Lightweight Encryption for Email

Ben Adida
MIT
ben@mit.edu

Susan Hohenberger
MIT
srhohen@mit.edu

Ronald L. Rivest
MIT
rivest@mit.edu

Abstract

Email encryption techniques have been available for more than a decade, yet none has been widely deployed. The problems of key generation, certification, and distribution have not been pragmatically addressed. We recently proposed a method for implementing a Lightweight Public Key Infrastructure (PKI) for *email authentication* using recent developments in identity-based cryptography and today's existing Internet infrastructure.

While this solution works well for email authentication, *email encryption* exhibits a different threat model that requires special treatment. In this work, we discuss how to achieve email encryption and present a realistic deployment and adoption process, while respecting the current functionality and expectations of email.

1 Introduction

Email is a mostly insecure communication medium. Email encryption solutions such as the well-known PGP [14] and S/MIME [10] have existed for more than a decade, yet neither has achieved widespread use. This is due, in large part, to the complexity of key management: a user must generate a keypair, certify and distribute his public key, and obtain a validated public key for all intended recipients.

Even identity-based encryption [12], which proposes to compute public keys directly from users' email addresses (or other identity-related strings), presents key management complications. No realistic, practical architecture has been proposed for making use of identity-based encryption in an Internet-wide setting.

Recently, we proposed and implemented a Lightweight PKI [2, 1] to manage keys for email authentication. We now propose extensions to this architecture for the purpose of email encryption. We call this approach Lightweight Encryption.

1.1 Prior Key Management Strategies

Public-key encryption has been around for 25 years. In its basic form, it is well understood: a public key allows for encryption, while an associated private (a.k.a. secret) key performs decryption. The complication lies in associating a public key with a user. How does Bob obtain Alice's public key? How can Bob be certain that the public key he has obtained is indeed Alice's, and not some eavesdropper's?

In classic public-key cryptosystems like RSA [11], El Gamal [7], or Cramer-Shoup [5], each user generates a keypair. The association between a public key and an identity is then certified by the digital signature of some authority. With S/MIME [10], these certification authorities form an organizational hierarchy. With PGP [14], an individual trusts a peer-to-peer certificate chain.

In identity-based public-key cryptosystems, first conjectured in 1984 [12] but only fully implemented in 2000 [4], a master authority generates a master keypair (MPK, MSK) and publishes MPK to the world. A user's public key is then the combination of MPK and the string id_string representing the user's identity. The user's secret key SK is computable from id_string only by a master authority in possession of MSK who delivers this key securely to the user. Though identity-based schemes simplify user-key management, there remains a domain-key management problem: the MPK-domain association must be safely distributed, and the user secret keys must be securely delivered.

1.2 The Lightweight PKI

We recently introduced Lightweight PKI, a mechanism for Internet-wide distribution of identity-based public keys for the purpose of email authentication [2, 1]. Each email domain becomes a master authority for an identity-based scheme *of its choosing* and generates a unique master keypair (MPK, MSK). Each MPK is dis-

tributed via the Domain Name System (DNS), as a TXT record associated with the hostname of the domain's Mail Exchange (MX) record. (Interestingly, Lightweight PKI automatically inherits security from any future improvements to DNS [6].) Email users obtain their secret key SK by email-based identification: the master authority delivers the key directly to the user's inbox. Key revocation is handled by using short-lived keys: the identity string of a public key includes an expiration date [2].

Thus, Alice can sign each of her messages with her secret key. Upon receiving a signed email from alice@wonderland.com, Bob can look up $MPK^{\texttt{wonderland.com}}$ via the DNS record for wonderland.com . Then, Bob can compute Alice's public key using $MPK^{\texttt{wonderland.com}}$ and the string alice@wonderland.com, and verify the signature.

Lightweight Signatures strike a practical compromise. They reasonably assume that Alice's mail server will not actively attack Alice. Then, using only the established infrastructures of DNS and email delivery, they make spoofing outgoing email from Alice as difficult as consistently intercepting Alice's incoming email.

1.3 Insufficient Security for Encryption

Consider deploying Lightweight PKI for encryption. In this case, a *passive* adversarial incoming mail server could easily decrypt and read a user's encrypted email. Moreover, even if Alice's mail server is honest, the compromise of the master authority's MSK would reveal all prior encrypted emails for *all* users of that domain.

We must take extra precautions to help honest domains secure their master key and honest users protect their privacy. We conclude two necessary principles for encryption:

1. a domain's MSK cannot exist on a single machine.
2. only Alice should know her decryption key SK_{Alice}.

