sayI: Trusted User Authentication at Internet Scale

Jon A. Solworth
University of Illinois at Chicago
solworth@ethos-os.org

Wenyuan Fei
University of Illinois at Chicago
feiwenyuan@yahoo.com

ABSTRACT

With Internet-scale user authentication, an organization authenticates users with which it has no prior association. Of necessity, the organization must rely on third parties, which make up the authentication infrastructure and can vouch for these users.

These third parties are trusted. And since different organizations have different adversaries and different security needs, it is up to the organization to determine which third parties to trust.

Unfortunately authentication infrastructures which meet the above trust requirements have been inefficient, suffering from high latency, excessive bandwidth, and high CPU load. These inefficiencies significantly impede wide-scale deployment.

We introduce sayI, a Public-Key based authentication Infrastructure (PKI). It is the first PKI which is efficient at Internet scale *and* enables organizations to determine their risk from third parties. It protects privacy and provides security. It is designed to minimize bandwidth and latency through a careful and novel integration of authorization and authentication. In sayI, irrelevant certificates do not negatively impact performance. An Internet user authentication is *guaranteed* to complete in a single Internet round trip, significantly faster than alternative authentication infrastructures.

1 Introduction

The Internet is so large and so geographically dispersed that its use depends on third parties. Since the Internet contains cyber criminals and other adversaries, the parties on which to depend must be selected carefully. Nowhere is this choice more important than in authentication, which binds (remote) users to their identity.

Authentication of a **user** is performed by an organization known as the **relying party**. The user and the relying party are called the principals of the authentication. A Trusted Third Party (TTP) is a non-principal which provides information used in an authentication. For simplicity we focus here on users, but the techniques described extend to hosts (see Appendix §B).

Figure 1 shows the structure of a user authentication (arrows show communication during an authentication). The user provides to the relying party an authenticator which uses her authentication credentials (e.g., public key). The relying party authenticates the user using the authenticator and information from TTPs. To authenticate a user, she must be known to one or more TTPs.

Historically, TTPs have included banks (for letters-of-credit, checks, and credit cards) and governments (for passports and

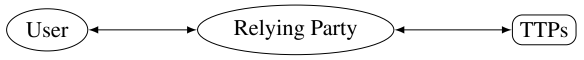

Figure 1: Authentication Architecture

driver's licenses) and are essential when the relying party does not know the user. In general, the relying party bears the risk of a mis-authentication [60][1]. Even when a TTP causes an authentication failure, the TTP may not suffer the consequences. When a TTP is an adversary, failures *benefit* the TTP.

One relying party's adversary is not necessarily another's. For example, a Chinese government relying party would not want to rely on a US TTP, but a US government relying party would. Indeed, a citizen may not want to rely on a TTP which can be pressured by his government, as this can be used for eavesdropping; this is not an abstract concern, there are commercial products to do just this [65]. Thus an authentication infrastructure must permit relying parties to select their TTPs.

The **trust model** determines the rules by which TTP information is used in an authentication. We'll say that a trust model is **strong** if relying parties can specify the third parties they trust for an authentication. Absent a strong trust model, serious problems have resulted. We give two examples using TTPs known in X.509 as Certificate Authorities (CAs): (1) Verisign, issued a certificate providing an erroneous Microsoft public key [54]; this allowed signed, but unauthentic software updates to millions of computers running Microsoft software. (2) DigiNotar provided bogus certificates for google.com, www.cia.gov, and torproject.org [71, 42]; these certificates were used in Iran, presumably the result of an Iranian intelligence service attack[2]. In the first case, Microsoft should not have used *any* TTP since it was both relying party (software to be updated) and user (signed updates); in the second case, relying parties should not use a small Dutch TTP to authenticate major US organizations. Trust is highly contextual, and trust is *always* a risk. Clearly, a strong trust model is essential for security.

As we shall describe in §3, only Public Key Infrastructures (PKIs) are known to support a strong trust model. In particular two PKIs—SDSI/SPKI and X.509—support strong trust models but do so inefficiently. We describe why X.509 is inefficient in §2.

In this paper we present a new Internet-scale user authentication architecture called sayI, based on Public Key Cryptography (PKC) [26, 59]. It *is the first PKI with a strong trust model which is efficient.* In particular, sayI bounds latency to one Interent round trip time and efficiently uses bandwidth and cryptographic operations. Long term, our goal is to create and field sayI on the Internet, initially for the web. We are inspired here by the success of OpenID, but wish to avoid its weaknesses (see §3). We also intend to integrate it with network services such as authorizing mail relays or

[1]The user too may share some risk with respect to a relying party. The user can avoid this risk by not using the relying party's services; she may mitigate it with contracts. But since the relying party is autonomous, she cannot control it.

[2]If Iran had its own CA embedded in browsers—as do the Chinese, US, and many other countries—the DigiNotar attack would have been unnecessary.

mail delivery agents.

Authentication in sayI is group-based, that is a user u is authenticated in the context of a group g. A **group** is a set of users; sayI's groups are specified by the relying party and are fully distributed. There are several advantages for this approach as groups

- enable a strong trust model with efficient authentication;
- support varying quality of authentication, enabling relying parties to trade off cost vs. quality of authentication;
- can also be used for authorization [33, 62]; and
- prevent naming attacks. For example, using a group of banks, only banks will be authenticated; hbsc.com (not a bank) could not be confused with hsbc.com (the bank).

Finally, sayI incorporates the best properties of existing PKIs. It provides privacy, is resilient to failure, has simple and unambiguous naming of users, and authenticates cryptographically (§5.5).

The remainder of this paper is organized as follows: §2 describes the strong trust model in X.509; §3 describes related work; §4 describes the attacker model; §5 gives an overview of sayI; §6 describes sayI's certificates; §7 gives sayI's algorithm; §8 evaluates sayI's security and performance on several large, world-wide groups, and measures its performance in terms of bandwidth, network latency, and CPU time. In Appendix §A we describe how we intend to deploy sayI and §B describes sayI generalizations.

2 Authentication in a strong trust model

In this section, we'll describe authentication in X.509.

In a PKI, TTPs are relied upon through their certificates. Each certificate is a statement, cryptographically signed using a private key k_i that belongs to i, a certificate issuer [48, 50]. An issuer is either the relying party or a TTP. The statements are of the form *"i says x's key is k_x"*. The certificates form a directed graph in which the nodes are issuers or user keys (k_i or k_x above) and the certificates are edges between them (e.g., $k_i \rightarrow k_x$).

An authentication requires a sequence of certificates, called a **certification path** because it is a path in the graph. It was first introduced and formally defined in a global authentication service without global trust [14]. The first certificate is signed by an issuer known to the relying party; the last certificate specifies the user's key. In general, each certificate in a path is signed by a different issuer.

Certification path construction is the problem of finding a certification path. Certificates which cannot be part of a certification path for that relying party we'll call **unproductive certificates**, since they do not contribute to the authentication. Processing unproductive certificates wastes Internet bandwidth and latency, as well as signature verification CPU time [27, 52].

X.509 originally only had a hierarchical trust model [43]. In a hierarchical trust model, each node has a single parent. Thus certification path construction is performed by going up the hierarchy from user to root; there is only one possible path. A hierarchical model is very efficient because it considers only productive certificates. However, it is not a strong trust model since the relying party has no choice in the selection of TTPs used in an authentication.

X.509's 3rd version [23, 24] added a graph-based, strong trust model [17]. X.509 enables policy and naming constraints to be embedded in a certificate, which a relying party uses to trim the search space [27, 52], and to determine which certificates to rely upon. But, certification path construction is challenging because the graph is created from the TTPs' perspective rather than a relying party's perspective:

1. In a graph search, at each node an edge must be selected to search next; many of these edges (certificates) will be unproductive. An Internet-wide PKI encompassing all users would have billions of users (and therefore certificates).
2. X.509's constraints may not even be useful. In an open Internet PKI, the TTP cannot anticipate the needs of millions of relying parties. It is difficult to determine which TTP to trust. Proposals have included rule-based mechanism [35] and metrics [58].

Constraints are like road signs when driving, they can give only a very limited view of what's ahead. Especially on a long trip, additional information is needed such as a map, a compass direction, or directions specifying which roads to take. One who relies *solely* on road signs will waste a lot of time going on unproductive roads.

Instead sayI uses a group mechanism to determine productive certificates. The group mechanism serves as a (directed) map containing only productive roads which lead to the destination. Because the relying party constructs the group, it is a strong trust model.

An Internet-scale authentication system must also enable high *reuse* to reduce administrative costs and enable broad use. In sayI user-key and domain-key are reusable; the group mechanism is also highly distributed and reusable. For example, an organization might make available a group of all its members, which anyone can use for authentication.

