Layering a Public Key Distribution Service Over Secure DNS: "Everybody Comes to RIKS"

John P. Jones, Daniel F. Berger, Chinya V. Ravishankar
Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA, 92521
{jjones,dberger,ravi}@cs.ucr.edu

Technical Report Number UCR-CS-2005-03059
March 18, 2005

Abstract

We present the Internet Key Service (IKS), a distributed architecture for authenticated distribution of public keys, layered on Secure DNS (DNSSEC). Clients wishing to manage or retrieve public keys use DNSSEC to securely discover the identities of the relevant IKS key registration and distribution servers, and send their key lookup / management requests directly to these servers using a special-purpose protocol. Clients can validate and authenticate keys retrieved from IKS servers using key commitments published in DNSSEC. Applications can use the public keys obtained from IKS to authenticate each other and establish secure end-to-end communications.

A feature of our approach is that IKS derives its authentication authority from the authority DNS domains have over names. The IKS architecture is loosely coupled with DNS to minimize the overhead on DNS servers. We also present the Riverside Internet Key Server (RIKS), a prototype IKS implementation.

1 Introduction

Digital communication has become pervasive, but there are few guarantees that such communications are secure and private. Indeed, security and privacy threats, long seen as hypothetical, are already real. In 2004, for the first time ever, an arrest was publicly acknowledged as having resulted from passive email monitoring [2].

Though cryptographic techniques exist that can address these concerns, no infrastructure is available to facilitate use by a variety of applications, and across the Internet. Cryptography has been most successfully deployed in protocols where a clear client-server relationship exists, such as Secure Socket Layer/Transport Layer Security (SSL/TLS) [25, 15], and Secure Shell (SSH) [68]. While proposals exist for securing less hierarchical applications, such as the Privacy Enhanced Mail (PEM) [41], and Secure Multimedia Internet Mail Extensions (S/MIME) [37] specifications for securing email, they have not been widely adopted.

1.1 The Internet Key Service

In this paper, we focus on a capability crucial to pervasive adoption of cryptography: *simple, scalable, authenticated public key distribution.* We present the Internet Key Service (IKS), a practical and deployable architecture for providing application-independent public key distribution layered on top of Secure DNS (DNSSEC). Our approach is flexible and extensible, it can store and serve a variety of key types, supporting a variety of applications.

The Internet Key Service differs substantially from earlier attempts to provide authenticated public key distribution. It bases its key-authentication authority on DNS's authority to manage Internet names. This is a significant feature, since all Internet names are ultimately DNS names, and DNS' hierarchical namespace is the Internet-wide standard for representing who has control over which names. IKS is also loosely coupled to DNS, so that it can provide specialized key distribution protocols without changes to or significant overhead on DNS.

The remainder of this document is organized as follows: Section 2 provides necessary background; the fundamentals of the domain name system (DNS), and a brief look at efforts to secure DNS. Section 3 briefly surveys related work, including previous proposals for key distribution. Section 4 gives a high-level view of our proposed solution, exploring the design philosophy and constraints. Section 5 delves deeper into the protocol design — detailing key query and registration, the server

location mechanism, message marshaling and transport, and authentication during key registration. Section 6 presents the Riverside Internet Key Server (RIKS) our proof-of-concept implementation of the IKS. Finally, Section 7 summarizes our findings, and points out avenues for future exploration.

2 Background

We assume familiarity with asymmetric (public key) cryptography, digital signatures and one-way hash functions. Readers unfamiliar with these topics are encouraged to consult a cryptography text, such as [56].

2.1 Key Authentication

Key authentication is the process of validating the binding of a cryptographic key to a named entity. Public key cryptosystems simplify, but do not solve, the problem of key distribution, since public keys must be authenticated to prevent impersonation and man-in-the-middle attacks. The most widely used approaches for solving the key authentication problem are the *certifying authority* model, exemplified by SSL/TLS, and the *web-of-trust* model, exemplified by Pretty Good Privacy (PGP).

Certifying Authorities
The certifying authority (CA) model assumes a small number of highly trusted individuals (or organizations) in the community. A new key is accompanied by an assertion (digital signature) from one of these trusted certifiers that the provided key is associated with the claimed identity. All participants wishing to verify keys signed by a certifying authority must somehow authenticate its public key. In practice, a small set of so-called *root certificates*, which are public keys for various recognized certifying authorities, are typically loaded into the cryptographic application prior to use.

Webs Of Trust
The web-of-trust model, relies on peers to vouch for the validity and trustworthiness of other peers. An unfamiliar key is accompanied by affirmations (digital signatures) from a set of community members who assert that the provided key is associated with the claimed identity. A recipient accepts the key only upon receiving enough verifiable affirmations from individuals that he trusts. Researchers have examined the characteristics of these sorts of trust systems in different contexts [48, 36].

IKS follows the certifying authority model, the IKS server for a domain acts as a CA for that domain and its public key can be authenticated by checking its key commitment with the one published via DNSSEC.

2.2 The Domain Name System (DNS)

The Domain Name System (DNS) [45] is the most effective and widely-used mechanism for name registration and resolution on the Internet. It has become a critical component of the Internet infrastructure. As of 2001, DNS root servers were handling a peak load of over 5000 queries per second [11].

In DNS names are assigned from a hierarchical namespace in which organizations are granted control over a sub-tree rotted at the domain they have registered. The DNS top-level domains (e.g. *.com*, *.org*, *.edu*, *.us*, *.uk*, etc.) are administered by ICANN (Internet Corporation for Assigned Names and Numbers). Domain administrators run DNS servers to provide authoratitive answers to queries regarding the domain and to participate in resolving DNS queries for clients belonging to the domain.

Efforts to Secure DNS
Unfortunately, security was not a primary consideration during the design and implementation of DNS. Its security shortcomings have long been understood, and first discussed at length in [7, 64]. In [7], Bellovin described attacks made possible by a combination of poor authentication and authorization techniques and inherent limitations of the DNS, and concluded that a healthy dose of skepticism and cryptographic authentication help mitigate the threats discussed.

The Internet Engineering Task Force (IETF) launched the DNSSEC effort in 1993 to secure DNS against these and other attacks. Presently, the DNSSEC working group proposal is nearing operational readiness, bringing with it the promise of a trustworthy name service. We use DNSSEC as a foundation for our key-distribution architecture.

2.3 DNSSEC Overview

DNSSEC is a collection of proposals for securing the data stored in DNS. Using cryptographic techniques, queries and associated responses can be strongly authenticated by the server and requesting client respectively, greatly reducing the potential for abuse present in the current DNS. An IETF draft by Atkins and Austein [5] enumerates the threats DNSSEC is intended to guard against. We focus here on the portions of DNSSEC relevant to our work. A detailed overview appears in [3].

Zone Signing
DNSSEC offers two fundamental improvements over traditional DNS: data origin authentication and data integrity verification. A DNSSEC-enabled DNS server responsible for a given domain (referred to as a *zone*) cryptographically signs the resource records comprising the zone with a public/private key pair bound to that zone,

and delivers those signatures to querying clients.

Resource Record SIGnatures are stored in a new DNS record type, *RRSIG*, which contains a cryptographic signature that authenticates a specific named set of resource records (termed an *RRSet*) for a specific duration. Each named resource in a secured DNS zone will have at least one associated *RRSIG* record.

DNSSEC responds to a query from a DNSSEC-enabled client with the DNS record for the name specified, along with the associated *RRSIG* record. The client obtains the public key associated with the zone containing the retrieved record and verifies the provided signature. If the signature is valid, the client can trust that the response was provided by the authoritative source.

