Secure Lookup without (Constrained) Flooding

Bobby Bhattacharjee[1]* Rodrigo Rodrigues[2]* Petr Kouznetsov[3]

[1] University of Maryland, College Park, Maryland, USA
[2] Instituto Superior Técnico (Technical Univ. of Lisbon) and INESC-ID, Lisbon, Portugal
[3] Max Planck Institute for Software Systems, Saarbrücken, Germany

1 Introduction

We present a new protocol for secure routing in overlay networks. Our protocol exports the same functionality as regular decentralized lookup protocols [9, 10, 12, 14, 15]. Moreover, the existing routing protocols can be enriched with the security primitives we introduce. Recall that a routing protocol exports a lookup operation that, given a key in a virtual identifier space (the *id space*), locates the node (or the group of nodes) that are, in a well-defined sense, the closest to the key. The routing primitive can then be used, e.g., to implement a secure Distributed Hash Table (DHT), a popular abstraction for publishing and retrieving items in a decentralized manner.

Originally, decentralized lookup protocols assumed that nodes are cooperative (follow their specifications unless they fail by crashing) and provided the lookup operation in $O(\log(N))$ network hops where N is the number of participating nodes. There have been a number of improvements to the base schemes in the cooperative scenario, including reduction of some of the operations to an amortized constant [7].

All of these protocols, however, are susceptible to various attacks if the underlying assumption that nodes are cooperative is violated. In particular, malicious nodes may mis-route or simply drop protocol messages. Even a small fraction of compromised nodes can adversely affect all routing guarantees. There have been a number of prior efforts towards securing a routing protocol. Castro et al. [2] and Fiat et al. [4] both consider systems where at most $\frac{1}{4}$ of the nodes are malicious. In order to securely forward messages, Castro et al. relies on *redundant routing*, which floods the message along multiple paths. Fiat et al.'s solution, S-Chord, groups sets of contiguous nodes into *swarms*. Nodes flood requests to every node in a swarm, which requires $O(\log^2 n)$ messages. A different attack model is considered in [8] — here, each node can be mapped to its autonomous system (AS), and adversaries are constrained to at most k. Within these ASs, there can be unlimited number of adversaries (perhaps even totaling more then $\frac{1}{4}$ of all nodes in the system). Like S-Chord, the protocol in [8], also divides the id space into a set of contiguous neighborhoods, and uses Byzantine agreement to secure each neighborhood.

In this paper, we propose a decentralized routing protocol that addresses the security issues by using novel challenge-response mechanisms and mobile proactive secret sharing. Our protocol was inspired by [8]; however, we address the general problem of an f-fraction of malicious nodes. Our system is secured using a system-wide public/private key pair, and the private key is stored in a distributed manner amongst system participants. We use threshold cryptography [6, 1, 16, 13] to ensure that no single (or small set of) node(s) has the system private key at any point in time. Since the private key is threshold distributed, some malicious nodes might hold key shares. Crucially, we show the set of shares held by bad nodes (including all shares over all time) can never be used to reconstruct the private key.

The key idea in our protocol is the notion of a *challenge* — when a (good) node receives a negative answer (e.g., when a lookup fails), it can challenge other nodes whether the routing data they provided is correct. The nodes must answer the challenge and sign the answer using the system-wide public key. Since good nodes must be involved in signing a message, a correctly answered challenge implies that the negative condition, in fact, exists in the system. This use of challenges and threshold cryptography was also inspired by previous work [11].

Our system provides several attractive features compared to prior approaches: the lookup path maintains the logarithmic complexity of original protocols, and we provide (easily) provable guarantees. In particular, we prove that (with configurably high probability) as long as challenges are correctly answered, the system state is not compromised. Furthermore, unlike previous secure routing protocols, we do not impose limits on the fraction of compromised nodes in the system.