3 Related work

In the following, we'll use A for Alice and B for Bob.

3.1 Network user authentication

Over-the-network user authentication is often based on passwords, despite the large variety of attacks against them, including guessing, spoofing, and key logging. (Non-networked use of passwords is considerably safer, especially when password security is improved [34, 66].) An extensive study of passwords vs. other schemes concluded that most are more secure than passwords [21]. Passwords do, however, require minimal infrastructure.

SSH is protocol and set of services which typically do not use an authentication infrastructure [73]. Two problems ensue from this lack of infrastructure; (1) is that passwords are often used despite the fact that SSH is public-key based. (2) there is no entity that can vouch for unknown users.

Kerberos provides authentication within the enterprise [55, 47, 56]. To authenticate A to B, A requests that the Key Distribution Center (KDC) create a symmetric key for it to communicate with B. To speak to the KDC, a key is generated from a password or by PKC using PKINIT. The KDC can thus impersonate either party or do a Man-in-the-Middle (MitM) attack between A and B. Kerberos is inappropriate on the Internet because the KDC (a TTP) would hold principal secrets and A and B must trust a common set of KDCs.

3.2 Authentication Infrastructure

The most important authentication infrastructure issues are scaling, naming, and trust. Scaling and naming are matters of performance. Trust and naming are security issues.

X.509 is the most widely deployed PKI. It is very flexible, but is also complex [3, 46], and its semantics are inconsistent [38]. Distinguished names are ambiguous, because there is no central naming authority. Hence, an additional name (e.g., an email address) is often used. The original design of X.509 relies on LDAP directories, a hierarchical structures and offline revocation. All of these are not suited for today's Internet use, and can result in naming, "Which directory?" and cross-certification issues which increase the complexity of authentication on Internet [36]. Gutmann analyzed many issues and challenges in X.509 and has suggested new approaches to PKI system [37].

In X.509, an Attribute Certificate (AC) [32] can name a group to which a user belongs. The attribute-based construction of groups makes it easy to test group membership, but it does not aid in certification path construction.

PGP [9] uses email addresses as names, and allows every user to create identity certificates. Email-like addresses are perhaps the one world-wide naming scheme in which names are unambiguous[3]. But PGP's reliance on individuals and email makes it difficult to scale, since it is both offline and each publisher knows relatively few entities. Moreover, PGP is radically decentralized and transitive; both amateurs and professionals may equally contribute to a web of trust. As a result, it is difficult to analyze and regulate trust in PGP. sayI adopts PGP's use of the email-style names.

SDSI/SPKI is a PKI which was designed to support a strong trust model using groups [61, 30, 31]. SDSI/SPKI's greatest contribution is its strong trust model [29] and attendant security analysis.

SDSI uses membership certificates to specify that an entity belongs to a group [61]. Unfortunately, in SDSI, name resolution—used to determine group membership—requires $O(lc^3)$ time where c is the number of certificates and l is the maximum path length in a name expression [22]. At Internet scale, c^3 is too large.

SDSI/SPKI performance is sensitive to SDSI/SPKI's local names [1, 39, 51, 28]. Local names significantly authentication costs *"since one SDSI name can be defined in terms of another, SDSI name resolution is fundamentally more complex than DNS name resolution"* [4]. sayI's names are based on domain names.

OpenID [57] is a web-based authentication protocol; it contains a billion credentials and is the largest authentication system ever built. A user is authenticated, for example, by password to an OpenID identity provider (a TTP). The provider then signs an authenticator which the relying party verifies. OpenID provides only user authentication, relying on X.509 to authenticate web sites and OpenID providers. The OpenID identity provider is privy to such private information as which service a user accesses, when and how it has been accessed, etc. OpenID adoption is affected by user concerns of privacy and credential exposure [67]. In general, the benefits discussed in §5.5 are not provided by OpenID.

The development of PolicyMaker/KeyNote overlapped with that of SDSI/SPKI. PolicyMaker [17], and its successor KeyNote [15, 16, 19] were designed to deal with the trust management issue, which combines distributed authorization and authentication, into a process called *compliance checking* [18, 12]. Trust management separates distributed authentication and authorization from application logic, leading to more modular (and hence more secure) systems. A large number of Trust Management Languages (TMLs) have been created to deal with various problems, e.g., privacy [20, 72, 44, 40, 13, 63].

Domain Name System Security Extensions (DNSSEC) [7, 8, 45] protects the domain name resolvers from forged Domain Name System (DNS) data. The primary goal of DNSSEC is protecting IP addresses, but other general-purpose certificates can also be protected by using CERT (certificate) records. DNSSEC responses (RRSIG record) are signed and can be verified by a certification path consisting of DS (containing key) and DNSKEY (containing DS record and DNS record) records. Such a sequence enables a domain to be reached from root DNS. While DNSSEC supports multiple trust anchors, they are all part of a single hierarchy (without relying part choice), and thus it is not a strong trust model.

A trust graph [2] specifies the trust relationships among entities. The nodes are entities on the network and the edges between any two nodes enable one node to certify the public key of another node. But the trust graph lacks a way to differentiate trust strength based on use and as far as we know has never been implemented.

[3]Of course, no naming scheme is perfect. Names may be reused or authorities may issue duplicate names. Still, such names are the best we have.

4 Attacker Model

The attacker model includes attackers who

- want to steal authentication secrets from TTPs,
- are third parties not trusted to provide accurate information for a given use (e.g., DigiNotar attacks),
- are TTPs that want to violate user privacy,
- want to fabricate authentication information by on-line attacks against a TTP, and
- use Denial-of-Service attacks against TTPs to prevent relying party authentications.

Specifically excluded from the attacker model is dishonest TTPs or insider attacks at honest TTPs, although such attacks can be audited even after the fact by examining the certificates provided. The relying party is assumed to take care in specifying which TTPs it uses. Similarly excluded are attacks on the relying party or user.

5 sayI overview

A PKI must specify the TTPs used in an authentication, name entities, define certificates, provision keys, and specify certification path construction.

In sayI, group construction—the mechanism for creating groups which can span organizations—determines the TTPs relied upon when using that group. The set of users that can be authenticated using a group is called the group's membership.

Authentication on the Internet is inherently of varying reliability. Using groups for authentication ensures that the authentication is reliable enough for the purpose it will be used. For example, authenticating a system administrator would typically not rely on *any* TTP, because such access is very security sensitive. On the other hand, a university library might lend a book based on authentications using many TTPs—perhaps from any university within the same state—enabling broader coverage but at lower integrity. Thus there is a tension between strength of authentication and coverage of users: large groups inherently have weaker authentication. Groups enable this tunable use-based authentication reliability.

In this section we give an overview of sayI. §5.1 describes certificates and the group tree; §5.2 describes certificate names; §5.3 describes distributed groups and certification path; §5.4 shows how certificates were designed for reuse, thus making the system more economical to use; finally we describe a number of general advantages of PKIs, in §5.5 to clarify some sayI design decisions.

5.1 Certificates

sayI constructs groups using user, key, and group certificates. The ultimate purpose of group construction is to define a set of user certificates to be used for user authentication. The types of certificates are:

cert.	purpose
user	binds a user name to a public key.
key	binds a TTP domain name to a public key.
group	names (1) members of the group, (2) key certificates and (3) other group certificates.

A group is specified by its root group certificate. The certificates of a group form a tree, called the **group tree**, rooted at its root group certificate. Only group certificates have children; user and key certificates must be leaves. The **group name** is the name of its root group certificate. Unlike a hierarchical PKI in which there is a single global tree, sayI has a tree per group and as a whole is a Directed Acyclic Graph (DAG).

Policy is set with group certificates, since the group certificate says which TTPs are trusted when authenticating that group. Of course, security depends on all the TTPs used in an authentication.

sayI groups are designed to minimize latency and bandwidth. The group tree contains *exactly* those certificates necessary to authenticate users of the group, and hence does not contain any unproductive certificates. As a result, the certification path construction algorithm never encounters an unproductive certificate—reducing network bandwidth and latency as well as signature validation computations.

5.2 Names

Naming is important for performance and security. sayI names have the same form as email addresses, $l@d$, where *l* is a local name and *d* is a domain name of a TTP. Such sayI names specify certificates and in the case of user certificates they also name users. They are unique, because the hierarchical administration of the Internet ensures uniqueness of *d*; and administration within the domain ensures the uniqueness of *l*. *d* specifies both the location from which to fetch a certificate and the signer of the certificate. We say that a certificate signed by *d* is **in *d***, since it is contained in *d*'s domain.

