Laribus: Privacy-Preserving Detection of Fake SSL Certificates with a Social P2P Notary Network

Andrea Micheloni, Karl-Peter Fuchs, Dominik Herrmann, Hannes Federrath
Security in Distributed Systems Group, Department of Informatics
University of Hamburg, Germany
andrea.micheloni@m7i.org, {fuchs|herrmann|federrath}@informatik.uni-hamburg.de

***Abstract*—In this paper we present Laribus, a peer-to-peer network designed to detect local man-in-the-middle attacks against SSL/TLS. With Laribus clients can validate the authenticity of a certificate presented to them by retrieving it from different vantage points on the network. Unlike previous solutions, clients do not have to trust a central notary service, nor do they have to rely on the cooperation of website owners. The Laribus network is based on a Social Network graph, which allows users to form Notary Groups that improve both privacy and availability. It integrates several well-known techniques, such as secret sharing, ring signatures, layered encryption, range queries and a Distributed Hash Table (DHT), to achieve privacy-aware queries, scalability and decentralization. We present the design and core components of Laribus, discuss its security properties and also provide results from a simulation-based feasibility study.**

***Keywords*—privacy; anonymity; MITM attack; SSL; P2P**

I. Introduction

The SSL/TLS protocol suite provides basic security mechanisms such as confidentiality, data integrity and especially authentication [1], [2]. There have been various attempts to attack these protocols, i. e., to eavesdrop on a connection. Some attacks (e. g., the BEAST attack [3] published by Rizzo and Duong in 2011) exploit security flaws in the protocols, implementation errors or weaknesses in the construction of the cryptographic primitives used (for a comprehensive overview, cf. [4]). The second type, which we are interested in, attacks weaknesses in the *authentication infrastructure*.

SSL and TLS rely on a X.509 PKI [5] for the purpose of *authenticating* a remote entity's key. This protects clients from disclosing data to adversaries that impersonate a designated destination. Authentication is delegated to Certification Authorities (CAs) that issue certificates by cryptographically signing the public key of website owners, guaranteeing the authenticity of the association $\langle public\,key, website\rangle$ or, in some cases, $\langle public\,key, website, organization\rangle$.

In a 2010 study, the Electronic Frontier Foundation found that browsers from Microsoft and Mozilla are (in-)directly trusting more than 650 CAs [6]. Unfortunately, the X.509 model allows *any* CA to certify *any* domain [7]. It only takes a single CA to misbehave (by being hacked, tricked, bribed or legally forced) to generate a seemingly valid **fake certificate**. Having obtained a fake certificate an adversary can mount a man-in-the-middle (MitM) attack impersonating any website of his choice. The MitM attack goes undetected because the fake certificate contains a genuine signature and thus appears legitimate. MitM attacks typically affect only users in a confined part of the Internet (cf. Fig. 1). The severity of this risk is demonstrated by security incidents involving the two CAs Comodo and DigiNotar. The attackers created fake certificates for high-profile sites (among them Google and Paypal) to intercept the SSL traffic of Internet users in Iran [8]. The currently used approach of solely relying on CAs for authenticating remote servers is fragile and insecure.

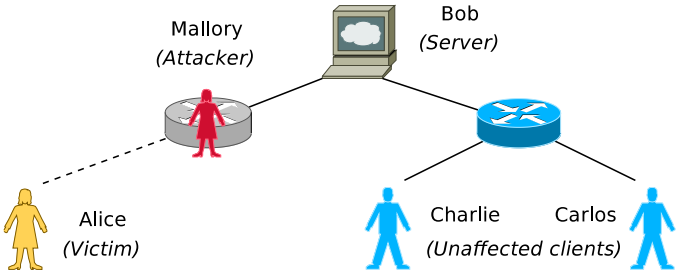

Fig. 1. An attacker presents a fake certificate during a MitM attack

Previous proposals aim to fix the CA-based authentication approach either by relying on other hierarchical structures (e. g., the DNS tree) or by introducing *notary servers* that validate certificates on the user's behalf. Both approaches are subject to considerable limitations (cf. Sect. II). In contrast, with **Laribus** (referring to guardian deities in ancient Roman religion), we propose that web clients should *collaborate* to validate certificates in a distributed manner. Our **contribution** consists of integrating well-known techniques to organize the clients in **a fully distributed peer-to-peer (P2P) network** that meets security, privacy and availability expectations of web clients. Laribus provides users with the ability to **model trust relationships that reflect their social relationships**, i. e., users can choose to trust their *friends*, and not unknown organizations, with certificate validation. The P2P architecture of Laribus improves **scalability** and provides **blocking-resistance**. **Privacy-preserving query techniques** protect the user's surfing behavior. A distributed storage mechanism offers **resilience to client churn** and increased performance via **caching**.

The rest of this paper is structured as follows. In Sect. II we review related work and motivate our design choices in Sect. III. We outline the architecture of Laribus in Sect. IV and focus on the details of the cryptographic mechanisms involved in Sect. V. We present results of an initial feasibility study in Sect. VI and discuss limitations in Sect. VII. We conclude in Sect. VIII.

II. Related Work

The problems with SSL and the CA trust model have been known for years, and various proposals have appeared in the literature to replace, amend or complement this model. Among those proposals, we have identified three common approaches: In Sect. II-A we describe those advocating to store certificate-related data in the Domain Name System. The proposals reviewed in Sect. II-B employ fixed *notaries* that certify and compare seen certificates. Finally, in Sect. II-C we describe methods that aim to validate certificates without having to trust third parties.

A. Utilizing the Domain Name System (DNS)

The approaches falling in this category aim to fix the broken CA architecture, but merely advocate another alternative hierarchy: **DANE** [9] and **CAA** [10] are IETF standards that propose to utilize the DNS infrastructure to validate certificates. Both designs standardize DNS resource records which can be used to either specify the SSL certificate of a website or to indicate the CAs that are supposed to issue its certificates.

The idea is to exploit an already existing decentralized hierarchy, as created by the DNS namespace, to authenticate website certificates. The security of these approaches depends on the integrity of the data supplied via DNS, i. e., DNSSEC [11] is a mandatory prerequisite in DANE. Indeed, as Verisign shows in [12], this solution would greatly reduce the attack surface for a MitM attack, since an adversary would have to compromise both a CA and the respective authoritative DNS server. However, leveraging DNS has some limitations in practice: Firstly, only few DNS servers make use of DNSSEC so far and its widespread deployment has proven to be challenging [13]. Moreover, the adoption of this approach can only be enforced by server administrators, not clients. Finally, parts of the DNS hierarchy are connected to the X.509 PKI infrastructure: A prominent example is Verisign itself, which maintains a Certificate Authority providing hundreds of thousands of SSL certificates [14]. At the same time Verisign acts as DNS registry for several top-level domains, among them `.com` and `.net`, which makes it a particularly weak spot.