The tradeoff of our protocol is in its computational complexity — threshold signature and key redistribution protocols are expensive, and our protocol must periodically incur a high key re-sharing cost. However, we believe this high overhead can be reduced using more sophisticated cryptography, and the current protocol is instructive in its design and in the simplicity with which it enables global security properties to be asserted. Indeed, our goal here is not to present a fully optimized (or perhaps even a deployable) protocol; instead it is to explore

*Work done while visiting the Max Planck Institute for Software Systems a new part of the secure routing design space using first principles.

As a side contribution, we propose an interesting replication scheme that may be used (outside the context of this paper) in the design of large-scale intrusion-tolerant systems. In particular, we use a proactive threshold signature protocol in a way that provides very strong safety properties (we can set the failure threshold arbitrarily high), but may incur liveness problems if there are more than 1/3 faulty replicas. To address the liveness problems we present a takeover protocol where the system regains liveness by having some groups taking over the responsibilities of groups that have halted.

In the rest of this paper, we explain how the protocol boots and maintains its invariants through node joins, leaves, and attacks.

2 Protocol

We model the system as a dynamic collection of nodes that are able to communicate with each other thorup exchanging messages.

Each node has a unique id that includes its IP address and its public key. There exists a one-way function h that maps every node (every object) to a unique point in the id space.

We assume a malicious adversary that is able to compromise a subset of nodes. We assume, however, that it is not in the power of the adversary to obtain many node ids (identities are "expensive") or to choose the positions of the nodes it controls in the id space.[1] Thus we assume incoming nodes can obtain a join certificate that can be validated by any system node. This assumption does not imply the need for an online global PKI. As long as the participants are willing to trust a certificate authority (CA), they only need to be seeded with this CA's public key. New nodes would get their ids generated at random and certified by the CA, and each node in the system could verify this new ID by verifying the CA's signature (using the CA public key they already have).

Further, we assume that at most fraction f of nodes, with ids chosen uniformly at random, can be compromised by the adversary within a bounded time period (a parameter of the system, called the *vulnerability window*).

The messages can be dropped, though the communication channels cannot produce or duplicate messages. We assume, however, that every two correct (non-compromised) nodes are able to eventually reliably communicate.

2.1 The Secure Routing Primitive

A secure key-based routing abstraction exports conventional membership operations for a new node to join and leave the system and the following operations: *lookup*(x) and *secure-lookup*(x). Both operation return a set of nodes of size t that are, in a certain sense, close to x in the id space (we will call these the *neighbors* of x). Operation *lookup*(x) is the best-effort lookup operation exported by the conventional (insecure) routing schemes [9, 10, 12, 14, 15] and, in the normal operation case, it returns the set of neighbors of x. Secure lookup is typically invoked when the conventional lookup does not return a satisfactory result. E.g., in a typical implementation of DHT used for storing and locating self-verifiable data, a secure lookup primitive can be invoked when the nodes returned by the conventional lookup claim not to have the required data (i.e., we suspect that either the lookup instance or the node are compromised).

The *secure-lookup*(x) operation guarantees that, under the condition that the system membership eventually stabilizes and liveness properties of our system rely upon the assumption that there exist bounds on relative processing speeds and communication delays (which are however unknown to the nodes), there is a time after which all *secure-lookup*(x) operations return the same group of nodes (which are the correct neighbors of x).

2.2 Spans

We dynamically partition the id space into sub-intervals called *spans*. Initially, there is a single span in the system that consists of the entire id space, but, as we will detail later, spans can be split or merged as the system membership evolves.

For each span there is a *span committee*, a subset of the span nodes that is responsible for keeping track of the span membership (i.e., the subset of the current system members whose ids lie within the span), and for producing and disseminating a *span certificate*, which is an authenticated description of the span membership.

We envision that the span committee is chosen when the span is formed, and its composition will only change when some number of its members leaves the system, before the liveness of the committee is affected. Details of how exactly this choice is made are left for future work.

Our system is still in its design phase, and it is not clear what the precise span and span committee sizes will be (partly because this depends on the security parameters specified by the application). In Section 3 we provide a resilience analysis for different committee sizes, and we expect, in practice, the committee sizes to be between 12–25 nodes.