We give examples of certificates and their names below

Type	Name	Example
Group	*l*@g.*d*	friends@g.smith.org
		smith.org provides a group of her friends
Key	*l*@k.*d*	ydom.com@k.xdom.com
		xdom.com asserts ydom.com's key
User	*l*@u.*d*	bob@u.sigsac.org
		the user known as bob at sigsac.org

There are three name spaces, so that group, key, and user names do not conflict.

While we use only ASCII names in the examples, there is no reason why names could not be Unicode (UTF-8). Indeed our encoding uses a length field for all names, and thus allows names with arbitrary contents. See [53] for a discussion on names.

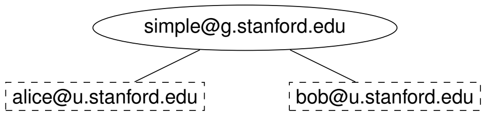

Figure 2: Group Tree for A Single Domain

Example 1. Figure 2 shows a group named simple@g.stanford.edu contained in domain stanford.edu. The group contains two users: alice@u.stanford.edu and bob@u.stanford.edu. Each node in the figure is a certificate: The elliptic node is a group certificate, and the dashed rectangular nodes are user certificates.

5.3 Distributed Groups and Certification Path

A group could be specified with a single group certificate which provides all the members of a group. Or it could use multiple group certificates, thus distributing group construction, possibly across domains. This modular structure enhances both large group construction and group re-use.

In sayI, a TTP could be an intermediary or a publisher. An **intermediary** is an issuer of group and/or key certificates (a relying party is always an intermediary). A **publisher** issues user certificates. An issuer may be both a publisher and an intermediary.

All groups specified by relying party *d* must have a name of the form *g@g.d*—that is, groups must be at the relying party. The relying party is seeded with a single key, k_d. Prior to use, a certificate is validated with its issuer's public key to detect fraudulent certificates.

Before a relying party can validate a certificate at *d'*, it must fetch the key certificate for *d'*. Thus in the group tree, key certificates must be fetched before certificates in another domain. Key certificates are inherently cross-domain certificates.

A certification path is a path in the group tree plus the necessary key certificates. A certification path starts with a group certificate (signed by the relying party) and ends with a user certificate. The relying party's key is the **anchor**; the **target** is the user being authenticated. Certification path is of the form:

$$c_1, c_2, \ldots c_n$$

Certificates $c_1, \ldots, c_{n-1}$ are either group certificates or key certificates; c_n is a user certificate. Each certificate c_j is either signed by the relying party or a signing key is provided in a key certificate $c_i, i < j$.

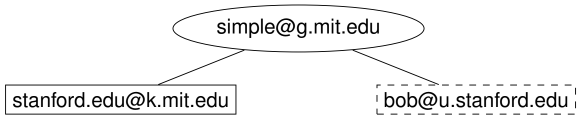

Figure 3: Cross-Domain Group Tree

Example 2. Figure 3 shows a cross-domain group tree for the group simple@g.mit.edu which spans two domains mit.edu, stanford.edu and has a user bob@u.stanford.edu. The key certificate stanford.edu@k.mit.edu is denoted by solid rectangular; it is referenced by simple@g.mit.edu. The anchor is $k_{mit.edu}$. The certification path is: simple@g.mit.edu, stanford.edu@k.mit.edu, bob@u.stanford.edu.

5.4 Reuse

Group certificates promote reuse, since they can be used as parts of other groups.

Keys are contained only in key and user certificates, so that keys can be changed (for example due to a compromise) without changing group certificates. Thus key provisioning is separated from group membership. This promotes reuse of key and user certificates in different groups, while making group certificates more stable.

Modularity and low cost encourages organizations such as governments, employers, schools, and financial institutions to share their authentication databases for which they have already done the most costly step, physically identifying users. Physically identifying users is typically performed locally (i.e., face-to-face), using paper credentials such as driver's licenses, passports and other techniques to prevent impersonation. Reuse is a kind of economic efficiency, in which the authentication infrastructure's fixed cost is spread over more uses.

Example 3. Figure 4 defines a group of organizations at financial institutions (finance@g.finance.com) consisting of banks (banks@g.federalreserve.gov) and credit card companies (creditcard@g.creditcard.com). Note that banks and creditcards are, in themselves, groups.

In this example, the bank keys are provided by the federal reserve bank, the regulator of banks in the US. If this was insufficient—perhaps you are transferring billions of dollars—you can eliminate this intermediary by going to your bank and getting its public key. The relying party decides whether group construction is delegated out to save cost and reduce the inconvenience of acquiring keys, or the relying party manages all keys and group certificates.

A group certificate can reference existing group certificates. An application which needs to authenticate a bank can make use of the existing bank group defined in Figure 4. The application just needs

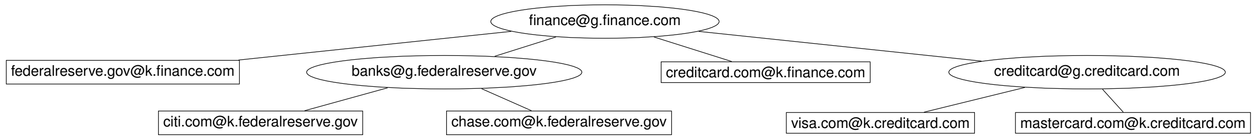
finance@g.finance.com
federalreserve.gov@k.finance.com
banks@g.federalreserve.gov
creditcard.com@k.finance.com
creditcard@g.creditcard.com
citi.com@k.federalreserve.gov
chase.com@k.federalreserve.gov
visa.com@k.creditcard.com
mastercard.com@k.creditcard.com

Figure 4: Group Tree for Multiple Domains

to create its own group certificate and sign the key for banks group (see example 4).

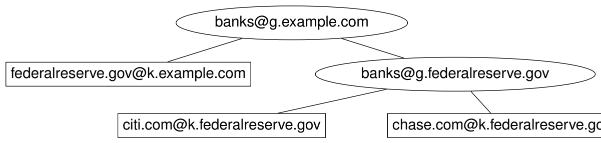
banks@g.example.com
federalreserve.gov@k.example.com
banks@g.federalreserve.gov
citi.com@k.federalreserve.gov
chase.com@k.federalreserve.gov

Figure 5: Reuse banks group

Example 4. Figure 5 shows a group of banks. example.com creates a group certificate banks@g.example.com which references banks@g.federalreserve.gov and a key certificate federalreserve.gov@k.example.com which provides the key for federalreserve.gov.

5.5 Other properties of sayI

We consider here some other properties of sayI.

1. There are no shared secrets.
   (a) TTP eavesdropping requires a bogus key binding which can be detected after the fact.
   (b) one key can be used with all relying parties.
2. Mutual authentication can be asymmetrical. For example, the customer can use the Federal Reserve to authenticate his bank; the bank can use its customer database to authenticate the customer.
3. Certificate signing can be performed off-line [50], preventing signing certificates without corrupting an insider. Thus, theft of PKC authentication databases or compromise of the on-line database server cannot result in faulty authentication.
4. Certificates can be stored in untrusted caches, since they are tamper evident. Tampering with the certificate would cause signature validation to fail. This has important performance and reliability implications as caching can be used to:
   - increase reliability when a TTP is unavailable, by using cached copies;
   - reduce bandwidth, latency, and CPU use.
5. Privacy from TTPs, as a TTP never needs to know when a user is being authenticated. The user certificate is directly provided by the user.

6 sayI Certificates

All certificate types—group, key, and user—share the same header format. The certificate header fields, sizes in bytes, and purpose are given below.

field	bytes	explanation
size	4	the size of the certificate in bytes
version	4	certificate encoding version
validFrom	16	
validTo	16	the certificate's validity period
typeHash	64	the type of the certificate
revServer	1 + len	revocation server name
publicKey	32	of the certificate signer, and
signature	128	over the certificate.

sayI certificates use Elliptic Curve Cryptography (ECC), providing high (128-bit) security—equivalent to the security of AES-128 [11]. ECC has been extensively studied since 1985, and since 2005 is the only PKC that NSA recommends for the protection of US government SECRET information. ECC also has smaller signatures and public keys than schemes such as RSA and DSA.

sayI is efficiently encoded. Given an 11-byte average domain name [64], a certificate header on average is 276 bytes. Use of ECC, a more efficient encoding, and elimination of unnecessary fields results in sayI certificates which are about a fourth the size of X.509 certificates. While sayI's compact encoding is certainly an advantage, its group structure has a far larger impact on performance.

One of the challenges of dealing with Internet scale is to support a range from small to extremely large (i.e., the whole Internet) groups efficiently. For **simple groups**, sayI uses prefetching to reduce authentication latency: This is an effective technique for all but the largest groups. For the largest groups, it is desirable to avoid the bandwidth and latency of prefetching. For very large groups, we introduce **segmented groups** which use more limited prefetching. Using simple and segmented groups, sayI efficiently supports groups of any size.