Much like Lightweight Signatures, our practical threat model does not defend against *active* mail server adversaries nor the total compromise of an end-user's personal computer. However, we do provide a reasonable privacy guarantee for users, using (primarily) the existing Internet infrastructure.

Good Enough Security? In certain limited cases, the unmodified deployment of Lightweight Signature keys may be good enough for encryption. It certainly achieves better privacy than the majority of users enjoy today. However, we can do better within the same deployment constraints.

1.4 Our (Two-Part) Solution

In this paper, we describe how to use the traditional approach of key splitting and distributed key generation [9] in our identity-based framework. We apply this technique in two ways, first to protect honest domains from single-attack compromise, then to protect honest users from overly curious incoming mail servers.

Protecting Honest Domains. A domain's MSK can retroactively decrypt all messages encrypted against its public counterpart. Thus, we consider MSK so valuable that it should never be stored on a single computer. However, it must remain functional enough to compute keys for new users on demand.

We propose that a domain maintain several servers that independently generate master key shares (MPK_i, MSK_i). The master public key shares are combined into a single MPK that is distributed via the DNS. Each server with MSK_i individually sends Alice secret key share $SK_{Alice,i}$. Alice can then combine all shares into a single secret key SK_{Alice}. As expected, SK_{Alice} correctly decrypts ciphertexts computed against MPK and Alice's identity string id_string.

Protecting Honest Users. Even with MSK split amongst multiple servers, Alice's secret key shares can be intercepted by her *passive*, incoming mail server. Effectively, Lightweight PKI is insufficient for encryption because it provides a single, imperfect communication channel: email is decrypted using the single key $SK_{Alice}^{\texttt{wonderland.com}}$ issued by the master domain for wonderland.com, which is known to both Alice and her incoming mail server.

Our solution is thus to set up multiple channels, all of which are necessary to perform decryption. Email-based identification can provide lightweight certification as long as it defines one of these channels. To ensure Alice's privacy, only she should have access to all channels simultaneously.

Alice can, in fact, create one of these channels on her own. She generates $(MPK^{Alice}, MSK^{Alice})$ using parameters compatible with $MPK^{\texttt{wonderland.com}}$ and publishes MPK^{Alice} via the mechanism of her choice, e.g. her web page. Then, her complete decryption key is a combination of $SK_{Alice}^{\texttt{wonderland.com}}$ and SK_{Alice}^{Alice}. Notice that, although Alice generates one key share on her own, she does not need to obtain certification for it, since an active adversary who spoofs it will not have $SK_{Alice}^{\texttt{wonderland.com}}$ to read her email. We return to these issues in Section 3.3.

