Chord-PKI: A Distributed Trust Infrastructure based on P2P Networks

Agapios Avramidis\(^a\), Panayiotis Kotzanikolaou\(^a\), Christos Douligeris\(^a\), Mike Burmester\(^b\)

\(^a\)Department of Informatics, University of Piraeus, Karaiskaki & Dimitriou 80, Piraeus 18534, Greece
\(^b\)Department of Computer Science, Florida State University, Florida 32306-4530

Abstract

Many P2P applications require security services such as privacy, anonymity, authentication, and non-repudiation. Such services could be provided through a hierarchical Public Key Infrastructure. However, P2P networks are usually Internet-scale distributed systems comprised of nodes with undetermined trust level, thus making hierarchical solutions unrealistic. In this paper, we propose Chord-PKI, a distributed PKI architecture which is build upon the Chord overlay network, in order to provide security services for P2P applications. Our solution distributes the functionality of a PKI across the peers, by using threshold cryptography and proactive updating. We analyze the security of the proposed infrastructure and through simulations, we evaluate its performance for various scenarios of untrusted node distributions.

Keywords: Peer-to-Peer (P2P) Networks, Security Services, Threshold Cryptography

1. Introduction

Peer to peer (P2P) networks have received considerable attention in the last few years. In particular, one class of P2P networks, namely structured overlays [1, 2, 3], seems a very attractive choice for building large scale systems. Almost all structured overlay networks utilize Distributed Hash Tables (DHTs). The DHT uses a collision-resistant hash function – e.g. a cryptographic hash function such as the Secure Hash Algorithm (SHA) family of functions [4] – in order to assign identifiers to nodes and keys\(^1\). Moreover, the DHT allows the look up operations (get and put) to be performed with logarithmic cost in terms of communication messages. DHTs offer a desirable set of properties for distributed applications such as:

- Load balancing. DHTs provide a degree of load balance, due to the cryptographic hash function which assigns (almost evenly) key identifiers to nodes.
- Decentralization. No peer is more important than any other
- Scalability. The cost of a get (and put) operation scales logarithmically to the number of nodes in the system.
- Availability. DHTs provide protocols that efficiently handle high churn situations where nodes join and leave the system continuously. These protocols guarantee the success of the lookup operation.

Until recently, the main focus of research regarding DHTs was targeted to the performance of the lookup protocols, the overlay topology (e.g. Rings, Tori, Butterflies, de Bruijn Graphs and Skip Graphs topologies), load balancing and search issues such as range queries, multi-attribute and aggregation queries. An excellent survey on the above issues can be found in [5]. Although the aforementioned properties make P2P networks desirable for building distributed applications, standardized and transparent security mechanisms are still required, in order to support their wide deployment.

As in any network architecture, security in P2P networks can be structured in a layer-based approach as illustrated in Figure 1. The first layer of security involves the overlay network. Security mechanisms at this

---

\(^1\)These keys correspond to indices to objects such as files, and are not keys in the cryptographic sense
layer will protect from attacks against overlay routing such as incorrect routing lookups (e.g., [6, 7, 8, 9, 10]), incorrect routing table updates (e.g., [6, 7, 9, 11, 10]), network partitioning attacks (e.g., [7]) and Sybil attacks at the overlay layer (e.g., [6, 8, 12, 13, 14, 9, 15]). Security at the overlay network (routing) layer is very important, since it guarantees the robustness of the infrastructure, i.e., network connectivity and message delivery. Note that routing security protocols typically require the employment of node authentication, data encryption and integrity services, which in turn require the employment of a key management service.

The second layer of P2P network security involves the application layer. The open nature of P2P applications makes the system vulnerable to malicious users. For example in P2P file sharing applications, malicious users might alter the content of a specific file (content infringement), deliberately provide files whose contents are different from their description (content poisoning), insert "bad" chunks/packets into a valid file (content polluting) or make use of the network without contributing resources (freeloading). Such application-layer attacks can be prevented or detected with higher layer security services such as user authentication, data integrity and encryption, non-repudiation of delivery and receive. As in the previous case, application-layer security services also require the employment of a key management service.

Key management services can be provided through trust infrastructures, which would enable peer nodes to authenticate and distribute cryptographic keys. A possible solution would be the use of a centralized Public Key Infrastructure. However, the open nature of P2P applications, imposes certain restrictions related to security. P2P users usually belong to different organizations, groups and locations in a worldwide scale. Thus, it is practically impossible to find a single entity that every user is willing to trust as the root of the trust hierarchy. Another issue with a universal centralized certification authority is that the cost may be prohibitive or impractical [16]. An interesting alternative is to explore possible solutions which distribute the certification functionality between the overlay nodes.

**Contribution**

In this paper we propose **Chord-PKI**², a distributed Public Key Infrastructure, which is embedded into the Chord [1] structured overlay network. Chord-PKI distributes the certification, revocation, storage and retrieval functionalities among the nodes by utilizing the Chord lookup protocol, thus making the architecture autonomous and fully distributed, in contrast to external and/or centralized PKI architectures. Our model guarantees security, under strong cryptographic assurances. The system uses well known cryptographic techniques as building blocks, such as threshold cryptography [18] and proactive updating [19]. Since our solution utilizes the Chord, it is designed to deal with security problems of the Chord routing protocol and it guarantees resistance to distributed attacks through redundancy.

Since Chord-PKI is an autonomous key management service, it can be used to support security services both into the Chord layer (e.g., certified node identifiers), as well as in the application layer for DHT based applications, such as distributed file sharing [20, 21], distributed filesystems [22, 23], multicasting [24, 25], caching [26] and content delivery networks [27, 28].

The rest of the paper is organized as follows. Section 2 describes related work on P2P security. Section 3 defines the functional, security and efficiency requirements for the implementation of a distributed PKI. Section 4 describes the building blocks of the proposed solution. Section 5 introduces the Chord-PKI architecture and describes the protocols utilized by the Chord-PKI.

²A high level description can be found in a preliminary version of this paper [17]
Section 6 analyzes the security of the Chord-PKI, while section 7 evaluates its performance. Finally section 8 concludes the paper.

2. Related work

Overlay networks are not secure since they are designed to be open and work with a large number of unknown parties. The distributed nature of overlay networks makes the security problem interesting since there is no central system to protect. Unfortunately, the existence of malicious users is very common, thus the system must be resilient to their presence. Moreover, security mechanisms must not compromise the scalability and the distributed nature of such systems.

In [7] a number of attacks against structure overlay networks are presented, where a malicious node may forward lookups to an incorrect or non-existent node. Moreover, a malicious node could corrupt the routing tables of other nodes by sending incorrect routing updates. Another type of routing attack exploits the way new nodes join the system and directs them to incorrect networks, thus partitioning the system. As a first defense against routing attacks the authors propose that node identifiers should be produced in a verifiable way (e.g. basing the node identifier on a cryptographic hash of its IP address). This approach is employed in Chord-PKI. Furthermore, the authors take into consideration the use of long-term identities for node identifiers based on public keys.

