

## Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSEC

D. K. Smetters and Glenn Durfee  
*Palo Alto Research Center*  
*3333 Coyote Hill Road*  
*Palo Alto, CA 94304*  
{smettters, gdurfee}@parc.com

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

Copyright ©2003 Palo Alto Research Center Incorporated. All Rights Reserved. This work was originally published by the USENIX Association.

# Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSEC

D. K. Smetters and Glenn Durfee  
*Palo Alto Research Center*  
*3333 Coyote Hill Road*  
*Palo Alto, CA 94304*  
{smettters,gdurfee}@parc.com

## Abstract

Effective widespread deployment of cryptographic technologies such as secure email and IPsec has been hampered by the difficulties involved in establishing a large scale public key infrastructure, or PKI. Identity-based cryptography (IBC) can be used to ameliorate some of this problem. However, current approaches to using IBC for email or IPsec require a global, trusted key distribution center. In this paper, we present DNSIBC, a system that captures many of the advantages of using IBC, without requiring a global trust infrastructure. The resulting system can be configured to require almost no user intervention to secure both email and IP-based network traffic. We have built a preliminary implementation of this system in Linux.

## 1 Introduction

Standards for end-to-end encryption and authentication of email messages and IP-based communication have been in place for several years [8, 18]. Implementations of those standards are provided by most mail clients and network stacks. For the most part, however, we still don't use them. A large reason for this is the difficulty of managing and distributing keys – of having an authentic copy of your desired communication partner's public key when you need it.

Traditional approaches to key distribution in the public key setting rely on a Public Key Infrastructure, or PKI, to authenticate the public keys of users and devices relative to a hierarchical organization of trust. PKIs can be complex and difficult to set up and manage. Even with a PKI, you have solved only part of the problem – it only

allows you to determine the authenticity of a user's certified public key once you actually have it. But, that user first has to create a key pair and have it certified, and you still have to obtain a copy of his certificate before you can communicate securely with him.

### 1.1 Identity-Based Cryptography

Identity-Based Cryptography (IBC), originally introduced by Shamir [26], was proposed as a means to solve this problem. In an identity-based cryptographic scheme, you don't have to obtain your communication partner's public key or certificate from anywhere – you already know it. In such a scheme, your public key is an arbitrary string – e.g. “smettters@parc.com” or “myhost.parc.com”. If that string is chosen to be some identity that your communication partner knows, he can encrypt a message to you using only that string and a set of global system parameters. You decrypt that message using the private key corresponding to your public identity string. That private key is derived from your public identity using the global system parameters and a “*master secret*” – a global system secret held by a trusted third party known as a *Private Key Generator*, or PKG. Given the master secret, the PKG can derive a private key corresponding to any desired public key or identity string. As a result, such a system provides automatic key escrow.

Although identity-based signature schemes have been known for some time (e.g. [11, 12]), practical and secure identity-based encryption (IBE) schemes have been described only recently [2, 6]. One such IBE scheme, that of Boneh and Franklin, is based on the Weil or Tate pairing on supersingular elliptic curves [2]. This work has been used as the basis of several identity-based signature schemes [3, 16, 23], as well as a number of identity-

based key agreement protocols [4, 5, 14, 25, 27]. Several of these protocols allow a pair of communicating parties to agree on a shared symmetric key with no interaction whatsoever [14, 25].

IBC seems like an appealing way to solve the usability problems inherent in traditional approaches to key distribution. Not only do you automatically know the public key of anyone or anything with which you might want to communicate, without having to look it up anywhere, but you can encrypt a message to them before that person has even obtained their private key [2, 26]. If software support makes it easy to do so, a user receiving an encrypted email message seems more likely to obtain the private key necessary to read it than they might be to go through the complex steps necessary to get a digital certificate and provide it to someone who wants to send encrypted email to them (see [28] for a prototype of such a system). Similarly, the noninteractive key exchange protocols available using IBC have been proposed as a means to secure network traffic [1].

Unfortunately, systems that provide the full benefits of traditional IBC – knowing any party’s public key with no interaction whatsoever – suffer from tremendous scaling and trust management problems. In order to use your public key in an IBC system, I need to know not only your identity, but also a set of *system parameters* – these include both basic cryptographic parameters like the choice of an elliptic curve, but also includes the public key of the PKG, derived from the PKG’s master secret [2]. In an extreme case, using an IBC-based noninteractive key exchange protocol requires not only that I know the system parameters associated with your public key, but that you and I share the same system parameters – and hence we must both trust a single PKG in possession of the corresponding master secret.

In general, traditional approaches to IBC assume that all users will share the same PKG. This means that everyone knows the global system parameters, and hence can immediately derive anyone’s public key, but also requires the establishment of a system of global trust, where all users obtain their private keys from the same PKG. The global PKG’s master secret can be distributed among several centers using threshold cryptography [2] – if everybody could agree on a set of entities to trust with such a secret. However, the mere existence of such a vulnerable global secret – and the resulting system of global key escrow – is simply unacceptable for most applications. Additionally, a global private key generator would be extremely difficult to make work in practice, as it would have to authenticate the “correct” recipient of each private key in a global system of identifiers.

To address the latter problem, recent work has proposed mechanisms for constructing IBC systems using a hierarchically-organized set of PKGs [14, 17]. Unfortunately, in some of these schemes, PKGs higher in the tree can recover the private keys of PKGs lower in the tree, resulting in a system that has a somewhat easier time of distributing keys, but still requires global trust. Even in schemes without escrow, the tree must still be constructed hierarchically – all the nodes in the tree above yours must be in place before your keys can be generated. The resulting system is even more difficult to implement than a global, hierarchical public key infrastructure – something which so far has been notoriously difficult to establish.

## 1.2 Our Approach

We would like to take advantage of the usability of IBC without requiring everyone to participate in a global trust model. Our goal is to design a system that balances security and usability in a manner resulting in much wider deployment of secure email and IP security (IPsec). In contrast to the approach of Appenzeller and Lynn [1], we attempt to integrate our approach into existing standards and software, so as to ease deployment. We design our trust model to mirror the trust and management divisions that exist in today’s deployed networks. We emphasize autoconfiguration and automatic update as much as possible to minimize practical barriers to use.

Building an IBC system that allows us to manage trust along appropriate boundaries is simple – we merely have each such trust domain run their own Private Key Generator. This means that to communicate securely with a given party, you must know not only that party’s identity, but you must obtain the system parameters of their trust domain. While this isn’t as simple and seamless as a traditional IBC system, it is considerably simpler than a traditional PKI, as those system parameters are shared by an entire trust domain. At worst, it’s equivalent to having to obtain the certificate of the certification authority (CA) serving a given trust domain, and having that, being able to immediately derive the public key of everyone else in that trust domain. It makes the distribution of private keys within a trust domain considerably simpler, as the namespace of identifiers is local to the trust domain, and a smaller population of key recipients needs to be authenticated. Making such a system easy to deploy in practice requires appropriate construction of those trust domains, and the design of software to take advantage of them.