Key Distribution in DNSSEC

To verify signatures, the client must have been either statically configured with the public key for the queried zone (the zone key), or must be able to somehow obtain and authenticate it. To facilitate distribution of these public keys, DNSSEC defines a *DNSKEY* resource record type. Interestingly, the *KEY* resource record, the predecessor to the *DNSKEY* type, was originally intended as a general purpose public key distribution mechanism [17] but was subsequently restricted to holding only DNSSEC keys [42] for reasons discussed in Section 2.4.

A DNS client can query for a zone key in the same way it queries for any other DNS record type. To authenticate the retrieved key, the *DNSKEY* record must be signed by a key which the client has previously authenticated, typically the key of the parent domain. As with SSL/TLS, clients are expected to be preconfigured with a small set of trusted keys. By recursively requesting keys while moving up the DNS name hierarchy, the client will either reach a trusted key, or exhaust the name space without reaching such a key, causing the key authentication attempt to fail. (While this description is conceptually sufficient, it is not technically precise. Full details are in [31].)

DNSSEC Implementation Status

DNSSEC has recently matured into an implementable system, and has been deployed in the medium scale. Its signing hierarchy has been revised based on this operational experience [31]. Some documents concerning operational and security concerns have also been written and published in [16].

An IETF draft exists that updates RFC 2535 and details the DNS protocol changes required to support DNSSEC [4]. A DNSSEC deployment working group has been formed with support of NIST and ICANN. In April of 2004, invitation was sent [13] to interested parties to build a road map for DNSSEC deployment. Consensus is growing that DNSSEC is largely ready for deployment, and that 2005 may see the beginnings of widespread adoption.

2.4 Barriers to Distributing Keys in DNS

Unfortunately, DNSSEC does not fully solve the authenticated key distribution problem. As observed in Section 2.3, the *KEY* record defined by DNSSEC was intended to store keys of many sorts, including end-user application keys. This decision was explicitly reversed in RFC 3445 for three primary reasons, scalability concerns, query interface limitations, and administrative authority mismatches [42].

Scalability

The original DNS RFC's proposal to use DNS to house per-user information clearly did not anticipate that the growth in Internet user population would far surpass the growth in DNS-registered host systems. Estimates for 2004 suggest about 945 million users [12], compared with 230 million hosts [38].

Adding DNSSEC signature records to a zone increases the size of the zone data by a factor of 8 or 9 [27], and adding per-user keys and their signatures would further increase the size of the zone data.

Finally, from a network perspective, DNS has been designed and optimized for very small query/response exchanges. Returning key data (and associated signatures) in DNS responses is expected to significantly increase network load, as would zone transfers between primary and secondary servers.

Query Interface

The second reason for reclaiming the *KEY* record was a mismatch between the resolver query interface and the requirements of an application seeking a particular key.

Different types of keys stored in *KEY* records were to be differentiated by subtype, so that a single named entity may have multiple key records, each storing a different type of key. Unfortunately, the DNS resolver interface does not support query by subtype, so the client was forced to retrieve all key records present for the named entity before sifting through the results for the "right one." Since DNSSEC internally requires keys retrieved from foreign servers, this affected not only applications but the efficiency of the name service itself.

Administrative Authority

A third significant issue is that the administrative model for DNS does not match the requirements for managing end-user keys.

DNS data tends to change slowly and is under the control of a domain administrator. Allowing users some level of direct control over their keys, (mediated updates of DNS key records, for example), would violate the existing administrative model. Supporting dynamic DNS

update in the context of DNSSEC is difficult in general; RFC 3007 [67] discusses it in detail and several researchers have contributed solutions [23, 65].

3 Related Work

Here we briefly survey previous approaches to key distribution, from application-specific to general approaches.

3.1 In-Band Key Transmission

A common approach to key distribution is to relegate it to the communication protocol. The SSH and SSL/TLS protocols both transmit the necessary keying information during connection setup, but they use different methods for authenticating the received key.

Secure Shell (SSH)
SSH performs initial key authentication by asking the user to certify the key-host association. A hash of the public key (a *key fingerprint*) is then stored locally. Subsequent connections use this stored fingerprint to authenticate known hosts without further user intervention.

This approach assumes that the end-user will know the appropriate key fingerprint during initial connection setup. While it limits the window for a successful attack to the initial connection, it does not eliminate the threat.

This is generally an acceptable level of risk mitigation when trust relationships are fairly static (users tend to repeatedly connect to the same small set of hosts). However, this sort of manual, out-of-band, process is not viable when the trust relationships are more dynamic (i.e. end-user to end-user communication).

Secure Socket Layer (SSL/TLS)
SSL/TLS uses the certifying authority model; the connecting client is provided with the server's certificate, signed by one or more certifying authorities. Clients (such as web browsers) are typically preconfigured with a number of "root certificates," which are public keys of trusted certifying authorities. If the certificate provided by the server has been signed by a statically known certifying authority, the connection is established without user intervention. In principle, the SSL/TLS model permits a hierarchy of intermediate signatories, but this feature is rarely used in practice.

3.2 Dedicated Key Distribution Services

Another approach to key distribution is to deploy a dedicated distributed service to handle the registration and query of public keys. Several proposals have been made, mainly differing on how keys are named and bound to individuals, how clients verify the responses from the service and how the servers distribute the responsibility of key distribution.

PGP/GPG
The MIT Pretty Good Privacy (PGP) [70] key server hosted at *http://pgp.mit.edu* [35] is perhaps the best known dedicated key distribution service. PGP and Gnu Privacy Guard (GPG) [29] support locating and publishing keys via the PGP key-server.

SPKI
Rivest and Lampson have proposed the Simple Distributed Security Infrastructure (SDSI) [50], an integrated solution to authentication and authorization based on capabilities. This proposal has subsequently been incorporated into the IETF's Simple Public Key Infrastructure (SPKI) working group's proposal [22]. In SPKI certificates bind specific authorizations to keys. Names in SPKI can be assigned to keys and can either exist in a local namespace or rooted in a global namespace such as the DNS namespace.

SPKI requres that applications switch from the current model of seperating authentication and authorization to the joint authentication/authorization model of SPKI; with this, the burden of key distribution is pushed onto the applications making authorization choices about the resources they control. The burden is upon the client applications to register or obtain keys and their bound certificates prior to requesting resources. These requirements have significantly impaired the fuller development and implementation of SPKI. In recent years no substantial anvances have been made by the SPKI working group.

COCA
Zhou, Shneider and Reese [69] have proposed COCA, a distributed Certificate Authority based system that seeks to improve security by distributing the role of a trusted CA over a collection of servers. While the implementation of the CA is distributed to provide security the logical functionality of the CA is flat, and does not utilize a hierarchical namespace. The issue of user key-registration in a large, distributed authority system, such as the Internet, is not addressed.

Scalable Key Distribution Hierarchy
McDaniel and Jamin [43] describe a scheme for a hierarchical set of certificate servers similar in capabilities to the certification authority requirements outlined in the Privacy Enhanced Email (PEM) specification. The authors describe their design, which is based on a well-meshed trust graph and is not directly related to the DNS namespace, and examine its behavior under hypothesized load. They do not discuss operational issues such as off-line signing keys and heterogeneous keys.

3.3 Distribution by Directory Service

Rather than inventing a dedicated key distribution service, many proposals have chosen to incorporate key dis-

tribution into existing directory services.