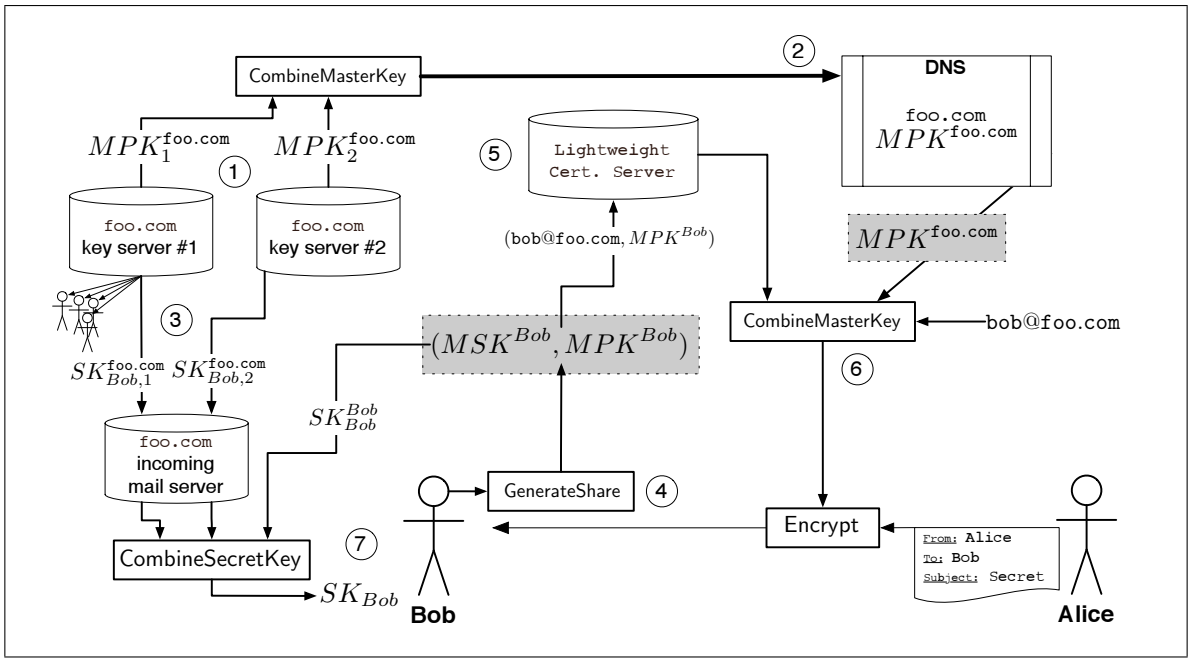

Figure 1: Lightweight Encryption: (1) The domain sets up two key servers, each in possession of a share of MSK, (2) The domain's shares of the master public key are combined into a single key $MPK^{\texttt{foo.com}}$ stored in the DNS, (3) each domain key server emails Bob a share of his secret key $SK_{Bob}^{\texttt{foo.com}}$ for that domain, (4) To achieve privacy from his mail server, Bob generates his own master keypair and stores his secret key SK_{Bob}^{Bob}, (5) Bob publishes his uncertified master public key MPK^{Bob} via a keyserver or simple web page, (6) To encrypt to Bob, Alice retrieves his two master public keys and combines them, (7) To decrypt, Bob combines his three secret keys.

Splitting and Combining Keys. An honest domain is protected by splitting and recombining the MSK, while an honest user is protected by creating a new master key share and recombining it with the domain's MPK. In both cases, our technical contribution is the application of these key splitting-and-recombination techniques in the context of identity-based encryption. We review a known approach for the popular Boneh-Franklin IBE scheme [4] and describe, for the first time, such a technique for the new Waters IBE system [13] as well.

1.5 Related Work

Since the early days of encrypted email with S/MIME [10] and PGP [14], the public-key infrastructure requirement has been noted as hampering widespread deployment [3]. Numerous approaches have been proposed in the intervening years. Most recently, Garfinkel suggested viral public key distribution and key continuity [8] in order to leverage existing communication channels and reduce the need for a public-key infrastructure. Both of these techniques can be combined with our work to strengthen the overall system.

1.6 Roadmap

In Section 2, we review identity-based cryptography. In Section 3, we present Lightweight Encryption at a high level. The technical details of this solution are in Section 4. Finally, we briefly discuss some extensions and variations of our solution in Section 5, before concluding in Section 6.

2 Preliminaries

With identity-based cryptography, a user's public key is the combination of MPK and id_string, usually the user's email address. More specifically, an identity-based key management interface offers the following calls:

- Generate(key_length): outputs (MPK, MSK), a master public key and corresponding master secret key of prescribed key length.
- ExtractSecretKey(MSK, id_string): outputs SK, the secret key that corresponds to the user id_string in domain MPK.
- VerifySecretKey(MPK, id_string, SK): outputs True only if SK matches id_string in the master domain designated by MPK.

The identity-based encryption (IBE) interface is then:

- Encrypt(MPK, id_string, m): encrypts m for user id_string in domain MPK.
- Decrypt(SK, c): decrypts c with secret key SK.

3 Lightweight Encryption

As previously discussed, there are two potential problems with applying the Lightweight PKI, in its basic form, to email encryption.

First, the compromise of an MSK causes significant damage for all users of the domain. Thus, our first goal is to provide domains with more safeguards for protecting this key. Second, Alice's incoming mail server for domain `wonderland.com` knows all of Alice's secret cryptographic keys, specifically $SK_{Alice}^{\texttt{wonderland.com}}$. Thus, all of Alice's incoming encrypted mail can be decrypted and read by her mail server. Our second goal is to provide users with more privacy, such that no passive adversary, not even Alice's incoming mail server, can ever read encrypted email destined for Alice.

Both of these goals can be achieved by either splitting (first case) or combining (second case) the public or secret keys of IBE schemes. We begin by describing this functionality.

3.1 Key Splitting and Combining

We rework the identity-based function calls from Section 2 to enable: (1) combining stand-alone MPK_i's into a single MPK, (2) combining stand-alone $SK_{id\_string,i}$'s into SK_{id_string}, and (3) splitting the key generation of MSK and the extraction of SK_{id_string} among many servers.

