10.6 LIGHTWEIGHT EMAIL SIGNATURES

Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest

Domain Keys and Identified Mail (DKIM) is a promising proposal for providing a cryptographic foundation to solve the phishing problem: domains are made cryptographically responsible for the email they send. Roughly, `bob@foo.com` sends emails via `outgoing.foo.com`, which properly identifies Bob and signs the email content. The public key is distributed via a DNS TXT record for `_domainkeys.foo.com`. The details of how DKIM should handle mailing lists, message canonicalization, message forwarding, and other thorny issues, are being resolved in the context of a recently-formed IETF Working Group [66].

We propose *Lightweight Email Signatures*, abbreviated LES, as an extension to DKIM. We envision LES as being fully compatible with DKIM, in that it should be supportable in principle within the flexible parameterized framework we forsee DKIM implementing and supporting. LES offers three significant improvements:

1. **Automatic Intra-Domain Authentication**: DKIM assumes that the domain `outgoing.foo.com` can tell its users `bob@foo.com` and `carol@foo.com` apart, which is not a safe assumption in a number of settings—e.g., university campuses or ISPs that authenticate only the sending IP address. (In Section 10.6.5, figure 10.20 highlights the concern.) By contrast, LES authenticates individual users within a domain without requiring additional authentication infrastructure within `foo.com`.

2. **Flexible Use of Email (Better End-to-End)**: LES allows Bob to send email via any outgoing mail server, not just the official `outgoing.foo.com` mandated by DKIM. This is particularly important when supporting existing use cases. Bob may want to alternate between using `bob@foo.com` and `bob@bar.com`, while his ISP

such that the secret key for one credential leaks the private key associated with another, and demonstrate how one party can *prove* this linkage of credentials.

For the particular case of identifier trees, we can employ a *verifiable unpredictable function* (VUF) [102] as described by Micali, Rabin, and Vadhan. This is a function f with a corresponding secret s such that $f_s(m)$ is efficiently computable for any message m; additionally, knowledge of s permits construction of an efficiently checkable proof of the correctness of $f_s(m)$. Knowledge of the value of f_s at any number of points does not permit prediction of f_s on a new point. Micali et al. demonstrate an RSA-based construction. Briefly stated, for RSA modulus N (with some short, additional public information), the value $f_s(m) = r^{1/p_m} \bmod N$, where p_m is a unique prime corresponding to message m according to a public mapping; no additional information is needed for verification.

Given this use of a VUF, a user can verify that the path corresponding to its identifier is consistent with an identifier-tree linked to the SSL certificate of a given server. If many users—or auditors—perform such verification, then it is possible to achieve good assurance of server compliance with the scheme.

Of course, it is feasible for a server to circumvent disclosure of its private SSL key by transmitting portions of its identifier tree. For example, with a tree of depth 80, 1024-bit digital signatures, and a base of 1,000,000 users, the size of the associated tree data would be slightly more than 10 GB. Thus, even without sharing its private key, a server can plausibly share its set of user identifiers. The more important aspect of our scheme is that, without sharing its private key, a server must share *updates* to the identifier tree when new users join. The resulting requirements for data coordination are a substantial technical encumbrance and disincentive in our view. Additionally, the ongoing relationship required for such updates would expose more evidence of collusion to potential auditors.

Remark VUFs have a property that is not essential for proprietary identifier-trees: The function f is deterministic, and thus the value $f_s(m)$ is unique. Without this property, a simpler scheme is possible. A random value r is assigned to the root node. Now, for any a child in position i, the associated value is computed as a digital signature on the value of the parent concatenated with i (suitably formatted). Digital signatures take the form of RSA signatures, with the SSL certificate defining the public key. Provided that the signature scheme carries the right security properties, an adversary cannot guess the value of unrevealed secrets in the identifier tree. This is true, for instance, for signature schemes that are existentially unforgeable under chosen-message attacks. Given this property, the ability to construct any unrevealed portion of the identifier tree implies knowledge of the private key for the SSL certificate. A simple signature-based scheme, however, lacks the crisp security properties of one based on VUFs. For example, the signer, that is, the creator of the identifier tree, can embed side-information in its signatures, perhaps undermining its security guarantees.