There is one key algorithmic difference between the two schemes. In the simple case, all key and group certificates are prefetched before any user authentication, while in the segmented case, all but the last level (in the group tree) of key certificates are pre-fetched. Either case requires one roundtrip latency (see §8). The segmented case reduces the prefetch size at the cost of a key fetch in user authentications.

We next describe the group certificates §6.1, the key certificates §6.2, and the user certificates §6.3.

6.1 Group Certificates

In addition to header fields, a group certificate contains the fields:

keyNames	a tuple of fully qualified key names (see §6.2)
groupCerts	a tuple of group certificate names
members	group members, each name is either:
	user name a single user or
	publisher every user at that publisher.

Wild cards can be used to denote sets of members or key names. For example, *.com means all .com names. Names with '*' in them are called **segmented names**; they are used only for segmented groups.

A **segmented certificate** is a group certificate which has exactly one segmented key certificate name; it may also contain one segmented member name as well as non-segmented key names. *Only* a segmented certificate can contain segmented names.

6.2 Key Certificates

A **fully qualified key name** specifies the name of a TTP and the location of the named TTP's key. For example, the fully qualified key name harvard.edu@k.example.com contains harvard.edu's key as asserted by example.com. Here, harvard.edu is called the **short name** of the key's owner. Short names are used to identify keys.

A key certificate specifies a set of pairs, where each pair is a

keyName	TTP's short name.
publicKey	TTP's public key.

6.3 User Certificates

User certificates have the same form as key certificates except the pairs consist of

userName	user's name.
publicKey	user's public key.

7 Algorithm

The entities involved in authentication are given in Figure 6, an expanded version of Figure 1. The user gets their user certificate from the publisher and supplies it to the relying party. The relying party requests the cache to create the certification path; this may require consulting intermediaries. An intermediary provides the needed group certificates and key certificates.

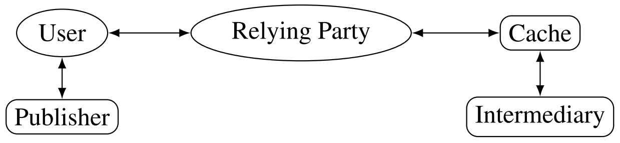

Figure 6: Architecture for sayI

Authentication takes place when a user u wants access to a service s provided by a relying party to user of group g. For u to be authorized to use s, u ∈ g. Authentication is done in the context of g, simultaneously authorizing the user to access the service and authenticating the user via the certification path. Thus g serves both authorization and authentication in a natural way.

Certification path construction has two phases. We describe here the simple case: The first phase, **group prefetch**, fetches the group and key certificates. The second phase, **lookup**, fetches the user certificate and computes the certification path.

The full algorithm overview is as follows:

Group prefetch Given the group name provided by the relying party, the cache fetches its root group certificate. From the group certificate the cache recursively fetches the key and group certificates until on each path either there is a segmented group certificate or there are no more key or group certificates to be fetched.

Lookup A user provides a user name, e.g. u@p.com, to the relying party, and then the relying party requests the user certificate from the user. Concurrently, the relying party forwards the user name to the cache. The cache returns a certification path from anchor to p.com's key certificate. (In the segmented case, the cache will also fetch the key certificate for p.com; for non-segmented groups this step has been performed during group prefetch.)

Since the user provides her own user certificate and since other certificates are cached, the TTP is unaware of user-relying party interactions, providing privacy to the user and the relying party.

Now that we have given an overview of the algorithm, we'll provide examples of simple and segmented groups.

Simple group example Figure 7 shows a simple (non-segmented) certificate example (corresponding to Figure 3) for the group name simple@g.mit.edu. We omit the header fields except "signer", which contains public key k_x for signer x. We show the certificates fetched for the user's name bob@u.stanford.edu. The group prefetch reads 7a and 7b. Upon the request from the relying party, certificate 7c is provided by the user. Lookup returns 7a and 7b which are used to validate 7c.

The group's members are a subset of the users at stanford.edu and mit.edu; for reasons of space we show only a single user's certificate, bob@u.stanford.edu.

signer	$k_{mit.edu}$
members	bob@u.stanford.edu
groupCerts	
keyNames	stanford.edu@k.mit.edu

(a) Group certificate simple@g.mit.edu

signer	$k_{mit.edu}$
publicKey	$k_{stanford.edu}$
keyName	stanford.edu

(b) Key certificate stanford.edu@k.mit.edu

signer	$k_{stanford.edu}$
publicKey	k_{bob}
userName	bob@u.stanford.edu

(c) User certificate bob@u.stanford.edu

Figure 7: Simple certificate example

Segmented Certificate Example A segmented certificate example is given in Figure 8, and is a truncated example for the group "everyone on the Internet". The group name is everyone@g.example.com. The group certificate edu@g.example.com specifies the subtree for signing all domain names ending with .edu. (If extended to the entire Internet, there would be a group certificate for each top-level domain.) During group prefetch, the certificates 8a, 8b, 8c, 8d, and 8e are fetched. Given a user's name bob@u.mit.edu, the user certificate 8g for it is provided to the relying party. During lookup the key certificate 8f is fetched (based on the segmented certificate 8b) and the certification path from 8a–8f is returned to the relying party.

Algorithm The algorithm builds a certification path as described in §5. The certification path contains group and key certificates, and its last certificate is a user certificate. For example, given bob@u.stanford.edu in Figure 7 the certificate path is the sequence of certificates with names

[simple@g.mit.edu, stanford.edu@k.mit.edu,
simple@g.stanford.edu, bob@u.stanford.edu].

The algorithm is based on groups. By phase:

prefetch does the group prefetch. It process group and key certificates using processGroupCert and processKeyCert.

lookup does the lookup phase, first fetching any needed key certificates and then in constructCertPath constructing the certification path.

The primary variables used in the certification are: g is a group, n is a (fully qualified) certificate name, gn is a group certificate name, gci is group certificate information; kn is a key certificate name, s is a (short) key name, k is a public key, c is a certificate, kc is a key certificate, and gc is a group certificate.

Four maps are defined for each group:

g.GroupCertInfoMap is a map from group certificate names to information about group certificates. Such information includes:

Keys a map from short key names to a [k, kn]. This map contains the keys that can be used to verify certificates named in the group certificate. The elements are the public key k and fully qualified key name kn of the signer of k. The short key names are from the keyName field of a key certificate.

SegmentedKeyName If a segmented group certificate, this field is used to store segmented key certificate name.

g.MemberMap A map from member names to the group certificate name which contains them.

signer	$k_{example.com}$
members	
groupCerts	edu@g.example.com, com@g.example.com
keyNames	

(a) Group certificate everyone@g.example.com

signer	$k_{example.com}$
members	*.edu
groupCerts	
keyNames	edu@k.example.com, *.edu@k.eduintermediary.org

(b) Group certificate edu@g.example.com

signer	$k_{example.com}$
publicKey	$k_{eduintermediary.org}$
keyName	eduintermediary.org

(c) Key certificate edu@k.example.com

signer	$k_{example.com}$
members	*.com
groupCerts	
keyNames	com@k.example.com, *.com@k.comintermediary.org

(d) Group certificate com@g.example.com

signer	$k_{example.com}$
publicKey	$k_{comintermediary.org}$
keyName	comintermediary.org

(e) Key certificate com@k.example.com

signer	$k_{eduintermediary.org}$
publicKey	$k_{mit.edu}$
keyName	mit.edu

(f) Key certificate mit.edu@k.eduintermediary.org

signer	$k_{mit.edu}$
publicKey	k_{bob}
userName	bob@u.mit.edu

(g) User certificate bob@u.mit.edu

Figure 8: Everyone certificate example

Algorithm 1 **prefetch**(groupName)

```
1: $g \leftarrow newGroup(groupName)$
2: $g.ParentMap[groupName] \leftarrow$ “”
3: $g.processGroupCert(\{anchor\}, groupName)$
4: return $g$
```

g.ParentMap a map from certificate name to the name of the certificate's parent in the group tree.

g.CertMap a map from certificate name to [c, kn], where c is the certificate and kn is the fully qualified key name of the signer. The map stores all fetched certificates.

The group prefetch phase is invoked by calling prefetch with the group name as the parameter (see Algorithm 1). The routine creates a group, g, and fetches the group and key certificates.

The procedure processGroupCert (see Algorithm 2) fetches the first group certificate, determines its key, and validates the certificate. It then iterates through the certificate's keyNames, fetching the key certificates if they have an unsegmented name and adding to the set of keys that can be used at that node. If there is a segmented key, it is saved in SegmentedKeyName. Then the group certificates are fetched and processed, and finally the member names are processed.

