicious parallel network. Kademlia does not face the

bootstrap node selection problem. Using the support of the Certification Service, $Likir$ offers a practical solution to this problem.

The CS maintains a list of active peers in a cache, where a generic entry stores the following information:

$$CacheEntry = (NodeId, IPaddress, UDPport, ts)$$

The CS probes nodes in the cache, controlling a DHT node, marked with a self signed $AuthId$, that runs a sequence of FIND-NODE RPCs for random generated keys. The CS adds to its cache the pointers to the nodes that replied to the FIND-NODE RPCs, then it can iterate the procedure until it gathers enough contacts for cache replacing. Likir implements a least-recently cache replacement policy, except that active nodes are never removed from the list: if the cache is full then the least-recently seen node is pinged. If it fails to respond, it is replaced with a newly discovered one. Otherwise, if the least-recently seen node responds, it is moved to the tail of the list, and the new contact is discarded.

4.6 IBS support

$Likir$ can profitably replace the classic public key cryptography with the IBS paradigm to sign messages. Since the $UserId$ must be sent in every communication session, the recipient of an RPC (request or response) always knows the user name of the sender. Using IBS is therefore possible to noticeably streamline the protocol overhead by omitting the information about the user public key in the $AuthId$; this is because the $UserId$ includes also the information of the public key. Suppose a scenario where there is a PKG (that could be an integral part of the CS) and where every system user has obtained from it a private key tied to his own user id; even the CS would have its own id and a related key pair. The CS should no longer certify the binding between a $UserId$ a public key and a $NodeId$, but it could just certify the binding between a $NodeId$ and a $UserId$ Using IBS, the $NodeIdReq$ message and the $AuthId$ related to a node A becomes:

$$NodeIdReq = UserId_A$$
$$AuthId_A = Sign(NodeId_A||UserId_A||exp, K^-_{CS})$$

4.7 Security Considerations

In this section we discuss how *Likir* strongly limits dangerousness of attacks described in Section 3.

Routing attacks. In Kademlia, the sender contact of every incoming message is added to the route table if there is enough room in the buckets. The contacts with a nodeId close to the local id are always added to the route table due to the splitting procedure. Combined usage of $AuthId$ and $Auth$ makes the communication between nodes authenticated, so the attacker can inject only its own contact into the target route table, and because the ids are randomly chosen by the CS, the attacker cannot generate its id "ad hoc". Routing attacks (including eclipse) are unfeasible. Moreover, it is unfeasible for an attacker to hide a content marked with a given key k by way of a node insertion attack, because the malicious node cannot register a substantial number of nodes with ids close to k: in fact, he cannot control id generation by his own.

Kademlia's lookup vulnerability is corrected by authenticated message exchange and random id generation. If the malicious FIND-NODE RPC receiver responds with a set of references to invalid nodes (i.e. devoid of $AuthId$s), the victim node is not able to contact any of them because the authentication protocol fails in signature verification. If the attacker responds with a set of valid colluding nodes, its attack results ineffective because the colluders' ids are scattered along the keyspace, so the lookup procedure proceeds properly.

Sybil attack. Every user can have multiple identities (e.g. many email addresses), so a user can bind each of his identities to a different node by sending many $NodeIdRequest$ to the CS, and then he can run all those nodes on the same machine. So the Sybil attack is not completely wiped out with this scheme. Nevertheless, each node corresponds to a different user account and the node initialization requires a verification procedure for that account. If the user authentication procedure requires a human interaction it would be difficult for an attacker to create many different nodes in an automated way, actually lowering the risk of Sybil Attacks. For this reason, we strongly suggest to adopt OpenId verification methods, that redirect the user agent to an identity provider, and that returns to the CS when the submitted identity has been correctly authenticated.

Storage attacks. Every storage entry in the DHT is bound with its $Cred$, created by the content owner with an unforgeable signature. A node performing a lookup operation returns to the application only those results that are bound with some $Cred$, and that has been previously verified. Therefore, the consumer application (or the human user himself) can interact with a reputation system to reward or penalize the owner of the consumed object depending on the quality of the content. The underlying node can be then instructed to exclude from network traffic those nodes whose reputation is too bad. The use of $Cred$ can contrast attacks like index poisoning, content pollution or even DDoS attacks based on redirection by punishing the malicious users who attempt these attacks.