B. Notary Approach

Perspectives [15] and **Convergence** [16] employ a set of network servers with the purpose of periodically fetching SSL/TLS certificates from Internet hosts. Users connect to one or more of the notary servers and retrieve from them the certificates they see at their vantage point. An attacker close to the client could then be easily detected by comparing the certificates presented to the notaries with the certificate the server presented to the client.

The notary model is based on the assumption that an attacker cannot interfere with both the connection of the users as well as the connection of (all the) queried network notaries. Unfortunately, notaries are full-blown dedicated servers which have to be maintained to be always up and running by their operators. As a result, there is only a limited number of them. However, a small fixed list of notaries for each user also means a greater chance for adversaries to attack users along with these notaries, effectively rendering their lists poisoned and ineffective. Moreover, users must put considerable trust into notaries: from a security perspective, users have to trust them not to lie (or collaborate with an adversary). From a privacy perspective, notaries have to be trusted to operate responsibly. Without any additional means, they learn all of a user's visited SSL domains. Privacy can be increased by forwarding queries to notaries via the Tor network [17], which requires additional software on clients, though.

Finally, the recently published **Sovereign Keys** [18] proposes to maintain a verifiable append-only data structure, which contains the history of all SSL-enabled domains. Server operators are supposed to push their authentication data (newly deployed certificates as well as blacklisted old certificates) to one of 20 redundant *Timeline Servers*. Clients interact with them to validate previously unseen certificates. Indeed, a read-only redundant and updated data structure could be the definitive answer to the trust problems in SSL, since attackers would have to publicly insert fake certificates into the data structure to be successful. However, this approach requires the cooperation of web server administrators.

C. End-to-End Schemes

Some approaches try to establish secure connections without a set of fixed, potentially untrusted third parties.

MonkeySphere [19] builds upon the PGP Web of Trust (WoT) concepts of a network of people who trust other people: Users who never met before can safely authenticate their identities due to the presence of a trust path in the network between them, established by friends who trust friends, etc. Whenever the MonkeySphere daemon encounters a self-signed or invalid certificate, it searches public key servers for a PGP key associated with that website's name. The certificate is trusted only if the daemon can construct a trust path from the user's key to the server's key. This approach could effectively abolish the need of external third parties and allow users to trust self-signed certificates, also providing a theoretically sound way of trusting remote certificates and detecting attackers. However, server administrators are required to sign certificates themselves and be connected into PGP's Web of Trust. Given the comparatively small user base of PGP/GPG keys, in many cases there will be no trust path between a user and a server.

Advocates of the **TOFU method**, on the other hand, simply reject the idea of having a third party issue certificates that have to be renewed periodically. Instead users are supposed to trust the certificate seen during the initial connection to a remote host (cf. the model used with SSH). Certificates should never change and never be renewed, unless exceptional circumstances occur. **DVCert** [20] expands this notion by allowing the renewal of a certificate whenever a user had already established a user name and password association with the end website.

Once users have connected to a website, indeed these proposals offer a great form of protection, which requires no extensive setup in comparison to other methods. On the other hand, the very first connection to a website cannot be protected with these models, and DVCert requires both existing users with website credentials and adoption by server administrators.

III. Building Laribus Upon Previous Work

Laribus is a combination and extension of techniques selected from the current state of the art (cf. Sect. II). It does not aim to replace the existing PKI infrastructure, but is supposed to provide an additional and independent layer that provides fail-safe functionality for the authentication function of SSL and TLS.

One of our central design goals is to achieve a **client-side solution that does not rely on the collaboration of website operators or other external organizations** for the purpose of detecting fake certificates. Therefore, Laribus is not based on DNS approaches that require additional resource records and whose security depends on the deployment of DNSSEC. Changes to DNS propagate very slowly and may never find widespread adoption.

The design of Laribus is inspired by the concept of notaries. However, another of our design goals is to provide a scalable **fully-distributed solution that does not require fixed entities**. Therefore, we do not want to be constrained to a set of dedicated notary servers. Running a dedicated notary server with adequate security and availability requires a high level of technical expertise and considerable resources – but it offers few benefits for its operator. In contrast, in Laribus each client acts as a low-availability component of a notary in Laribus.

Another design goal consists of providing users with **intuitive means to assess the trustworthiness** of the result of the certificate validation. MonkeySphere's approach, which relies on the Web of Trust infrastructure of PGP, seems promising, but relies on the cooperation of website operators, which conflicts with our client-side approach. Moreover, MonkeySphere requires its users to adopt PGP, which has not seen widespread adoption due to its intrinsic complexity and the resulting usability issues. Instead, in Laribus users can point out their friends based on real-world social relationships in order to define straightforward trust relations. These relations are used to find trustworthy clients for certificate validation.

IV. The Laribus Proposal

In this section we will present an overview of the architecture of Laribus and the interactions of the involved components. We will start out with a naïve approach for client-based certificate validation in order to explain the issues that must be overcome. After that we will explain how the design of Laribus addresses these challenges.

A. Shortcomings of a Naïve Approach

Given the initial scenario in Fig. 1, Alice could ask one of the unaffected users, e. g., Carlos, to retrieve the certificate for the server she wants to connect to (Bob). If Alice receives different certificates from Carlos and Bob, she can detect the MitM attack.

This naïve approach, however, has several shortcomings. First of all, it does not meet users' **security** expectations. Alice wants to be sure that Carlos is neither cooperating with Mallory and therefore not telling her the truth (Problem 1: intentionally false testimonies), nor that Carlos is unknowingly also affected by Mallory's attack (Problem 2: unintentionally false testimonies). Alice can solve the first problem by asking only friends she trusts to validate certificates. The second problem can be solved by asking users that are dispersed throughout the world, e. g., friends living in different countries.