- GenerateShare$(params)$
  outputs (MPK_i, MSK_i), a share of a master public and corresponding master secret key.
- CombineMasterKey$(MPK_1, \ldots, MPK_n)$
  outputs MPK, the master public key corresponding to the n input public key shares.
- ExtractSecretShare(MSK_i, id_string)
  outputs SK_i, the user key share that corresponds to the user id_string and the master key share MSK_i.
- VerifySecretShare(MPK_i, id_string, SK_i)
  outputs True or False, depending on whether SK_i corresponds to the secret key share for input id_string against master key share MPK_i.
- CombineSecretKey$(SK_1, \ldots, SK_n)$
  outputs SK, the secret key corresponding to the n input secret key shares.

We stress an important feature: a secret key share $SK_{id\_string,i}$ generated against a master share MPK_i can function as a stand-alone key, with MPK_i the stand-alone master. Each key share is a fully functional key, if need be. This feature enables the reverse operation of key recombination from existing, independent, fully-functional keys.

3.2 Protocol for Email Domains

Using the functionality from Section 3.1, a given email domain can use multiple servers to manage its master keypair. These servers can all be online to send Alice her secret key shares, yet any adversary wanting to hijack the domain must hack into *every* server to reconstruct MSK. Individuals wishing to send Alice an encrypted email can retrieve the pre-combined MPK through the DNS, against which they can form an encryption. They need not know that the corresponding MSK is shared among multiple servers.

3.3 Protocol for Users

Alice, with address `alice@wonderland.com`, can use functionality from Section 3.1 to prevent her mail server from reading her incoming emails as follows:

1. generate a fresh master keypair $(MPK^{Alice}, MSK^{Alice})$ using the same $params$ as those of `wonderland.com`.
2. distribute $(\texttt{alice@wonderland.com}, MPK^{Alice})$ via a keyserver, web site, etc. The association between `alice@wonderland.com` and MPK^{Alice} need not be certified.
3. generate a secret key complement SK_{Alice}^{Alice} using the string `alice@wonderland.com` and MSK^{Alice}. Run CombineSecretKey on $SK_{Alice}^{\texttt{wonderland.com}}$, from her domain, and the newly created SK_{Alice}^{Alice} to obtain a single secret key SK_{Alice}.

Then, when Bob wishes to send Alice an encrypted email at address `alice@wonderland.com`, he performs the following actions:

1. obtain $MPK^{\texttt{wonderland.com}}$ using the established DNS-based mechanism of the Lightweight PKI.
2. obtain MPK^{Alice} from a keyserver or Alice's web site.
3. combine the two master public keys: MPK = CombineMasterKey$(MPK^{\texttt{alice@wonderland.com}}, MPK^{Alice})$.
4. encrypt a message m for Alice: Encrypt$(MPK, \texttt{alice@wonderland.com}, m)$.

With this approach, Alice prevents her mail server from decrypting and reading her email with only $SK_{Alice}^{\texttt{wonderland.com}}$. Even an adversary (other than Alice's mail server) who spoofs Alice's uncertified key

MPK^{Alice} *cannot* decrypt and read her email. Our threat model does not protect against a mail server that spoofs its own user's webpage or keyserver entry, as this type of attack is unlikely in practice.

However, an active attacker – other than Alice's mail server – could mount a denial-of-service attack against Alice by publishing spurious master public keys associated with Alice's email address. We suggest methods for preventing this attack in Section 5.

3.4 Adoption and Deployment

Lightweight encryption offers flexible deployment options. Privacy increases with each domain deployment of a Lightweight PKI, and even naive users get some privacy. We explore two usage cases:

Scenario One: Naive Users. Using the Lightweight PKI, non-savvy users can use an updated mail client which automatically encrypts their outgoing mail and decrypts their incoming mail. To encrypt to `bob@foo.com`, Alice's client simply needs to obtain $MPK^{\texttt{foo.com}}$ via DNS. Recall that even if this domain splits its master secret key among many servers to safeguard against hackers, it need only post this combined key in the DNS. Decryption is even easier. The mail server for `foo.com` emails Bob his secret key $SK_{Bob}^{\texttt{foo.com}}$. Bob's email client can transparently recognize and process such emails, then using the included key to decrypt the email from Alice.