X.500, LDAP

ISO and CCITT maintain a set of recommendations for building distributed replicable directory services under the umbrella name X.500 [39]. Clients typically access these directories using the "lightweight directory access protocol" as defined in RFC 1487.

Configuration and maintenance of an X.500/LDAP directory is perceived as difficult and complex. Although standard schemas exist for a wide variety of object types, including X.509 certificates [9], implementors often ignore or are unaware of these standards. As a result, X.500 implementations meet the directory-related requirements of the owning organization, but may not be interoperable across administrative domains. Additionally, LDAP is often not permitted across network boundaries, resulting in disconnected islands of information.

Perhaps most damagingly, X.500 complexity is exposed beyond implementor and administrator. Users searching an X.500/LDAP directory must specify values for unfamiliar terms such as "Search Base" and "Search Scope." Correct values are required to obtain useful search results, and most tools provide little guidance. Such factors have prevented X.500/LDAP from becoming a practical Internet-wide key distribution tool.

DNS

Efforts have been made to standardize storing keys of various types [18, 21, 19, 20, 55] and X.509 certificates [21] within DNS. Recently, Yahoo! has submitted an IETF draft [14] that describes using DNS to distribute public keys for authenticating email delivery.

The FreeS/WAN Project [24], an open source IPSec implementation, includes support for "opportunistic encryption;" by automatically retrieving host keys from DNS, end-to-end IPSec encryption could be setup without user intervention. While the FreeS/WAN solution made retrieving keys from DNS invisible, it did not address key publication.

In [26], Galvin presented an overview of DNSSEC and briefly discussed the potential for using DNSSEC to distribute end-user public keys [26]. In a subsequent RFC [6] the author describes a DNS key exchange record type with semantics similar to DNS mail exchange records. Though focus was on IPSec, the author briefly describes the potential for this mechanism to delegate authority to a more general key distribution center.

3.4 Identity Based Encryption (IBE)

Identity-based cryptography addresses the authenticated key distribution problem by allowing a sender to directly derive a public key from a recipient's name. Each recipient obtains its secret key from a trusted key generator, which generates this private key from the receiver's name, public system parameters, and a system secret. Since the key generator knows all private keys, this system implies key escrow. It also requires secure and trusted access to the system parameters, which a sender needs to construct a recipient's public key.

The work in [59] describes a domain-level key-distribution scheme using the identity based encryption scheme of Boneh and Franklin [10]. Since identity based encryption implies key escrow, this work side-steps the problem of key registration. Requiring key escrow, however, is often undesirable or unacceptable, and even careful implementations carry significant risks [1].

4 The Internet Key Service

We now introduce the Internet Key Service (IKS), a dedicated key registration and distribution service capable of publishing keys bound to DNS names. In this section, we provide the design rationale and guidelines for IKS. In Section 5, we present an overview of the IKS protocol. In Section 6, we describe RIKS, a prototype implementation of IKS we have developed to evaluate IKS.

DNS's role in naming is a fundamental aspect of the Internet, so any mechanism to bind keys to named entities on the Internet must derive its authority from DNS. All names are ultimately DNS names, and all authority to bind names and derivatives to objects must ultimately derive from the authority DNS has over names. This basic observation drives our design of IKS.

Previous mechanisms for authenticated key distribution in the Internet have failed either because they were unable to root their authentication mechanisms in DNS, or because they attempted to use DNS directly to manage keys, causing the problems discussed in Section 2.4.

The Internet Key Service overcomes these deficiencies by using DNSSEC to effectively transfer DNS's authority over name resolution to a specialized service designed to meet the requirements of authenticated key distribution. Prior to DNSSEC becoming available, no natural secure delegation mechanism existed for the Internet. The imminent deployment of DNSSEC has made the key distribution problem tractable.

4.1 IKS Overview

The IKS model allows public keys to be registered and retrieved for any entity that can be assigned a DNS name, such as a host, user, or service port. These keys are stored in and managed by IKS servers, which may be discovered securely through DNSSEC. The IKS server responsible for a domain `D` accepts key registration requests for keys being bound to names contained in `D`, and provides authentic responses to lookup queries for names contained in `D`.

A client wishing to register or retrieve a public key

for an entity with a DNS name first uses DNSSEC to discover and verify the identity of the IKS key server responsible for the entity's domain. It then sends the key-registration or lookup request to this server using the IKS protocol, and authenticates the server's responses using key commitments placed in DNS, and authenticated by DNSSEC's zone signatures. Such authentication guarantees that the keys retrieved from IKS are registered to the indicated name, allowing these keys to be used for establishing secure communications channels or validating digital signatures. Figure 1 shows high-level information flows in the Internet Key Service architecture.

Since each domain administers its own IKS server (or delegates this task to a trusted organization) there is no communications between IKS servers of different domains. This is substantially different from the structure of DNS. This property prevents issues with bottlenecks that plague DNS scalability. This is possible since IKS can rely on DNS to provide service discovery of the appropriate IKS server for clients.

4.2 Design Requirements/Constraints

We now discuss some requirements for an authenticated and secure key distribution service for the Internet.

Authority: The service's authority to bind DNS names to keys must derive from DNS's naming authority.

Scalability: The service must not result in a substantial increase in load on DNS.

Compatibility: Changes to DNS must be avoided. In particular, the service must not create new resource record types, as such a change requires additional *per-client* software deployment and reconfiguration.

Flexibility: Domain- and application- specific mechanisms for authenticating users during key registration must be supported. Some services and domains may be amenable to on-line authentication, while others may require out-of-band authentication.

Efficiency: The protocol should simplify the common case, and allow client applications to perform authentication without user intervention. The number of required messages must be small for performance and reliability reasons.

Generality: The query and registration mechanisms must be application-independent. It must provide a generic key service useful to any application. In keeping with the end-to-end principle, we should not care what the application is.

Security: The server response must serve as validation of all messages in a key registration or lookup protocol. Further, to protect the private signing keys, key registration and query servers should have limited contact with the system's key-signing keys.

Consistency: The key-authentication guarantees expected by an end-user must be consistent with the guarantees actually provided by the system. Mismatches between the system model and user model [46] can be disastrous.

IKS attempts to balance these requirements.

4.3 IKS Architecture

Our use of DNSSEC for *authenticated delegation* provides for both a secure hand-off between DNS and IKS servers, and a mechanism to authenticate server responses.

Each participating DNS domain delegates to one or more IKS servers the responsibility for handling key query and registration requests. This delegation is accomplished by adding resource records to the DNS zone for the domain being delegated, and is under the domain administrator's direct control.

Clients wishing to lookup or register keys for a name contained in a given domain learn of this delegation through authenticated DNS queries, via DNSSEC. Clients communicate with the IKS servers via HTTP, retrieving a URL to perform queries, and sending SOAP datagrams [30] to perform registration.

To allow clients to validate responses, IKS servers sign all keys returned with a named key-signing key (KSK), the public half of which is committed in DNSSEC. Clients can retrieve this commitment securely from DNSSEC, and use the KSK to verify IKS signatures on query results.

All registration acknowledgments, and query responses indicating query failure, are signed with a response-signing key (RSK). The RSK, like all keys, can be retrieved from the IKS and is signed by a KSK. This indirection allows the server to provide the client with authentic responses without risking exposure of KSKs. If an RSK is attacked the worst an adversary can do is convince the client that an entity has no published keys, or that an unfulfilled key registration request has been fulfilled. While this is a legitimate problem it is substantiably less problematic than an adversary obtaining a KSK, with which it can arbitrarilly substitute keys. The adversary's abilities with an RSK however are not much stronger than any adversary's ability to launch a DoS attack on an IKS server.