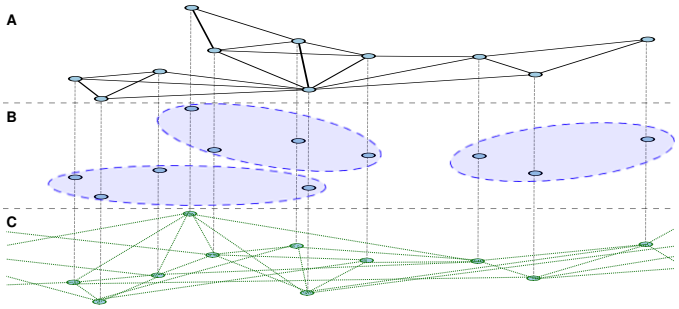

Fig. 2. Laribus consists of independent layers: a Social Network (A), Notary Groups (B) and a Distributed Hash Table for storage (C)

Secondly, the naïve approach does not meet users' **privacy** expectations. If Alice asks her friends to retrieve certificates on her behalf, her friends will know which servers she connects to. While it is a reasonable assumption that her friends will honestly answer requests for certificates, they must be considered to be *curious*; spying on an acquaintance may be more attractive than snooping on a stranger. Security and privacy seem to be conflicting goals in this respect. Solutions for this dilemma do however exist: Alice could ask a client she trusts, but she doesn't know personally, e. g., a client run by a university. Alternatively, she could use a suitable privacy-enhancing technique to hide her identity, e. g., Tor [17].

Thirdly, the naïve approach does not meet users' **availability** expectations. Alice cannot assume that a friend is online every time she wants to validate a certificate. Typically, Alice will also not be willing to wait until a friend comes online again. This availability issue can either be resolved by caching recently obtained validation results or by resorting to delegate certificate validation to less trustworthy clients.

Finally, certificate validation must be **scalable**. Asking one's friends to validate certificates scales well under three assumptions. Firstly, users that *consume* validation services, i. e., requesting other clients to perform certificate validation on their behalf, do also offer validation services to others, i. e., there is no *free-riding*. Secondly, friends trust each other *mutually*, i. e., they are willing to consume and offer validation services among themselves. Thirdly, the available resources offered by a group of friends are sufficient to handle the aggregate query volume issued by all of its members. However, in reality it is difficult to force users to honor these assumptions. Therefore, a practical system should be able to cope with unbalanced trust relationships as well as uneven resources and loads.

B. Layered Network Structure

Laribus is a decentralized, distributed peer-to-peer system comprised of clients that collaborate for certificate validation. Each client offers validation on behalf of others and each client can request validation by other clients.

The Laribus P2P network is structured into multiple layers of abstraction (cf. Fig. 2). Clients are requested to explicitly

point out their friends from the real world, to limit the influence of adversaries. These social relationships are captured in the **Social Network layer**, a directed graph that reflects the *friendship relations* expressed by the clients. The set of close friends of a user (also referred to as his *clique*), makes up his **Trusted Core** in Laribus. The resulting friendship graph is typically not fully-connected. In fact, some parts of the graph may be isolated from the rest, e. g., in case of cliques that do not point out any friends apart from members of their trusted core. We assume that adversaries will be unable to enter the trusted core, because this typically involves establishing a close social relationship with users in the *real world*.

Apart from pointing out their friends, clients organize themselves into *Notary Groups*. Group memberships are reflected in the **Notary Group layer**. Clients within a group collaboratively act as a notary. The members of a Notary Group are not supposed to be *fully congruent* with the set of clients within a Trusted Core.

Finally, clients are expected to contribute resources for distributed storage of validation data. The **storage layer** consists of a global Distributed Hash Table (DHT) based on Kademlia [21], in which Laribus stores all information regarding group memberships and trust relationships. Apart from providing decentralized storage this layer serves as a cache to provide cheap access to the results obtained from previous certificate validations. Data stored in the DHT is cryptographically signed (cf. Sect. V-C).

C. Interactions During Certificate Validation

Figure 3 extends the initial scenario by the components of Laribus. Alice connects to Bob's HTTPS website (Step A in Fig. 3). We assume that Alice is subject to a MitM attack perpetrated by Mallory. Other Laribus users, such as members of Notary Groups 1, 2, and 3 are supposed to be unaffected by the MitM attack. Formally, Alice tries to connect to a server $\mathcal{SRV}$ and is presented with a fake certificate $cert_M$. Some unaffected clients are able to fetch the authentic certificate $cert_B$. In the following we will only briefly sketch the relevant interactions. For conciseness we will defer the treatment of the involved security and privacy techniques to Sects. IV-E and IV-G.

The straightforward way to validate certificates in Laribus is via the **Direct Queries** technique. A requestor, Alice, sends a Direct Query to a Notary Group of her choice (Step B in Fig. 3). She may choose a random group for this purpose, or a group containing one of her friends, asking members of the selected Notary Group to retrieve the certificate of $\mathcal{SRV}$ from their respective vantage points. The Notary Group exchanges the obtained certificates in order to reach a consensus about the certificate of $\mathcal{SRV}$ by means of a majority vote: The winning certificate is signed by the group's *Notary Signature* and returned to Alice (not shown). Alice validates the certificate presented to her by Bob by comparing it with the result obtained from the Notary Group.

Issuing Direct Queries for every certificate validation is inefficient. An alternative way for a client to validate certificates consists of retrieving certificates that have been obtained by other clients via Direct Queries with the **DHT Lookup** technique. Every time a Notary Group obtains a consensus

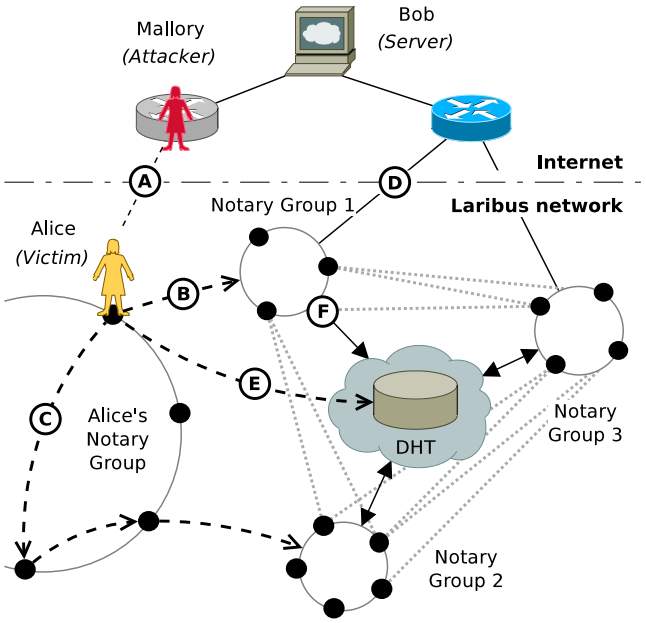

Fig. 3. Components of Laribus and their interactions

about a server's certificate, it signs and stores it in the DHT (Step F in Fig. 3, further discussed in Sect. V-C). Instead of issuing a Direct Query, clients can look up both the unknown certificate's hash or hostname of $\mathcal{SRV}$ in the DHT (Step E). Summing up, in Laribus a client performs the following steps to validate a certificate:

1) Alice performs a DHT Lookup to determine whether $cert_M$ has been seen previously. If there are entries for $cert_M$, she retrieves them and (depending on how much she trusts the Notary Groups that stored the entries into the DHT) validates the certificate presented to her. If no trusted Notary Group has seen that particular certificate before, she proceeds with a Direct Query.
2) Alice contacts a number of Notary Groups with a Direct Query asking them to report what certificate server $\mathcal{SRV}$ is offering. The queried group's members connect to $\mathcal{SRV}$ and get $cert_B$, and then sign with a Notary Signature the fact of having seen $cert_B$ at the moment of fetching.
3) The queried Notary Groups store the obtained certificates in the DHT.