10.5.12 Implementation

A *write* to a user's browser cache works by having a user visit a page that injects a series of URLs in the browser cache of the user. This is achieved by redirecting the user through a corresponding series of URLs, all displayed in a hidden iframe. There are two methods for this redirection: A server-side redirect and a client-side redirect. A server-side redirect means that the browser receives an HTTP 302 message, which forces a redirection before anything is displayed. This then redirects the browser to a second URL, etc. Each redirect inserts a URL in the history. The second approach involves use of client-side pull, and uses the meta-refresh approach to cause redirection. This second approach can manipulate either history or cache—the latter by downloading content.

We note that different browsers allow a different number of redirects per iframe. For Safari, this is 16, for other browsers, it is 20. Of course, with several iframes, any number of redirects can be achieved.

We performed a number of write experiments using MAC OS X clients, with a Gentoo Linux server and a LAN connection with download speed of 10 Mbits/s. For Mozilla, we observed a write speed of about 20 bits in 3 seconds; for Safari, 16 bits in 2 seconds. Using multiple iframes, however, the number of redirects can be increased and writes performed in parallel, so that considerably greater throughput is possible, as we have determined in preliminary experiments.

A *read* from a user's browser cache may be accomplished using a CSS approach that detects contents of the history file. Researchers have recently posted an example online to illustrate browser-sniffing attacks [73]. If we wish to read the cache instead of the history, we can do this using the above-mentioned meta-refresh technique on the client side, in which the tags corresponding to both (or all) edges from a node are listed. The one that is already downloaded and resides in the cache will not be requested. Note that no URLs will be served: This could cause either a 200 or 401 error, but neither will be displayed in the hidden frame, and thus the operation will be invisible to the user. The server side, however, learns the desired contents of the cache. Experiments yielded read speeds comparable to the write speeds cited.

10.6 LIGHTWEIGHT EMAIL SIGNATURES

Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest

Domain Keys and Identified Mail (DKIM) is a promising proposal for providing a cryptographic foundation to solve the phishing problem: domains are made cryptographically responsible for the email they send. Roughly, `bob@foo.com` sends emails via `outgoing.foo.com`, which properly identifies Bob and signs the email content. The public key is distributed via a DNS TXT record for `_domainkeys.foo.com`. The details of how DKIM should handle mailing lists, message canonicalization, message forwarding, and other thorny issues, are being resolved in the context of a recently-formed IETF Working Group [66].

We propose *Lightweight Email Signatures*, abbreviated LES, as an extension to DKIM. We envision LES as being fully compatible with DKIM, in that it should be supportable in principle within the flexible parameterized framework we forsee DKIM implementing and supporting. LES offers three significant improvements:

1. **Automatic Intra-Domain Authentication**: DKIM assumes that the domain `outgoing.foo.com` can tell its users `bob@foo.com` and `carol@foo.com` apart, which is not a safe assumption in a number of settings—e.g., university campuses or ISPs that authenticate only the sending IP address. (In Section 10.6.5, figure 10.20 highlights the concern.) By contrast, LES authenticates individual users within a domain without requiring additional authentication infrastructure within `foo.com`.

2. **Flexible Use of Email (Better End-to-End)**: LES allows Bob to send email via any outgoing mail server, not just the official `outgoing.foo.com` mandated by DKIM. This is particularly important when supporting existing use cases. Bob may want to alternate between using `bob@foo.com` and `bob@bar.com`, while his ISP