There is no implicit delegation in IKS, a domain that does not explicitly publish delegation records is choosing not to participate in IKS. DNSSEC will notify any requesting clients that IKS is not configured for the domain through an authenticated response indicating that no IKS servers have been designated.

Our scheme is conceptually decoupled from DNSSEC, relying only on the presence of a trustworthy name service, not any particular implementation. The

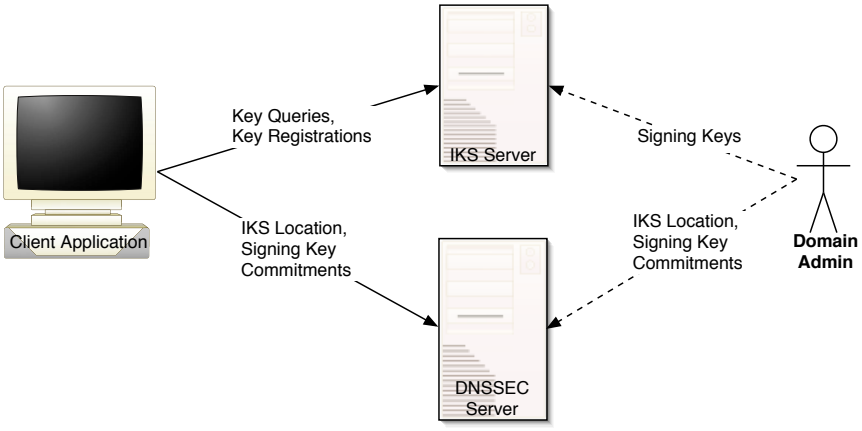

Figure 1: IKS Architecture: Naming and authentication authority is given by domain administrators to DNS and IKS. The client uses them to resolve names securely.

impact of changes to DNSSEC standards on our proposal is likely to be minimal. Further, unlike previous attempts to distribute keys via DNS(SEC), our proposal does not suffer from any of the three pitfalls enumerated in Section 2.4: scalability, poor query-interface, and mismatch of administrative authority.

Scalability
By layering IKS on to the hierarchical structure of DNS, we decentralize the task of key distribution without impeding scalability. The additional data placed in DNS, consisting of delegation records and KSK commitments, is negligible and does not increase with the number of keys active in the system. The number of DNS requests needed to learn of the delegation and retrieve KSK commitments are comparable to the number for resolving other services (e.g., for publishing the location of a website or FTP server). The delegation mechanism permits simple weighted load balancing across an arbitrary set of IKS servers, enhancing system scalability. IKS permits domain administrators wide latitude in distributing the key management workload, so that each deployment site can adopt the approach that best fits its particular requirements.

Query Interface
IKS uses a specialized query and registration protocol which provides the appropriate level of expressiveness for key registration and key distribution. Clients can perform narrow searches, based on attributes such as key length, cryptographic algorithm, and key-container format, to discover keys suitable for their purposes.

Administrative Authority
Unlike proposals to distribute keys in DNS, IKS places minimal burden on DNS administrators, and does not cause rapid changes to DNS zone data. The delegation records for a given domain are expected to be static, and publication and revocation of key-signing keys are expected to be infrequent events, not driven by end-user behavior.

5 Protocol Overview

We introduce IKS by showing how to fetch and register a key corresponding to a name. We briefly discuss the issue of key revocation in IKS.

5.0.1 Key Lookup

A name `N = (E,D)` consists of two components: the *entity* part `E` that designates the user, host, or communication endpoint, and the *domain* part `D`, which is a DNS domain name. A name may be associated with one or more keys. A query specifies a name `N` and a set of key *selection criteria* `C`, and is processed as follows.

1. Issue a DNS query for the IKS server registered to handle queries for domain `D`. If the query succeeds, and the response can be authenticated via the DNSSEC mechanisms described in Section 2.3, send a key-query message to the IKS server specifying entity `N`.
   IKS will respond with metadata for all public keys registered for `N`, signed with a key-signing key (KSK), whose name is included in the response.
2. Validate the IKS response as follows:
   (a) Fetch the indicated KSK from the IKS server.
   (b) Request the commitment for this KSK from DNS. Validate this response using DNSSEC zone signatures.
   (c) Validate the KSK using this commitment.
   (d) Verify the IKS signature on the metadata for `N`'s keys using the KSK.

3. The key metadata is now processed at the client end, and metadata for keys matching the selection criteria `C` are extracted. These keys are fetched from IKS and verified using the appropriate KSK.

In the event that `N` does not have any keys published in IKS the query server will need to generate a signed negative response. This negative response is signed with a response-signing key that is registered in IKS so that the client may validate the signature without requiring the query server to risk exposure of the KSK by keeping it available on-line in order to sign these responses.

Trust Guarantees for IKS Responses

In accordance with our discussion in Section 4.2, we explicitly state the guarantee made by the Internet Key Service. A valid signature on a query response indicates the following.

1. The IKS server for domain `D` asserts that the key provided is bound to name `N` in domain `D`.
2. The administrators of domain `D` have verified, to the extent **they** see fit, that the key in question was registered in the IKS for domain `D` by the user or agent in control of name `N`.

A signed IKS response **makes no guarantees** about how the verification was performed during key registration. The client who receives a validated query response must decide what degree of trust to place in the mechanisms that are in place in domain `D`. Though IKS insulates the end-user from the complexities of the mechanics of key distribution, it is not intended to provide guarantees about the trustworthiness of individual domains, which ultimately hold contorl over names in that domain.

5.0.2 Key Registration

Given a target name `N = (E,D)` and a public key `K`, registration proceeds as follows:

1. Issue a DNS query for the IKS server registered to handle queries for domain `D`. If the query succeeds, and the response can be authenticated via the DNSSEC mechanisms described in Section 2.3, send a key-registration message to the IKS server specifying entity `N`, the key `K`, relevant metadata (permitted uses, expiration date, and so on), and authentication information.
2. If authentication succeeds, and the registration is authorized, the server returns a success message signed by a named response-signing key (RSK). The client can authenticate this RSK and the response as it does for key lookups.
3. If authentication or authorization fails, the IKS responds with a list of supported authentication methods, as guidance for the client.

In the event that the authentication fails, the registration server will generate a signed response indicating acceptable methods of authentication in order to guide the client in completing this transaction. As with negative query responses these authentication failure responses are signed with a response-signing key (RSK) rather than the KSK.

Authentication During Registration

The flexibility constraint (Section 4.2) requires that the registration server have wide latitude with respect to how to perform authentication when keys are registered. Flexibility, however, works against the efficiency constraint, which requires us to simplify the common case. Increasing flexibility can decrease security, and increase both complexity and the likelihood that human intervention will be needed.

To understand the challenges in designing authentication mechanisms for key registration, let us consider some authentication schemes that IKS may need to support.

- It is common to authenticate on the basis of a shared secret, typically a username and password. While weak by modern cryptographic standards, this mechanism has survived and is widely used.
- Variations on this theme abound, including one-time password schemes such as S/Key [33] and OTP [34], as well as two-factor schemes involving hardware devices such as RSA's SecurID [52]. These schemes often appear identical to username/password schemes from the system perspective.
- The coarsest authentication model we address is "proxy" authentication, in which a trusted, well-authenticated party is allowed to vouch for the identity of others. This may be a designated system administrator attesting for the credentials for the requesting client out of band, and authenticating himself to the system to authorize the request.