Scenario Two: Advanced Users. Suppose Alice and Bob are more advanced users, able to follow the protocol in Section 3.3. Bob publishes an additional public key on his web site. Alice combines this public key with the one defined by $MPK^{\texttt{foo.com}}$. Her email to Bob is then encrypted such that even Bob's incoming mail server is unable to decrypt it: only Bob can. Note that Bob never needs to certify any of this additional key material.

4 How To Split IBE Master Keys

We present methods for splitting, among a group of trustees, the key generation and verification algorithms for two predominant IBE systems. Both key types [4, 13] are based on bilinear maps and support efficient identity-based encryption and signature schemes [1].

4.1 Bilinear Maps

Let BM_Setup be an algorithm that, on input the security parameter 1^k, outputs (q, g, h, G_1, G_2, e), where e is a function mapping $G_1 \times G_1$ to G_2, where both G_1 and G_2 are groups of prime order $q = \Theta(2^k)$, and elements g and h both generate G_1. The function e has the properties [4]: (*Bilinear*) for all $g, h \in G_1$, for all $a, b \in \mathbb{Z}_q$, $e(g^a, h^b) = e(g, h)^{ab}$; (*Non-degenerate*) if g is a generator of G_1, then $e(g, g)$ generates G_2; and (*Efficient*) computing $e(g, h)$ is efficient for all $g, h \in G_1$.

4.2 Waters Key Pairs

We present these algorithms for (a reformulation of) the Waters key pairs [13, 1] when *all trustees are assumed to be honest*. This is the case, for example, when a company wants to split its MSK among a number of servers that it owns and operates in order to make stealing the MSK more difficult for hackers. In Section 4.4, we address the possibility of malicious servers.

Single Server Key Algorithms

- Generate(1^k) outputs $MPK = (params, g^b)$ and $MSK = b$ for $params = (q, g, h, G_1, G_2, e, H)$ where H is a function mapping strings in $\{0,1\}^k$ to elements in G_1 and the remaining parameters are generated by running BM_Setup(1^k). We assume that the discrete logarithm of h with respect to g is unknown. (Here, H is a particular implementation of a hash function, not a generic random oracle. We defer to Waters for the details [13].)

- ExtractSecretKey(MSK, id_string) outputs the user secret key $SK = (h^b H(id\_string)^r, g^r)$, for a random $r \in \mathbb{Z}_q$.

- VerifySecretKey(MPK, id_string, SK) parses SK as (A, B), outputs True if and only if $e(A, g)/e(B, H(id\_string)) = e(h, g^b)$.

Multiple Server Key Algorithms

- Setup(1^k): $params = (q, g, h, G_1, G_2, e, H)$.

- GenerateShare$(params)$ outputs a master key share $MPK_i = g^{b_i}$ and the corresponding secret key share $MSK_i = b_i$.

- CombineMasterKey$(MPK_1, .., MPK_n)$ outputs

$$MPK = \prod_{i=1}^{n} g^{b_i} = g^b.$$

- ExtractSecretShare(MSK_i, id_string) outputs

$$SK_i = (h^{b_i} H(id\_string)^{r_i}, g^{r_i})$$

for a random $r_i \in \mathbb{Z}_q$.

- VerifySecretShare(MPK_i, id_string, SK_i) parses SK_i as (A_i, B_i), outputs True only if

$$e(A_i, g)/e(B_i, H(id\_string)) = e(h, g^{b_i}).$$

- CombineSecretKey($SK_1, \ldots, SK_n$) outputs

$$\begin{aligned} SK &= (\prod_{i=1}^{n} h^{b_i} H(ID)^{r_i}, \prod_{i=1}^{n} g^{r_i}) \\ &= (h^b H(ID)^r, g^r). \end{aligned}$$

4.3 Boneh-Franklin Key Pairs

Boneh and Franklin [4] briefly discuss how the key pairs for their encryption scheme can be generated and verified in a distributed fashion. The key pairs are simpler than the above Waters scheme. The master key pairs are of the form $(MPK, MSK) = (g^s, s)$ and user key pairs of the form $(PK, SK) = (H(id\_string), H(id\_string)^s)$, where $H : \{0,1\}^* \rightarrow G_1$ is a hash function. For the remaining details, we defer to the Boneh and Franklin [4].

4.4 Dealing with Untrusted Servers

There are two ways that a malicious server (or servers) can undermine the security of the previous schemes.

Key Generation Issues. It is possible for a set of colluding servers to deviate from the basic GenerateShare protocols above and choose their shares MPK_i in such a way as to bias the final master key MPK. An adversary might post a malicious second-channel key MPK^{evil} which "cancels out" Alice's first-channel key from $MPK^{\texttt{wonderland.com}}$, making the combined SK_{Alice} a known value.