Algorithm 2 g.**processGroupCert**(keys, gn)

```
1: $groupCert \leftarrow requestCert(gn)$
2: $key \leftarrow keys[publisher(gn)]$
3: if $validate(key, groupCert)$ then
4:     $g.CertMap[gn] \leftarrow [groupCert, key.kn]$
5:     $gci.Keys \leftarrow keys$
6:     for all $kn \in groupCert.KeyNames$ do
7:         $g.ParentMap[kn] \leftarrow gn$
8:         if $segmented(kn)$ then
9:             $gci.SegmentedKeyName \leftarrow kn$
10:        else
11:            $gci.Keys \leftarrow g.processKeyCert(gci.Keys,\ kn)$
12:        end if
13:    end for
14:    for all $gcName \in groupCert.GroupCerts$ do
15:        $g.ParentMap[gcName] \leftarrow gn$
16:        $g.processGroupCert(gci.Keys, gcName)$
17:    end for
18:    $g.GroupCertInfoMap[gn] \leftarrow gci$
19:    for all $member \in groupCert.Members$ do
20:        $g.MemberMap[member] \leftarrow gn$
21:    end for
22: end if
```

Algorithm 3 g.**processKeyCert**(keys, kcName)

```
1: $kc \leftarrow requestCert(kcName)$
2: $key \leftarrow keys[publisher(kcName)]$
3: $Map \leftarrow keys$
4: if $validate(key, kc)$ then
5:     $g.CertMap[kcName] \leftarrow [kc, key.kn]$
6:     // keyName in kc is short key name
7:     for $i \in \{0...\#kc.keyName\}$ do
8:         $Map[kc.keyName[i]] \leftarrow$
           $[kc.publicKey[i], kcName]$
9:     end for
10: end if
11: return $Map$
```

The other certificate processing routine is processKeyCert (see Algorithm 3). The key certificate is fetched, validated, and then added to the CertMap. The name-key map is returned, to be included in the group certificate's Keys map.

Now we consider the lookup phase, lookup (see Algorithm 4). The lookup is triggered when a user connects to the relying party, sending a user name. The relying party requests the user certificate for the user name and in the mean time forwards the name to the cache. The cache then does the lookup, which primarily deals with segmented certificates. If the publisher of userName is in a segmented certificate, its fully qualified key name, kn, is created by instantiate, and then, if the key certificate is not in the cache, it is fetched and processed. Next the non-segmented part of lookup is invoked and returned as the result of the lookup.

Finally, the certification path is constructed in constructCertPath (see Algorithm 5). Most of this algorithm creates the certificate chain, certs (excluding user certificate). First, the certificate tree is traversed from userName to groupName, creating the tuple certNames, containing group certificate names. Next certNames is traversed, putting the necessary key certificates and group certificates into certs.

Algorithm 4 g.**lookup**(userName)

```
 1: // Publisher Info
 2: $p \leftarrow publisher(userName)$
 3: $gn \leftarrow g.match(p)$
 4: $gci \leftarrow g.GroupCertInfoMap[gn]$
 5: if $gci.SegmentedKeyName \neq$ ”” then
 6:     $kn \leftarrow instantiate(gci.SegmentedKeyName, p)$
 7:     if $gci.Keys[p]$ $not$ $found$ then
 8:         $gci.Keys \leftarrow g.processKeyCert(gci.Keys, kn)$
 9:         $g.ParentMap[kn] \leftarrow gn$
10:     end if
11: end if
12: // User Info
13: return $g.constructCertPath(gci.Keys, userName)$
```

Algorithm 5 g.**constructCertPath**(keys, userName)

```
 1: $certs \leftarrow []$
 2: $certNames \leftarrow []$
 3: $certName \leftarrow g.match(publisher(userName))$
 4: while $certName \neq$ ”” do
 5:     $certNames \leftarrow [certName] + certNames$
 6:     $certName \leftarrow g.ParentMap[certName]$
 7: end while
 8: // Now construct cert chain
 9: $keySet \leftarrow \{anchorKey\}$
10: for all $n \in certNames$ do
11:     $kn \leftarrow g.CertMap[n].kn$
12:     $keyChain \leftarrow []$
13:     while $kn \notin keySet$ do
14:         $keySet \leftarrow keySet \cup \{kn\}$
15:         $keyChain \leftarrow [g.CertMap[kn].c] + keyChain$
16:         $kn \leftarrow g.CertMap[kn].kn$
17:     end while
18:     $certs \leftarrow certs + keyChain + [g.CertMap[n].c]$
19: end for
20: return $certs$
```

Finally, the relying party takes the path returned by constructCertPath and the user certificate and verifies the certificate chain. (If the cache is trusted, it can verify the certificate chain as each cert is fetched.) We have not described caching of user certs (alternatively user certs could be sent on connect), two obvious means of reducing Internet roundtrips.

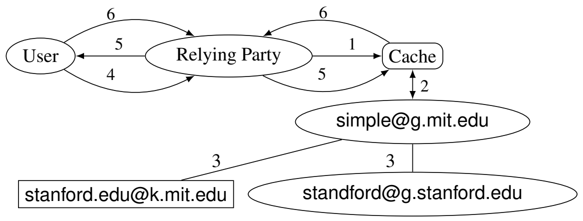

Figure 9: Authentication sequence for cross-domain group

Figure 9 shows how the algorithm works on the simple cross-domain group given in Figure 3. The numbers in the figure indicate the sequence of operations. Operations with the same number can be performed in parallel.

1. The relying party requests the group simple@g.mit.edu to be fetched by the cache.
2. The cache fetches root group certificate simple@g.mit.edu.
3. The cache fetches the second level key and group certificates.
4. The user sends a user name to the relying party.
5. The relying party requests
   - the user certificate from the user and
   - the certification path (except user certificate) from the cache.
6. The certification path is returned in two replies to step 5 and is then validated by the relying party.

8 Evaluation

In the previous sections, we have shown that: sayI is designed to provide a strong trust model which can support efficient global-scale user Internet authentication. sayI needs at most one Internet roundtrip for a user authentication, once the group has been prefetched.

In this section we evaluate certification path construction, in which 10,000 organizations and more contribute to an authentication database. We'll also show how the entire Internet can be structured to authenticate *anyone*—across hundreds of millions of organizations and billions of users. While the reliability of Internet-wide authentication is inherently weak—it is vulnerable to each of the many TTPs it relies upon—it demonstrates scalability far beyond current state-of-the-art techniques.

We evaluate sayI from a number of perspectives:

group prefetch the bandwidth (bytes fetched) and time needed before any query can be answered.

lookup the bandwidth and time to do a lookup.

complexity the algorithmic complexity.

X.509 study sayI is compared to X.509

security the security improvements.

CPU performance was measured on a computer with a 3.2 GHz AMD Phenom II X4 965 processor and 8 GB of memory. For the purpose of the experiment, we only use one core. Internet latency numbers were measured on PlanetLab, a world-wide networking testbed.

sayI is written in the Go programming language, developed at Google for systems programming. Our initial implementation is 1,879 lines of code—not including libraries and tools from a related project. We are still working on some extensions, see §B; when completed we will release sayI as open source.

8.1 Groups evaluated

The cost of group prefetch depends both the group size and the number of intermediaries. We focus here on substantial groups, which are the largest we conceived. These are the worse case for our scheme, smaller groups will be proportionately less expensive. To test scalability, it is necessary to stress the performance.

We use as examples the following large, world-wide groups which have need to communicate with each other. To our knowledge, such groups do not exist today; our goal was to stress sayI with these out-sized groups. Thus, we would expect almost all groups to be far smaller. The first four groups, numbering from 9,290 to 36,722 organizations are:

Universities	9,290 worldwide [68] [41]
Banks	13,371 worldwide [69]
Hospitals	18,000 worldwide [6]
Municipal Governments	36,722 worldwide [5]

(The source for these numbers is Wikipedia, the most readily available source; the purpose of these numbers is to provide a rough estimate of large group sizes.) Such groups would enable sharing educational transcripts, medical records, financial information and intergovernmental communication. These groups might have millions, even billions, of users.

We also consider three other synthetic groups

Small Group	of 160 organizations, based on [74]
Mega Group	a very large group with 5,000,000 organizations
Internet	which is composed of hundreds of millions of organizations and about two billion users [25][49]

Thus we have small, large, and exceptionally large groups on which to measure sayI. These groups are named in terms of the needed group sizes. The largest PKI group today is the DoD authentication database composing tens of millions of users [10]. It has a hierarchical structure—an adequate trust model only because all of its relying parties share the same adversaries, those wishing to attack the defense establishment of the US.