Our challenge is to permit these and other authentication schemes to be incorporated in a flexible, yet manageable, manner. We must be careful that our protocol will facilitate the building of clients that will operate correctly in different domains. We have standardized three authentication mechanisms for IKS.

Usename/Password: A client may authenticate to an IKS server using a shared secret, such as a username and password. A public RSA encryption key for IKS would be published in IKS. A client may use this key to securely send the the secret to the server in a key-registration request.

User Key Management: A client acting on behalf of a user `U` may authenticate its key-registration re-

quests by signing it with the private half of a special purpose key, the public half of which has been previously registered to `U` using IKS. The registration server can then verify the request against this key.

Third-Party Authentication: IKS supports other, possibly domain-specific, authentication methods, by allowing the authentication of registration requests based on signatures using other certifying keys, again published via IKS. The private half of these keys can be distributed to a number of third-party authentication servers, which can implement arbitrary authentication protocols and use signatures with these keys to validate their authentication with the IKS registration server. These external authentication protocols are outside of the scope of IKS.

In practice, it is likely that the Username/Password and User Key Management authentication methods will be useful and sufficient for the majority of IKS installations. The third-party authentication method provides an extensible approach for supporting arbitrary authentication procedures. For example, IKS usage for VoIP services is likely to require some form of key escrow to support legal wire-tap warrants by law enforcement agencies in the United States. Third-party authentication could be setup to require this key escrow prior to authenticating a VoIP key-registration request.

5.0.3 Key Revocation

When a key is registered in IKS an expiration time is optionally provided to allow for the gracefull degredation of old keys. In the exceptional case in which a key's security is compromised and it must be forceably revoked before the end of its intended lifetime IKS supports a simple mechanism for revoking the key. Before IKS revokes a published key the key's client must detect that the key has been compromised and submit an authenticated key revocation request to the IKS server. Machanisms for timely dissemination of key revocation information is both application specific, and a generally open problem outside the scope of IKS. Once a key revocation request is authenticated and accepted by the IKS server the key's IKS entry is updated and a signed key revocation response is prepared for responding to future queries for this key.

The timing requirements between the acceptance of a key revocation request and the publication of the signed key revocation response are open but implementations should seek to minimize this delay in order to mitigate the dangers presented by a compromised key. It is required that no subsequent non-revoked key certificates are published by IKS after a key has been revoked. IKS is designed so that implementations may severely limit the lifetime of key certificates in order to limit the effect of caching keys at clients. IKS servers using long lived key certificates avoid continued lookup queries from clients, who may cache authenticated responses for longer time periods, but are less able to quickly respond to key revocations.

5.1 Locating an IKS Server

A fundamental issue is how a client identifies the IKS server responsible for handling requests for a given domain. We have chosen to use the existing *SRV* record type, intended for service location [32].

DNS SRV Records

DNS *SRV* records are intended to allow clients to perform service discovery using DNS. As defined in RFC 1700 [49], a client locates a server for service `S` running a protocol `P` in domain `D`, through a DNS query for `_S._P.D`. The response includes a list of `(host, port)` pairs, along with a priority for each matching record and a weight used to distribute load across servers of the same priority.

Key Service and Protocol Names

To locate a server using DNS *SRV* records, a client must first know the service and protocol names. Since, in principle, key lookups may be handled by a different set of servers than registration requests, we will use two distinct service names. The choice of service names is entirely capricious. We have currently chosen `ikqs`, for "Internet Key-Query Service", and `ikrs`, for "Internet Key-Registration Service," respectively.

Hence, a client wishing to locate a server capable of handling queries for names in domain `D` would query for an *SRV* record matching `_ikqs._tcp.D`. Similarly, to locate a server capable of handling registration requests for names in `D`, *SRV* records matching `_ikrs._tcp.D` would be requested.

The IKS for a given domain need not actually be hosted in or by the owner of the domain. The domain administrator may delegate this function by adding the required *SRV* records. We see the potential for organizations, possibly existing certifying authorities who already have a good understanding of the operational security issues involved in key management, to offer IKS services to domain administrators.

5.2 Message Marshalling and Transport

We have chosen XML as the format for IKS messages to ensure compatibility with the dominant message format protocol and the dominant class of applications on the Internet. This approach conforms to current industry best practices and standards for remote service location and invocation. It is relatively straightforward for a client to be able to parse and to generate the simple XML messages used in IKS.

Marshalling
The World Wide Web Consortium (W3C) has published a multi-part recommendation called SOAP (Simple Object Access Protocol) specifying an interoperable means of using XML to exchange structured and typed information in a distributed environment or application [30].

SOAP specifies message formatting, including the overall structure of the message as an XML document (called the SOAP *envelope*), the structure of an optional header carrying non-payload information, and the structure of the body carrying the message payload. SOAP is transport-independent, dictating only the high-level message format and providing bindings to various transport mechanisms, including HTTP. A more complete overview of SOAP, with supporting examples, can be found in [44].

Since SOAP is an emerging standard, and XML is already a dominant message exchange format on the Internet, we have chosen SOAP to encode registration messages. Relevant details of the SOAP message structure will be explained as they are required.

Transport
With the increasing popularity of the Web Services model of remote service invocation, HTTP and HTTPS are fast becoming the de facto standard transport protocols for remote procedure call. One of the primary reasons for this adoption is that HTTP and HTTPS are typically permitted through firewalls and across different administrative domains within an organization. This is reinforced by the relative simplicity of the protocol, as well as availability of client and server implementations. Given SOAP's transport independence, other transport mechanisms may be supported in the future.

Query Optimization
Due to the relative simplicity of query operations, and the need to optimize this common operation, we have chosen to provide a simplified and optimized interface to lookup operations. Query requests are mapped, by the IKS client, into HTTP requests for static XML documents using a URL-safe encoding of the queried object's name [53]. This optimization allows the query server to return the query response in the form of a precomputed response. The response XML format is similar to the SOAP formats used by the registration server but without the overhead of the SOAP envelope, and without the need for dynamic response generation.

5.3 Query and Registration URIs

Determining the name of the host providing the desired service, as described in 5.1, is insufficient to actually contact the service end-point. Since SOAP services are differentiated by URI, we use the service names established previously as URI components. The path "*/ikqs*" references the key-query service and "*/ikrs*" references the key-registration service. Given the host and port information obtained from the *SRV* records, a client can construct complete URIs for the desired service. Modern HTTP servers provide flexible mechanisms to map the published URI space into an arbitrary server side structure, so the mandate of public endpoint names does not dictate server side details.

5.4 Authenticating Key-Signing Keys

As mentioned in Section 4.3, query response messages are signed by one of the domain's key-signing keys (KSK). To verify this signature, the client must fetch the KSK from IKS as well as the commitment for that KSK from DNSSEC. To avoid polluting a domain's DNS name space with a slew of standardized names, we allow the query response to identify the signature's KSK by name, and require only that a commitment to the KSK be published in DNSSEC by the domain being queried.

While it is tempting to store these keys using the *KEY* resource record format for DSA keys described in RFC 2536 [18], this strategy runs afoul of RFC 3445 [42], which prohibits storing keys not directly consumed by DNSSEC in *KEY* records. Instead, we store a key-commitment (a SHA-1 hash) of the KSK in DNS, instead of the entire key. (Recent cryptanalytic results against SHA-1 mandate re-evaluating the use of SHA-1 as a secure hash function [8, 66].) The guarantees made by DNSSEC will continue to apply, the achieved security is equivalent, as long as a secure hash function is used.