### 1.2.1 Bootstrapping IBC with Domain-Based Trust

Currently, user identities for email (email addresses), and the identities of hosts (names or IP addresses) are managed at the level of network domains, as described by the Domain Name System, or DNS. DNS delegates management of parts of the Internet name space to individual domains of control. We propose to divide the responsibility for authenticating those email addresses and host identities along exactly the same lines, by having each DNS domain responsible for creating a set of IBC system parameters and distributing private keys to its own users. This is directly analogous to having a given domain run its own Certification Authority and issue certificates to its users and machines.

Recent security extensions to DNS, known as DNSSEC, allow a DNS server to digitally sign the responses to queries, so that they cannot be modified or spoofed [9]. Each DNS server providing DNSSEC services offers up a KEY record for its domain containing its public key, signed by the key of the name server above it in the domain hierarchy. When fully deployed, the DNSSEC hierarchy will terminate in a root key trusted by all DNS clients and servers. In the meantime, parts of the DNSSEC hierarchy can be authenticated using cross-certification. As DNSSEC-capable name servers are already capable of providing and authenticating cryptographic data, they have been suggested as the most practical distribution method for cryptographic keys and certificates to be used by IPsec, TLS, secure email, and other protocols [9, 10, 24]. FreeS/WAN, a standard Linux IPsec distribution, has attempted to use these mechanisms to bootstrap an approach to “opportunisticly” encrypt all network traffic, by combining distribution of host IPsec keys in DNS records with records that indicate what machines can act as “security gateways” (IPsec termination points) for machines that cannot terminate IPsec themselves [13].

Using DNSSEC to store and distribute a set of authenticated IBC parameters for a domain, retrievable under the domain entry (e.g. “parc.com”) is a simple extension of these approaches. However, we suggest that the resulting combination of IBC and DNSSEC-based parameter distribution has a number of advantages over a traditional PKI, even one that uses the DNS as a key distribution mechanism.

First, it minimizes the amount of information stored in and retrieved from the DNS. IBC parameter information is global to the domain. It is generated once, and only re-generated in the case of master secret compromise. Updating of cryptographic information in the DNS is done

once for the domain, with intermittent additional updates (e.g. we use a small amount of transient data, a “salt” to provide revocation of keys through key expiration, see Section 2.1). This can be done under administrator control, and doesn’t require either clients or a CA to be able to publish information to the DNS, as would likely be necessary if clients put their certificate or key information into the DNS.

Second, as parameters (and salt) are global to the domain, clients wishing to communicate with multiple parties in the domain (or the same party more than once) must only pull one copy of the domain’s system parameters. These it can cache over time, reducing the load on the DNS servers.

Third, individual domains can deploy such a system incrementally – if a domain does not provide parameter information in its DNS entry, it obviously does not participate. And though DNSSEC is some ways from being completely deployed, it is designed to allow trust to be constructed incrementally, subtree by subtree, until the roots are in place. A system that bootstraps trust from DNSSEC will grow naturally along with it.

Fourth, client configuration can be dramatically simplified and automated. Clients don’t have to obtain their private key from the domain key distribution center until they need it. In the case of email, this means that the motivation is on the “right foot” – a user having received encrypted email is interested in reading it, and will go and get their private key, while a user who wants to send encrypted email to someone else in a non-IBC system is hard pressed to get that person to go and get themselves a certificate [2]. In the case of IPsec, clients can be designed to auto-configure themselves, and automatically request their private key at installation time. We believe the more setup steps that can proceed independently, the simpler the system will be to deploy in practice.

And finally, the fact that we can simply and automatically generate the public key of any party whose domain participates in the system (and the lack of domain parameters in the DNS will tell us whether they participate) means that we can use simpler cryptographic protocols, and can attempt to automatically secure all of our email and network traffic.

### 1.2.2 Overview

In the remainder of the paper, we will present a practical system for deploying identity-based cryptography (shown in Figure 1). In Section 2.1, we begin with an

overview of the design issues important in embedding IBC into existing protocols. In Sections 2.3, and 2.2, we describe the components of our system. In Section 3, we describe an email client that uses our approach to secure mail, and in Section 4, we describe how to use this system to secure IPsec traffic. Finally, we present our (in-progress) Linux implementation of this system in Section 5, and finish with related work and conclusions.

## 2 System Design

We have designed a system for using domain-based trust to implement identity-based cryptography. The overall structure of this system can be seen in Figure 1. Such a system consists of a number of components: first, a set of system parameters and a domain master secret, created by a setup procedure. Second, a private key generator, or PKG, that distributes private keys to authenticated members of the domain. Third, a modified nameserver capable of providing copies of the system parameters to communicating peers. And finally, client software capable of using DNS-based IBC (DNSIBC) to secure communications.

### 2.1 IBC Setup

The first step in enabling DNSIBC in a domain is the creation of the domain’s IBC system parameters (labeled setup in Figure 1). In what follows, we focus on IBC systems implemented using operations over supersingular elliptic curves [2], as there are a large number of encryption, signature, and key exchange protocols than can be used with a single private key pair derived for such a scheme.<sup>1</sup>

To specify the domain’s system parameters, we first specify a set of elliptic curve group parameters (“group-params”). These consist of the choice of the curve itself, the field it is defined over, and a generator point, referred to as  $P$ . These can be considered as analogous to the group parameters used in standard Diffie-Hellman systems, whether defined over an elliptic curve or over a prime field (where the corresponding choice would be of a prime,  $p$ , and a generator,  $g$ ). Like the group parameters used with Diffie-Hellman schemes, these parameters can be shared by many domains, and sets of such parameters can be predefined by standards bodies for general

<sup>1</sup>Note that in the presentation here and in what follows, we use notation common for the mathematics of elliptic curves –  $P$  is a point on a curve,  $x$  is an integer drawn from the field over which the curve is defined, and  $\langle xP, yP, xyP \rangle$  is the elliptic curve equivalent of the standard Diffie-Hellman tuple  $\langle g^x, g^y, g^{xy} \rangle$ .

use. This is done, for example, for Diffie-Hellman parameters for use in IPsec [15], allowing hosts that choose to use the standard groups to simply transmit short identifiers in place of the group parameters.

Once the group-params have been selected, each domain creates its own master-secret,  $s_d$ , which is a random value in a range specified by the group-params. The group-parameters and the master-secret are used to derive a corresponding public domain-public-key,  $s_dP$ . The master-secret is used later by the PKG to derive the private-key corresponding to any identity string,  $id$ , by first converting that  $id$  to a point  $Q_{id}$  on the curve using a hash function  $\text{mapToPoint}$ ,  $Q_{id} = \text{mapToPoint}(id)$ , and calculating the private-key as  $S_{id} = s_dQ_{id}$ .

The resulting domain-params consist of:

domain-params := (group-params, domain-public-key)

The master-secret must be stored securely for use by the PKG, while the domain-params are published publicly using the DNS, as shown in Figure 2.