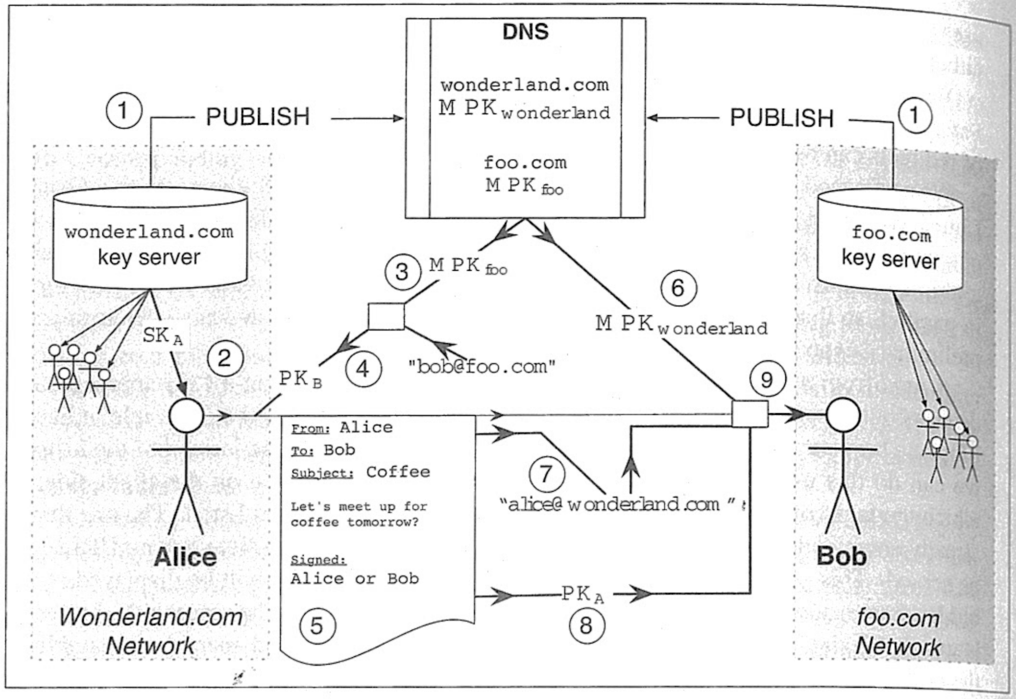


Figure 10.17 LES: (1) The domain keyservers for Alice and Bob publish their $MPKs$ in the DNS. (2) Alice's domain sends Alice her secret key SK_A, via email. (3) Alice obtains the MPK for Bob's domain, `foo.com`. (4) Alice computes Bob's public key PK_B. (5) Alice signs her email with a ring signature and sends it to Bob. (6) Bob obtains the MPK for Alice's domain, from the DNS. (7) Bob extracts the `From:` field value, `alice@wonderland.com`, from the email. (8) Bob computes Alice's public key PK_A, using the claimed identity string "`alice@wonderland.com`". (9) Bob verifies the signature against the message and PK_A.

might only allow SMTP connections to its outgoing mail server `outgoing.isp.com`. Bob may also use his university's alumni forwarding services to send email from `bob@alum.univ.edu`, though his university might not provide outgoing mail service.

3. **A Privacy Option**: LES enables the use of repudiable signatures to help protect users' privacy. Bellovin [18] and other security experts [114, 24] warn that digitally signed emails entail serious privacy consequences. We believe the option for repudiable signatures can alleviate these concerns.

In a nutshell, LES provides more implementation flexibility for each participating domain — in particular, flexibility that addresses *existing legitimate uses of email* — without complicating the domain's public interface. A LES domain exposes a single public key in the DNS, just like DKIM. Among its users, a LES domain can implement DKIM-style, server-based signatures and verifications, or user-based signatures and verifications where each user has her own signing key.

The LES Architecture We now describe the LES architecture as diagrammed in Figure 10.17.

The DKIM Baseline A LES-signed email contains an extra SMTP header, called `X-LES-Signature`, which encodes a signature of a canonicalized version of the message. We leave to the DKIM Working Group the details of this canonicalization—which includes the `From:` field, the subject and body of the message, and a timestamp—, as they do not impact the specifics of LES. Verification of a LES-signed email is similar to the DKIM solution: The recipient retrieves the sender domain's public key from a specially crafted DNS record, and uses it to verify the claimed signature on the canonicalized message.

Limitations of DKIM In its basic form, a DKIM domain uses a single key to sign all of its emails. This simple architecture is what makes DKIM so appealing and easy to deploy. Not surprisingly, it is also the source of DKIM's limitations: Users must send email via their approved outgoing mail server, and this outgoing mail server must have some internal method of robustly distinguishing one user from another to prevent `bob@foo.com` from spoofing `carol@foo.com`.

In fairness, we observe that DKIM can support multiple domain keys or user-level keys by placing each public key in the DNS. However, this approach places a strain on DNS administrators, who would need to repeatedly change a large amount of data in the DNS. For example, keeping short expiration dates on user-level keys would be difficult.

LES aims to overcome these limitations while retaining DKIM's simplicity.

User Secret Keys with Identity-Based Signatures LES assigns an individual secret key to each user, so that `bob@foo.com` can sign his own emails. This means Bob can use any outgoing server he chooses, and `outgoing.foo.com` does not need to authenticate individual users (though it may, of course, continue to use any mechanism it chooses to curb abusive mail relaying.)

To maintain a single domain-level key in the DNS, LES uses *identity-based signatures*, a type of scheme first conceptualized and implemented in 1984 by Shamir [134]. A LES domain publishes (in the DNS) a master public key MPK and retains the counterpart master secret key MSK. Bob's public key, $PK_{\texttt{Bob}}$, can be computed using MPK and an identification string for Bob, usually his email address "`bob@foo.com`". The corresponding secret key, $SK_{\texttt{Bob}}$, is computed by Bob's domain using MSK and the same identification string. Note that, contrary to certain widespread misconceptions, identity-based signatures are well tested and efficient. Shamir and Guillou–Quisquater signatures, for example, rely on the widely used RSA assumption and are roughly as efficient as normal RSA signatures.