[1] Note that this assumption rules out the famous Sybil attack [3]. Relaxing or validating this assumption is left for future work.

2.3 Threshold Cryptography

To authenticate span certificates, we use a proactive threshold signature scheme [6, 1, 16, 13], which allows spans certificates to be validated with a single, well-known public key, without relying on any particular committee member knowing the corresponding private key (since if it were faulty it could expose it).

In an (n,t) proactive threshold signature schemes, each one of n nodes (in this case, the committee members) holds a share of a secret, and the protocol will only generate a correct signature if t of these nodes agree on signing the same statement.

Furthermore, these protocols include a mechanism for share refreshment that produces a new set of shares from the old ones. In particular, we require a protocol where the set nodes that hold the shares can change as part of the share refreshment protocol (and for this reason we intend to use MPSS [13]).

Share refreshment is triggered when there is a change in the span committee. After the share refreshment protocol ends, the old nodes can discard their shares. This allows the protocol to work correctly (meaning informally that malicious nodes cannot produce a valid signature) provided that less than t in each set of share holders (committee members) are compromised during a window of vulnerability, which is the time interval during which these nodes are holding their shares.

2.4 System Operation

In this section we describe how the system works in the normal case, assuming, for now, a steady-state operation where there are no membership changes. This will clarify the importance of span certificates and span committees.

We explain our system in terms of the Pastry [12] protocol, where nodes maintain a leaf set (a set of neighboring nodes in the id space) and a routing table (a set of nodes in distant locations of the id space).

Nodes also maintain the current span certificate for their own span, and, for each entry in the routing table, a cached copy of the span certificate for the corresponding span.

The *lookup*(x) operation works exactly as in Pastry. Therefore it does not provide any guarantee that the answer is correct. On the contrary, the *secure-lookup*(x) operation must ensure that it returns the current set of neighbors of id x. This primitive starts by performing a normal lookup but the reply from the neighboring nodes must be accompanied by the span certificate for that span that includes id x.

Obtaining the span certificate is not enough, though, since this could be an old certificate, where a large fraction of the members had left the system or even be compromised by now. To ensure the freshness of the span certificate the client must issue a *challenge* to the span committee members. This is a random nonce that the client sends, which the span committee members must sign (with the threshold signature protocol) along with some digest of the span membership. Only the current span committee can produce such reply since old committee members would no longer hold the shares of the secret required to sign.

2.5 Join and Leave

Another crucial function of the span committee is to keep track of the changes in system membership. We now describe how the system handles nodes joining and leaving the system.

To join the system, the node must first obtain its join certificate and the address of one or more current system nodes (the bootstrap nodes) using some out-of-band mechanism.

Then it asks the bootstrap node to perform a lookup to the incoming node's id, and it also asks for the corresponding span certificate. This will enable the incoming node to find out about its leaf set and span committee. The incoming node can verify that the span committee is current in two ways: it can challenge it right now, or it can wait until the join operation succeeds to see if a new span certificate is produced containing itself.

The incoming node contacts the span committee members, asking them to join the span, and sending them its join certificate. Span committee members run the threshold signature protocol to produce a new span certificate that includes the new node. As they do this they exchange the join request among themselves, to deal with requests that did not reach all committee members. If multiple joins occur concurrently, the span committee members add all incoming nodes to the span certificate they are trying to produce.

The consequence of this operation is that the node is added to the span membership, and a new span certificate is produced and disseminated to the span members Possibly this may trigger a span split, which we describe later.

Once the joining node receives the new certificate containing itself, it does a normal Pastry join, which will update the Pastry structures. We need to deal with a bad incoming node that will run the operation on the span committee but will not contact its Pastry neighbors. To handle this case, system nodes will update their leaf sets when they realize that there is a new node in the span certificate that should be part of the respective leaf set.