Castro et al. [6] define the basic requirements for secure routing in structure overlay networks. They state that secure routing requires secure assignment of node identifiers, secure routing table maintenance and secure message forwarding. As a solution to secure assignment of node identifiers they propose to delegate the problem to a central, trusted authority. A trusted set of certification authorities (CAs) assign node identifiers to principals and sign nodeId certificates, which bind a random nodeId to the public key that represents the principal and its IP address. The CAs ensure that the identifiers are randomly chosen from the id space and prevent nodes from forging identifiers.

An alternative choice to the central certification authority is a distributed trust infrastructure which exploits the trust relationships between the P2P participants. Wolf [29] proposes a Public Key Infrastructure for P2P systems. The proposed solution exploits the underlying overlay network (Chord) to provide efficient search for certificates. The author uses Maurer’s PKI [30] to model complex trust-relationships which arise in a P2P environment. The proposed solution does not handle certificate revocation and expiration since the employed trust model does not deal with these aspects.

In [17] a distributed PKI is presented, which distributes the cost of certificate storage and retrieval among the peer nodes of the Chord overlay network. The cost of signing node’s certificates is also distributed among a group of trusted nodes, by employing cryptographic techniques such as threshold cryptography. This model is extended in this paper. In the same spirit, Lesuer et al. [31] propose a distributed certification system for overlay networks, where the operation of signing a certificate is distributed to a fixed ratio of nodes that hold a share of a secret key. While the ratio is fixed, the actual number of nodes required to sign a certificate increases (decreases) as new nodes enter (leave) the system. Thus, the certification costs are distributed among the nodes of the overlay network.

Another decentralized trust management approach can be build upon reputation-based mechanisms [32, 33, 34, 35, 36]. Although reputation models fit well with the nature of P2P applications and provide adequate solutions in several cases, they are not always appropriate. Reputation-based models are usually based on the local trust views of the users (peer nodes) in order to build the global trust view of the network. Distributing the global trust view to all the nodes is not always easy to achieve. Moreover, in reputation-based systems the trust views (either local or global) of the nodes are not protected with strong integrity protection mechanisms, such as cryptographic integrity checks or digital signatures. The integrity of the trust values is usually protected by storing the trust views of a node to several other nodes and by using majority voting in case of disputes. Although this may work with loose security models, in strong security models where nodes may alter stored or transmitted data, reputation-based models are not applicable. The reputation based approaches could support security services such as integrity, encryption, and non-repudiation, only if they are combined with public key based solutions. Thus, the design of a decentralized, autonomous and embedded trust infrastructure which provides authentication and cryptographic services for P2P systems, remains an interesting challenge.

3. Requirements for a P2P Public Key Infrastructure

Several requirements need to be satisfied, for the implementation of a distributed PKI within a P2P network such as Chord. These can be divided into functional, security and efficiency requirements.
3.1. Functional requirements

These concern the design of the required PKI services and include:

1. **Node Certification.** Any node must be able to request a public key certificate. Upon receipt of a request, the PKI service issues the certificate, provided that the requesting node satisfies some predefined properties (e.g., a trust or reliability metric).

2. **Node Revocation.** Any certificate issued in the past by the PKI service can also be revoked by the PKI service, if the corresponding node fails to satisfy some predefined properties.

3. **Certificate storage.** The PKI service should store both the valid and the revoked public key certificates.

4. **Certificate retrieval.** The PKI service should provide retrieval functionality both for the valid and the revoked certificates.

3.2. Security requirements

These concern the security controls that should be in place, in order to protect the infrastructure from various attacks.

1. **Availability.** The PKI service must always be available to nodes requesting certification. Moreover, the issued certificates (valid or revoked) must always be available to other nodes for retrieval. A more relaxed assumption is that the certification service may be unavailable only for a limited amount of time and for a small fraction of certification and retrieval requests.

2. **Resiliency.** The PKI service must be resilient to malicious and non-malicious faults, provided that the number of faults is bounded by a threshold. This means that the system should not be vulnerable to single point failure or to failures of a limited number of nodes.

3. **Unforgeability.** Malicious nodes must not be able to generate a forged certificate (or a forged certificate revocation), provided that the number of malicious nodes is bounded by a threshold.

4. **Proactive security.** The system must be protected from an active adversary which is able to adaptively compromise nodes over long time periods. In particular the system must provide forward security [37], i.e., nodes that have been compromised in a certain time period should not be able to collaborate with nodes compromised in another period.

5. **Support secure communication.** The PKI service must provide nodes with the ability to support confidentiality, integrity, authentication and non-repudiation of origin, of submission and of delivery services.

3.3. Efficiency requirements

Finally, the PKI service must be efficient in terms of certification, revocation, storage and retrieval costs and it must scale well in terms of the expected number of participating nodes. More particularly, the following requirements must be satisfied:

1. **Scalability.** The cost of the PKI service for certification, revocation, storage and retrieval must efficiently scale to the number of nodes in the system.

2. **Distribution of functionality.** The certification, revocation and storage functionality of the PKI service should be evenly distributed among the peers and no peer node should be more burdened than any other, over the lifetime of the system. This can be further divided into the following requirements:

   (a) **Load balancing of certification and revocation.** The PKI service must efficiently issue or revoke a certificate. Ideally, the cost for certification and revocation should be evenly distributed among the peer nodes. A more relaxed requirement would be the circulation of the certification and revocation functionality. More specifically, some peer nodes may have added functionality for certain time periods, which is recirculated between all the peers. In this way in the long term the certification and revocation functionality will be evenly distributed among all the peers.

   (b) **Load balancing of certificate storage.** The PKI service must efficiently store the certificates (valid or revoked) among the nodes. Ideally, the storage cost should be evenly distributed among the peer nodes.

3. **Tolerance to frequent changes.** The PKI service must tolerate frequent joins and leaves, since P2P networks face high churn situations where nodes join and leave the system continuously.

---

3Note that for strong guarantees, several non-repudiation services may also require a time-stamping service
4. Building Blocks

Our goal is to build a distributed PKI for the Chord structured overlay network. Since our system enhances the Chord protocol, we briefly describe the functionality of Chord. After that, we shall describe the cryptographic primitives that will be used in order to distribute the cryptographic protection mechanisms and in particular threshold cryptography and proactive key updating.