One might argue that a typical hierarchical certificate mechanism, where the domain certifies user-generated keypairs, would be just as appropriate here. There are some problems with this approach. First, a user's public-key certificate would need to be sent along with every signed message and would require verifying a chain of two signatures, where the identity-based solution requires only one signature and one verification operation. Second, with user-generated keypairs, it is much more difficult to use ring signatures (or any of the known deniable authentication methods) between a sender and a receiver who has not yet generated his public key. The identity-based solution ensures the availability of any user's public key.

Distributing User Secret Keys via Email LES delivers the secret key $SK_{\texttt{Bob}}$ by sending it via email to `bob@foo.com` [50], using SMTP/TLS [65] where available. Thus, quite naturally, only someone with the credentials to read Bob's email can send signed emails with `bob@foo.com` as `From` address. Most importantly, as every domain already has *some* mechanism for authenticating access to incoming email inboxes, this secret-key delivery mechanism requires no additional infrastructure or protocol.

Privacy with Deniable Signatures Privacy advocates have long noted that digital signatures present a double-edged sword [18, 114, 24]: signatures may make a private conversation publicly verifiable. The LES framework supports many forms of deniable authentication [30] through its use of identity-based keys: Alice can create a deniable signature using her secret key $SK_{\texttt{Alice}}$ and Bob's public key $PK_{\texttt{Bob}}$. Only Bob can meaningfully verify such a signature. We note that this approach does not provide anonymity beyond that of a normal, unsigned email. However, unlike DKIM and other signature proposals, LES does not make the signature publicly verifiable: only the email recipient will be convinced.

A Prototype Implementation To determine the practical feasibility of deploying LES, we built a basic prototype, including a key server and a plugin to the Apple Mail client. We deployed a real MPK in the DNS for `csail.mit.edu`, using the Guillou–Quisquater identity-based scheme [59] for its simplicity and using ring signatures for deniability. We then conducted a small test with nine users. Though our test was too small to provide complete, statistically significant usability results, we note that most participants were able to quickly install and use the plugin with no user-noticeable effect on performance.

Detailed performance numbers, in Section 10.6.5, show that basic ring signature and verification operations perform well within acceptable limits—under 40 ms on an average desktop computer—even before serious cryptographic optimizations. A small keyserver can easily compute and distribute keys for more than 50,000 users, even when configured to renew keys on a daily basis.

Previous and Related Work The email authentication problem has motivated a large number of proposed solutions.

End-to-end digital signatures for email have repeatedly been proposed [10, 146] as a mechanism for making email more trustworthy and thereby preventing spoofing attacks such as phishing. One proposal suggests labeling email content and digitally signing the label [64]. Apart from DKIM, all of these past proposals require some form of Public-Key Infrastructure, e.g., X.509 [46]. Alternatively, path-based verification has been proposed in a plethora of initiatives. Those which rely on DNS-based verification of host IP addresses were reviewed by the IETF MARID working group [67, 91, 90]. The latest proposal in this line of work is SIDF [104].

A number of spam-related solutions have been suggested to fight phishing. Blacklists of phishing mail servers are sometimes used [142, 95], as is content filtering, where statistical machine learning methods are used to detect likely attacks [128, 99, 101]. Collaborative methods [39] that enable users to help one another have also been proposed. LES can help complement these approaches.

10.6.1 Cryptographic and System Preliminaries

We now review the cryptographic and system building blocks involved in LES.

Identity-Based Signatures In 1984, Shamir proposed the concept of identity-based signatures (IBS) [134]. Since then over a dozen schemes have been realized based on factoring, RSA, discrete logarithm, and pairings. (See [17] for an overview, plus a few more in [8].)

Most IBS signatures can be computed roughly as fast as RSA signatures, and those based on pairings can be 200 bits long for the equivalent security of a 1024 bit RSA signature.

IBS schemes were introduced to help simplify the key management problem. Here, a single master authority publishes a master public key MPK and stores the corresponding master secret key MSK. Users are identified by a character string id_string, which is typically the user's email address. A user's public key PK can be publicly computed from MPK and id_string, while a user's secret key SK is computed by the master authority using MSK and the same id_string, then delivered to the user.