Node departure. Due to space constraints, we briefly summarize the protocol actions when a node departs (or crashes). We assume that nodes leave ungracefully, i.e. without notifying others. Instead, all nodes implement a protocol such as Rosebud [11] or PeerReview [5] to monitor and robustly detect when a neighbor has died or misbehaved. Once sufficient nodes declare a span member to be faulty, they contact the span committee and a new span certificate is computed.

2.6 Span Split and Merge

There are two system parameters that define the minimum and maximum number of elements in a span, s_{min} and s_{max}. This is important to avoid overloading span committees if the span is too populated, and to avoid creating a liveness problem if there aren't enough nodes in the span to form a committee.

After each join or leave operation concludes, nodes in the span must verify if the number of nodes in the span is still within these limits. If the number of nodes in the span is too high, the span is split in half, and two new committess are formed (one for each span). In this case the old committee must run two parallel instances of the share refreshment protocol with each one of the new committees.

If, on the other hand, the number of nodes in the span becomes too low, the span must merge with its neighboring span. In this case we can just pick one of the committees to take over the responsibility of the entire span to avoid running another share refreshment protocol.

2.7 Span Takeover

In some cases, a span committee may experience liveness problems. This could happen because there aren't enough non-faulty replicas to meet the threshold for signing statements, or because the membership turnover required a change in the composition of the committee, and the MPSS protocol requires more than 2/3 non-faulty replicas to provide liveness (even though there is no such bound for providing safety).

Our system includes a mechanism for recovering liveness in such cases called a *span takeover*. The idea is that periodically each span committee monitors the liveness of one of its adjacent span committees (e.g., in clockwise direction of the circular id space). Monitoring liveness consists of using the challenge mechanism described in Section 2.4. If a span committee detects that its neighboring span is not responding to challenges, it takes over the neighboring span. This means that the monitoring span will force a merge with its neighboring span, becoming responsible for its id interval.

Note that since we do not rely upon strong synchrony assumptions, the takeover protocol may be initiated against a live (but slow) span. This might lead to bounded periods when the ranges of two neighboring spans overlap. Such a situation is considered normal, since eventually, the spans' live nodes will not suspect each other and spans will resolve the conflict. In some cases, e.g. if a network partition persists for a long period, we may have two disjoint systems with independent routing guarantees. This is not different from what would occur in a normal routing overlay.

3 Discussion

Space constraints will not permit us to present a comprehensive evaluation of either the security or the overhead of our protocol. Instead, we begin with a brief summary of overheads using the base share refreshment protocols (without optimizations). Next we present two specific attack scenarios and how our protocol is resilient — we believe the general security "theme" will be apparent from our informal analysis. Finally, we conclude this paper with a discussion of open areas of work.

3.1 Overhead

The threshold key redistribution protocol incurs $O(n^4)$ overhead, where n is the size of the span committee. Recall that in an (n, t) sharing scheme, n members get a key share, and at least t members are required to produce a valid signature. Hence, for the key to be exposed, at least t members in the committee have to be corrupt. Thus, given a fixed (f_e) expected fraction of malicious nodes, we can choose the values of n and t to make the probability of key exposure arbitrarily low.

In Table 1, we present the probabilities of key exposure for different values of f_e and n. For example, suppose 10% of the nodes are assumed to be malicious (with malicious node ids picked uniformly at random), and if the committee size is 20, then the expected number of bad nodes in the committee is 2. However, suppose we set the threshold t to be 10. Then the probability that all 10 nodes are corrupt is 0.0003 (second line in the table). These values were derived by upper bounding the mass in the tail of the PDF using the Chernoff-Hoeffding bounds.

Assuming reasonable key sizes (1024 bit DSA private keys), and the $O(n^4)$ overhead of the rekeying protocol, the total amount of data that is exchanged by a node during the re-key is approximately 8 MB (with a 20 node committee). The overhead drops to 4 MB per node if the committee size is reduced to 16.