4.1. Chord

Chord [1] is a distributed lookup protocol, suitable for dynamic P2P networks with frequent node arrivals and departures. The protocol supports one basic operation: given a key (e.g. a file name), it maps that key onto a node. In particular, Chord uses a hash function (e.g. one of the SHA family [4]) to assign to each node and key an \( m \)-bit identifier. The node’s identifier is chosen by hashing the node’s IP address, while the key identifier is produced by hashing the key. Identifiers are ordered on an identifier circle modulo \( 2^m \) (Chord Ring). A key with identifier \( K_i \) is assigned to the first node (clockwise) which satisfies the following condition \( K_i \leq N_j \), where \( N_j \) is the identifier of node \( N \). Node \( N \) is the so called successor node of key \( K_i \). Note that by using a collision-resistant and pre-image resistant hash function a malicious node cannot select an identifier of its choice. Figure 2 presents an example of a Chord Ring with \( m = 6 \).

Routing in Chord. Chord offers a scalable lookup operation with an average hop count of \( 1/2 \log_2 |N| \), where \(|N|\) is the number of nodes in the system. Each node in the system tracks its predecessor (the previous node in the circle), its successor (or a list of successors) and a routing table called finger table. Each entry in the finger table contains the identifier of a node and its IP address (and port number). The finger table contains \( m \) entries, where \( m \) is the bit length of the identifier produced by the hash function. The \( i_{th} \) entry in the finger table of node \( N \) contains the identifier of the first node \( S \) that succeeds \( N \) by at least \( 2^{i-1} \) on the identifier circle (i.e., \( S = successor((N + 2^{i-1}) \mod m) \)), with \( 1 \leq i \leq m \). Using the finger table, each hop is at least half the remaining distance around the ring to the target node. Figure 3 illustrates how the lookup protocol works in a system when the identifier bit length is 6. In particular, we demonstrate the scenario where node 8 (N8) is looking for key 58 (K58). Dotted arrows denote the possible routes from a node (8 and 42) while the solid arrows show the actual path from node 8 towards the successor node for key 58.

Join. A new node \( N' \) joins the system by asking any existing node \( N \) to find its immediate successor. This process sets the successor pointer for \( N' \) but does not make the rest of the network aware of \( N' \). This task is part of the stabilization protocol [1].

4.2. Threshold Cryptography

Threshold cryptosystems [18] enable the distribution of a cryptographic operation, such as a digital signature, among a set of entities. With a \((t, n)\) threshold signature scheme, a set of \( n \) entities is able to share the secret key \( SK \) of a public/secret key pair \((PK, SK)\), in such a way that any \( t \) (or more) entities can use their shares of \( SK \) to jointly generate a digital signature which can be verified with the public key \( PK \). We denote the \( j_{th} \) share of the key as \( SK_j \), \( j \in [1, n] \). On the other hand, any set of less than \( t \) entities is unable to generate a valid signature. In
order to sign a message $M$, any set of $t$ share holders use their shares $SK_j$ of the key $SK$ to compute partial signatures $SIG_{SK}(M)$. The partial signatures are sent to one of the $t$ entities which combines them with its own partial signature to produce the signature $SIG_{SK}(M)$ on the message $M$, as shown in Figure 4. Note that any subset of $t$ (or more) valid key shares can be used to produce a valid signature and that any entity can act as a combiner. By using the corresponding public key $PK$, any entity (and in particular the combiner) can verify the validity of the generated signature. If the signature fails (due to an incorrect partial signature), another subset $t$ of key shares can be used to produce the signature.

The basic threshold schemes cannot deal efficiently with the problem of incorrect partial signatures. This problem can be efficiently solved by using robust threshold schemes [38, 39]. A robust threshold scheme provides verifiability of the correctness of the partial signatures, since they incorporate mechanisms to distinguish valid from invalid partial signatures. Although this increases the computation costs, it provides resilience to dishonest or misbehaving share holders, since invalid partial signatures can be identified. Another enhancement of threshold cryptography, first proposed in [40], enables the computation and sharing of the partial keys without a trusted dealer (also known as randomized threshold cryptography). In this case, all the parties participate in the generation of the shared secret in such a way that no one knows the secret key to be shared, even in the key generation phase.

4.3. Proactive Cryptography

A threshold cryptosystem can also be enhanced by using a proactive update of key shares. If an adversary is given the ability to compromise share holders over a long time period, it may be easier for the adversary to ultimately compromise the secret key. With proactive schemes (e.g. [19, 41, 42]) the entities that hold the key shares periodically engage in a share refreshing (updating) protocol. This enables the entities to update their shares, without changing the secret key itself. The updated shares determine a new $(t, n)$ sharing of the secret key $SK$. After the share updating, the entities discard the old shares and use the new shares to generate partial signatures. Note that it is not possible to combine partial signatures generated by key shares of different update periods in order to produce a valid signature with the key $PK$. Briefly, an update relies on the homomorphic property that if $SK_1, SK_2, ..., SK_n$ is a sharing of $SK$ and $SK_1', SK_2', ..., SK_n'$ is a sharing of $SK'$, then $SK_1 + SK_1', SK_2 + SK_2', ..., SK_n + SK_n'$ is a sharing of the key $SK + SK'$. If $SK' = 0$, then the result is a new sharing of $SK$. Several proactive updating schemes [42] also allow the threshold scheme to change its configuration from $(t, n)$ to $(t', n')$. This adds flexibility to the number of sharers as well as to the security parameter of the key sharing.

5. The Chord-PKI

The Chord-PKI implements the functional requirements described in Section 3, by combining and extending the building blocks described in the previous section. Our solution minimizes the burden imposed by the use of public key cryptography by distributing the cryptographic functionality within the peers through threshold cryptography. It also minimizes the storage and retrieval requirements for the public keys, by exploiting the distributed storage and retrieval functionality of the Chord protocol. The certificate and certificate revocation list (CRL) storage is evenly distributed among the system nodes and is implemented as a Chord put operation, thus balancing the storage cost. Moreover, the lookup of a certificate also exploits the Chord functionality and is implemented through a simple Chord get operation. Below we describe the specific assumptions for the deployment of the Chord-PKI service, as well as the required protocols that implement the system operations.

5.1. Assumptions

Before the actual operation of the Chord-PKI a bootstrapping protocol is executed asynchronously by a number of nodes (this can be triggered by a node requesting other nodes to participate in the bootstrapping). The nodes will partition the network identifier space into a number of continuous virtual segments. We assume that after this partitioning, the size of each segment will be publicly known to every node. We assume that all nodes involved in the bootstrapping, are mutually trusted. Furthermore, we assume that each segment
contains a threshold of at least \( t \) trusted nodes (a system parameter).

Although the assumption on the initially trusted nodes during the bootstrapping is a “strong” assumption, it is not possible to distribute and manage trust in any system, without assuming that some initial trust exists. Also, this assumption is feasible in any P2P environment, since it is reasonable to expect that some nodes will have previous off-line trust relationships (e.g. [12]).