A named KSK `K` for domain `D` must be a DSA key in PEM format stored on the IKS server serving queries for domain `D` for name `K`. The hash of the PEM key data is stored in a DNS text record with the name `sha1_K`. This record will contain a hexadecimal representation of the SHA-1 hash value in big-endian byte order.

To verify the results of a query, the client first obtains the KSK by requesting the key named in the query response from the server. Subsequently, the client retrieves the commitment of that KSK from DNSSEC and confirms that the retrieved key matches the commitment. Finally, the KSK is used to verify the query results.

We do not require KSKs to explicitly have a lifetime. A KSK's lifetime is bound by the validity period of its commitment's DNSSEC signatures and its appearance in query responses. If a KSK is compromised, or is being routinely refreshed, query responses signed with the old key must be re-signed with the new key. Clients may cache a KSK until its DNSSEC signature expires, and use this cached key to validate any retrieved records that use it.

5.5 Key Metadata Specification

To facilitate the selection and use of keys stored in IKS, we have specified the standardized metadata attributes included in IKS key entries. We have selected a standardized set of key information that describes the valid uses, lifetime, and storage format of the keys stored in IKS. Table 1 shows the supported attributes.

Attribute	Description
`Name`	Fully qualified entity name
`Service`	Service name
`Id`	Unique key identifier
`Format`	Key storage format
`Algorithm`	Signing algorithm
`Length`	Key length
`Allowed Use`	one of {privacy, authenticity, privacy+authenticity}
`Valid After`	Start of key lifetime
`Valid Until`	End of key lifetime/`null`
`Revoked At`	Key revocation time/`null`

Table 1: Key Metadata Supported by IKS

At the moment, other key information must be encoded into the stored key itself, requiring a suitable key format specification, and requires fetching the key in order to extract.

6 The Riverside Internet Key Server

We have built a prototype implementation of IKS, which we call the Riverside Internet Key Server (RIKS). In this section, we will describe the issues, the design choices, and our preliminary experience with this system.

We decided early on to leverage as much existing code as possible in building RIKS. Our discussions below identify the third-party components used, and point out relevant details of each.

Our presentation is in four parts: securing a zone using DNSSEC, the RIKS server components, the RIKS client library, and our experience writing simple test clients.

6.1 DNSSEC Theory into Practice

We now discuss several issues that we encountered in using DNSSEC effectively in our work. We point out difficulties, and describe how we addressed them.

Securing a Zone

The first issue was the provisioning of a security-aware DNS server. The latest release candidates of BIND (9.3rc1, at the time of this writing) are closely tracking the latest DNSSEC IETF drafts, and include utilities to perform key generation and zone signing [63]. The details of configuring DNS are outside the scope of this paper, and the details of configuring DNSSEC are guaranteed to change over time, and are therefore omitted.

One general problem we observed was that while DNSSEC implementations seem to be maturing quickly, the documentation for implementors is often nonexistent or inaccurate. We found the operational HOWTO by Olaf Kolkman [40] useful guidance in configuring secure resolvers and zones.

Secure Resolution

Secure resolution involves three actors: the security-enabled server, the verifying resolver trusted by the client, and the requesting client. The client, which may be unaware of DNSSEC, submits a query to the verifying resolver, which returns validated results. If validation is successful, the resolver responds with the requested records, setting the authenticated data (AD) bit to indicate the data has been verified.

The BIND distribution includes a lightweight resolver daemon, called `lwresd`. This daemon, intended to be run on each client host and service only local requests, provides implementations of the standard resolution interfaces as well as extensions that give clients additional control over name resolution. The documentation on `lwresd` configuration and use is sparse, and improved documentation would be helpful.

Secure Resolver Interface

Typically, the mechanics of name resolution are hidden by networking libraries, allowing developers to work in terms of higher-level network operations. DNSSEC throws a proverbial monkey-wrench into the gears. Existing applications are unaware of DNSSEC. Consequently, they have no way to interpret information about the validity of query responses, and the application programmer interfaces (APIs) they use have no means to express this information.

In the longer term, clients may expect that all DNS responses will be validated by DNSSEC, and non-verifiable responses will not be returned to them. In the short term, it is unclear how applications that are interested in the security state of their name-resolution requests will interface with the verifying resolver. It is undesirable to have each application embed a verifying resolver, but the standard library interfaces are inadequate. Further, validation requires cryptographic code, which is unlikely to find its way into the standard system libraries. The interface between clients and resolvers is still a topic of active development and an evolving IETF draft [28].

6.2 The RIKS Server

The RIKS server is composed of three components, one to handle query requests, one to handle registration and

revocation requests, and a separate update process to generate signed query responses for the query handler (and in the darkness BIND them.) For security reasons, only the update process is given access to the domain's KSKs. The components of the server communicate through a relational database.

RIKS is designed to make key lookups efficient. All valid keys are stored in an SQLite [60] database as XML objects already signed with the KSK. A lookup is simply a retrieval from the database. A separate activity (the update process) periodically ensures that these objects are current and carry valid signatures.

We have identified three signature generation strategies, which differ in the time at which the KSK is needed and which processes have access to it.

On-line: The key-registration handler signs keys with the KSK immediately upon their acceptance by the system. This method has the advantages that it is easy to implement and updates are immediately available to clients.

On-demand: The key-lookup handler checks the database for a response object. If it exists and is signed, it is returned. Otherwise, it is immediately signed and returned to the requesting client. This method has the advantage that no unnecessary signatures are computed.

Off-line: All signatures are generated by an off-line process that runs periodically. This method has the advantages, because the key-signing key can be kept offline during operation, it is most secure and updates can be scheduled at regular intervals.

RIKS currently supports only the off-line method of signature generation. However, it would be very straightforward for us to add the other signature methods, and for RIKS deployments to select one as a configuration option. Figure 2 shows the current RIKS Architecture.

a

6.2.1 Implementation Toolkit

We chose to implement RIKS in Python [51], using the Zolera Soap Infrastructure (ZSI) [54], *mod_python* [62]. Requests from clients take the form of HTTP queries, and are handled using the Apache [61] web server. The M2Crypto [58] wrapper provides Python access to the cryptographic functionality in the OpenSSL library. The server uses SQLite [60], an embeddable SQL'92 compliant RDBMS engine, and *pysqlite* [47], a Python DB-API [57] compliant interface layer, for underlying data storage.

During our work, we identified and corrected defects in the ZSI framework code, and exposed additional OpenSSL functionality to the M2Crypto Python cryptographic library. These improvements have been submitted back to the respective package maintainers.

6.2.2 Registration Handler

New key-registration requests and key-revocation requests are sent to the Registration handler. These requests must be authenticated and authorized before execution. Likewise, the server's response to the client is signed by a response-signing key (RSK), as confirmation to the client that its request was received by the registration handler.

6.2.3 Update Process

Before the effects of registration and revocation operations performed by the registration handler are made visible to querying clients, the corresponding signed key-query response messages have to be generated by the update process, which is granted, potentially temporary, access to at least one of the domain's KSKs. Additionally, as query response messages expire, replacement signatures must be generated.

The update process runs in two stages, in the first phase it queries the database to generate a worklist of responses to generate. This is followed by a work phase in which query responses are prepared and signed for the query handler.

6.2.4 Query Handler

When a request for keys registered under a given name arrives, the query handler simply looks in the database for a pre-signed message with this information already placed there by the update process (see Section 6.2.3). If no such object is found in the database, the query handler returns a failure response signed with the RSK.