**Revocation** To add the ability to revoke identities in this system, we add a form of key expiration [2]. Instead of using the identity  $id$  as the public key of a user or host, we use  $\text{salt}||id$ , where  $\text{salt}$  is a random string long enough to be unlikely to be chosen at random again (say, ten bytes), and  $||$  indicates concatenation. For instance, if your  $id$  was  $\text{smetters@parc.com}$ , and the current salt for  $\text{parc.com}$  was  $\text{OVQpMJJPpgZn}$ , your public key for this time period would be  $\text{OVQpMJJPpgZnsmetters@parc.com}$ . The salt is published in the DNS along with the domain-params. When the domain’s salt changes, keyholders in the domain know to automatically contact the PKG to update their private keys. By using lifetimes in the DNS (see section 2.2) to control the interval at which we have members of the domain and communicating peers check for an updated salt value, we can control the revocation interval for keys in this system. Because peers will automatically update their cached copy of this domain’s system salt, we can easily revoke keys on any schedule with much lower bandwidth requirements than, say, the distribution of Certificate Revocation Lists (CRLs).

### 2.2 DNS

We extend the Domain Name Service [21, 22] to support publication of the domain-params and salt. We do this by adding two resource record (RR) types to the

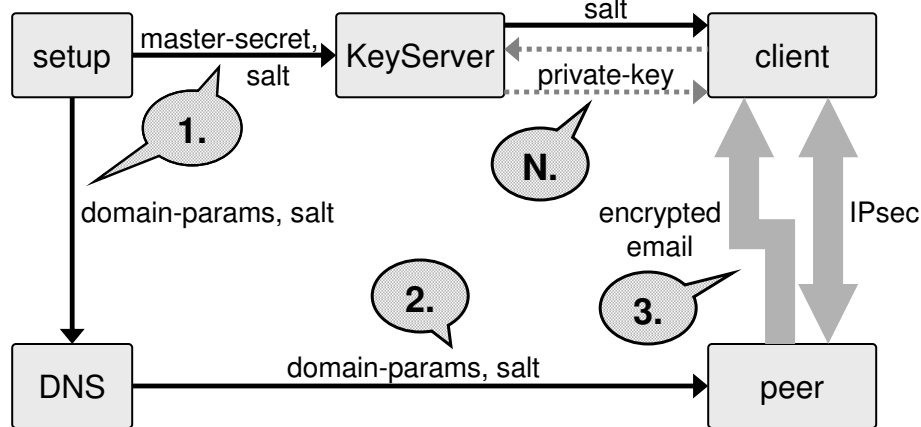


Figure 1: The architecture of our DNSIBC system. The step labeled “N.” can take place any time after step “1.” Encrypted email is sent normally: i.e., through a standard mail server, without direct interaction between sender and recipient.

DNS. These are PARAMS, which encodes the domain-params, and PSALT, which encodes the salt to be used with these parameters. In this section, we discuss the format and values of the RR fields associated with the new RR types.

The NAME field of the PARAMS and PSALT RRs must be the domain name for which these are the domain-params and salt, respectively. The use of DNS abbreviated names is allowed (although care must be taken so that this does not interact poorly with signing of RRs, discussed below.)

The TTL field of the PARAMS RR should be set to an interval that adequately protects against compromise of the master-secret. In particular, if an unauthorized party obtains the master-secret, she can compute the private-key of any user or machine in the affected domain. If the master-secret is compromised, use of the IBC system should be immediately discontinued until a new master-secret has been generated and corresponding domain-params have propagated to the DNS system. At this point, all previous private-keys and the original master-key are now useless to the attacker. One way to guarantee this recovery happens quickly is to set the TTL field of the PARAMS record to an interval short enough to force frequent updates due to TTL expiration. However, given the typical size of the PARAMS record (approximately 286 bytes with standardized group-params and 690 bytes without), and the unlikelihood of the master-secret compromise (which is presumably on a closely-monitored and well-protected machine accessible from only inside the domain), one can set the PARAMS TTL to a reasonably long period of time (e.g., on the order of

days).

The TTL field of the PSALT RR should be set to an interval that protects against compromise of an individual private-key. If a private-key has been compromised, use of IBC on the compromised machine (or user’s email account) must be discontinued until a new salt has been generated and propagated to the DNS system. Note that other machines/users may safely continue using IBC-protected IPsec/email. The use of a new salt necessitates (eventual) distribution of a new private-key to every machine/user enrolled in the system. This might place a heavy load on the key server if the update is performed all at once; however, an update could be performed incrementally, as machines discover they are using expired salts, or by having several PSALT records available for a domain used by disjoint blocks of addresses<sup>2</sup>. Since the compromise of an individual private-key is more likely than compromise of the master-secret, and the PSALT RR is significantly shorter than the PARAMS RR, it makes sense to use a much shorter time for the TTL of the PSALT RR, perhaps on the order of several hours. We note that even after the TTL of a PSALT or PARAMS record runs out, it does not necessarily have to be changed. For efficiency, we suggest keeping PARAMS and PSALT records unchanged for long periods of time or until replacement becomes necessary. For the case of the PSALT, this would be until the occurrence of an actual key compromise, or several weeks have passed. The PARAMS could be expected to remain unchanged unless

<sup>2</sup>For example, the PSALT record could be augmented with a field indicating it is to be used for all email address starting with letters a–h. For conciseness, we will not discuss the many interesting variants on the PSALT field that might arise in such a system.

```

struct group_params {
    big_int p, q;
    point_Fp P;
};

struct domain_params {
    struct group_params gp;
    point_Fp domain_public_key;
};

struct DNS_PARAMS_RR {
    unsigned char *NAME;
    uint16      TYPE, CLASS;
    uint32      TTL;
    uint16      RDLENGTH;
    uint16      flags;      // RDATA start
    uint8       protocol;
    uint8       algorithm;
    struct domain_params *domain_params;
};

```

Figure 2: Internal format of the PARAMS DNS resource record and related types.

the master-secret was compromised.

To simplify implementation, the RDATA (record type-specific) portion of the PARAMS resource record uses the same format as that of the KEY RR [9] (see Figure 4). This record contains fields for key usage flags, a protocol and an algorithm (see Figure 2). We suggest using only a subset of the flags possible for the KEY record: bits 0 and 1 (prohibition of use of the domain-params for authentication and confidentiality, respectively), and bit 8 (allowing use of domain-params for IPsec). The protocol field, which further specifies how this key can be used, should allow an additional value beyond those available in the KEY record: an index for “IBC”. The algorithm field indicates which basic IBC system is being used, in this case an index indicating the use of IBC based on supersingular elliptic curves.

One might notice that instead of creating a new RR specifically to house the domain-params, we could have extended the KEY resource record type. However, we feel this would be inappropriate for several reasons. First, DNS servers are only required to handle two KEY RRs associated with a DNS name [9]; presumably one is already used as the DNSSEC signing key for this domain, so spending the other on IBC unnecessarily limits further use of the KEY RR type. Second, we wish to use DNSSEC to verify the integrity of a signature on the PARAMS record; this will be signed by the domain

```

parc.com. IN PSALT OVQpMJJPpgZn
parc.com. IN PARAMS 256 1 1 MIICrgKBG2c\
331fS7BexMEzGkWGycIBPrIH915TnE6c06Ifg\
fnZBK1cz/PGrF36Z7n1hrHGFHb0hmmHBZb17a\
YjEG2+MbxvN801DFE6sihKXwORlLkk5DtuD...

```

Figure 3: Example zone file containing PSALT and PARAMS DNS resource records.

KEY, and it would be improper to have one KEY record be used to sign a KEY record at the same level. Third, the domain-params are not a public key, and should not be treated as such (for the purposes of revocation, selecting a TTL, etc.) Placing them in a KEY record encourages confusion in this regard, and is in any case deprecated [19].