5.2. Protocols

The Chord-PKI utilizes the following protocols to support the functionality of the distributed PKI service.

5.2.1. Bootstrapping

The nodes participating in the bootstrapping protocol partition the Chord identifier (ID) space into \( s \) virtual segments, as shown in Figure 5. Thus, if the identifier space of a Chord overlay network is \([0, 2^n - 1]\), the identifier space of each segment is \( 2^n / s \). The segments are continuous, i.e. \( SEG_i = ((i-1) \cdot (2^n / s), i \cdot (2^n / s) - 1) \), \( i \in [1, s] \). The value \( s \) is a system parameter and by using an asymptotic analysis model with an identifier space of \( 2^n \), a realistic number of \( s \) is of order \( o(m) \) (for example \( s \) could be 256 if SHA256 is used as a hash function). Suppose that after the partitioning, in each segment \( SEG_i \) there are (at least) \( n \geq 2t + 1 \) nodes, where \( t \) is the system threshold for trusted nodes. These nodes are the initial certification nodes of each segment, which in turn will be able to certify any other node within \( SEG_i \), for a certain period. After this period the set of the certification nodes will be updated by other nodes.

After the partitioning of the Chord identifier space, in each segment \( SEG_i \), \( i \in [1, s] \), the bootstrapping protocol is executed. This can be performed asynchronously within the segments. During this protocol, the initial \( n \) certification nodes securely generate and certify the public/secret segment key pair \((PK_i, SK_i)\) of \( SEG_i \), using a public key cryptosystem, (such as RSA or DSS). The secret key \( SK_i \) will be used for the certification of any existing or forthcoming node within the segment \( SEG_i \), as described in Section 5.2.2. The public segment key \( PK_i \) is self-certified and is used for the verification of any node certificate signed with the segment secret key.

The secret key \( SK_i \) of the segment \( SEG_i \) is jointly generated by the \( n \) nodes, by using a verifiable random secret sharing algorithm (e.g. the Joint-Exp-RSS algorithm described in [43]), using a \((t,n)\) threshold sharing, where \( n \geq 2t + 1 \). In this way, the actual secret key is not known to any node, while any set of at least \( t \) nodes will be able to generate a signature with \( SK_i \). Less than \( t \) nodes will not be able to generate a valid signature. Alternatively, and since all the initial nodes are trusted, one node can act as a dealer, generate and distribute the key shares and the verification information and then discard the secret key. At the end of this protocol, each certification node \( C_{i,j} \in SEG_i \), \( 1 \leq j \leq n \) will have a key share \( SK_{i,j} \) of the key \( SK_i \). After the successful sharing, the nodes (at least \( t \) of them) will use their shares to generate a self-signed certificate\(^4\) \( CERT_i = SEGSK_i(i, PK_i) \).

\( CERT_i \) is exchanged to all the certification nodes of the other segments. This can be achieved in several ways. For example, the certification nodes can exchange the segment public keys with out-of-band methods. This is scalable since only a fraction of nodes of the order \( o(m) \) need to exchange keys. Also, in P2P environments it is common that the nodes have some form of off-line trust relationship with some other nodes, such as the first point of contact needed in order to join the network. Another solution is to use a node (acting as a point of contact during the bootstrapping), through which all the segment public keys are exchanged.

After all the certification nodes have exchanged

\(CERT_i \) with the nodes of the other segments, each segment \( SEG_i \) registers a new cryptographic key pair \((PK_i, SK_i)\). After that, each segment is able to sign any certificate with the private key \( SK_i \) and use the public key \( PK_i \) as its certification key.

\(^4\) All the certificates used (segment or node certificates) may be formatted following the ITU-T X.509 standard, also containing other attributes such as certification time, expiration time etc. For simplicity we omit these values.
the segment public keys, they generate and store a certified list of the public segment keys as $CL_i = SIG_{SK_i}(PK_1, PK_2, ... PK_j)$ and each one stores the list. The certified list $CL_i$ is also stored at the nodes that correspond to the Chord identifier $H(CERT_i)$, $1 \leq i \leq s$, where $H$ is a cryptographically secure hashing function used by the Chord implementation for node and Chord-key assignment [1].

Finally, in each segment, the certification nodes generate a list $List_i$ containing: the identifiers and IP addresses of the current certification nodes, the expiration date and time of the current threshold sharing and the segment’s boundaries, i.e. $List_i = [ID(C_{i,j}), IP(C_{i,j}), ThresExpTime, ID_{start-i}, ID_{end-i}], \forall j \in [1,n]$. The list is also signed with the segment secret key $SK_i$ and is exchanged between all the current certification nodes.

Remark: Although for simplicity the bootstrapping protocol is assumed to take place in a single step of execution, in realistic implementations, it is possible to execute the bootstrapping asynchronously in different segments at different time periods. Note however that the segmentation of the network shall be decided during the first step of the bootstrapping. Until all the segments have performed their bootstrapping protocol, the nodes belonging into segments that have not yet performed the bootstrapping, will not be able to use the certification services.

5.2.2. Node Certification

In order to certify the public key of a node $N$, the following protocol is executed.

1. Segment identification. First, the node $N$ identifies the segment $SEG_i$ to which it belongs, the number of segments $s$ and the segment certificates.

2. Key pair generation. The node $N$ generates a pair of public/secret keys $(pk_N, sk_N)$ of a public key cryptosystem such as RSA or DSS.

3. Lookup for a certification node. The node $N$ finds a certification node of the segment $SEG_i$ it belongs to, according to the protocol described in Section 5.2.6. For simplicity and without loss of generality, we assume that the contacted certification node is $C_{i,1}$ (i.e. the certification node of $SEG_i$ that holds the segment secret key share $SK_{i,1}$).

4. Certificate issuing. Node $N$ sends to $C_{i,1}$ its public key, along with a certification request and a proof of knowledge of the corresponding secret key $sk_N$ (for example following the PKCS#10 certificate request format). If $C_{i,1}$ decides to certify $N$, then it can act as a combiner of a threshold signature (such as threshold RSA or DSS) and issue a certificate for the node $N$. For this purpose, the combiner generates a (partial) signature $SIG_{SK_{i,1}}(i, N, pk_N)$. Additionally, it contacts with $t-1$ other certification nodes of its segment (these can be selected randomly from $List_i$) and requests partial signatures from these nodes for the message $(i, N, pk_N)$. After the combiner has received the partial signatures $SIG_{SK_j}(N, pk_N),..., SIG_{SK_i}(N, pk_N)$, the combiner is able to produce a valid certificate for the node $N$. By combining the $t$ partial signatures, the combiner generates a valid certificate $cert_{j} = SIG_{SK_i}(i, N, pk_N)$ for the node $N$.