We can prevent these attacks by requiring that each server posting an MPK also posts a proof that they know the corresponding value MSK *without revealing* MSK. This can be done by standard cryptographic techniques outside the scope of this paper.

Threshold Issues. After a set of n servers have published a final master key MPK, one or more malicious servers may refuse to provide Alice with their shares of her user secret key SK_{Alice}. It would be better if some subset of the servers, say any group of $k+1$ out of n for $n/2 \leq k < n$, can provide enough secret shares to reconstruct SK_{Alice} for Alice. This safeguard can be achieved by a direct application of techniques due to Gennaro et al. [9] *when the majority of servers are honest*, as previously noted for Boneh-Franklin key pairs [4].

5 Extensions and Variations

Stronger Lightweight Certificates. In Section 3, Alice creates her second encryption channel by posting MPK^{Alice}. An active adversary might post spurious MPK^{evil} in order to hijack this second channel. Since the adversary does not know the secret key $SK_{Alice}^{\texttt{wonderland.com}}$ for the first channel, the best he can achieve is a denial-of-service attack. In practice, we want to prevent such attacks too. Alice can do so by using her existing lightweight signing key $SK_{Alice}^{\texttt{wonderland.com}}$ to sign her second-channel MPK^{Alice}. If Alice's mail server were actively malicious, it could spoof this signed $MPK^{\texttt{wonderland.com}}$. However, as stated earlier, our threat model does not account for this (unlikely) attack.

Double Email-Based Identification. Lightweight encryption works because Alice has two encryption channels: one from her email domain, and one from herself. As we saw in Section 4, the two keys from these channels can be combined to allow for a single encryption by the sender and a single decryption by the recipient.

Another way to set up two encryption keys is to have Alice advertise two email addresses at different domains, `alice@wonderland.com` and `alice@school.edu`. As the two domains `wonderland.com` and `school.edu` are unlikely to be using the same key parameters (e.g. the same bilinear map in Boneh-Franklin or Waters), the two secret keys obtained from each domain cannot be recombined. However, the sender can simply use double-encryption: encrypt the message first against $MPK^{\texttt{wonderland.com}}$, then encrypt that ciphertext against $MPK^{\texttt{school.edu}}$. Neither incoming mail server can decrypt Alice's email single-handedly, yet Alice can perform two decryptions to recover the message. (Note that this only provides added security if Alice isn't simply forwarding her email from `alice@school.edu` to `alice@wonderland.com`. She must log in to each incoming mail server independently.)

6 Conclusion

We introduced Lightweight Encryption as a means of realizing email encryption with a realistic adoption and deployment process. Certainly, more work is necessary, including user interface considerations, the widespread distribution of secondary user private keys, and the real-world validation of our deployment ideas.

Authentic, private email can help improve the quality of all Internet-based communication. Lightweight encryption provides a solid building block towards making email encryption practical.

Acknowledgments. Susan Hohenberger's work was supported by an NDSEG Fellowship.

References

[1] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-Hoc Group Signatures (including ID-based ones) from Almost Any Collection of Key Pairs, 2005. Available at http://theory.lcs.mit.edu/~rivest/publications.html.

[2] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Lightweight Signatures for Email, 2005. Available at http://theory.lcs.mit.edu/~rivest/publications.html.

[3] Steven M. Bellovin. Cryptography and the internet. In *CRYPTO*, volume 1462 of LNCS, pages 46–55, 1998.

[4] Dan Boneh and Matt Franklin. Identity-based Encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[5] Ronald Cramer and Victor Shoup. A pratical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, volume 1642 of LNCS, pages 13–25, 1998.

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

[7] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO '84*, pages 10–18, 1984.

[8] Simson L. Garfinkel. *Design Principles and Patterns for Computer Systems that are Simultaneously Secure and Usable*. PhD thesis, MIT, April 2005.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, volume 1592 of LNCS, pages 295–310, 1999.

[10] IETF. S/MIME Working Group. http://www.imc.org/ietf-smime/index.html.

[11] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining dig. signatures and public-key cryptosystems. *Com. of the ACM*, 21,2:120–126, 1978.

[12] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, volume 196, pages 47–53, 1984.

[13] Brent Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, volume 3494 of LNCS, pages 114–127, 2005.

[14] Phil Zimmerman. Pretty Good Privacy. http://www.pgp.com.