The RDATA field of the PSALT RR type consists of simply the bytes that make up the salt.

### 2.2.1 DNSSEC Records for IBE

In order to bootstrap trust in a domain’s IBC domain-params there must be a way to verify the validity of the PARAMS and PSALT RRs retrieved from the DNS. We recommend using DNSSEC [9]. A SIG record should be added for the PARAMS resource record owned by this domain. If present, the SIG record must be verifiable using the domain’s (traditional) cryptographic public key, which must be available as a KEY record.

### 2.3 Private Key Generator

In our system, the Private Key Generator (PKG) is a service that computes an entity’s private-key using the groups-params, the master-secret, and the client’s identity (which could be an email address or a hostname), and the current salt. The PKG obtains the group-params and master-secret from the output of the setup procedure (see Section 2.1). This setup procedure could be run automatically as part of (PKG) initialization. The PKG then waits for key retrieval requests from clients.

**Initial Client Key Retrieval** Perhaps the most difficult problem in designing a system to easily deploy IBC is enabling clients to automatically retrieve their keys with sufficient security. It is very easy to have clients automatically retrieve their private keys – when a machine or piece of client software (such as an email program)

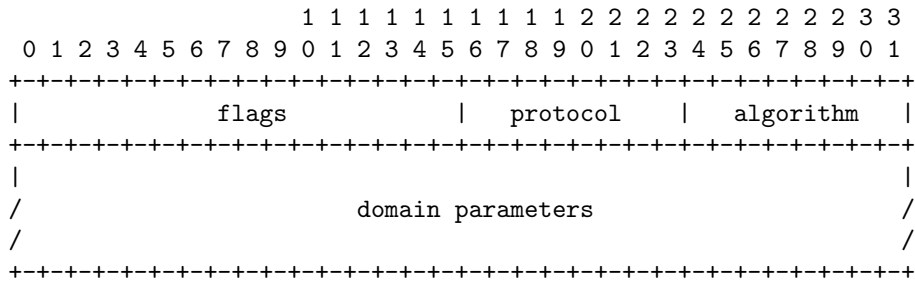


Figure 4: The wire format of RDATA for the PARAMS DNS resource record.

first realizes that it doesn't have a necessary private key, *e.g.* the first time an IPsec-enabled host boots, or the first time a mail client receives IBC-encrypted mail for a particular user, it attempts to retrieve that key from the domain's PKG. While it would be very easy to create a system to allow a user to securely retrieve that key, for example through a password-protected web site linked to the domain's user database, we would like to enable automated key retrieval.

A host wishing to automatically retrieve its private-key must first find the domain PKG. This can be done, for example, through the use of a standard configuration file general to the domain, or by using a designated record similar to an MX record in the domain's internal DNS to indicate the host acting as the PKG. Then, the client must create an encrypted and preferably authenticated connection to the PKG – encrypted to protect its private-key from eavesdroppers, and authenticated to make sure that an intermediary is not attempting to hijack its key *en route*. Interestingly, a rogue host attempting to pretend to be the PKG is no more than a nuisance, as the client can verify the authenticity of the private-key it receives simply by encrypting and decrypting a message to itself.

In practice, we simply use SSL/TLS to secure communication between the client and the PKG. The PKG can use a traditional server certificate authenticated through any number of standard means – *e.g.* traditional DNSSEC mechanisms for distributing keys, certificates, or CA keys; an internal domain PKI; an external trusted CA; a modified IBC-based version of SSL, or it could even use a self-signed certificate if the internal infrastructure is deemed sufficiently resistant to spoofing attacks.

More importantly, the PKG must be able to authenticate that the client requesting the key for a certain id is actually the client that should have that id. There are a number of ways of accomplishing that. At one extreme, clients could be equipped with private keys and certificates, which they use to authenticate themselves to the

PKG. Clients can also be identified using any secrets they already share with the domain (*e.g.* user passwords, or machine domain credentials). It would also be very simple to design mechanisms that use temporary client passwords or “cookies” provided to clients through user registration or administrator action, that they can use to authenticate themselves to the PKG. More practically, simple mechanisms that seem inherently insecure when used globally may be more than sufficient when used inside a trust domain, especially when combined with the ability to verify over time that the correct client received the correct key. In the case of email, this could mean using the ability of a user to receive mail at a particular address to be an indicator that they really are the valid “owner” of that address. Similarly, if the PKG is only allowed to communicate with hosts within a trusted domain (say, behind a firewall on a trusted piece of the network) that in and of itself may be sufficient to authenticate host identity.

**Client Key Update** Clients must receive new private-keys when the salt (and rarely, the domain-params) are updated (see Section 2.1). Clients can use DNS TTL values to automatically determine the intervals at which they ought to check for new salt or domain-params values (see Section 2.2). While client key updates can obviously be performed in the same manner as initial client key retrieval, some optimization is possible when only the salt has changed. In such a situation, a client whose old private-key has not been compromised can be sent its new private-key encrypted under its old id, which it can use its old private-key to decrypt. The security of the system is maintained as long as the initial private-key exchange was secure.

We note that clients will want to keep a list of several previous private-keys and associated salts. This is necessary to to decrypt email which was sent before, but not read until after, the domain underwent a domain-params or salt change.

### 3 Secure Email Client

In this section, we describe our approach to providing secure email using domain-based administration of identity-based cryptography. Suppose a sender Alice wishes to send email to a recipient, say Bob with email address “bob@parc.com”. Alice’s email client first retrieves from the DNS the `PARAMS` and `PSALT` resource records for `parc.com`. These decode into a `domain-params` and `salt`, respectively. Alice then uses identity-based encryption [2] with the `domain-params` and “salt||bob@parc.com” as the public key to encrypt a symmetric cipher key. This is in turn used to encrypt (and MAC) the email. Alice sends the encrypted email, along with the salt and a digest of the `domain-params` used to Bob’s mail server (which was presumably learned from a MX record in the same DNS query.) Note that Bob is not involved in this process. Indeed, up until the encrypted email is sent, Alice has no need to communicate with any machine in `parc.com`; the DNS will cache all appropriate parameters until their TTLs run out.

Bob’s email client then pulls the encrypted email into his mailbox, checks that the digest of the `domain-params` matches its current knowledge of the `domain-params`, and pulls from a private store the private-key corresponding to the salt encoded in the message. If either the `domain-params` digest fails to match, or no entry exists for the salt used in the message, the client asks the DNS for the latest `domain-params` and salt to see if it needs an updated private-key. If so, it contacts the PKG for a new private-key (using SSL with encryption and mutual authentication); otherwise, it rejects the email as invalid. It then decrypts the message and presents it to Bob.

### 4 IPsec Client

We would like to use identity-based cryptography to secure IP-based network traffic. Previous work has suggested the use of non-interactive identity-based key exchange protocols to secure traffic between hosts in the same IBC trust domain [1], but did so by inventing a new set of protocols. We’d prefer to use IBC in a way that works easily with existing standards and software, and supports hosts using different sets of IBC parameters.<sup>3</sup> This means using IBC to secure IPsec [18].