Note that if a verifiable threshold signature scheme is used, then any fault during the partial signature generation – malicious or not – will be traced.

5. Verification. The combiner $C_{i,1}$ verifies the signature of the certificate and if it is not valid, it repeats the previous step with another subset of certification nodes.

6. Certificate Delivery. The combiner sends the certificate to node $N$. The node $N$ stores the certificate $cert_N$ and it also stores the contact information for the certification node $C_{i,1}$ in its local cache for future communication.

5.2.3. Certificate Revocation

The certification nodes of each segment $i$ receive revocation requests concerning nodes that belong to their own segment, originated by certified nodes. A revocation request is considered as valid only if it is signed by a certified node, which may belong to any segment. The certification nodes periodically examine the revocation requests and may decide to revoke a node certificate. In order to revoke the certificate $cert_N$ of a node $N$ the following protocol is executed.

1. Selection of revocation set. A certification node $C_{i,1}$ decides to act as a combiner and selects $t-1$ other certification nodes from $List_i$.

2. Revocation signing. The revocation set will sign a revocation message $rev_N = SIG_{SK_i}(cert_N, RevTime)$ of the public key of the misbehaving node, where $RevTime$ is the time of the revocation. The revocation message must then be stored in a certificate revocation list (CRL).
5.2.4. Certificate and CRL Storage

The certificate and the CRL storage is distributed among all the Chord nodes, as shown in Figure 6. Each node has a local certificate and CRL store and is responsible to retain a number of certificates of other nodes. The certificates and the CRLs are stored in a distributed manner as follows:

1. **Certificate Generation.** Node \( N \) follows the protocol described in 5.2.2 and acquires its certificate \( cert_N \).

2. **Certificate Storage.** Node \( N \) requests node \( Y_1 \) to store its certificate, where \( H(cert_N, 1) \rightarrow Y_1 \). Thus any node that wants to verify a claimed certificate of another node, knows where the received certificate should be stored.

3. **CRL Storage.** In the case when the certificate \( cert_N \) is revoked by the certification nodes, the revocation \( rev_N \) is also stored in the local CRL of the node \( Y_1 \) that corresponds to the Chord identifier, \( H(cert_N, 1) \rightarrow Y_1 \). Thus, any node that requires to verify a certificate \( cert_N \) will check the local certificate and CRL store of \( Y \).

4. **Redundancy.** For redundancy, the certificate \( cert_N \) or the revocation \( rev_N \) is stored into \( t \) instead of one nodes. The node \( N \) selects the storage nodes (including the first storage node) \( Y_1, ..., Y_t \) as:

\[
H(cert_N, i) \rightarrow Y_i, \quad i = 1, ..., t.
\]

5.2.5. Certificate – CRL retrieval and verification

As explained above, a certificate and a CRL retrieval, is the same as a Chord look up. Any node is able to look up and verify a certificate of another node belonging to any segment, as shown in Figure 6. The following protocol is applied when node \( X \) receives a claimed certificate \( cert_N \) presented by node \( N \) as valid.

1. **Look up the appropriate storage node.** The node \( X \) applies the Chord hash function to the claimed certificate \( cert_N \), in order to find the \( t \) storage nodes of the certificate. Without loss of generality, say that node \( X \) starts from the first storage node \( Y_1 \) (i.e. \( H(cert_N, 1) \rightarrow Y_1 \)).

2. **Look up the Certificate.** Node \( X \) will query the certificate and CRL store of node \( Y_1 \) for the particular certificate. If the CRL store of \( Y_1 \) contains a revocation of the certificate, then \( X \) rejects \( cert_N \) as revoked. If the certificate store of \( Y_1 \) does not contain the certificate then goto step 3, else goto step 4.

3. **Redundant Look up of Certificate.** Look up if node \( Y_i \rightarrow H(cert_N, i) \), \( i = 2, ..., t \) contains the certificate in its store. If the certificate is found goto step 4. If not, while \( i \leq t \) repeat this step. If after all the \( t \) storage nodes have been examined the certificate is not found, then \( X \) rejects \( cert_N \).

4. **Certificate Verification.** Node \( X \) verifies the signature of the certificate \( cert_N \) by using the corresponding segment certificate and by checking the expiration time of the certificate. If all checks are successful then the node \( X \) accepts \( cert_N \) as genuine.

5. **Check for Certificate Revocation.** Finally, the node \( X \) will check the CRL store of the rest of the \( t \) storage nodes (the storage nodes that were not checked during step 2), for a possible revocation \( rev_N \) of the certificate. The certificate is considered as revoked if at least one valid (signed) revocation is found. Else, if no revocation is found, the certificate is considered as valid.

5.2.6. Certification node look up by ordinary nodes within a segment

The certification of a public key (or a revocation claim for an issued certificate) requires the communication between an ordinary node \( N \) and a certification node \( C_{i,j} \). This can be implemented with the following protocol. Assume that node \( N \) wishes to find a certification node of segment \( SEG_i \). Then it performs the following steps:

1. **Segment Identification.** Node \( N \) calculates the bounds of \( SEG_i \) as \([i - 1) \cdot (2^m/s), i \cdot (2^m/s) - 1]\.)
2. **Local Look up.** The node $N$ looks up in its finger table for a node whose identifier falls between the segments bounds (e.g. $[(i-1) \cdot (2^n/s)] \leq X \leq i \cdot (2^n/s) - 1$). If such a node exists go to step 4, otherwise continue.

3. **Random Selection.** Node $N$ selects a random identifier $id$ with $[(i-1) \cdot (2^n/s)] \leq id \leq i \cdot (2^n/s) - 1$ and locate the node $X$ that is responsible for the random id (i.e. $successor(id) = X$).

4. **Request Propagation.** The node $N$ sends a message to $X$, requesting the ID of a certification node within the same segment.

5. **Finding Certification Node.** Node $X$ can either be:
   a. A certification node. In this case $X$ returns its identifier and IP address to $N$.
   b. An ordinary node who has already been certified and has a certification node in its cache. In that case $X$ pings its cache entry and if the entry is alive it returns to $N$, the identifier and the IP address of the certification node. Otherwise proceeds as in step 5c.
   c. An uncertified node who does not know any certification node. In that case $X$ forwards the request to its immediate successor and the protocol continues from step 4.

6. **Request for Certification.** $N$ sends a certification request to the node returned by the previous step.

5.2.7. **Communication between certification nodes within a segment**

All the certification nodes that belong to the same segment can trivially lookup for each other, by accessing the current list of the certification nodes $List_i$, generated during bootstrapping (section 5.2.1) or updated after an updating period (section 5.2.9).