The Mega Group is at the far end of what might constitute a reasonable group: 5,000,000 organizations could easily describe billions of users. We don't believe that such large, geographically dispersed group would be worth distinguishing from the Internet as a whole. Nevertheless, we wanted to stress our infrastructure to ensure that it can handle even extreme use scenarios.

The next factor is the number of intermediaries for a group. The number of intermediaries needed depends on geographic size, organization size, and other factors. For example, larger organizations need fewer intermediaries since they are available in more places, and hence able to do the physical identification step at more places. We have modeled 400 intermediaries for groups that span a medium number of organizations—these intermediaries are one per country except for the most highly populated countries (over 100,000,000) where an intermediary is provided for population units of 100,000,000. We use 2,000 intermediaries for groups that span a large number of organizations, approximately one organization per 3.5 million people. The Internet is a special case, since there are now 280 top-level domains [25], and it is one of the few places that names can be used to partition the space. In all cases, we have used 3 levels of intermediaries; if more levels were used, it would only affect group prefetch latency, a one-time cost for a group.

8.2 Group Prefetch

Table 1 shows the cost of the group prefetch using simple (i.e., non-segmented) and segmented groups. For each of simple and segmented, the table gives:

- the number of key/group certificates,
- the bytes those certificates use (and hence the bandwidth necessary to transmit them), and
- the time to fetch them, which is a combination of latency and limited bandwidth.

For all but the largest groups, the simple group prefetch is modest, less than 3.4 million bytes. For the largest groups, the value of segmentation is substantial: The whole Internet is less than 200,000 bytes. The Mega group is larger, at 61 million bytes, but we do not believe anyone will actually want to construct groups of this size because it is unwieldy and we believe unneeded. However, it is feasible to support such groups in sayI.

Each group (other than the Internet) explicitly names all of its member organizations. Segmented groups have a smaller prefetch, since they do not fetch publishers' keys; but if all the members are fetched, segmented groups will use more total bandwidth. We used 11-byte names, the average size of an Internet name [64].

The time measures the group prefetch phase using a 1 megabit/second connection plus the latency, calculated by the depth of the group tree multiplied by t_w, the worst case round-trip latency between two nodes on the Internet. We determined t_w, by running tests on PlanetLab. For node distances of between 8,000–12,205 miles, we measured $t_w = 1.45$s. This is an overestimate (pessimistic for our case), since in general the latency and bandwidth will overlap in time, distances are less than the maximum, and PlanetLab's multiplexing means that a fraction of t_w is spent waiting for a time quantum. We had originally tried to get performance number for sayI on PlanetLab, but were unable to do so[4].

We used 3 levels of intermediaries. This means that of the times given, $3 \cdot 1.45 = 4.35$ seconds is for latency. The remaining time is for bandwidth. In the simple case, prefetch is bandwidth limited while in the segmented cases prefetch is primarily (except for Municipal Government and Mega Group) latency limited.

sayI can support almost 6,000 certificate validations per second (on a single core). During the lookup phase, these certificate validations can be overlapped with network operations, and thus do not add to total time.

8.3 Lookup

The time and number of bytes to do a lookup depends only on (1) whether the necessary certificates are cached and (2) whether the group is segmented or not. Table 2 shows the number of bytes fetched and Internet latency. It also shows the number of authentications per second per CPU.

If the certificates are not cached then one Internet fetch (the user certificate) is needed for a simple group and two concurrent Internet fetches (the user certificate and the publisher's key certificate) are needed for a segmented group. When not cached, signature verification time dominates authentications per second, limiting performance to 5,882 path discoveries per second in the simple case and 2,941 in the segmented case. ECC signature verifications are more expensive than RSA, but ECC signatures and public keys are smaller, saving transmission bandwidth and making more effective use of caches.

We also measured the number of authentications per second when the user certificate is cached, and hence no verifications are needed. So the number authentications is much larger, 78,259 in the simple case and 49,195 in the segmented case, yielding an improvement of a factor of 13.3–16.7 over the uncached case.

	Group Type	Bytes	Internet Latency	Auth/ Second
Not Cached	Simple	345	1.45 s	5,882
	Segmented	692	1.45 s	2,941
Cached	Simple	0	0	78,259
	Segmented	0	0	49,195

Table 2: sayI lookup latency

8.4 X.509 performance studies

Performance studies of any scale on the X.509 graph model are scarce. For example, Elley et al. [27] which considers the problem of X.509 certification path does not report any numbers. We conjecture this is because (1) there are a large number of options in path discovery and (2) bandwidth limitations mean that graph deployments are all small scale.

Zhao and Smith have studied X.509 certification path of an 160 organization graph [75, 74] using a network simulator; these papers provided key insights in the design of sayI. They show a network latency of 36 Internet round trips on average in X.509; the same size group requires 4 round trips in sayI (including 3 levels of group

[4] We had two problems: First, sayI's memory usage exceded PlanetLab limits; we reduced sayI's memory footprint. Second, the multiplexing of PlanetLab was so high (and non-transparent) that we were essentially measuring job schedule queuing time rather than network latency. We suspect that a reduced memory footprint would result in smaller queuing delay, but that would have required a complete rewrite of sayI.

		Inter-	Simple			Segmented		
Groups	**Publishers**	**mediaries**	**Certs**	**Bytes**	**Time (sec)**	**Certs**	**Bytes**	**Time (sec)**
Universities	9,290	400	1,201	787,190	10.36	801	378,155	7.24
Banks	13,771	400	1,201	1,038,126	12.27	801	431,927	7.65
Hospitals	18,000	400	1,201	1,274,950	14.08	801	482,675	8.03
Municipal Gov.	36,722	2,000	6,001	3,388,982	30.21	4,001	1,772,939	17.88
Small Group	154	6	19	13,170	4.45	13	6,119	4.40
Mega Group	5,000,000	2,000	6,001	281,332,550	2,150.75	4,001	61,332,275	472.28
Internet	200,000,000	280	841	11,200,193,750	85,455.05	561	196,835	5.85

Table 1: Groups which span the Internet, their structure as sayI groups and group prefetch bandwidth and latency measures

prefetch and a lookup). Using PlanetLab latencies, their time is 52.20 seconds vs 4.45 seconds for sayI (see "Small group" in 1), both of these numbers use 1.45 seconds round trip time.

In their paper, Zhao and Smith used a round-trip time of 0.22 seconds; using this smaller number they achieve 7.7 seconds vs. 0.88 seconds for sayI. Note that even in this case, network latency far outweighs crypto operations.

It is interesting to compare the certificates fetched. sayI fetches the same (simple) group with 19 certificates plus 1 certificate per user; in contrast, Zhao uses 100 certificates, about 5 times as many. For sayI, after the first user is authenticated, at most 1 certificate (and 1 Internet round-trip latency) per user authentication is required. X.509, on the other hand, may still need to fetch more than a user certificate if the user is at a different organization. In other words, sayI group prefetch fetches all non-user certificates while X.509 path certification is only a partial fetch.

The comparison of bandwidth is even more striking. sayI's bandwidth consumed on this problem is about $\frac{1}{20}$ of X.509's. A factor of 4 is due to smaller, better encoded certificates and a factor of 5 is due to fewer certificates. Of course, X.509 can use ECC but sayI's encoding would still be significantly smaller due to fewer fields and implicit, as opposed to ASN.1's explicit (describing the encoded fields), encoding. X.509 needs more certificates because there are many paths to the same key.

What happens as the size of the graph grows? Using bidirectional search, the best algorithm for this problem, search time grows by $b^{d/2}$ where b is the branching factor and d is the distance between the two nodes [70]. Clearly as the graph grows, either b or d needs to increase resulting in an asymptotic increase in certification path.

Hence, search—bandwidth and latency—grows with the square root of the graph size. In sayI, bandwidth and latency is proportional to the group size which even for large groups is modest. (And on segmented groups is modest even for the largest groups).

8.5 Algorithm Complexity

Based on Table 2, the CPU time is crypto-computation bound for uncached certificates. Let the time for certificate validation be v. In the group tree, let the number of group certificates plus key certificates be N and the total number of bytes of the certificates be B. The time complexity for group prefetch is $O(N \cdot v + B)$.

During the lookup phase, the chain is built from group head to just before the user certificate. We use P to denote the number of bytes in the certificate chain. So the time complexity for look up phase is $O(P + v)$ for the segmented case.

Thus the cost is linear in the size of the input plus the output, plus the cost of a verification each time a certificate is brought in the cache.

8.6 Security

sayI provides security as stated in Section §1 by allowing relying party to choose whom to trust. We also require groups to be well formed, defined as

- Keys can only be used from ancestor nodes and
- Member names cannot be replicated along different paths in the group tree.

The second requirement prevents DigiNotar-like attacks.