Man in the middle. An attacker who's able to intercept and alter the messages flowing between two nodes has no way to act as one of the endpoints or to fool correct nodes into accepting forged messages. $AuthId$ and $Auth$ cannot

be modified since they are signed, and the RPC cannot be altered or replaced because the Authenticator contains the RPC hash code. An attacker cannot effectively replay an intercepted $Auth$ because it includes a nonce which validity is limited to a node interaction session; moreover authenticators are addressee-specific, because they include the recipient node id. Finally, the nonce based two-way authentication scheme grants protection against common interleaving attacks as Oracle session attacks, parallel attacks and offset attacks.

5 Evaluation

In this section we propose a quantitative analysis of the Likir protocol in order to evaluate the feasibility of the proposed approach in a large-scale distributed environment. Compared to Kademlia, Likir shows an overhead that affects both the number and the size of messages exchanged between nodes, besides the computational cost introduced by cryptographic operations. In particular, any interaction between nodes involves the dispatch of two additional messages for the nonce exchange (messages I and II) and the addition of $AuthId$ and $Auth$ to the request and response RPCs (messages III and IV). Moreover, a node must produce a signature for each sent RPC (two signatures when the content is submitted by its owner, because of the production of credentials) and verify two signatures for each received RPC. In the following, we will state the costs in terms of messages size (Section 5.1), computational cost due to cryptographic primitives (Section 5.2) and network latency (Section 5.3).

5.1 Spatial Analysis

The size of every Likir message is greater than the size of ordinary Kademlia RPC, due to the addition of the signed tokens. $AuthId$, $Auth$ and $Cred$ are composed by different data; the whole set of elements that composes these tokens, together with their size, is listed in Table 1.

$NodeId$	:	20 bytes
$UserId$	:	128 bytes
K^+	:	128 bytes
exp	:	8 bytes
$Signature$	:	128 bytes
$Nonce$	:	16 bytes
$Hash$	:	20 bytes
ts	:	8 bytes
TTL	:	8 bytes
key	:	20 bytes

Table 1. Protocol elements size

Every message contains at least an $AuthId$ and an $Auth$, which constitutes the message header. In addition to this, the STORE RPC request includes a $Cred$ associated to the object to be stored and the FIND-VALUE RPC response payload contains a $Cred$ attached to every content returned to the querier. Because the number of contents per FIND-VALUE response is variable due to the availability of objects corresponding to the requested key stored in the replica nodes, the overhead for such RPC is not constant; we define n as a variable representing the number of $Cred$ per FIND-VALUE response. If an IBS scheme is used, the $Auth$ does not include the public key of the sender, thus streamlining the spatial overhead for every RPC. Table 2 shows the overhead for every RPC, when RSA and IBS schemes are used. It is worth to notice that the additional header size is smaller than 1KB in the worst case, and the FIND-VALUE payload dimension overhead is linear with the number of retrieved contents.

RPC	Likir RSA		Likir IBS	
	Request	Response	Request	Response
PING FIND-NODE	596	596	468	468
FIND-VALUE	596	596 + 312 · n	468	468 + 312 · n
STORE	908	596	780	468

Table 2. RPC Spatial Overhead (in bytes)

5.2 Cryptographic Microbenchmark

The node interaction protocol instructs both sender and receiver to generate and to verify signatures. Table 3 shows the number of cryptographic primitives to be performed by a node during a whole session, for every RPC. We refer to gen as the signature generation, and to $check$ as the signature verification. The SHA-1 hashing operations are not considered due to the non-influential cost. The receiver must perform a check for any received request and must produce a signature for every sent response, independently of the RPC performed. Conversely, the operations required to the sender depend on the RPC type. In particular, the sender of a FIND-VALUE RPC should verify the signature of the $Cred$s associated to the n received contents included in the response.