5.2.8. **Communication between certification nodes of different segments**

If a certification node $C_{i,j}$ of the segment $SEG_i$ wants to find a certification node $C_{k,l}$ of another segment $SEG_k$, then it executes a protocol similar to the one described in section 5.2.6.

5.2.9. **Updating the segment key shares**

At the end of the current threshold period $ThresExpTime$ the certification nodes will run the following update protocol, after which both the certification node set and the key shares will have been updated.

---

1. **Availability Check.** The certification nodes of $SEG_i$ communicate with each other to verify that at least $t$ nodes are active. If some nodes do not participate then the rest of the nodes will generate their shares (i.e. through Lagrange interpolation), provided that at least $t$ nodes participate. This procedure is also followed in the case where some nodes drop out half-way through the updating. They are also treated as non-participants and the rest of the nodes will generate their shares. At the end of this check all the $n$ shares of the key will be available.

2. **Updating the Set of the Certification Nodes.** The current certification nodes use a random function to select the new set of $n$ certification nodes within the segment for the next period. This can be constructed as follows. Each certification node provides a random binary string $R_j$ of sufficient length as an input to the hash function $^5$. Say that the result of the hash function is a Chord identifier $ID = H(R_1, ..., R_n)$. If $ID$ is an identifier outside the limits of the segment $i$ then repeat the process. Else, the successor node for the identifier $ID$ is a candidate certification node. In order to prevent a malicious node from manipulating the candidate certification node selection procedure, all the current certification nodes will sign their share and send it to all the other segment certification nodes. In this way a malicious node cannot replace the randomness selected by the others. Also, a malicious node cannot manipulate the selection by exhaustive search, since the hash function is pre-image resistant and each share $R_j$ has sufficient length (e.g. 256-bits). Note that if a significant portion of the input of a hash function is random, then the output is random.

   a. Lookup for the node with the identifier $ID$. If the node has already been certified and agrees to participate, one good candidate is found. Repeat the previous step until $n$ nodes are found.
   b. Each old certification node is assigned to a new certification node. Then, each old certification node encrypts its segment key share $SK_i$ with the public key of its assigned new certification node and sends the encrypted share to the new node, along with a proof of correctness. If a share is not valid, the new

---

^5For this purpose, a timer may be used, to provide loose synchronization.
node will raise a claim and the share can be reconstructed by at least \( t \) certification nodes of the old sharing.

3. **Proactive update.** The new nodes will proceed in a proactive update [19] of the key shares and then they will verify their new shares. After the proactive update the old shares are discarded both by the old and the new certification nodes.

6. Security Analysis

6.1. Threat Model

Our threat model assumes a Byzantine adversary. All entities, including the adversary, have polynomially bounded resources. We distinguish between two types of entities: trusted and untrusted. Trusted entities behave as expected (adhere to protocol requirements). Untrusted entities may deviate from their expected behavior. An untrusted entity is malicious if it actively attempts to undermine the security of the system, e.g., it attempts to disrupt message delivery or to manipulate the certification procedure. Compromised entities are untrusted entities whose private keys have been compromised by the adversary.

All the untrusted entities are controlled by the adversary. We consider both passive and active adversaries. A passive adversary will only eavesdrop on the messages exchanged by the entities. An active adversary may also corrupt or inject messages.

To deal with such adversaries we shall use appropriate cryptographic mechanisms. We will assume that the number of untrusted entities is bounded, less than a constant \( t \) — a system parameter, and will use threshold cryptography and redundancy to guarantee the security of our protocols.

6.2. Security of the Chord-PKI

We examine how the proposed Chord-PKI infrastructure satisfies the security requirements given in section 3.2, against the examined adversarial model. We analyze the security of the Chord-PKI in two layers; (i) security of the key management services provided by Chord-PKI (i.e. application-layer security) and (ii) security against Chord routing attacks (i.e. network-layer). Application layer security shows the robustness of the proposed infrastructure, while network layer security analysis shows the availability and reliability of the Chord-PKI in the presence of denial of service attacks on the underlying network infrastructure.

6.2.1. Security of the key management services

a) Resilience to untrusted nodes. Within each segment, the certification nodes are responsible to issue/revoke certificates, for a certain time period. At the end of the certification period, the set of the certification nodes of each segment is replaced by a new set of nodes. We examine the resilience of the Chord-PKI to untrusted nodes, both during the certification (working) periods and during the certification set updating periods.

**Theorem 1** (Resilience during the certification periods). If the number of untrusted nodes within a certification node set is less than \( t \), then the certification/revocation process provided by Chord-PKI cannot be controlled by the adversary.

**Proof.** This follows directly from the fact that a \((t, n)\) threshold system is used. Thus, in order to prove the resiliency of the system during the certification periods, it suffices to prove that for a given probability distribution \( p \) of untrusted nodes, the adversary can control \( x \geq t \) certification nodes only with negligible probability, if a proper threshold setting \((t, n)\) is applied. We use probability theory, in order to compute the proper \((t, n)\) set.

During a certification period, any node within a segment can either belong to the trusted or to the untrusted set. Since the nodes may change from one state to the other during their lifetime, we assume that within a period a node may belong to the untrusted set with a probability no more than \( p \), while it may belong to the trusted set with a probability of \( q = 1 - p \).

In each working period, \( n \) nodes out of the segment set have been randomly selected\(^6\) to act as certification nodes. Since the nodes may be re-selected in a future period, we use the binomial probability distribution. Thus the probability that \( x \) out of the \( n \) selected nodes belong to the untrusted set is given by the equation:

\[
Pr(P = x) = \binom{n}{x} p^x q^{n-x}.
\]  

The cumulative probability that at most \( t - 1 \) out of the \( n \) certification nodes belong to the untrusted set can be computed as:

\[
Pr(P < t) = \sum_{x=0}^{t-1} \binom{n}{x} p^x q^{n-x}.
\]  

\(^6\)Random selection is guaranteed since in the certification update process, the certification nodes are selected randomly – see Theorem 2.
By using Equation 2 we can compute, for given values of $p$, $n$ and $t$ the probability $Pr(P < t) \geq 1 - \varepsilon$ where $\varepsilon$ is negligible. Table 1 shows the probability that at most $t - 1$ untrusted nodes belong to the threshold set for various threshold settings $(t, n)$ and for various distributions $p$ of untrusted nodes. For example, in a network where the expected population of untrusted nodes is at most $p = 0.2$, an $(24, 49)$ threshold setting guarantees with probability $Pr(P < 24) \geq 1 - 5.3 \times 10^{-6}$, that the certification process cannot be practically compromised.

Thus, the proposed Chord-PKI architecture is resilient to untrusted nodes, for appropriate values $(t, n)$ of threshold sets.