Ring Signatures from Any Keypairs Ring signatures [37, 125] allow an individual to sign on behalf of a group of individuals without requiring any prior group setup or coordination. Although rings can be of any size, consider the two party case. Suppose Alice and Bob have keypairs $(PK_{\texttt{Alice}}, SK_{\texttt{Alice}})$ and $(PK_{\texttt{Bob}}, SK_{\texttt{Bob}})$ respectively. Alice can sign on behalf of the group "Alice or Bob" using her secret key $SK_{\texttt{Alice}}$ and Bob's public key $PK_{\texttt{Bob}}$. Anyone can verify this signature using both of their public keys. We require the property of *signer-ambiguity* [8]; that is, *even if Alice and Bob reveal their secret keys*, no one can distinguish the actual signer.

There exist compilers for creating signer-ambiguous ring signatures using keypairs of almost any type [8]. That is, Alice may have a PGP RSA-based keypair and Bob may have a pairing-based identity-based keypair, yet Alice can still create a ring signature from these keys! For our purposes here, it does not matter *how* this compiler works. It is enough to know that: (1) the security of the resulting ring signature is equivalent to the security of the weakest scheme involved, and (2) the time to sign (or verify) a ring signature produced by our compiler is roughly the sum of the time to sign (or verify) individually for each key involved, plus an additional hash.

Using ring signatures for deniable authentication is not a new concept [125, 24]. The idea is that, if Bob receives an email signed by "Alice or Bob," he knows Alice must have created it. However, Bob cannot prove this fact to anyone, since he *could* have created the signature himself. In Section 10.6.2, we describe how ring signatures are used to protect a user's privacy in LES.

Email Secret-Key Distribution Web password reminders, mailing list subscription confirmations, and e-commerce notifications all use email as a semi-trusted messaging mechanism. This approach, called Email-Based Identity and Authentication [50], delivers semi-sensitive data to a user by simply sending the user an email. The user gains access to this data by authenticating to his incoming mail server in the usual way, via account login to an access-controlled filesystem, webmail, POP3 [113], or IMAP4 [38]. For added security, one can use SMTP/TLS [65] for the transmission.

10.6.2 Lightweight Email Signatures

We now present the complete design of LES, as previously illustrated in Figure 10.17.

Email Domain Setup Each email domain is responsible for establishing the cryptographic keys to authenticate the email of its users. The setup procedure for that master authority run by `wonderland.com` is defined as follows:

1. Select one of the identity-based signatures (IBS) discussed in Section 10.6.1. (For our Section 10.6.5 experiment, we chose the RSA-based Guillou–Quisquater IBS [59] because of its speed and simplicity.)

2. Generate a master keypair $(MPK_{\texttt{wonderland}}, MSK_{\texttt{wonderland}})$ for this scheme.

3. Define key issuance policy $Policy$, which defines if and how emails from this domain should be signed. (Details of this policy are defined in Section 10.6.3.)

4. Publish $MPK_{\texttt{wonderland}}$ and $Policy$ in the DNS as defined by the DKIM specifications.

User Identities Per the identity-based construction, a user's public key PK can be derived from any character string id_string that represents the user's identity. We propose a standard format for id_string.

Master Domain In most cases, `bob@foo.com` obtains a secret key derived from a master keypair whose public component is found in the DNS record for the expected domain, `foo.com`. However, in cases related to bootstrapping (see Section 10.6.3), Bob might obtain a secret key from a domain *other than* `foo.com`.

For this purpose, we build a $issuing\_domain$ parameter into the user identity character string. Note that `foo.com` should always refuse to issue secret keys for identity strings whose $issuing\_domain$ is not `foo.com`. However, `foo.com` may choose to issue a key for `alice@wonderland.com`, as long as the $issuing\_domain$ within the identity string is `foo.com`. We provide a clarifying example shortly.

Key Types The LES infrastructure may be expanded to other applications in the future, such as encryption. To ensure that a key is used only for its intended purpose, we include type information in id_string. Consider $type$, a character string composed only of lowercase ASCII characters. This type becomes part of the overall identity string. For the purposes of our application, we define a single type: `lightsig`.

Key Expiration In order to provide key revocation capabilities, the user identity string includes expiration information. Specifically, id_string includes the last date on which the key is valid: $expiration\_date$, a character string formatted according to ISO-8601, which include an indication for the timezone. For now, we default to UTC for timezone disambiguation.