RPC	Sender	Receiver
PING FIND-NODE	$gen + check$	$gen + check$
FIND-VALUE	$gen + (n + 1) \cdot check$	
STORE	$2 \cdot gen + check$	

Table 3. RPC Cryptographic Overhead

Used in this analys, we have built an initial implementation in C in order to quantify the cost of cryptographic operations in a real platform. We use the GNU *openSSL* library for all standard cryptographic operations, 1024-bit RSA for

signing and SHA-1 for hashing. We adopt the *Pairing-Based Cryptography* (PBC) library[3] for all identity-based primitives. In particular, we used *Boneh-Lynn-Shacham* (BLS) scheme. All experiments are conducted using a Intel Quad-Core Xeon 2.5 GHZ with 4 GB RAM, running the Linux Ubuntu 7.01 operating system. Each test is performed 1000 times and the average value is presented in Table 4 along with the standard deviation σ. Note that the σ estimation for the FIND-VALUE RPC considers a ϵ value that represents the standard deviation component due to the n $check$s performed.

Operation	Likir RSA		Likir IBS	
	Cost	σ	Cost	σ
gen	2.026	0.164	17.695	0.045
$check$	0.1	0.014	26.5	0.051
PING	4.244	0.214	88.372	0.126
FIND-NODE	4.634	0.218	88.624	0.156
FIND-VALUE	$4.634 + check \cdot n$	$0.218+\epsilon$	$88.624 + check \cdot n$	$0.156+\epsilon$
STORE	6.834	0.256	106.009	0.175

Table 4. RPC Cryptographic Costs (in ms)

From the results, the additional cryptographic costs are affordable for a real system in both RSA and IBS schemes. It is worth noting that there are evident differences between RSA and IBS, mainly for the $check$ primitive; anyway this gap does not compromise the feasibility of the IBS approach.

5.3 Computational Effort

As depicted previously, the Likir protocol introduces an additional cost due to the exchange of a nonce value between the participant nodes. In order to quantify the impact of this further message, we deploy the Likir middleware in a couple of nodes running on the *PlanetLab*[4] network. Accordingly, we select two nodes (i.e., *planetlab1.di.unito.it* and *planetlab1.cs.ubc.ca*) and we executed all RPCs measuring the overall session time. Each test is performed 500 times and the results show the average values.

In Figure 3 we show the comparison between the session time for the basic Kademlia and the Likir with IBS or RSA support. As expected, for all RPCs the overhead introduced by the Likir protocol is about 2 times for the RSA scheme and 2.5 times for the IBS. This gap is basically due to the nonce exchange.

As described above, the number of received *Cred*s n can affect the overall performance mainly due to the additional $check$s required. Therefore, in the following we will examine the behavior of the FIND-VALUE RPC according to the n parameter. Figure 4 shows that the RSA approach is slightly

[3]http://crypto.stanford.edu/pbc/

[4]http://www.planet-lab.org/

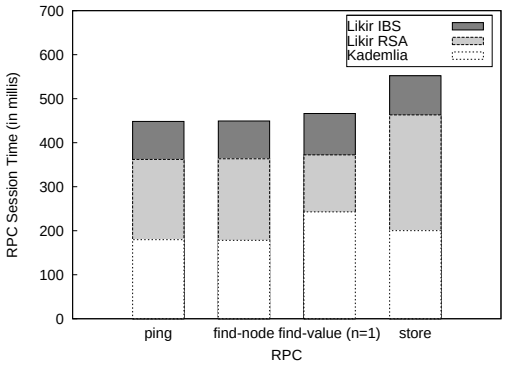

Figure 3. RPC Session Overhead

affected by the n variation while the IBS scheme introduces a linear degradation when n grows. Anyway, an improvement to the IBS scheme could be gained by the adoption of an aggregate signature [7] scheme by means of a group of signatures that can be verified in a single step. This can (significanlty) reduce the cost of the signature verification phase, even if this alternative needs further analysis due to the different pairing system used.

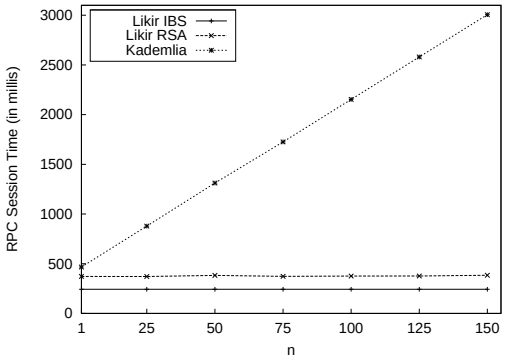

Figure 4. FIND-VALUE **RPC Session**

5.4 Discussion

In relation to the high level of security that the proposed architecture guarantees, the cost of the $Auth$ signature production for the outgoing messages is affordable.