### Table 1: Probability of selecting less than $t$ untrusted certification nodes for various distributions of untrusted nodes

<table>
<thead>
<tr>
<th>Threshold scheme $(t, n)$</th>
<th>$p = 0$ (trusted set)</th>
<th>$p = 0.1$</th>
<th>$p = 0.2$</th>
<th>$p = 0.3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6, 15)</td>
<td>$&gt; 0.999$</td>
<td>$&gt; 0.969$</td>
<td>$&gt; 0.834$</td>
<td></td>
</tr>
<tr>
<td>(12, 25)</td>
<td>$&gt; 1 - 1.49 \times 10^6$</td>
<td>$&gt; 0.998$</td>
<td>$&gt; 0.959$</td>
<td></td>
</tr>
<tr>
<td>(24, 49)</td>
<td>$&gt; 1 - 5.10 \times 10^6$</td>
<td>$&gt; 1 - 5.30 \times 10^6$</td>
<td>$&gt; 0.993$</td>
<td></td>
</tr>
<tr>
<td>(48, 97)</td>
<td>$&gt; 1 - 8.50 \times 10^6$</td>
<td>$&gt; 1 - 8.59 \times 10^6$</td>
<td>$&gt; 1 - 8.51 \times 10^6$</td>
<td></td>
</tr>
<tr>
<td>(96, 195)</td>
<td>$&gt; 1 - 2.95 \times 10^{12}$</td>
<td>$&gt; 1 - 3.02 \times 10^{12}$</td>
<td>$&gt; 1 - 7.5 \times 10^9$</td>
<td></td>
</tr>
</tbody>
</table>

#### b) Unforgeability

This comes directly from the system’s resilience to untrusted nodes. Since the adversary is polynomially bounded, forging a certificate (or a CRL) requires that the adversary controls at least $t$ certification nodes of a segment within a threshold period. However, as shown before, this is possible only with a negligible probability, for properly selected $(t, n)$ parameters. Therefore, the adversary cannot generate signatures with the segment secret key and thus forge certificates.

#### c) Proactive security

The Chord-PKI provides proactive security, since an active adversary cannot combine key shares from certification nodes which have been compromised in different threshold periods, as it is shown by the following theorem.

**Theorem 3** (Limitation of the attack time-frame). The adversary can successfully forge a signature, only if he can forge $x \geq t$ certification nodes within a threshold period.

**Proof.** Recall from Section 5.2.9 that during the certification set updating period, after the new threshold certification set has been selected, the old nodes proactively refresh their shares by adding sharings of zero. Then, the refreshed shares are securely transferred to the new certification nodes along with a proof of correctness (each new node receives one share) and the old shares are discarded. In this way, an active adversary who has compromised key shares in a period $i$ cannot combine them with shares compromised in period $j > i$ in order to construct the segment signature key, since the adversary will not know the zero sharing contained in each refreshed key share. Thus, the adversary can successfully attack the segment signature key only if he compromises at least $t$ shares within the same threshold period.

#### d) Availability of certificate storage

We examine the denial of service attacks from untrusted storage nodes, which refuse to provide certificates of other nodes from their local store. Chord-PKI preserves availability in the presence of malicious storage nodes through redundancy. The certificate (or the revocation of a certificate) of each certified node is stored into the $t$ distinct nodes, indicated by $ID_i = H(cert_i, i)$ for $i = 1 \ldots t$, as described in 5.2.4. Since each certificate is stored into $t$ nodes, the requested certificate can be retrieved if at least of them is trusted. Note that since the storage nodes are randomly selected by applying the collision-resistant hash function and since nodes may be selected multiple times, by applying the binomial distribution
the probability that all $t$ storage nodes are untrusted is given by $Pr(P = t) = p^t$. For example, with an expected population of untrusted nodes $p = 0.2$, and for a $(24, 49)$ threshold setting, the probability that all $t$ nodes storing a requested certificate are untrusted is $Pr(P = 24) = 0.2^{24} \approx 1.6 \times 10^{-17}$. Thus, it is practically impossible for the adversary to control all of them, for a given certificate.

6.2.2. Security against routing attacks

Since messages related with the Chord-PKI key management functionality are signed and encrypted before they are exchanged between nodes, the security attacks of the network layer basically involve routing attacks against the availability of the infrastructure, i.e. the lookup of the stored certificates. Chord inherently provides availability in the case of non-malicious faults, since if a node is not available at a given time, the first successor of this node automatically takes over the role of the failed node, through the Chord stabilization protocol. However, in our adversarial model we also consider malicious nodes that may attack routing in two ways: a) deliberately provide incorrect routing lookup and b) deliberately provide incorrect routing update. We examine the availability of the infrastructure for the aforementioned routing attacks.

a) Protection from incorrect routing lookups.

We examine how Chord-PKI addresses incorrect routing lookup attacks. By using an iterative implementation of Chord, the initial node performing a lookup (the requesting node) can always monitor the lookup process. In each hop of the lookup, the requesting node checks whether the request is getting closer to the destination until the requested data are returned. If the distance towards the destination is not decreased at each hop, the initial node can select another entry from its routing table to forward the query from an alternative path. The probability that an alternative path from the requesting node to the destination exists is high, if the malicious nodes cannot select their location (i.e. Chord identifiers). Since in Chord-PKI the node identifier is derived by applying a pre-image resistant hash function over the node IP address, a malicious node cannot select an ID of its choice. We consider a system with $N$ nodes, where the untrusted nodes are randomly distributed around the identifier space with probability $p$, $0 \leq p \leq 1$. We examine the probability of successful certificate lookup.

The probability of successful communication between two honest nodes (the node requesting a certificate and the node storing the certificate) is $P_s = (1 - p)^h$, where $h$ is the average path length ($h = 1/2log_2N$ in Chord). In case of failure, the initial node can forward the query to another routing table entry. Each chord node has $k = logN$ [1] entries in its routing table. A node will fail to lookup a certificate if and only if all its routing table entries are compromised. This probability can be computed as $Pr(x = k) = \frac{k}{x} (1 - P_s)^x P^x_s$.

Thus, the probability of finding at least one valid path towards the destination is $P(x < k) = 1 - Pr(x = k)$. Since each certificate is replicated into $t$ storage nodes, the probability of finding at least one valid certificate is $1 - Pr(x = k')$. For example, in a network with $N = 1024$ nodes, a fraction of untrusted nodes $p = 0.2$ and an $(6, 13)$ threshold setting, the probability of finding at least one valid certificate is approximately $1 - 4.5 \times 10^{-11}$.

b) Protection from incorrect routing table updates.