Since the group is prefetched, and any wildcards are contained in the prefetched certificates, the group can be analyzed after prefetch to ensure the above conditions. Of course, this means that groups might be rejected due to errors in their construction. To avoid problems, sayI uses a two-stage approach to group prefetch. First, the new version of the group is fetched and checked. If it passes, it replaces the old version of the group. This can be done by the expediency of creating a new version of the intermediaries, all signed by the relying party.

9 Conclusion

An authentication infrastructure needs a strong trust model to enable a relying party to control the risk of accidentally or adversarially induced authentication failures. Hence, all modern PKIs support strong trust models.

However, the cost for strong trust models has been high in terms of latency, bandwidth, and CPU time. On the Internet, where roundtrip latencies can be over a second, extra round trips impact the user experience. Bandwidth costs make it less attractive for organizations to share authentication databases and more expensive to use them. CPU time adds to latency and costs.

sayI's use of groups for authentication provides high security. Its group construction specifies the TTPs that are relied upon, implementing a strong trust model. Its groups also enable TTP choice—and therefore cost—to be determined depending on the purpose of the authentication. Thus very important groups can have high reliable authentication while less important groups can use less reliable, but more cost effective, authentication. It prevents attacks such as the ones using DigiNotar and Verisign. It prevents naming attacks, which are often used in phishing. The groups, which are constructed by the relying party can also be used for authorization.

sayI groups are extremely efficient. Its groups

- enable shorter path lengths, reducing the number of roundtrips, and hence latency and
- enable group prefetch which can be done before an authentication.

After a group prefetch, at most one round trip latency is needed for an authentication.

sayI avoids unproductive certificates, reducing bandwidth and latency while increasing computational efficiency. A sayI authentication requires information which is proportional to the number of organizations contributing authentication information.

sayI shows substantial speedup even for small groups relative to X.509. A 160 organization PKI showed 8.75 fold speed-up and a 20 fold reduction in bandwidth. sayI's advantage increases with increasing group size, as X.509 needs to do a search over increasingly larger graphs which contain mostly unproductive certificates.

We plan to release sayI as open source. In Appendix §A we describe our plans to further sayI deployment. Future work includes integration of the variations described in §B and integration with both general purpose services, such as the web, and specific services.

10 References

[1] Abadi, M. On SDSI's linked local name spaces. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop* (1997), IEEE Computer Society Press.

[2] Adams, C., Burmester, M., Desmedt, Y., Reiter, M., and Zimmermann, P. Which PKI (public key infrastructure) is the right one? (panel session). In *Proceedings of the 7th ACM conference on Computer and communications security* (New York, NY, USA, 2000), CCS '00, ACM, pp. 98–101.

[3] Adams, C., and Lloyd, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*, 2nd ed. Addison-Wesley, 2002.

[4] Ajmani, S., Clarke, D. E., Moh, C.-H., and Richman, S. Conchord: Cooperative SDSI certificate storage and name resolution. In *IPTPS* (2002), pp. 141–154.

[5] Answers. The number of cities in the world. `http://wiki.answers.com/Q/Total_number_of_cities_in_the_world`, 2009.

[6] Answers. The number of hospitals in the world. `http://wiki.answers.com/Q/How_many_hospitals_are_there_in_the_world`, 2011.

[7] Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. DNS Security Introduction and Requirements, March 2005.

[8] Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. Resource Records for the DNS Security Extensions, Mar. 2005.

[9] Atkins, D., Stallings, W., and Zimmermann, P. RFC 1991: PGP message exchange formats, Aug. 1996. Status: INFORMATIONAL.

[10] augustShell. August schell awarded 5-year, $37m agency-wide contract to supply select red hat products to the department of defense. http://www.augustschell.com/news/august-schell-awarded-5-year-37m-agency-wide-contract-supply-select-red-hat-products-departme-0, 2011.

[11] Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. Recommendation for key management—part 1: General (revised), Mar. 2007.

[12] Bauer, L., Garriss, S., and Reiter, M. K. Distributed proving in access-control systems. In *IEEE Symposium on Security and Privacy* (2005), pp. 81–95.

[13] Bertino, E., Ferrari, E., and Squicciarini, A. Trust negotiations: Concepts, systems, and languages. *Computing in Science and Engg. 6*, 4 (2004), 27–34.

[14] Birrell, A., Lampson, B. W., Needham, R. M., and Schroeder, M. D. A global authentication service without global trust. In *IEEE Symposium on Security and Privacy* (1986), pp. 223–230.

[15] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A. RFC 2704: The KeyNote trust-management system version 2, Sept. 1999.

[16] Blaze, M., Feigenbaum, J., and Keromytis, A. D. The role of trust management in distributed systems security. In *Secure Internet Programming, Security Issues for Mobile and Distributed Objects* (1999), J. Vitek and C. D. Jensen, Eds., vol. 1603 of *Lecture Notes in Computer Science*, Springer, pp. 185–210.

[17] Blaze, M., Feigenbaum, J., and Lacy, J. Decentralized trust management. In *Proc. IEEE Symp. Security and Privacy* (1996).

[18] Blaze, M., Feigenbaum, J., and Strauss, M. Compliance checking in the PolicyMaker trust management system. In *Financial Cryptography* (Feb. 11 1998).

[19] Blaze, M., Ioannidis, J., and Keromytis, A. D. Experience with the KeyNote trust management system: Applications and future directions. In *International Conference on Trust Management (iTrust)* (2003), P. Nixon and S. Terzis, Eds., vol. 2692 of *Lecture Notes in Computer Science*, Springer, pp. 284–300.

[20] Bonatti, P., and Samarati, P. Regulating service access and information release on the web. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security* (New York, NY, USA, 2000), ACM Press, pp. 134–143.

[21] Bonneau, J., Herley, C., van Oorschot, P. C., and Stajano, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE Symp. Security and Privacy* (2012), pp. 553–567.

[22] Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., and Rivest, R. L. Certificate chain discovery in SPKI/SDSI. *J. Comput. Secur. 9*, 4 (2001), 285–322.

[23] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, May 2008.

[24] Cooper, M., Dzambasow, Y., Hesse, P., Joseph, S., and Nicholas, R. RFC 4158: Internet x.509 public key infrastructure: Certification path building, Sept. 2005.

[25] Couvering, A. V. How many top-level domains are there? `http://www.mindsandmachines.com/2009/03/how-many-top-level-domains-are-there/`, 2009.

[26] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (Nov. 1976), 644–654.

[27] Elley, Y., Anderson, A., Hanna, S., Mullan, S., Perlman, R., and Proctor, S. Building certification paths: Forward vs. reverse. In *Proc. of the Symp. on Network and Distributed Systems Security (NDSS)* (San Diego, CA, 2001), Internet Society.

[28] Ellison, C. Establishing identity without certification authorities. In *Proceedings of the sixth annual USENIX Security Symposium, focusing on applications of cryptography* (San Jose, California, 1996), USENIX, Ed., USENIX, pp. 67–76.

[29] Ellison, C., and Schneier, B. Ten risks of PKI: What you're not being told about Public Key Infrastructure. *Computer Security Journal 16*, 1 (2000), 1–7.

[30] Ellison, C. M. *RFC 2692: SPKI requirements*. The Internet Society, Sept. 1999.

[31] Ellison, C. M., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T. RFC 2693: SPKI certificate theory, Sept. 1999.

[32] Farrell, S., Housley, R., and Turner, S. An Internet attribute certificate profile for authorization, Jan. 2010.

[33] Ferraiolo, D. F., and Kuhn, R. Role based access control. In *15th National Computer Security Conference* (Baltimore, MD, 1992), pp. 554–563.

[34] Forget, A., Chiasson, S., van Oorschot, P. C., and Biddle, R. Improving text passwords through persuasion. In *Proceedings of the 4th symposium on Usable privacy and security* (New York, NY, USA, 2008), SOUPS '08, ACM, pp. 1–12.

[35] Gligor, V. D., Luan, S.-W., and Pato, J. N. On inter-realm authentication in large distributed systems. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 1992), SP '92, IEEE Computer Society, pp. 2–.

[36] Gutmann, P. PKI: It's not dead, just resting. *IEEE Computer 35*, 8 (2002), 41–49.

[37] Gutmann, P. How to build a PKI that works. `http://www.cs.auckland.ac.nz/~pgut001/pubs/howto.pdf`, 2003.

[38] Gutmann, P. How to build a PKI that works. In *3rd PKI workshop* (2004). Invited talk.

[39] Halpern, J. Y., and van der Meyden, R. A logic for SDSI's linked local name spaces. *Journal of Computer Security* (Jan. 28 2000).

[40] Herzberg, A., Mass, Y., Michaeli, J., Ravid, Y., and Naor, D. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)* (Washington, DC, USA, 2000), IEEE Computer Society, p. 2.