Total shares	f_e	Probability of more than t bad nodes		
20	.1	> 8	$<$	0.0061
		> 10	$<$	0.0003
		> 12	$<$	1.01e-05
	.25	> 15	$<$	0.001
16	.05	> 6.4	$<$	0.0004
		> 7.2	$<$	8.1e-05
	.1	> 9.6	$<$	0.0001
		> 11.2	$<$	5.05e-06

Table 1: Probability of Key Exposure, assuming an expected fraction f_e of bad nodes.

3.2 Security Analysis

We consider two different attacks: the eclipse attack (in which the malicious nodes try to take over the entire neighborhood of a good node, hence isolating it from the system), and a data erasure attack (in which a malicious node pretends that data that was published in a DHT does not exist).

Eclipse Attack. Suppose node i joins the system and i's ID is hashes to a malicious node j (i.e. j is the closest node to i in the system). Node j wants to isolate i, and provides i with a "bad" neighborhood certificate C. Note that if C is simply signed with the wrong key, i can immediately discard it. However, suppose C is signed with the correct key, but is old. Many more nodes have joined (and left) the system since C was produced, and moreover, many (or even all) of the alive nodes pointed to by C are now corrupt, but perhaps belong to different spans. If i were to use this certificate to seed its routing table, then it could be isolated (or eclipsed), since all its neighbors would always return other bad nodes as neighbors and so on. However, before accepting the certificate, i produces a nonce and challenges j to produce a valid signature that binds the nonce to C. Since the bad nodes do not have sufficient shares to sign (any statement), the challenge will fail, and i will not accept C. The only certificate that can be signed is the current valid one, which will ensure that i joins the group correctly.

Data Erasure. Now consider a DHT implementation based on our secure routing primitive. Assume that a data item d is (successfully) published, and then i performs a lookup for it. The lookup reaches malicious node j, which tries to convince i that the item does not exist. First, j has to convince i that j belongs to the span which covers d. If j does belong to the correct span, then it can just produce the current (valid) certificate. It is possible that j does not belong to the correct span, but has an old cached span certificate which did cover d. In both cases, when j returns a negative answer, i will issue a challenge, which j will not be able to respond to. Node i can then sequentially contact other nodes in the j's span and either get a pointer to a closer span or get the item d itself. (Note that the same argument also applies during forwarding, in case j asserts that it is not in d's span and is not aware of a closer span id).

4 Summary

We have presented a new protocol for securing distributed hash tables. Our protocol does not penalize the lookup path, but does currently impose a periodic heavy overhead (due to the proactive secret sharing protocol we have used). Perhaps more importantly, the security properties of our protocol are relatively easy to verify, and a single challenge mechanism is used to secure the protocol against all forms of attacks.

References

[1] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proc. 9th (ACM) conference on Computer and Communications Security*, pages 88–97. (ACM) Press, 2002.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[3] J. Douceur. The Sybil attack. In *IPTPS*, 2002.

[4] A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. In *ESA*, 2005.

[5] A. Haeberlen, P. Kouznetsov, and P. Druschel. The case for Byzantine fault detection. In *HotDep*, 2006.

[6] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *ACM Conference on Computer and Communications Security*, pages 100–110, 1997.

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

[8] R. Morselli. *Lookup Protocols and Techniques for Anonymity.* PhD thesis, University of Maryland, College Park, 2006.

[9] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA*, 1997.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *SIGCOMM*, 2001.

[11] R. Rodrigues and B. Liskov. Rosebud: A scalable byzantine-fault-tolerant storage architecture. MIT LCS TR/932, Dec. 2003.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[13] D. Schultz, B. Liskov, and M. Liskov. Mobile proactive secret sharing. MIT CSAIL Research Abstract. `http://publications.csail.mit.edu/abstracts/abstracts06/das/das.html`, 2006.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[15] B. Zhao, K. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California, Berkeley, 2001.

[16] L. Zhou, F. B. Schneider, and R. van Renesse. Apss: Proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3):259–286, August 2005.