D. User-defined Trust

Laribus bases its security upon the Social Network of its users. Like in the PGP Web of Trust, users are required to define social relationships, i. e., determine to which extent they trust particular users (*user trust level*).

Alice can assign the following *user trust levels* to a friend (Charlie):

Connection: Alice receives and processes Charlie's messages. The consequence is that Charlie gets to know Alice's IP address and connection times.

Direct query: Alice includes Charlie in the list of users

she will (try to) send Direct Queries to. As a result, Charlie may learn about Alice's (group's) surfing habits.

Transitive trust: Alice trusts Charlie's friends to some extent and would receive and forward their messages. In consequence, if Charlie authenticates untrustworthy users, both Alice and Charlie are affected.

Group trust: Alice vouches for Charlie's inclusion into her group (however, Charlie cannot join the group until all members vote for his inclusion). If Charlie is an attacker, he can perform denial-of-service attacks against Alice's group, i. e., by not forwarding messages or not participating in creating Notary Signatures.

E. Security Measures

In order to prevent outsiders from forging messages or launching impersonation attacks, Laribus clients make use of public-key cryptography for **message authentication** and identification purposes. Before connecting to the Laribus network for the first time each client n_i creates a key pair (S_i, P_i, ID_i), where ID_i is a self-signed pseudonym, (ID_i, P_i) is the public key and S_i is the secret key. ID_i does not necessarily have to reveal the actual identity of a user, but friends should be able to discover each other based on these pseudonyms.

As Direct Queries are quite expensive, they could be used to mount denial of service attacks. Therefore, they cannot be issued anonymously, but have to be authenticated by the requestor by a digital signature. This allows Notary Groups to reject queries received from misbehaving clients or to only accept queries from trusted groups. However, if clients sign Direct Queries themselves, validation requests can be linked and tracked back to their pseudonym. We will discuss ways to overcome this privacy issue in Sect. IV-G1. Moreover, the certificate records stored within the DHT are also signed by the Notary Groups in order to guarantee their authenticity.

F. Availability Measures

Given the P2P approach of Laribus, clients cannot be expected to be online at all times. We address this problem with three mechanisms: a ring signature scheme, a threshold signature scheme and a DHT (i. e., the storage layer, cf. Sect. IV-B).

The **ring signature scheme** allows Alice to sign a Direct Query, i. e., to ask another Notary Group to retrieve a certificate, even if Alice is the only (online) member of her group (cf. Sect. V-A). With this scheme Alice can prove that she is a member of her group, without disclosing her identity (cf. Sect. IV-G).

The purpose of the **threshold signature scheme**, called *Notary Signatures* in this paper (cf. Sect. V-B3), is to allow groups to sign replies to Direct Queries when only a subset of the group members is online. We propose to employ an efficient threshold scheme by Lesueur et al. [22].

The most important availability feature of Laribus is **the DHT**. It serves as a global public *timeline* of the certificates seen by different groups at different locations. Since the DHT is distributed among *all* clients (i. e., independent from the Social Network and Notary Group structures), it assures availability of the signed records (i. e., the past validations) of a Notary Group, even if not a single member of that group is online (cf. Sect. V-C).

G. Privacy-Preserving Certificate Validation

To meet the users' privacy expectations, i. e., to prevent that network nodes will get to know who wants to validate which certificates, we use well-known and approved techniques from the privacy-enhancing technologies research community. Laribus contains privacy-preserving mechanisms for both *Direct Queries* and *DHT Lookups*.

1) Preserving Privacy for Direct Queries: If Alice asked a Notary Group to validate a certificate directly (Step B in Fig. 3), she could be easily identified by her IP address or by her signature of the Direct Query. To prevent identification via signatures, we use **ring signatures**, i. e., other Notary Groups can validate that *some member* of a certain group has signed a request, but not which member exactly (details follow in Sect. V-A).

To obfuscate IP addresses, simple solutions like Convergence's approach to route requests via a low-latency anonymity system (Tor) could be employed. This approach however conflicts with our design goal of a decentralized solution and would be subject to performance problems of such anonymity systems [23]. To this end, we have designed a scheme that utilizes the group structure of our network to build anonymity sets. Before a request is sent to another Notary Group, it is passed around between the members of Alice's group (cf. Step *C* in Fig. 3). As a result, members of other groups will not be able to get to know whether a request was initiated by the requesting host itself, or whether the request was sent on behalf of another member of the group.

This mechanism offers privacy towards other groups (e. g., *Group 2* in Fig. 3), but it cannot protect Alice from the members of her own group. To this end, we use a layered encryption scheme (*LES*, cf. [24]). With a *LES*, Alice chooses N *hops* (i. e., friends) to forward her requests (source routing). *LESs* hide the routing information required by her friends to relay a request. Each hop removes one layer of encryption and, if it is an intermediate node, receives the address of the *next hop* (i. e., the next group member), or the destination address (i. e., the address of the *destination group*) if it is the final hop. We propose to use the *Sphinx* scheme [25] that hides the actual path length (messages have the same length at each hop) and is very compact.

2) Preserving Privacy of DHT Lookups: To meet the users' privacy requirements for DHT Lookups, we use a *Range Query approach* [26]: Alice will not request a specific entry using its key, but instead a set of results by querying for a prefix that refers to a certain subtree within the DHT. As a result, an attacker that is able to observe this process will only get to know that Alice is interested in one of the records of the subtree, but not in which exactly. The security parameter k determines the size of the result set, which allows to balance performance and privacy (cf. [21] and Sect. V-C).

V. Laribus Cryptography Mechanisms

In this section we will present details about the cryptographic mechanisms of Laribus, especially ring signatures and

Notary Signatures. We describe how Notary Group key pairs are generated and (dynamically) shared as well as the structure and data format of the DHT.