<sup>3</sup>We are primarily interested in hosts using domain-based parameters, but the details of how we incorporate IBC into IPsec should be generic across almost any IBC trust distribution mechanism.

IPsec is an IETF standard protocol providing mechanisms for encrypting and authenticating IP packets. This protection is provided using algorithms and symmetric keys negotiated using the Internet Key Exchange protocol, or IKE [15, 20, 24].<sup>4</sup> Clients using IPsec generally implement IKE in a user-space daemon, which negotiates security associations (SAs) and keys with the corresponding IKE daemon on the hosts with which it wishes to communicate securely. The negotiated SAs and keys are then provided to the IPsec implementation in the network stack, which uses them to secure IP packets.

Using IBC to secure IPsec traffic means describing IBC-based key exchange protocols to be used as part of IKE, and implementing them in the IKE daemon. It does not require modification to the network stack components in the kernel. Luckily, it turns out that IBC can be easily accommodated in the existing IKE protocol, with no change to packet structure or protocol flows.

#### 4.1 Structure of IKE

IKE is defined as a particular instantiation of the Internet Security Association and Key Management Protocol (ISAKMP [20]). This is a very complex family of protocols, designed to provide negotiability of algorithms and parameters, optional identity protection for the participating parties, and a number of authentication options.

IKE is divided into two phases. *Phase 1* is used by two peers to establish a secure, authenticated channel over which to communicate; this is referred to the ISAKMP Security Association (SA). During *Phase 2*, those peers go on to negotiate Security Associations to be used by IPsec or other services. Phase 2 traffic is secured using the symmetric keys agreed on as part of the SA negotiated in Phase 1, and therefore is unchanged in our scheme. As part of authenticating each other during Phase 1, the two parties exchange their “identities”, in one of several forms, e.g. fully-qualified (DNS) domain names (FQDN), or IP addresses.

*Phase 1* can be accomplished in two ways, described as “modes”. *Main mode* provides identity protection for the communicating parties by protecting the identifying information they send to each other under either a key derived from an ephemeral Diffie-Hellman exchange, or a public key that they have previously exchanged. *Aggressive mode* is designed to be more efficient, and in

<sup>4</sup>At this writing, IKEv2 is currently under development. In order to experiment with our approach using current software, we have focused our efforts on IKE. Similar modifications should be possible with IKEv2.

<b>Initiator</b>	<b>Responder</b>
$x_i P$	$\longrightarrow$
	$\longleftarrow x_r P$
$K_i = \hat{e}(S_i, x_r P) \hat{e}(Q_r, x_i s_{rd} P)$	
$K_r = \hat{e}(S_r, x_i P) \hat{e}(Q_i, x_r s_{id} P)$	
$K = K_i = K_r = \hat{e}(x_r S_i + x_i S_r, P)$	
$K_{psk} = \text{hash}(K, x_i x_r P)$	

Symbols are defined as follows:

$P, q$	subgroup generator and subgroup, part of domain's group-params
$x_i$	initiator's ephemeral elliptic curve Diffie-Hellman private value, $x_i \in_R \mathbb{Z}_q$
$x_i P$	initiator's ephemeral elliptic curve Diffie-Hellman public value
$x_r, x_r P$	analogous values for responder
$s_{id}, P_{id} = s_{id} P$	initiator's domain's master-secret and domain-public-key
$s_{rd}, P_{rd} = s_{rd} P$	responder's domain's master-secret and domain-public-key
$Q_i, Q_r$	mapToPoint of the initiator and responder's identities, i.e., $Q_i = H(ID_i), Q_r = H(ID_r)$
$S_i, S_r$	initiator and responder's private keys, $S_i = s_{id} Q_i, S_r = s_{rd} Q_r$
$\hat{e}$	an admissible pairing function, $\hat{e}(\text{point}_1, \text{point}_2)$
$K_i, K_r$	method used by initiator and responder, respectively to compute the shared key
$K$	resulting shared key computed by both parties
$K_{psk}$	final shared session key without key escrow

Figure 5: Identity-based key exchange algorithm using different domain-public-keys but the same group-params [5].

general does not provide identity protection – participants' identities are sent in the clear as part of their first exchanges.<sup>5</sup>

Finally, Phase 1 (both main and aggressive modes) can be authenticated using one of four protocols: signature-based authentication, two forms of authentication using public key encryption (both of which do provide identity protection in aggressive mode), and authentication using a pre-shared key.

All components of IKE are designed to support a variety of cryptographic algorithms, key lengths, and parameters. Acceptable choices for these variables are listed by the initiator in *proposal* payloads, as part of the first security association (SA) negotiation message it sends to the responder; the responder replies with the single proposal of its choice. This extensibility allows us to incorporate IBC seamlessly into Phase 1, simply by identity-based algorithms as alternatives in these proposal payloads, as long as they can fit into the flows used by IKE's

<sup>5</sup>IKE uses two other "modes" – *quick mode* is what is used to perform Phase 2 key exchanges, and *new group mode* can be used after a Phase 1 exchange to change the cryptographic group used by the participants. As cryptographic groups can be negotiated during Phase 1, we present our discussion of group management in that context, and do not discuss either of these modes further.

three authentication protocol types. In the next section, we show how to map domain-based IBC onto each of these authentication protocols.

## 4.2 IBC-Based IKE

Using IBC in the signature and public-key based authentication modes for IKE Phase 1 is extremely straightforward. It will work even if the participants come from different domains, using unrelated domain-params to issue private keys. All that is required is to select an identity-based signature algorithm (e.g. [3, 16, 23]), and/or an identity-based encryption algorithm (e.g. Boneh and Franklin's IBE algorithm [2]). Given the selection of appropriate algorithm identifiers and a fixed format in which to exchange the resulting encryptions and signatures, these can be dropped directly into the standard protocol flows provided by IKE [15].

When using identity-based cryptography, the authenticity of a peer's public key is given merely because they prove possession of the private key corresponding to a given identity (sent by the peer as part of IKE, or known *a priori*), relative to the public system parameters of the domain that they claim to be a part of. As a result, the optional IKE messages provided for the ex-

change of certificates can be omitted. Additionally, all of IKE's key exchange protocols can provide perfect forward secrecy (PFS) by generating session keys not from long-term cryptographic secrets (e.g. IBC private keys), but instead using an optional ephemeral Diffie-Hellman key exchange authenticated by those long-term secrets. Combining perfect forward secrecy with IBC automatically avoids the key escrow facilities present in identity-based systems.

#### 4.2.1 IBC and Pre-shared Keys

If the domain-params of the two parties are related, we have another option. At the limit, if the two parties belong to the same domain (*i.e.* have the same domain-params – the same group-params and domain-public-key), and they know each others' identities *a priori*, they can use noninteractive IBC-based key exchange protocols to establish a shared secret key without sending any messages at all [14, 25]. This approach is appealing (*e.g.* [1]), but only applicable to members of the same security domain, and results in a key that is subject to escrow. In practice, hosts using IKE to establish security associations already have to exchange a number of preliminary messages, *e.g.* nonces for freshness, proposals for choices of algorithms, or keying information for PFS. Therefore, they may not be able to take best advantage of the noninteractive nature of these protocols. Additionally, the hosts involved, the responder in particular, may not know the other's identity *a priori* unless it is available as the IP address in current use, or a hostname available through reverse DNS. While these noninteractive protocols can be slightly more computationally efficient than other approaches to using IBC, the narrow set of circumstances in which they can be used, and the potential difficulty in determining whether those circumstances actually apply, make them less appealing.