Caching can play the same role in reducing load on IKS that it plays with DNS. Clients should actively cache responses to avoid unnecessary queries to the IKS query server. Caching will also improve the latency seen by the application, since signatures need not be verified for every request. Unlike DNS, where cache misses frequently result in a query to a root name server, IKS cache misses are sent directly to the IKS server responsible for the indicated domain. No bottlenecks result since requests are distributed across a large number of domains, managed by different IKS servers.

6.2.5 Performance

We ran a series of tests to measure the registration, update, and query performance of our RIKS prototype. These tests were run on a single CPU (1.5GHz Pentium 4M) laptop machine with 512MB of RAM. The tests were run with a moderate-sized database, containing between 50,000 and 60,000 entries (10 keys registered to each user). This database was approximately 300 MB in size. Table 2 summarizes RIKS performance.

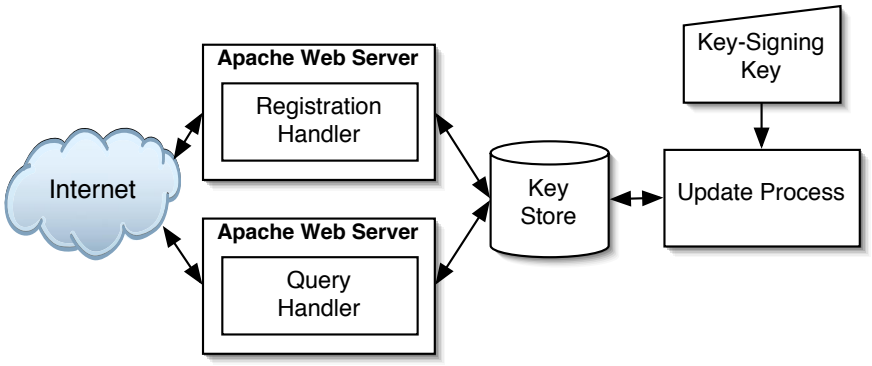

Figure 2: RIKS Architecture using off-line signature generation.

Operation	Registration	Query	Update
Transactions/sec	6.1	295	68

Table 2: RIKS Performance Summary

The RIKS prototype query handler was able to serve 295 key lookup requests per second, a number we regard as adequate for use in practice. When IKS caching is used on client sites, we expect the workload on the IKS servers to be reduced dramatically. We also expect IKS throughput to increase significantly once we optimize our implementation further. A large, high-traffic, domain can easilly parallelize their IKS server in order to provide higher performance.

The RIKS registration handler was able to complete 6.1 registration requests per second. The bulk of its time was spent in parsing incoming requests, serializing success responses and sending them back to the client. Approximately 12% of the registration handler's time was spent authenticating requests, storing the new keys in the database, and signing responses. Registrations are relatively rare compared to key lookups, and we see this performance as reasonable.

The update activity identifies keys that must be registered, re-signed, expired, or purged, and then process them. The RIKS update process took 70 seconds to identify entries requiring reprocessing, with the database on disk. Once the database was loaded into memory, this same operation took 2.0 to 2.5 seconds. After constructing this worklist, the update process completed generating and signing query responses at a rate of 68 per second.

A successful registration of a 1024 bit DSA key requires approximately 4 KB of SOAP messages to be sent between the client and server. XML query responses were approximately 1.8 KB each.

6.3 RIKS Client Library

Currently, the only complete client library available for RIKS is a Python module that shares core portions of the server's code base. While this client library is functional, it is not appropriate for inclusion in most client applications. We are re-implementing our client library in ANSI C, and expect this effort to be completed shortly.

We have taken measures to limit the complexity of the client library so that we can reasonably expect any cryptographically aware application to include it in order to publish and lookup keys in IKS. Aside from the cryptographic operations provided by the OpenSSL toolkit most IKS operations are handled by libraries included with languages such as Python or Java and are easily available for C or C++.

Python DNSSEC Resolver Status
Python lacks DNSSEC resolver support, and we are in the process of building a bridge to the native-code `lwresd` library. In the interim, our client library uses a dummy DNSSEC resolver implementation which resolves DNS queries from its own trusted data source (provided by the client).

6.4 Sample Client Code

Several small client applications have been written in Python using the RIKS client library in order to validate the correctness of the server and client as well as to perform the performance measurements reported in Section 6.2.5. Each of these performance tests were simple, using only a few dozen lines of Python to drive the RIKS. We feel that the API being exposed to the client is easy to use and appropriate for adding IKS support to existing applications.

6.5 Discussion

Our focus to date has been on ensuring the correctness of the RIKS implementation. We have a fully functional

and usable prototype, but several issues need to be addressed before it is ready for production deployment. For example, our current implementation is suceptable to SQL injection attacks that can compromise the integrity of the key database. Stringent filtering of user input would protect the database from such un-authorized changes.

Numerous optimizations are also possible on the database. Performance may suffer when domains become very large. Horizontal partitioning of the database is a viable option in this case. Our architecture allows the individual RIKS processes to run independently on separate CPUs.

7 Conclusions & Future Work

Security and privacy are becoming major concerns as Internet continues to grow. Powerful cryptographic tools exist to address such concerns, but have not been widely used since no convenient infrastructure is available for distributed authenticated key distribution. IKS is intended to accelerate the development and widespread adoption of cryptographically-enabled applications by addressing this need. IKS is a simple, scalable public key distribution service, and its protocols have been designed specifically to meet the requirements of this domain, conforming to current industry best practices and standards for remote service location and invocation.

We rely on DNSSEC to provide authenticated and trustworthy name resolution and delegation, conforming to DNS's domain delegation semantics, while keeping the functional overhead of key distribution outside the critical DNS infrastructure. This strategy allows us to use the name service infrastructure to provide authenticity guarantees while avoiding the scalability, efficiency, and administrative pitfalls of earlier DNS-based mechanisms. Furthermore, we use DNS names directly, and not a namespace orthogonal to it, facilitating its integration into the existing Internet infrastructure.

We have presented the design and implementation of RIKS, the Riverside Internet Key Server, a prototype implementation of IKS. RIKS consists of approximately 4000 lines of Python code, and demonstrates performance adequate to justify confidence in our approach. The RIKS client library API provides a simple interface to IKS, making it easier to incorporate key authentication into existing collaborative tools.

Future Work

With our continuing work on IKS, we hope to progress to the development of an IKS standard specification, to incorporate input from the community and motivate deployment in tandem with DNSSEC.

We will continue improving RIKS in order to improve its performance, security, and manageability. Much work remains to be done to allow RIKS to serve the needs of large ISPs. We feel that that the current RIKS key-lookup performance is adequate for use in the context of a large ISP, but that we will need a five-fold increase in registration performance and a smaller improvement in update performance. We believe these requirements are achievable.

To verify the ease with which existing applications can be extended to use IKS, we are planning the deployment of a secure application. While distributed applications, such as email and VoIP, will benefit most from IKS in the longer term, it should be straightforward to deploy IKS within a single domain, even with the current deployment status of DNSSEC. Centralized applications, including certain Instant Messaging applications, could easily be secured using IKS today.

As DNSSEC gains adoption and penetration, we believe IKS will facilitate authenticated public key distribution, improving the security of existing network applications and protocols, and enabling new developments. When Alice must locate Bob's key, she can turn to IKS.

Acknowledgments

This work was supported in part by the Defense Advanced Projects Research Agency under contract F30602-01-2-0536.