A. Ring Signatures

In Laribus, direct queries have to be signed as Notary Groups are not required to answer direct queries from any other group (cf. Sect. IV-E). However, we also want to provide privacy for Direct Queries (cf. Sect. IV-G1). We use ring signatures to solve this problem [27], [28]. Ring signatures allow a single member of a group to sign data (m) on behalf of all group members. The signer requires only his own private key (S_s) and the public keys of the other ring members (P_r) as input to create a signature.

To achieve this given some signing scheme based on trapdoor one-way permutations, such as RSA, the signer constructs a *verification equation* $C_{k,v}(g_1(x_1), g_2(x_2), \ldots, g_r(x_r)) = z$ combining a hash of the data to be signed and the ring's public keys functions with trapdoor $g_1, g_2, \ldots, g_r$. The equation is infeasible to solve for all inputs without inverting any trapdoor function g_i (i. e., not possessing the relative secret key to a key in the ring), therefore, the signature proves the signer's membership in the ring. Only the signer can provide the solution to the equation as the trapdoor is available only to him [27].

B. Notary Signatures

When Alice asks a Notary Group to retrieve a certificate on her behalf (*Direct Query*), the Notary Group has to sign its answer (*Notary Signature*) to guarantee authenticity (cf. Sect. IV-C). In contrast to the case of *ring signatures*, a majority of the Notary Group must participate to perform the signature. On the other hand, to meet the availability requirements of Laribus, we must ensure that Notary Signatures can be obtained even if some group members are offline. A suitable solution for this problem is *threshold cryptography* [29].

With *threshold cryptography*, a public/private key pair (P, S) can be jointly generated by n parties. After key generation, each party knows P and holds a share s_i of S, but no party knows S entirely. Signatures can be obtained as long as a *threshold* t of shares is available, i. e., t parties participate. However, t and n must be chosen before the distributed key generation is initialized, and cannot be changed afterwards. As a result, using a *standard* $(t-n)$-threshold scheme directly would require to generate a new key pair every time n changes, i. e., whenever a Notary Group is resized. Furthermore, the overhead for generating keys increases drastically with the number of nodes [30] and would thus be impractical for our case as we assume that groups may consist of up to about 15 (and at least 3) members.

To overcome this problem, we propose to use a dynamic scheme by Lesueur et al. [22] that allows share merging and splitting. After an initial key generation process (e. g., to generate 3 shares) the number of shares can be adjusted to match the actual number of nodes. Furthermore, its performance is sufficient for larger group sizes, in fact even much larger than required for Laribus [22]. In the following, we will explain details about key generation, dynamic share handling, signing and Notary Group protection measures against Sybil attackers.

1) Key Generation: To create a key pair for a Notary Group, we employ Boneh et al.'s efficient method of distributed RSA key pair generation [30]. The protocol allows a set of parties to construct an RSA modulus $N = pq = \sum p_i \sum q_i$ where N is publicly known, and each member N_i only knows about p_i and q_i, not about the factorization of N. To enable sharing of the secret key, they then calculate shares of $d = e^{-1} mod \varphi(N)$ for any given RSA encryption exponent e. Once each of the n group founders has obtained an initial share e_i of the secret d, to which we'll refer as S_G, the following holds: $\sum_{i=1}^{n} e_i = S_G$ [30]. In Laribus, we parameterize Boneh et al.'s method as a $(3-3)$-threshold scheme, i. e., a group will create three shares and all three shares are required to obtain a signature. To meet the security and availability requirements, the number of shares is adjusted with the protocol described in the next section.

2) Dynamic Share Handling: To dynamically assign shares to group members, we use the scheme of Lesueur et al. [22]. With this scheme, even though a group is composed of n members, the number E of shares can vary in respect to the number $t' \leq n$ of online members. The scheme allows to specify a fixed *ratio* r of nodes that are required to recover S_G (not a fixed *number* of nodes as in *classical* threshold schemes [22]). To achieve this, shares (e_i) are split and merged and may be replicated on different nodes. Nodes that are assigned the same share e_i compose a *sharing group*.

The share and merge operations require distributed sums and subtractions, since $split(e_i) = (e_{i_1}, e_{i_2})$ and $merge((e_j, e_k)) = e_{jk}$. To prevent an attacker from reconstructing S_G from *old* shares, *old* shares must be rendered useless after each split and merge operation. To this end, if a newly created e_i is mixed with another share e_j, their owners collaboratively calculate $e_i := e_i - \Delta$ and $e_j := e_j + \Delta$, maintaining $\sum_{i=1}^{E} e_i = S_G$. Δ is a chosen random value that hides the original share (cf. [22]).

The ideal number of shares is $E = r \times t'$, with each sharing group composed of $g = \frac{1}{r}$ nodes knowing the same share [22]. We require clients to participate in merge operation only if the number of shares E would not fall under a minimum of $E_{min} = 3$. This way, at least three colluding attackers must be present in a group to recover S_G and forge Notary Signatures. In practice, when only three group members are online and one member wants to disconnect, the whole group is shut down. The last three shares are assigned to a third of the group members and stored in the DHT encryptedly. Nodes can recover the shares once they re-connect. The Notary Group ratio is set to $r = \frac{1}{2}$. We will motivate this choice in Sect. VI.

3) Obtaining a Notary Signature: In order to obtain a Notary Signature for a message D, there must be at least one member per *Sharing Group* available and willing to sign D, i. e., each share e_i is required for the signature computation. Following the signature protocol (cf. Fig. 4), the group signature of D is $D^{S_G}[m]$. With the equality

$$D^{S_G}[m] = D^{(\sum_{i=1}^{E} e_i)}[m] = \left(\prod_{i=1}^{E} D^{e_i}[m] \right)[m]$$

the signature can be calculated collaboratively. One node per *Sharing Group* signs D with its share, i. e., calculates $s_i =$

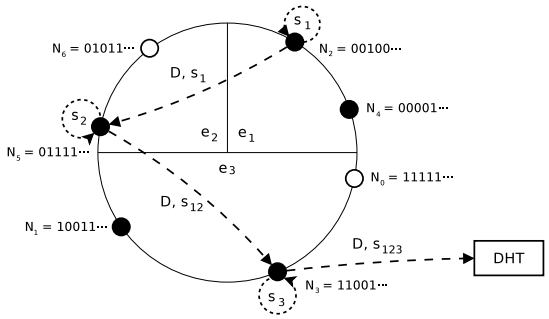

Fig. 4. A Notary Group collaborating for the signature of a message D: one node per *Sharing Group* signs D and forwards it along with the calculated incremental product to the next *Sharing Group*. Shares are stored encryptedly in the DHT for offline nodes (empty circles).