If the group-params of the two parties are the same, regardless of whether their domain-public-keys are different, then they can use a key exchange protocol similar to (but slightly less efficient than) the noninteractive protocols described above [4, 5]. This would happen if they belonged to different domains, and those domains used the same choice from among the standard sets of group-params. This protocol is illustrated in Figure 5. The resulting protocol avoids the shortcomings of the noninteractive protocols – it is applicable to hosts from different domains, and does not suffer from key escrow. The resulting protocol is effectively a pre-shared key protocol that uses additional elliptic curve Diffie-Hellman information in the computation of the session key. These additional Diffie-Hellman values are directly analogous

to the Diffie-Hellman values used in IKE to provide PFS (and in fact do act here to provide PFS), and can be exchanged in the same key exchange (KE) message that standard Diffie-Hellman values would be. The resulting protocol fits neatly into IKE's pre-shared key authentication method, and is illustrated in Figure 6.

The only limitation of the IKE pre-shared key protocol in general is that the two peers do need to know each other's identities – whether they are using IBC (so they can compute the key) or share a traditional static key (so they know which key to use). That means that either they must use aggressive mode so that the identities are exchanged in the first set of messages, or the initiator must know the identity of the responder, either *a priori* or because the responder's identity is either its IP address or a hostname available through reverse DNS lookup.

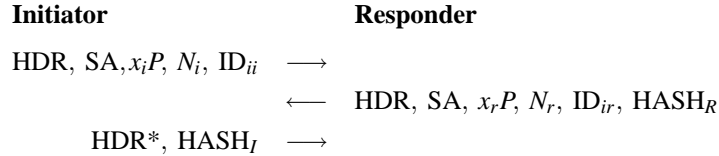
To use IBC in PSK mode, both peers must know that they are using the same IBC group-params. To achieve this, they exchange information about the group-params in the proposal payloads they use during SA negotiation to suggest the use of PSK. IKE provides standard mechanisms for exchanging group information in the proposal payloads, which were designed originally to identify the Diffie-Hellman groups used for achieving PFS. The same approach can be used to identify IBC group-params, and supports both the description of arbitrary group-params and the use of short identifiers that indicate the use of commonly used standard sets of group-params. Such standardized sets of parameters are used by most hosts for Diffie-Hellman exchanges in IKE, and we anticipate that such standard group-params would be used by most domains in DNSIBC.

## 5 Implementation

We have built a preliminary version of this system under Linux. Our implementation takes advantage of extensive DNSSEC support present in both the standard implementation of `bind`, the Unix DNS server program, and in `pluto`, the IKE daemon used by Free S/WAN, the most common IPSEC implementation for Linux.

### 5.1 IBC Libraries

For our initial implementation we wrote a 100%-Java implementation of the low-level field, elliptic curve, and Tate pairing operations necessary to perform identity-based cryptography. This work is based on the C implementation of identity-based encryption available at [28]. Our Java library is used by our PKG server and email



Keys and authentication values used in protocol are computed as follows:

$$\begin{aligned}
 \text{SKEYID} &= \text{prf}(K_{psk}, N_i \parallel N_r) \\
 \text{HASH}_I &= \text{prf}(\text{SKEYID}, x_iP \parallel x_rP \parallel \text{CKY-I} \parallel \text{CKY-R} \parallel \text{SA}_i \parallel \text{ID}_{ii}) \\
 \text{HASH}_R &= \text{prf}(\text{SKEYID}, x_rP \parallel x_iP \parallel \text{CKY-R} \parallel \text{CKY-I} \parallel \text{SA}_i \parallel \text{ID}_{ir})
 \end{aligned}$$

Symbols are as in Figure 5, with additional symbols defined as follows:

HDR	ISAKMP header
$nx_iP, x_rP$	as in Figure 5, sent in an ISAKMP KE payload
HDR*	ISAKMP header, payload encrypted under $x_i x_r P$ (also used in computation of $K_{psk}$ )
SA	SA negotiation payload with one or more proposals from initiator, one choice from responder
$N_i, N_r$	initiator and responder nonces
HASH <sub>I</sub> , HASH <sub>R</sub>	initiator and responder hashes
prf(key, msg)	keyed pseudorandom function
CKY-I, CKY-R	initiator and responder cookies, respectively
SA <sub>i</sub>	the body of the entire SA payload sent by the initiator

Figure 6: Identity-based version of IKE’s pre-shared key authentication protocol. Aggressive mode is illustrated here, main mode is similar.

client. We are working to complete a C port of our library for use in our IKE implementation.

## 5.2 DNS Support

We have modified a DNSSEC-compliant version of the Unix name server program, BIND, to support the distribution of signed parameter and salt records, as shown in Figure 4. These parameters are initially inserted in the DNS during the setup phase of the PKG, and are updated as necessary.

The PARAMS resource record type is implemented as a modified KEY record, with RR type 44. The PSALT resource record type is implemented as a modified text (TXT) record type, with RR type 45.

## 5.3 Private Key Generation Service

In a fully deployed system, there are many ways to implement a PKG that provide different amounts of auto-configuration and different levels of protection on the domain master secret. In our implementation, we have chosen to maximize ease of use and simplicity of setup, in order to encourage deployment.

Our PKG is a standalone program written in Java. On

first configuration of a domain (or re-keying of an existing domain), the PKG runs a setup sub-program that allows an administrator to select one of the standard sets of domain parameters (see Section 2.1) or to generate her own. The administrator also indicates how the minimum interval permissible before compromised keys can be revoked; this is controlled by the salt lifetime (see Sections 2.1,2.2). The setup program then creates a master secret and initial salt, and stores both these and the system parameters in two files: one appropriate for use by the PKG service, and another suitable for incorporating into a DNS zone file. This latter step could also be implemented using DNS dynamic update.

The PKG service then starts on a machine inside the domain network. It listens for connections on a known port (5599), and secures them using SSL/TLS, using a self-signed certificate (obtained from Java’s `keytool`) that was previously distributed to clients. Clients connect to the service to obtain their keys either on first initialization, or on change of salt. Authentication of clients is done using the simple “in-vs-out” determination described in Section 2.3, based on the desired identity (email address or FQDN) provided by the client. Private keys, parameters, and the current salt are returned to clients as XML-encoded data protected by the SSL tunnel. Clients then store their new private keys in the location and manner appropriate to them.

To support salt updating, the current salt is passed either on the command line or in a configuration file. Updating the PKG to issue private-keys derived from the new salt is a simple matter of restarting the PKG.

## 5.4 Email Client

As a preliminary proof of concept, we have implemented a standalone mail client in Java that can send and receive email encrypted with IBE using domain parameters pulled from a DNS server modified as above.

To send encrypted email, our client uses the `dnsjava` package [29], which we modified to accept and parse the new `PARAMS` and `PSALT` DNS resource records. Our client encrypts the message using our Java IBC libraries, encodes it as an XML string, and sends it to the recipient's mail server using the `javax.mail` package.