[41] Infoplease. Number of U.S. colleges and universities. `http://www.infoplease.com/ipa/A0908742.html`, 2005.

[42] IOerror. Diginotar damage disclosure. `https://blog.torproject.org/blog/diginotar-damage-disclosure`, 2011.

[43] iso. ISO/IEC 9594-8 information technology—open systems interconnection—the directory: Authentication framework, 1993. also published as ITU-T X.509 Recommendation.

[44] Jim, T. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P 2001)* (Washington, DC, USA, 2001), IEEE Computer Society, p. 106.

[45] Josefsson, S. Storing Certificates in the Domain Name System (DNS), March 2006.

[46] Kaufman, C., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World*. Prentice-Hall, 2002.

[47] Kohl, J. T., Neuman, B. C., and Ts'o, T. Y. The evolution of the Kerberos authentication service. In *Proceedings of the Spring EurOpen'91 Conference* (Tromso, 1991).

[48] Kohnfelder, L. M. Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman, May 1978.

[49] Labnol. Total domain name registrations. `http://www.labnol.org/internet/total-web-domain-names/18395/`, 2010.

[50] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computing Systems (TOCS) 10*, 4 (1992), 265–310.

[51] Li, N. Local names in SPKI/SDSI. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop* (2000), IEEE Computer Society Press.

[52] Lloyd, S. Understanding certification path construction, Sept. 2002.

[53] McKenzie, P. Falsehoods programmers believe about names. `http://goo.gl/eIfr`, 2010.

[54] Microsoft. Erroneous verisign-issued digital certificates pose spoofing hazard. `http://technet.microsoft.com/en-us/security/bulletin/ms01-017`, Mar. 22 2001. Microsoft Security Bulletin MS01-017.

[55] Miller, S. P., Neuman, B. C., Schiller, J. I., and Saltzer, J. H. Kerberos authentication and authorization system. Tech. rep., MIT, 1987.

[56] NEEDHAM, R. M., AND SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *CACM: Communications of the ACM 21* (1978).

[57] RECORDON, D., AND REED, D. Openid 2.0: a platform for user-centric identity management. In *DIM '06: Proceedings of the second ACM workshop on Digital identity management* (New York, NY, USA, 2006), ACM, pp. 11–16.

[58] REITER, M. K., AND STUBBLEBINE, S. G. Authentication metric analysis and design. *ACM Transactions on Information and System Security (TISSEC) 2*, 2 (1999), 138–158.

[59] RIVEST, R., SHAMIR, A., AND ADLEMAN, L. On digital signatures and public key cryptosystems. *Communications of the ACM (CACM) 21* (1978), 120–126.

[60] RIVEST, R. L. Can we eliminate certificate revocations lists? In *Financial Cryptography* (1998), pp. 178–183.

[61] RIVEST, R. L., AND LAMPSON, B. SDSI — a simple distributed security infrastucture. Tech. rep., MIT, Apr. 1996.

[62] SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. Role-based access control models. *IEEE Computer 29*, 2 (1996), 38–47.

[63] SEAMONS, K., WINSLETT, M., YU, T., SMITH, B., CHILD, E., JACOBSON, J., MILLS, H., AND YU, L. Requirements for policy languages for trust negotiation. In *Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)* (2002).

[64] SEARCHENGINEKNOWLEDGE. Will domain name length affect my search engine rankings? `http://www.searchengineknowledge.com/domains/length.php`, 2006.

[65] SOGHOIAN, C., AND STAMM, S. Certified lies: detecting and defeating government interception attacks against SSL. In *Proceedings of the 15th international conference on Financial Cryptography and Data Security* (Berlin, Heidelberg, 2012), FC'11, Springer-Verlag, pp. 250–259.

[66] STOBERT, E., FORGET, A., CHIASSON, S., VAN OORSCHOT, P. C., AND BIDDLE, R. Exploring usability effects of increasing security in click-based graphical passwords. In *Proceedings of the 26th Annual Computer Security Applications Conference* (New York, NY, USA, 2010), ACSAC '10, ACM, pp. 79–88.

[67] SUN, S.-T., POSPISIL, E., MUSLUKHOV, I., DINDAR, N., HAWKEY, K., AND BEZNOSOV, K. What makes users refuse web single sign-on?: an empirical investigation of openid. In *Proceedings of the Seventh Symposium on Usable Privacy and Security* (New York, NY, USA, 2011), SOUPS '11, ACM, pp. 4:1–4:20.

[68] UNIV.CC. Universities worldwide except united states. `http://univ.cc/world.php`, 2011.

[69] WIKIPEDIA. The number of banks in the world. `http://en.wikipedia.org/wiki/Lists_of_banks`, 2011.

[70] WIKIPEDIA. Bidirectional search. `http://en.wikipedia.org/wiki/Bidirectional_search`, 2012.

[71] WIKIPEDIA. Diginotar. `http://en.wikipedia.org/wiki/DigiNotar`, 2012.

[72] WINSLETT, M., YU, T., SEAMONS, K. E., HESS, A., JACOBSON, J., JARVIS, R., SMITH, B., AND YU, L. Negotiating trust on the web. *IEEE Internet Computing 6*, 6 (2002), 30–37.

[73] YLONEN, T. SSH—secure login connections over the Internet. In *Proc. of the USENIX Security Symposium* (San Jose, California, 1996), pp. 37–42.

[74] ZHAO, M. Performance Evaluation of Distributed Security Protocols Using Discrete Event Simulation. Tech. Rep. TR2005-559, Dartmouth College, Computer Science, Hanover, NH, October 2005.

[75] ZHAO, M., AND SMITH, S. W. Modeling and evaluation of certification path discovery in the emerging global PKI. In *Public Key Infrastructure: EuroPKI 2006* (2006), Springer-Verlag LNCS.

APPENDIX

A Deployment

We plan to release sayI as open source software, so that anyone who wants to use it or study it can do so. Software availability is particularly useful to enable test drives and early deployments.

Public-key authentication enables a single key to be securely used to communicate with an arbitrary number of different services, avoiding a plethora of passwords. Or a variety of different pseudonyms can be used for different purposes—such as work and personal—and anonymity. Seeding of such software, once it is established on the first machine can be done by Near-Field Communication (NFC). Creating a certificate can be done by bringing one's public key to a publisher, along with a suitable photo identification, and asking for a key to be made. We note that financial institutions, governments, universities, and employers often provide paper credentials based on a very similar mechanism. Thus public-key authentication increases set up on the owner's devices it simplifies the per-server issue of pasword management.

We believe sayI will see its first adoption locally in organizations, growing bottom up (because it is useful) rather than top down (by fiat). If successful, it can then federate to the Internet. So our immediate concern is how to make sayI useful within the enterprise.

We have begun to integrate sayI with secure networking software which provides authentication, encryption, and authorization. Using this network software in an application would both protect its communication and integrate with sayI.

We plan to then integrate this with web services. We services typically integrate a number of authentication mechanisms including X.509, kerberos, and password. We also intend to integrate it with applications—as in the mobile space—and services within an organization in which sayI's strong trust model is an easy fit.

sayI can be deployed within the enterprise: deployment does not assume the existence of a global infrastructure. An enterprise, or part of an enterprise, can implement a sayI service, and deploy application and tools which use the service. As other organizations deploy, the group mechanism can be used to federate across organizations.

B Variations

sayI is a very simple scheme. It is possible to generalize it in a number of directions to make it more flexible and useful. For example, to provide

- Segmented group names
- Hierarchical name space
- Host names
- Multiple signers for segmented certificates

Segmented group names The first variation is a generalization of segmented groups which allows organizations (publishers) to decide which of their users are in a specified segmented group (they can do this in the current scheme only for simple groups). For example, there may be a working group which spans organizations, and each organization determines which of its employees are members of the working group.

In sayI, a segmented certificate does not contain any group certificate names. By allowing segmented group certificate names, such as name@g.*.com in a segmented certificate, each .com can efficiently provide a group certificate determining group members from their organization.

Hierarchical name space The second variation allows a hierarchical name space at a publisher. Thus, one could have employees.example.com, executive.employees.example.com, and customers.example.com. This enables the publishers to create attributes (such as 'employee') which can be easily used to construct groups.

User and host names sayI can be extended to support host names, in addition to user names. We intend to provide a certificate to represent a host.

Multiple signers The current scheme is for a single signer based on the name of the publisher. For simple groups this is not an issue, but for segmented groups this may be a problem: in the case of *.com, this would mean that every .com key binding would be signed by the same intermediary. This is not good for security; more signers would have to compete for customers, lower prices, and would encourage signers to increase security because if they had security failures they might lose their signing privileges.

This generalization would enable segmented key names to specify a set of signers.