If a certain percentage of "corrupted" messages could be accepted as valid, it would be possible to set up the nodes to verify the signatures of inbound messages to sample. A more detailed study about the relationship between the percentage of corrupt messages on the overlay network, the signature control frequency and the percentage of corrupt messages that succeed in avoiding the verification could show what is the optimal control threshold for systems subjected to any degree of risk.

The introduction of a centralized control in a completely distributed system is another potential weakness of the system. However, it is worth noting that the CS intervenes only in the node registration procedure; besides, the expiration time specified in the $AuthId$ could reasonably be considered valid for months, or even for several years. The CS is essential only for those users who have yet to make the registration process, or whose $AuthId$ is expired; all other nodes can join the system using a list of previously known nodes as bootstrap list. Thus, despite its presence, the CS is not a single point of failure and the overlay remains a fully distributed environment which is almost independent from the central control.

6 Conclusion and Future Work

Leveraging *Likir* functionalities, every content in the system will be associated with its owner's identity in a non-repudiable way. Moreover, IDs will be certified and randomly selected preventing routing and storage attacks. In such a way, the previously identified weaknesses are strongly mitigated. Finally, applications' interoperability with other Web 2.0 services is granted by integrating authentication through external identity providers. A Java API has been implemented and it will be released very soon. Further performance and scalability tests, conducted in a real network environment, are under way. As future work, we plan to improve Likir architecture by providing the Certification Service distribution, in order to completely remove the dependence of the system on a central authority. Moreover, Likir security properties will be methodically analyzed in a more formal way.

Acknowledgments

This work has been partially supported by the Italian Ministry for University and Research (MIUR), within the framework of the "PROFILES" project (PRIN).

References

[1] *Sybil-Resistant DHT Routing*, volume 3679 of *LNCS*. Springer, 2005.

[2] H. R. al. Limiting sybil attacks in structured peer-to-peer networks. Technical Report NAS-TR-0017-2005, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, 2005.

[3] I. Baumgart and S. Mies. S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing. In *Proc. of P2P-NVE 2007 in conjunction with ICPADS 2007, Hsinchu, Taiwan*, volume 2, Dec. 2007.

[4] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 299–314, New York, NY, USA, 2002. ACM.

[6] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Proc. of the 8th IMA Int. Conf. on Cryptography and Coding*, pages 360–363, London, UK, 2001. Springer-Verlag.

[7] B. L. D. Boneh, C. Gentry and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Eurocrypt '03, LNCS 2656, Springer-Verlag*, pages 416–432, 2003.

[8] J. Douceur. The sybil attack, 2002.

[9] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. *SIGMETRICS Perform. Eval. Rev.*, 33(1):38–49, 2005.

[10] A. S. et al. Eclipse attacks on overlays: Threats and defenses. In *Proc. of the 25th IEEE InfoCom 2006*, Barcelona, Spanien, April 2006. IEEE Computer Society.

[11] J. L. et al. The index poisoning attack in p2p file sharing systems. In *INFOCOM*, 2006.

[12] M. C. et al. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[13] M. S. et al. Analyzing peer behavior in KAD. Technical Report EURECOM+2358, Institut Eurecom, France, Oct 2007.

[14] M. S. et al. Exploiting kad: possible uses and misuses. *SIGCOMM Comput. Commun. Rev.*, 37(5):65–70, 2007.

[15] M. S. et al. A global view of kad. In *IMC '07: Proc. of the 7th ACM SIGCOMM*, pages 117–122, New York, NY, USA, 2007. ACM.

[16] S. R. et al. Leveraging identity-based cryptography for node id assignment in structured p2p systems. In *Proc. of AINAW '07*, pages 519–524, Washington, DC, USA, 2007. IEEE Computer Society.

[17] T. C. et al. Induced churn as shelter from routing-table poisoning. In *Proc. of NDSS 2006, San Diego, California, USA*, 2006.

[18] Y. K. et al. Admission control in peer groups, 2003.

[19] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS 2002*, pages 53–65, 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *LNCS*, 2218:329–??, 2001.

[21] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proc. of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[22] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 261–269, London, UK, 2002. Springer-Verlag.

[23] R. Steinmetz and K. Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *LNCS*. Springer, 2005.