Since any node builds its routing table by consulting others, a malicious node may corrupt the routing tables by sending incorrect routing updates. The most vulnerable time for incorrect routing update attacks is during the joining operation. Since at this time joining nodes do not know anything about the overlay topology, they are at the mercy of their bootstrap node for building their routing tables. In such a scenario, a malicious node could provide routing table entries pointing to other malicious nodes and thus totally eclipse the honest node from the rest of the network. This attack can be prevented, by requiring that the join procedure is performed by the certification nodes. Recall from section 5.2.1 that for each segment $i$ the certified list $\text{List}_i$, provides the location of the current certification nodes.

Also, during the working period, a malicious node could return a routing update entry pointing to a wrong or non-existent node. A node receiving such updates can easily detect them by contacting the node to the claimed IP address and port number.

In addition, a malicious node could return routing table entries pointing to a distant, but honest node. This attack is possible but it can only affect the performance of the lookup. Furthermore, if the returned entries point to other malicious nodes, then a portion of the routing tables are controlled by the attackers. If these routing entries remain undetected, the probability of incorrect routing table entries, increases in each update of the routing table. In particular, in a network where the fraction of untrusted nodes (returning incorrect routing table entries) is $p$, then the probability that a routing table entry (of an honest node) is faulty is $Pr_f = p + (1 - p) p$. Since in a chord network with $N$ nodes, each routing table contains approximately $k = logN$ distinct entries,
then the probability that a node is eclipsed from the rest of the network is $P_r(x = k) = (P)^k = (p + (1 - p))^k$. For $k = 10$ and $p = 0.2$ this probability is approximately $3.6 \times 10^{-5}$.

In theory, in each routing table update operation the probability that a routing entry is incorrect may be increased, if each node does not verify the correctness of each entry. Since the routing table entries are used to effectively lookup certificates, an honest node can always determine if an entry of its routing table points to a valid path towards a valid certificate. If this is not the case, then this entry is recorded (not to be used in subsequent routing table updates) and removed from the routing table.

### 7. Evaluation-Simulations

#### 7.1. Computation costs

In order to evaluate the efficiency of the proposed system, we have implemented an RSA threshold signature scheme with 1024 bit modulus in two variations: a basic threshold scheme and a verifiable (robust) threshold scheme, where each partial signature can be verified. In the second case, malicious nodes participating in a certification set can be identified and excluded. Since a large-scale P2P network will be based on typical computers, we run our experiments on a typical workstation with a Pentium IV processor at 2.8GHz.

Then, we run a set of simulation scenarios for the threshold sets, $(t, n)$ of $(6, 13)$, $(12, 25)$, $(24, 49)$, $(48, 97)$ and $(96, 193)$. We evaluated the computation and communication costs for a Chord network consisting of $s = 256$ segments, while each segment contains up to 10,000 nodes. Since the network must be resilient to malicious nodes, we evaluated scenarios of networks where the probability of a node belonging to the untrusted node set ranges from $p = 0\%$ (a trusted nodes scenario) up to 30%. All the scenarios were tested both for the basic and for the verifiable threshold scheme.

The computational costs for the basic threshold signature scheme is presented in table 2. In particular, we measure the processing time required to generate the keys, the time required to sign a message (processing time of $t$ partial signatures) of variable length ($1K$, $2K$, $4K$ and $8K$) and the processing time required to verify the signature. The times presented are the mean times of the 10,000 operations that where executed. As illustrated in table 2 the processing times are independent from the message size. The results show that all the examined threshold sets are applicable for a real-time system. For example in the $(96, 193)$ threshold set, the processing time for a node certification is approximately one minute, while in the $(24, 49)$ threshold set, the processing time is approximately 15 seconds (without considering the communication delay). This order of cost can be considered as reasonable time, since the certification occurs only once per user.

The signing and verification costs for a verifiable threshold scheme are multiplied with a factor of 2. Although, in the verifiable threshold scheme the time to acquire a certificate is doubled in respect to the typical scheme, the costs for the partial signatures in the presence of malicious nodes (which are presented bellow), show that the verifiable scheme is the only viable solution.

In figures 7 (a) and (b), we show the number of partial signatures required to certify 10,000 nodes under the basic threshold scheme for various settings. It can be observed that even with a small percentage of malicious nodes the number of partial signatures is significantly increased for all the examined threshold sets. The exponential increase in the number of partial signatures can be justified by the fact that under the basic threshold scheme, the combiner cannot identify the malicious certification nodes that produce the invalid signatures, hence cannot exclude them from the next selection. Note that if the percentage of malicious nodes $p$ exceeds 10% of the population the basic threshold scheme is practically unusable.

In order to make the system usable under the presence of a large number of malicious nodes we use a verifiable threshold scheme, where the combiner can verify the validity of each partial signature and exclude the malicious node from future selections. As we can observe from the results in Figure 8, the number of partial signatures is substantially smaller from the basic threshold scheme.

#### 7.2. Communication costs

In order to measure the communication costs of the certification procedure we examine the number of mes-
sages exchanged between the certification nodes in order to certify 10,000 nodes within a segment.

As it is shown in Figures 9 and 10 the use of the basic threshold scheme is not suitable under the presence of malicious nodes. The results follow the same patterns as in the case of the partial signatures experiments. Thus, the use of the verifiable threshold is preferred.
7.3. Certificate lookup and verification costs

In the case of non-malicious faults, the cost for a certificate (or revocation) lookup is equal to the cost of an ordinary Chord lookup, which is logarithmical to the number of nodes. In case of malicious storage nodes, a lookup may take up to \( t \) times the cost of an ordinary lookup, since a certificate can be stored to the \( t \) successor nodes indicated by the hash function.

The verification cost of a node certificate is equal to the cost of an ordinary signature verification, in the case where the certificate has not been revoked. For the verification of the revocation, a second signature verification is required.

8. Conclusions

In this paper we propose Chord-PKI, a decentralized PKI that exploits the characteristics of the Chord protocol in order to meet the requirements of P2P networks, namely scalability, efficiency and resilience to compromised nodes. Our system provides certification to the Chord nodes through a distributed protocol that enables the collaboration of the nodes themselves, without the need for an external PKI. By relying on cryptographic techniques such as threshold cryptography our system can tolerate a significant fraction of malicious nodes within each segment. Moreover, by segmenting the network, the possible damage will be restrained within one segment. By exploiting the natural load balance property of the consistent hash function of Chord the certificates and CRL storage is evenly distributed among the system nodes. The lookup for a certificate (or CRL) is accomplished in \( \log_2 N \) steps, where \( N \) is the number of nodes in Chord-PKI.

An important issue for any trust infrastructure such as the Chord-PKI is fine-tuning the trust and revocation models. Several papers in the literature use reputation models for P2P systems \([32, 35, 44]\) to address these issues. Unfortunately, many of these have open scalability issues when the number of nodes is large. Thus, it is important to design a trust model which fine-tunes certification and revocation decisions and which exploits the scalability and functionality distribution provided by the Chord-PKI.


307–315.