References

[1] H. Abelson, R. Anderson, S. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. Neumann, R. Rivest, J. Schiller, and B. Schneier. The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption, 1998.

[2] D. Akin. Arrests key win for NSA hackers. The Globe And Mail, April 2004.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. IETF draft: dnsext-dnssec-intro-09, February 2004.

[4] R. Arends, M. Larson, R. Austein, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. IETF draft: dnsext-dnssec-protocol-06, May 2004. Updates RFC 2535.

[5] D. Atkins and R. Austein. Threat Analysis of the Domain Name System. IETF draft: dnsext-dns-threats-07, April 2004.

[6] R. Atkinson. Key Exchange Delegation Records for the DNS. RFC 2230, November 1997.

[7] S. Bellovin. Using the Domain Name System for System Break-ins. In *Proc. Fifth USENIX Security Symposium*, June 1995.

[8] E. Biham and R. Chen. Near-Collisions of SHA-0. In *Proc. of CRYPTO 04*, 2004.

[9] S. Boeyen, T. Howes, and P. Richard. Internet X.509 Public Key Infrastructure LDAPv2 Schema. RFC 2587, June 1999.

[10] D. Boneh and M. Franklin. Identity-based Encryption from the Weil Paring. In *Proc. of CRYPTO 01*, pages 213–229, 2001.

[11] N. Brownlee, kc Claffy, and E. Nemeth. DNS Measurements at a Root Server. In *Proc. of Globecom 2001*. IEEE, November 2001.

[12] ClickZ Stats. Population Explosion!, April 2004.

[13] S. Crocker and R. Mundy. An Invitation: Building a Road Map for DNSSEC Deployment, April 2004.

[14] M. Delany. Domain-based Email Authentication Using Public-Keys Advertised in the DNS (DomainKeys). IETF draft: delany-domainkeys-base-00, May 2004.

[15] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[16] D. Eastlake. DNS Security Operational Considerations. RFC 2541, March 1999.

[17] D. Eastlake. Domain Name System Security Extensions. RFC 2535, March 1999.

[18] D. Eastlake. DSA KEYs and SIGs in the Domain Name System (DNS). RFC 2536, March 1999.

[19] D. Eastlake. Storage of Diffie-Hellman Keys in the Domain Name System (DNS). RFC 2539, March 1999.

[20] D. Eastlake. RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS). RFC 3110, May 2001.

[21] D. Eastlake and O. Gudmunsdsson. Storing Certificates in the Domain Name System (DNS). RFC 2538, March 1999.

[22] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Yolen. SPKI Certificate Theory. RFC 2693, September 1999.

[23] P. Eronen. Applying Decentralized Trust Management to DNS Dynamic Updates. In *Proc. Third NordU/USENIX Conference*, 2001.

[24] FreeS/WAN Project. FreeS/WAN. www.freeswan.org/.

[25] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol Version 3.0, November 1996.

[26] J. Galvin. Public Key Distribution with Secure DNS. In *Proc. Sixth USENIX Security Symposium*, 1996.

[27] R. Gieben. DNSSEC in NL, January 2004.

[28] R. Gieben, G. Guette, and O. Courtay. DNSSEC Resolver Interfacer to Applications. IETF draft: gieben-resolver.txt, January 2004. Expired July 1, 2004.

[29] Gnu Privacy Guard Project. www.gnupg.org.

[30] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework, June 23.

[31] O. Gudmundsson. Delegation Signer (DS) Resource Record (RR). RFC 3658, November 2003.

[32] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR For Specifying the Location of Services (DNS SRV). RFC 2782, March 1999.

[33] N. Haller. The S/KEY One-Time Password System. RFC 1760, February 1995.

[34] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289, February 1998.

[35] M. Horowitz. A PGP Public Key Server. Undergraduate Thesis, MIT, 1996.

[36] J. Hubaux, L. Buttyán, and S. Capkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proc. 2nd ACM International Symposium on Mobile ad hoc Networking and Computing*, pages 146–155, 2001.

[37] IETF S/MIME Working Group. S/MIME Mail Security Charter.

[38] Internet Software Consortium. Internet Domain Survey Host Count, May 2004.

[39] ISO and CCITT, editors. *Recommendation X.500: The Directory: Overview of Concepts, Models and Services*. ITU, 1993.

[40] O. Kolkman. DNSSEC Operational HOWTO. www.ripe.net/disi/, November 2002.

[41] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421, February 1993.

[42] D. Massey and S. Rose. Limiting the Scope of the KEY Resource Record (RR). RFC 3445, December 2002.

[43] P. McDaniel and S. Jamin. A Scalable Key Distribution Hierarchy. Technical Report CSE-TR-366-98, E.E. & C.S. Department, University of Michigan, March 1998.

[44] N. Mitra. SOAP Version 1.2 Part 0: Primer, June 23.

[45] P. Mockapetris. Domain Names – Concepts and Facilities. RFC 882, November 1983. Superceded by RFC 1034.

[46] D. A. Norman. *The Design of Everyday Things*. MIT Press, 1998.

[47] M. Owens and G. Häring. PySQLite. pysqlite.sourceforge.net.

[48] M. Reiter and S. Stubblebine. Path Independence for Authentication in Large-Scale Systems. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 57–66, 1997.

[49] J. Reynolds and J. Postel. Assigned Numbers. RFC 1700, October 1994.

[50] R. L. Rivest and B. Lampson. SDSI – A Simple Distributed Security Infrastructure. theory.lcs.mit.edu/~rivest/sdsi10.ps, April 1996.

[51] G. Rossum. Python. www.python.org.

[52] RSA Security. RSA SecurID Authentication.

[53] E. S. Joseffson. The Base16, Base32, and Base64 Data Encodings. RFC 3548, July 2003.

[54] R. Salz. Zolera Soap Infrastructure. pywebsvcs.sourceforge.net.

[55] J. Schlyter and W. Griffin. Using DNS to Securely Publish SSH Key Fingerprints. IETF draft: secsh-dns-05, September 2003.

[56] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, second edition, October 1995.

[57] P. D. SIG. Python Database API Specification 2.0. www.python.org/peps/pep-0249.html.

[58] N. P. Siong. M2Crypto. sandbox.rulemaker.net/ngps/m2.

[59] D. K. Smetters and G. Durfee. Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSEC. In *Proc. Twelfth USENIX Security Symposium*, pages 215–230, August 2003.

[60] SQLite.org. SQLite. www.sqlite.org.

[61] The Apache Foundation. The Apache HTTP Server Project. httpd.apache.org.

[62] G. Trubetskoy. mod_python: Apache/Python Integration. www.modpython.org.

[63] P. Vixie. Berkeley Internet Name Domain. www.isc.org/index.pl?/sw/bind/.

[64] P. Vixie. DNS and BIND Security Issues. In *Proc. Fifth USENIX Security Symposium*, June 1995.

[65] X. Wang, Y. Huang, Y. Desmedt, and D. Rine. Enabling Secure On-line DNS Dynamic Update. In *Proc. Annual Computer Security Applications Conference*, December 2000.

[66] X. Wang, Y. Yin, and H. Yu. Collision Search Attacks on SHA1, February 2005.

[67] B. Wellington. Secure Domain Name System (DNS) Dynamic Update. RFC 3007, November 2000.

[68] T. Ylonen and D. Moffat. SSH Protocol Architecture. IETF draft: secsh-architecture-15, October 2003.

[69] L. Zhou, F. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.

[70] P. Zimmerman. *The Official PGP User's Guide*. MIT Press, 1995.