TABLE I. THE INCLUSION OF A NODE IN A NOTARY GROUP

row type	node	timestamp
Inclusion	(P_i, ID_i)	t_x

signature
$sig_{grp,inc}$

$D^{e_i}[m]$ and multiplies the result (mod m) with the result from the previous group (cf. Fig. 4 and [22]).

4) Notary Group Protection Measures: Like any other P2P system Laribus has to address Sybil Attacks that consist of a malicious user joining the network with multiple nodes and *fake* identities in order to control (large) parts of the network. We plan to employ Gatekeeper [31] as an admission control system to counter Sybil Attacks. Gatekeeper uses a *ticket distribution algorithm* to detect attackers and is a suitable choice since, like Laribus, it is based on a social network and does not require users to have a global view of the network.

When a Notary Group decides about the inclusion of a new candidate (P_i, ID_i), already present members will take into account their own friendship relations (cf. Sect. IV-D) as well as Gatekeeper. If the new candidate passes both tests, his public key is collaboratively signed by the Notary Group and stored in the DHT to prove its membership. From that moment on the candidate's NodeID in the group and DHT is fixed as $sig_{grp,inc}$. Table I shows the resulting data structure.

C. Certificate Timeline and Storage Data Format

To improve performance and availability of Laribus, clients can retrieve certificates signed by other groups, *Certificate Validation Records* (CVR), from the DHT instead of issuing a Direct Query (cf. Sect. IV-C). The DHT stores $\langle key \rightarrow value \rangle$ pairs (cf. Table II), where multiple values (i. e., CVRs) per key can be returned, e. g., records from different groups or from different times (*certificate timeline*). Laribus employs the Kademlia DHT [21], i. e., keys are truncated to 160 bits (cf. Table II).

DHT lookups require either a hash of the certificate in question ($cert_X$) or a hash of the domain name ($\mathcal{SRV}_X$, cf. Sect. IV-C), i. e., the DHT stores two different keys for CVR lookups (cf. Table II). As mentioned in Sect. IV-G2, to meet the users' privacy requirements, clients may request a subtree of the DHT by submitting only a prefix with $160 - k$ bits

TABLE II. THE STORED $cert_X$ VALIDATION RECORDS IN THE DHT

DHT keys		
$SHA_{256}(cert_X)[0 \cdots 159]$	$\Longrightarrow$	CVR
$SHA_{256}(\mathcal{SRV}_X)[0 \cdots 159]$	$\Longrightarrow$	

TABLE III. CERTIFICATE VALIDATION RECORD STORAGE FORMAT

row type	certificate	server	timestamp
Validation	$SHA_{256}(cert_X)$	$\mathcal{SRV}_X$	t_x

signature
sig_{grp}

(*Range Query*), i. e., they will receive all CVRs with a key that starts with the prefix.

The data format of a CVR is shown in Table III. The row type *Validation* separates *blacklisted* from *whitelisted* CVRs. The *certificate* field identifies the certificates by their SHA_{256} hash. The row type *Server* consists of $\mathcal{SRV} = \langle hostname, port \rangle$ values that identify the server which offered $cert_X$. The *timestamp* field states when $cert_X$ was validated. All CVR fields are signed with the respective Notary Group's signature (sig_{grp}).

VI. EVALUATION

The goal of this section is to provide an initial feasibility study of Laribus. We focus on the computational cost of cryptographic processes as well as availability aspects.

A. Cryptographic Processes

Laribus makes use of four cryptographic schemes: Distributed key generation (cf. Sect. V-B1), Notary Signatures (cf. Sect. V-B3), ring signatures (cf. Sect. V-A) and a LES (cf. Sect. IV-G1).

The **distributed key generation** scheme is by far the most expensive sub-protocol of Laribus. It requires several rounds of private distributed computation, e. g., to generate an RSA modulus and for biprimality testing. However, key generation is performed only once per group, when a new group is initialized. Given the results of Congos et al. [32], generating a 2048 bit key that consists of three shares can be expected to be finished in less than 4 minutes on commodity hardware and with a total network traffic of less than 25 MByte. As this process is required only once per group and has no strong real-time requirements, we expect no practical limitations no matter if keys are generated in a LAN or via Internet. The **Notary Signature** scheme consists of share assignment and distribution and the signing process itself. The distribution of shares is not expensive, as it only consists of distributed sums and subtractions (cf. Sect. V-B2 and [22]). The signing process requires a single *normal* signature per sharing group, as well as a simple multiplication of the resulting signatures (cf. Sect. V-B3 and [22]). The overhead of both processes seems negligible. The **ring signature scheme** [28] is practical for our scenario as well: It requires only between n and $2n$ modular multiplications, where n is the group size and can be performed locally by a single node (cf. Sect. V-A). The **layered encryption scheme** of Laribus, *Sphinx* (cf. Sect. IV-G1), requires essentially $2r$ public key operations to create a message (r is the number of hops) and again 2 public key operations on each of the r hops (i. e., group members) that forward the