Upon receiving the first encrypted email message, our client pulls the current salt and its own private-key from the PKG and stores them in a keystore in the local filesystem. On subsequent encrypted email messages received, the client queries the PKG for the latest salt if the current salt's TTL has expired; if the salt changes, it requests a new private-key. In our implementation, the domain-params are included in the private-key, so there is no need to perform a separate check for changed domain-params.

We note that decryption of email is completely transparent to the user: no interaction whatsoever is required pull keys and decrypt messages.

In future work, we would like to incorporate support for domain-based IBC parameters into the existing IBE-based plugins for Outlook and Eudora [28].

## 5.5 IKE Client

The basis of our initial implementation of an IBC-enabled IKE daemon is a modified version of `pluto`, the IKE daemon provided as part of FreeS/WAN [13]. This implementation was chosen because FreeS/WAN is widely used under Linux, and already provides extensive support for retrieving public keys from DNSSEC. As `pluto` does not provide complete support for use of public-key encryption to authenticate IKE exchanges, we are concentrating on implementing the IBC signature-based and preshared-key based modes described in Section 4. Again, our modified `pluto` is designed for autoconfiguration, requesting its private keys

as necessary from the PKG the first time it runs, and updating them on expiration of the domain's salt.

We use the fully-qualified domain name FQDN to identify IPsec hosts, because it allows us to easily support IBC-based IPsec to hosts that use DHCP to obtain their addresses, even if those hosts are currently roaming outside their home domains (a "road warrior" configuration). For those modes of IKE where the responder does not send the initiator their identity before it is needed to derive their public key (*e.g.* both modes authenticated with public key encryption, and main mode authenticated with pre-shared keys), the initiator must already know the responder's identity, or be able to use reverse DNS on their IP address to determine their identity. This is not an extreme limitation, as you frequently know with whom you are initiating a communication with. If it is an unacceptable limitation, an IP address can be used as a host's id without change to any of the above protocols.

## 6 Related Work

While the value of identity-based encryption for securing email has been recognized for some time [2], only recently have other uses for IBE, and IBC in general, begun to be explored. Much of this work has been at the level of cryptographic primitives, focusing on identity-based signature schemes ([3, 16, 23]) and key exchange protocols ([5, 25, 27]). While much of the work on identity-based cryptography has focused on the model where there is one global trust infrastructure, and one trusted IBC key generator, more recent work has begun to describe primitives that work with less restrictive trust models. This began with work on hierarchical organizations of IBC trust centers [14, 17], and has continued with the design of protocols which work between users in different trust domains that share some of their mathematical parameters [4, 5]. We have been able to make use of this latter work in our own (see section 4.2.1).

Appenzeller and Lynn [1] have suggested using the non-interactive identity-based key exchange protocols suggested by Sakai et al. and others [14, 25] to secure network traffic. While their work is very much in the spirit of ours, it suffers from a number of critical limitations. First, it will only support communication between hosts in the same IBC trust domain. As we note in the introduction, a global IBC trust system is not a realistic deployment scenario. Second, it is a non-standard, special-purpose protocol. As such, it has not been analyzed in any detail, and has no deployment support. In contrast, our approach is to support communicating peers who do

not belong to the same trust domain, and to enable IPsec to use IBC to secure network traffic. While IPsec itself is not a perfect protocol, it is extremely widely deployed, studied and supported, and is subject continuing improvement. Therefore, the general approach to IBC-enabled IPsec presented here seems the most effective way to leverage IBC's deployment advantages to secure network traffic.

## 7 Advantages Over Alternative Approaches

We believe that our approach offers a number of advantages over existing methods for key distribution to secure email and network traffic. In particular, we believe that our scheme, with its emphasis on autoconfiguration, makes it simple enough to deploy these technologies that they could begin to see much more widespread use. In this section, we compare DNSIBC to alternative approaches.

### 7.1 Distribution of Trust

An important feature of DNSIBC is the idea of domain-based trust. This is in contrast to standard IBC approaches requiring the establishment of a global trust system [2, 1], which engenders a tremendous management problem – who gets to manage such a system, even if it is distributed, and how do they authenticate requests for private keys – and results in a facility for global key escrow and key compromise.

Our approach is based on a hierarchical distribution of trust, similar to that used by traditional PKIs and hierarchical IBC schemes [14, 17].<sup>6</sup> In contrast to traditional PKIs, however, our approach links its hierarchical organization directly to that of the DNS, rather than having organizations create PKI nodes as a function of their internal organizational structure. Using a domain-based approach, whether for IBC or even a traditional PKI, has the advantage that the things we are intending to authenticate are email senders and network hosts, whose identities derive directly from domains as structured in the DNS. This approach also dramatically simplifies the task faced by someone outside of a given domain who wishes to communicate securely with someone inside that domain, by making it easy for them to find out whether or not there is a cryptographic trust system in place (e.g. IBC system or PKI) in that domain, and to know where to look for credentials in that trust system.

<sup>6</sup>See Section 1.1 for discussion of the shortcomings of current hierarchical IBC schemes.

A domain-based approach also eases autoconfiguration and system setup, both for the system administrator, and the end user. If basic credentials for securing email and network traffic are organized according to the domain, a simple default implementation of a system to create and manage such credentials can be provided along with the other tools used to manage a domain, such as DNSSEC tools now come along with the name server, bind. Domains with more sophisticated security needs and resources can replace these simple implementations with something more complex, but they may be good enough for many domains that currently find themselves unable to set up and manage a PKI from “scratch”. A domain-based system which uses DNSSEC to root its trust has the added advantage that it removes yet another energy barrier to deployment. Although it is rolling out slowly, there are very good practical reasons for full deployment of DNSSEC. Trust infrastructures that inherit from DNSSEC (e.g. by using your domain or zone's DNSSEC keys to sign and hence authenticate your domain IBC parameters) can take advantage of this momentum, and are therefore much easier to deploy in practice than setting up yet another trust hierarchy whose organization mirrors that of the DNS.

### 7.2 Use of Identity-Based Cryptography

We have argued strongly above for the practical advantages of domain-based, standardized trust systems. Why, then should we implement such a system with IBC, rather than say, having each domain directly certify the keys [9] or digital certificates [10] of end-entities with DNSSEC, and distribute them through the DNS? Or perhaps have an LDAP server running which maintains a list To see the advantages of IBC in these situations, it is illustrative to focus on the clients – IBC has its strongest advantages there.

#### 7.2.1 Versus Storing Certificates in DNS

Why use IBC, rather than distributing keys or certificates via the DNS? Distribution of a domain root certificate via the DNS would give us a domain trust model similar to DNSIBC, and would make it easy for clients from different domains to find the trust root for their desired communication partner. We could even, in the extreme, automate a domain's certification authority so that clients (email users and network hosts) could automatically request certificates when they needed them. Such a system is actually currently implemented in Microsoft Windows 2000<sup>TM</sup> Active Directory-based domains that run a Microsoft Certification Authority [7].

Machines belonging to the domain can be configured to automatically request an IPsec certificate when they first join the domain, and that certificate is stored in Active Directory, which also can be used to store and distribute user email certificates. That particular approach is limited to a particular vendor's client and server software, and limits access to the stored certificates to members of the domain, but it could obviously be generalized.