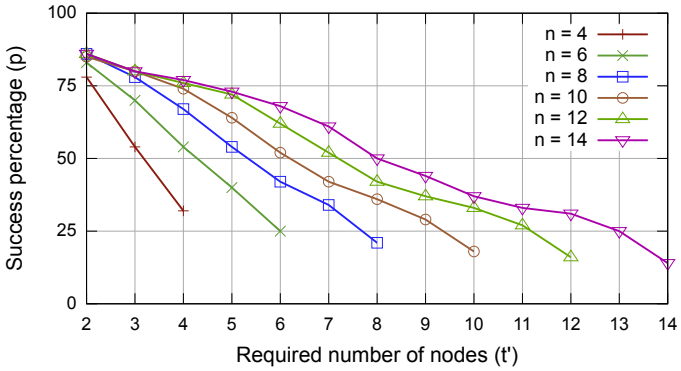

Fig. 5. Simulations of online availability for *normal* users

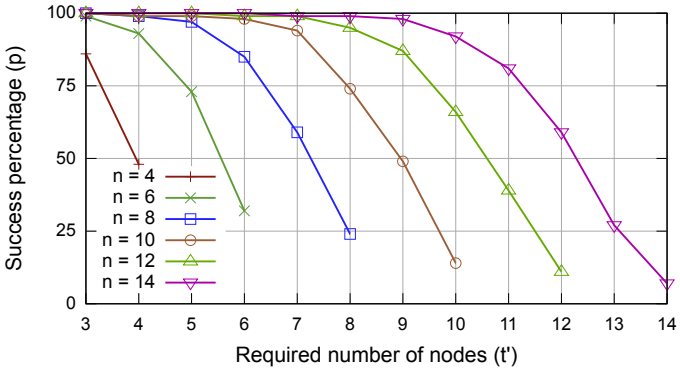

Fig. 6. Simulations of online availability for *power* users

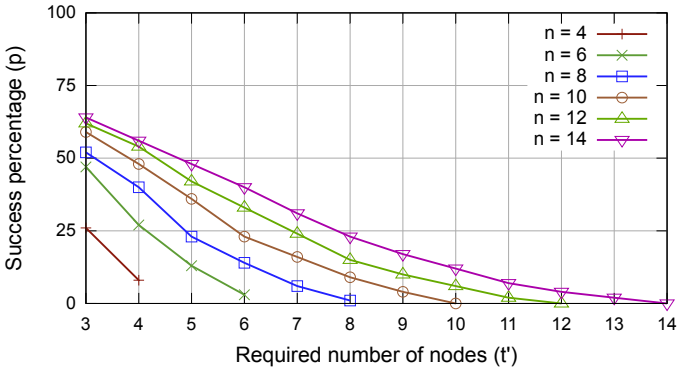

Fig. 7. The simulation of online availability for mixed users

message [25]. It can be used in conjunction with the very efficient Curve25519 elliptic curve library [33]. Results of a recent study [34] indicate that the delay introduced by Sphinx and the relaying of messages should be well below one second.

In conclusion, we do not expect performance problems due to cryptographic overhead. The only sub-protocol of Laribus that introduces considerable delay is the distributed key generation, which takes place only once a group is initialized.

B. Availability Aspects

Whether a group is operational (i. e., can for example answer direct queries) depends on three factors: the group size n, the number of nodes necessary to recover the group secret (*threshold* t') and the user behavior (i. e., online and offline times). We have performed a simulation-based study to determine adequate values for n and t' and to assess whether these values are indeed practical or not. Source code and configuration files will be made available by the authors on request.

Since Laribus is not deployed yet, we have to estimate the **user behavior** of clients. We model four different types of users: *casual*, *normal*, *office* and *power users*. *Casual users* connect to the Internet with short session times (on average, ten minutes per session) during daytime (between 8:00 and 21:00) and spend 60 minutes on the Internet per day on average. *Normal users* are connected 5 hours per day on average, split into two sessions, one in the morning (around 10:00) and evening (around 19:00). *Office users* connect between 9:00 and 10:00 and end their session between 17:00 and 18:00. *Power users* are either running a server or leaving their computer online most of the day, with 4 hours of *downtime* per day on average. User connection times and session durations are sampled from a Gaussian distribution (standard deviation 0.5), except for casual and power users, whose connection times are drawn uniformly from the $[8, 21]$ and $[0, 24]$ hour ranges, respectively. We assume that Laribus is run as a system service, i. e., that clients are available whenever a user is connected to the Internet.

Figures 5, 6 and 7 show the *success percentage* p for different group sizes n, thresholds t' and (mixes of) user types. The *success percentage* is defined as the probability that, whenever at least one node of a Notary Group is online, the group is operational (i. e., it can perform a Notary Signature), i. e., at least t' of its nodes are online.

Figures 5 and 6 present the success ratios of groups with uniform type of users, respectively *normal* and *power* users. Figure 7 shows results for a more realistic case with mixed user types, i. e., 80% normal and office users and 20% casual and power users. As we can see from Fig. 5 and Fig. 6, user behavior has a strong influence on the *success percentage*. The results for *normal users* (Fig. 5) suggest that for $p = 50\%$ and $n = 6$, $t' \approx 0.7n$ would be a reasonable choice ($\frac{t'}{n} = \frac{4}{6} \approx 0.7$), while the results for *power users* (Fig. 6) suggest $t' \approx 0.9n$. The simulation of the most realistic case, the mixed case (Fig. 7), suggests $t' \approx 0.5n$. As expected, larger groups can tolerate a bigger fraction of offline nodes for the desired *success percentage* of 50%. For instance, a group with $n = 14$ nodes can tolerate 9 offline nodes in the mixed case, which is about 64%, while a group with $n = 6$ nodes can tolerate only about 3 offline nodes, i. e., about 50% (cf. Fig. 7).

In conclusion, the results suggest that $t' \approx \frac{n}{2}$ is a reasonable choice, if $n \geq 6$, i. e., groups should consist of at least 6 members to be operational in 50% of cases.

VII. Limitations and Discussion

In this section we will discuss limitations and challenges of Laribus. We focus on the attacker model as well as practical problems, performance tradeoffs and bootstrapping.

Laribus cannot protect against attackers that control large parts of the Internet. By design, and like any other notary-based MitM detection approach, *Laribus can solely detect local attacks*, i. e., the majority of notaries must not be affected by an attack and must be able to retrieve valid certificates.

Attacks that affect a single country (e. g., a repressive regime) or continent can be detected as long as notaries from different parts of the world are available. Moreover, strong attackers may try to block all traffic pertaining to certificate validation. While this is straightforward given static, centralized notaries, Laribus is less vulnerable to blocking due to its decentralized architecture.

A general problem of collaborative distributed certificate validation is that some hosts and especially Content Distribution Networks (CDNs) *use different valid certificates for the same host*. As a result, a 1:1-comparison of certificates is not appropriate. However, since the *timeline* of seen certificates of Laribus allows to store several certificates for a host (cf. Sect. V-C), clients can still judge the validity of a certificate by the number of notaries that have seen the same certificate.

Newly issued or renewed certificates cannot be answered with DHT Lookups as the *timeline* will not contain any records for them. In this case Direct Queries (cf. Sect. IV-C) have to be issued, i. e., enough notaries that the client trusts have to be available.

The results of our initial feasibility study suggest that groups should consist of at least 6 members to be operational in 50% of cases (cf. Sect. VI-B). Composing a group of at least 6 friends should not be a problem in practice. An average 50% availability of a Notary Group should be practical as well, especially since the storage and caching mechanisms of the DHT do allow to access the *timeline* of the certificates seen by different groups even if all members of those groups are offline. Further, previous studies indicate that the popularity of web hosts follows a power-law distribution [35], i. e., a small set of popular hosts is responsible for the vast majority of all requests, while the *long tail* of remaining hosts is requested rarely. Thus, we expect that the DHT's cache hit ratio will be high; Direct Queries will not be needed in most cases. Our analysis of the cryptographic processes of Laribus showed that *cryptographic overhead is moderate*, except for distributed key generation which is required only during initialization (cf. Sect. VI-A).

In conclusion, results for both availability and performance are quite promising. They indicate that a Laribus deployment would indeed be practical. However, future work should focus on an analysis of the cache hit ratio and the resulting user-perceived latencies. The results would be of particular interest for the parameterization of Laribus, especially the security parameter k of the Range Query protection mechanism (cf. Sect. IV-G2).

To offer adequate performance and availability during the initial deployment of Laribus, we suggest to *bootstrap the network* with a set of dedicated servers operated by different universities around the world until enough users participate. In fact, a permanent operation of these servers could be an option as well as it would allow to combine the benefits of P2P (especially blocking resistance) and client-server solutions and thus further increase the attractiveness of Laribus. Furthermore, snapshots of the *timeline* (including group signatures) could be mirrored and made publically available for all users on the Internet. This resembles the approach of *Sovereign Keys* (cf. Sect. II and [18]) and would also help to reduce lookup latencies as well as the size of the DHT.

VIII. Conclusion

In this paper we presented Laribus, a social peer-to-peer network that addresses the problem of man-in-the-middle attacks on SSL. Laribus integrates approved techniques from current state-of-the-art solutions and combines them with well-known proposals from the privacy enhancing technologies research community to improve both privacy and availability.

To the best of our knowledge, Laribus is the first fully distributed solution for the detection of fake SSL certificates. It does not require cooperation of website owners and is blocking-resistant due to its decentralized architecture. Further, clients do not have to trust a central notary service. The Laribus architecture is fundamentally different from previous solutions since users can model trust relationships that reflect their social relationships and may create virtual notaries that consist of their friends. The resulting groups of friends are utilized both as anonymity sets and to increase availability.

Our initial evaluation results are promising and suggest that Laribus is feasible. Future work will focus on an implementation of its components as well as an in-depth analysis of the effectiveness of caching and user-perceived latencies in order to prepare the practical deployment of Laribus.

References

[1] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, 2008.

[2] A. F. ad P. Karlton and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," RFC 6101, 2011.

[3] J. Rizzo and T. Duong, "Here Come The XOR Ninjas," Unpublished manuscript, May 2011.

[4] C. Meyer and J. Schwenk, "Lessons Learned From Previous SSL/TLS Attacks – A Brief Chronology Of Attacks And Weaknesses," Cryptology ePrint Archive, Report 2013/049, 2013, http://eprint.iacr.org/.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008.

[6] Electronic Frontier Foundation, "EFF SSL Observatory," https://www.eff.org/observatory, last fetched 2013-02-28.

[7] C. Soghoian and S. Stamm, "Certified lies: detecting and defeating government interception attacks against SSL," in *FC 2011*, vol. 7035. Springer, 2012, pp. 250–259.

[8] N. Leavitt, "Internet Security under Attack: The Undermining of Digital Certificates," *IEEE Computer*, vol. 44, no. 12, pp. 17–20, 2011.

[9] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698, 2012.

[10] P. Hallam-Baker and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record," RFC 6844, 2013.

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033, Mar. 2005.

[12] E. Osterweil, B. Kaliski, M. Larson, and D. McPherson, "Reducing the X.509 Attack Surface with DNSSEC's DANE," SATIN, 2012. [Online]. Available: http://conferences.npl.co.uk/satin/papers/satin2012-Osterweil.pdf

[13] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, "Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 5, pp. 656–669, 2011.

[14] "Licensing VeriSign Certificates," Verisign, Tech. Rep., 2005. [Online]. Available: http://www.verisign.com/static/001496.pdf

[15] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing," in *USENIX Annual Technical Conference*. USENIX, 2008, pp. 321–334.

[16] "Convergence," http://convergence.io/, last fetched 2013-02-28.

[17] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*. USENIX, 2004, pp. 303–320.

[18] "The Sovereign Keys Project," https://www.eff.org/sovereign-keys, last fetched 2013-02-28.

[19] "The Monkeysphere Project," http://web.monkeysphere.info/, last fetched 2013-02-28.

[20] I. Dacosta, M. Ahamad, and P. Traynor, "Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Parties," in *ESORICS*, ser. LNCS, vol. 7459. Springer, 2012, pp. 199–216.

[21] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *IPTPS*, ser. LNCS, vol. 2429. Springer, 2002, pp. 53–65.

[22] F. Lesueur, L. Mé, and V. V. T. Tong, "A Distributed Certification System for Structured P2P Networks," in *AIMS*, ser. LNCS, vol. 5127. Springer, 2008, pp. 40–52.

[23] R. Dingledine and S. J. Murdoch, "Performance Improvements on Tor, or, Why Tor is Slow and What We're Going to Do About It." 2009. [Online]. Available: https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf

[24] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, 1981.

[25] G. Danezis and I. Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," in *S&P*. IEEE, 2009, pp. 269–282.

[26] Y. Lu and G. Tsudik, "Towards Plugging Privacy Leaks in the Domain Name System," in *Peer-to-Peer Computing*. IEEE, 2010, pp. 1–10.

[27] R. L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," in *ASIACRYPT*, ser. LNCS, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 552–565.

[28] R. L. Rivest, Y. Tauman, and A. Shamir, "How to Leak a Secret: Theory and Applications of Ring Signatures," in *Essays in Memory of Shimon Even*, ser. LNCS, vol. 3895. Springer, 2006, pp. 164–186.

[29] A. Shamir, "How to Share a Secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[30] D. Boneh and M. K. Franklin, "Efficient Generation of Shared RSA Keys (Extended Abstract)," in *CRYPTO*, ser. LNCS, vol. 1294. Springer, 1997, pp. 425–439.

[31] D. N. Tran, J. Li, L. Subramanian, and S. S. M. Chow, "Optimal Sybil-resilient node admission control," in *INFOCOM*. IEEE, 2011, pp. 3218–3226.

[32] T. Congos and F. Lesueur, "Experimenting with distributed generation of RSA keys," in *Proceedings of the 5th International Workshop on Security and Trust Management (STM)*. Elsevier, 2009.

[33] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography*, ser. LNCS, vol. 3958. Springer, 2006, pp. 207–228.

[34] K.-P. Fuchs, D. Herrmann, and H. Federrath, "Introducing the gMix Open Source Framework for Mix Implementations," in *ESORICS*, ser. LNCS, vol. 7459. Springer, 2012, pp. 487–504.

[35] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *Networking, IEEE/ACM Transactions on*, vol. 10, no. 5, pp. 589–603, 2002.