We suggest a number of reasons why IBC might be a better approach. First of all, it minimizes the number of interacting parties in the system, and in particular, the number of parties that need to update the DNS zone information. Using IBC, domain parameters need to be made available by the DNS, but no per-client information needs to be there. In a certificate-based approach, each client needs to place their certificate information into the DNS. An IBC approach also dramatically reduces the bandwidth required to access peers' credential information. To communicate with any number of peers in a given domain, I only need to obtain that domain's parameters once per revocation interval. I can then communicate securely with any email user in the domain, or any domain host, for which I know an address. I can also cache that information and make it available to a population of querying hosts using standard DNS caching software.

Another advantage of this approach is that using it, I can communicate securely with any host or email user whose address I know – but only those whose addresses I know. If every user in a domain has their email address directly represented in the DNS in the form of their digital certificate, “fishing expeditions” to find user identities or the distribution of hosts become much easier.

And finally, this approach preserves the appealing use model of IBC. I can send encrypted email to a user that has not yet bothered to get the private key necessary to decrypt it, or even perhaps to install the software or plug-in necessary to support IBC. Having received such an encrypted email, that user is then considerably more motivated to perform the necessary steps to decrypt it, after which he will continue to seamlessly participate in the system. Similarly, it becomes possible to support both autoconfiguration of IPsec hosts who can retrieve their own keys as part of their setup process, and seamless IPsec termination by trusted proxies provided by the domain for devices not capable of terminating IPsec on their own.

## 7.2.2 Versus Dynamic Certificate Generation

Lastly, we might consider using instead a system with dynamic or “lazy” certificate generation. An LDAP certificate server could be set up which, if a user or host already has a certificate, returns it. If not, it generates a key pair, makes the certificate available to the outside world, and keeps the private key to be later transmitted to the user or host.

Our IBE-based approach has several advantages over a system such as this. First, in IBE, the process of generating a user's private key is decoupled from the generation of their public key (which is, of course, just their identity.) This allows us to introduce an “air gap” in between the private key generator and the outside world: the private key generator need only be accessible by users or machines within the domain. In contrast, our hypothetical LDAP certificate server, which is on the outside of a domain's firewall, must maintain connectivity with the private key generator at all times, introducing a possible path for an attacker to the private key repository.

Furthermore, this LDAP certificate server must either validate requests or generate key pairs for every request that is made. For instance, the LDAP server either maintains an up-to-date list of email addresses, which an attacker could then quickly probe; or, it generates key pairs for every requested email address, which opens up vulnerability to denial-of-service by flooding the server with bogus requests. In our IBE-based approach, no such attacks are possible.

Lastly, a major strength of integrating identity-based encryption parameters into the Domain Name Service is the propagation and redundancy the DNS provides via caching. In our hypothetical LDAP system, a single service must be contacted in order to send encrypted email to a user in a domain. In our scheme, the identity-based encryption parameters for that domain propagate through the DNS and can be cached locally.

## 8 Conclusions

We have presented an approach to protecting email and network traffic using identity-based cryptography and domain-based trust. We think that this system provides a simple and easy way to establish widespread support for secured communication, through its thorough support for autoconfiguration, and identity-based cryptog-

raphy's novel solution to the key distribution problem. We have built an initial implementation of this system in Linux as a proof of concept of its effectiveness and usability.

## Acknowledgments

The authors would like to thank the referees for their many helpful comments.

## References

- [1] G. Appenzeller and B. Lynn. Minimal overhead IP security using identity-based encryption. Submitted for publication, <http://rooster.stanford.edu/~ben/pubs/ipibe.pdf>.
- [2] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Proc. CRYPTO 01*, pages 213–229. Springer-Verlag, 2001. LNCS 2139.
- [3] J. Cha and J. Cheon. An identity-based signature from gap diffie-hellman groups. <http://eprint.iacr.org/2002/018>.
- [4] L. Chen, K. Harrison, N. P. Smart, and D. Soldera. Applications of multiple trust authorities in pairing based cryptosystems. In *Proceedings of Infrastructure Security: InfraSec 2002*, pages 260–275. Springer-Verlag, 2002. LNCS 2437.
- [5] L. Chen and C. Kudla. Identity based authenticated key agreement from pairings. <http://eprint.iacr.org/2002/184>.
- [6] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, pages 360–363. Springer-Verlag, 2001. LNCS 2260.
- [7] J. de Clercq. PKI comes of age. *Windows & .NET Magazine*, pages 47–53, May 2002.
- [8] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. *S/MIME Version 2 Message Specification*. IETF - Network Working Group, The Internet Society, March 1998. RFC 2311.
- [9] D. Eastlake. *Domain Name System Security Extensions*. IETF - Network Working Group, The Internet Society, March 1999. RFC 2535.
- [10] D. Eastlake and O. Gudmundsson. *Storing Certificates in the Domain Name System (DNS)*. IETF - Network Working Group, The Internet Society, March 1999. RFC 2538.
- [11] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
- [12] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 186–194. Springer, 1987. Lecture Notes in Computer Science No. 263.
- [13] Free S/WAN Project. Free S/WAN. <http://www.freeswan.org>.
- [14] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Advances in Cryptology - Asiacrypt 2002*. Springer-Verlag, 2002.
- [15] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*. IETF - Network Working Group, The Internet Society, November 1998. RFC 2409.
- [16] F. Hess. Exponent group signature schemes and efficient identity based signature schemes based on pairings. <http://eprint.iacr.org/2002/012>.
- [17] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Proc. EUROCRYPT 02*, pages 466–481. Springer-Verlag, 2002. LNCS 2332.
- [18] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*. IETF - Network Working Group, The Internet Society, November 1998. RFC 2401.
- [19] D. Massey and S. Rose. *Limiting the Scope of the KEY Resource Record*. IETF - Network Working Group, The Internet Society, December 2002. RFC 3445.
- [20] D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*. IETF - Network Working Group, The Internet Society, November 1998. RFC 2408.
- [21] P. Mockapetris. *Domain Names – Concepts and Facilities*. IETF - Network Working Group, The Internet Society, November 1987. RFC 1034.
- [22] P. Mockapetris. *Domain Names – Implementation and Specification*. IETF - Network Working Group, The Internet Society, November 1987. RFC 1035.
- [23] K. Paterson. ID-based signatures from pairings on elliptic curves. <http://eprint.iacr.org/2002/004>.
- [24] D. Piper. *The Internet IP Security Domain of Interpretation for ISAKMP*. IETF - Network Working Group, The Internet Society, November 1998. RFC 2407.
- [25] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Proceedings of the Symposium on Cryptography and Information Security (SCIS 2000)*, Okinawa, Japan, January 2000.
- [26] A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. C. Chaum, editors, *Proc. CRYPTO 84*, pages 47–53. Springer, 1985. Lecture Notes in Computer Science No. 196.
- [27] N. Smart. An identity based authenticated key agreement protocol based on the weil pairing. *Electronics Letters*, 38:630–632, 2002.
- [28] Stanford Applied Cryptography Group. IBE secure e-mail. <http://crypto.stanford.edu/ibe>.
- [29] B. Wellington. dnsjava: An implementation of DNS in Java. <http://www.xbill.org/dnsjava/>.