Constructing Identity Character Strings An id_string is thus constructed as

$$\langle issuing\_domain\rangle, \langle email\rangle, \langle expiration\_date\rangle, \langle type\rangle$$

For example, a 2006 LES identity string for email address `bob@foo.com` would be

```
foo.com,bob@foo.com,2006-12-31,lightsig
```

If Bob obtains his secret key from a master authority different than his domain, e.g., `lightsig.org`, his public key would necessarily be derived from a different id_string:

```
lightsig.org,bob@foo.com,2006-12-31,lightsig
```

Notice that `lightsig.org` will happily issue a secret key for that identity string, even though the email address is not within the `lightsig.org` domain. This is legitimate, as long as the $issuing\_domain$ portion of the id_string matches the issuing keyserver.

Delivering User Secret Keys Each domain keyserver will choose its own interval for regular user secret key issuance, possibly daily, weekly or monthly. These secret keys are delivered by email, with a well-defined format—e.g., XML with base64-encoded key, including a special mail header—that the mail client will recognize. The most recent key-delivery email is kept in the user's inbox for all mail clients to access, in case the user checks his email from different computers. The mail client may check the correctness of the secret key it receives against its domain's master public key, either using a specific algorithm inherent to most IBS schemes, or by attempting to sign a few messages with the new key and then verifying those results. (For more details, see Section 10.6.1.)

The Repudiability Option The downside of signing email is that it makes a large portion of digital communication undeniable [18, 114, 24]. An off-the-record opinion confided over email to a once-trusted friend may turn into a publicly verifiable message on a blog! We believe that repudiable signatures should be the *default* to protect a user's privacy as much as possible and that non-repudiable signatures should be an option for the user to choose.

Numerous approaches exist for realizing repudiable authentication: designated-verifier signatures [75], chameleon signatures [86] , ring signatures [125] , and more (see [30] for an overview of deniable authentication with RSA). In theory, any of these approaches can be used. We chose the ring signature approach for two reasons: (1) it fits seamlessly into our identity-based framework without creating new key management problems, and (2) our ring signature compiler can create ring signatures using keys from different schemes, as discussed in Section 10.6.1. Thus, no domain is obligated to use a single (perhaps patented) IBS.

Let us explore why ring signatures are an ideal choice for adding repudiability to LES. Most repudiation options require the sender to know something about the recipient; in ring signatures, the sender need only know the receiver's public key. In an identity-based setting, the sender Alice can easily derive Bob's public key using the $MPK_{\texttt{foo.com}}$ for `foo.com` in the DNS and Bob's id_string. Setting the $issuing\_domain$ to `foo.com`, the $type$ to `lightsig`, and the email field to `bob@foo.com` for Bob's id_string is straightforward. For $expiration\_date$, Alice simply selects the current date. We then require that domains be willing to distribute back-dated secret keys (to match the incoming public key) on request to any of their members. Few users will take this opportunity, but the fact that they *could* yields repudiability. Such requests for back-dated keys can simply be handled by signed email to the keyserver.

This "Alice or Bob" authentication is valid: if Bob is confident that *he* did not create it, then Alice must have. However, this signature is also repudiable, because Bob cannot convince a third party that he did not, in fact, create it. In Section 10.6.3, we discuss what Alice should do if `foo.com` does not yet support LES, and in Section 10.6.3, we discuss methods for achieving more repudiability.

Signing and Verifying Messages Consider Alice, `alice@wonderland.com`, and Bob, `bob@foo.com`. On date `2006-07-04`, Alice wants to send an email to Bob with subject $\langle subject \rangle$ and body $\langle body \rangle$. When Alice clicks "send," her email client performs the following actions:

1. Prepare a message $\mathcal{M}$ to sign, using the DKIM canonicalization (which includes the `From:`, `To:`, and `Subject:` fields, as well as a timestamp and the message body).

2. If Alice desires repudiability, she needs to obtain Bob's public key:

(a) Obtain $MPK_{\texttt{foo.com}}$, the master public key for Bob's domain `foo.com`, using DNS lookup.

(b) Assemble $id\_string_{\texttt{Bob}}$, an identity string for Bob using `2006-07-04` as the *expiration_date*: `foo.com,bob@foo.com,2006-07-04,lightsig`

(c) Compute $PK_{\texttt{Bob}}$ from $MPK_{\texttt{foo.com}}$ and $id\_string_{\texttt{Bob}}$. (We assume that $PK_{\texttt{Bob}}$ contains a cryptosystem identifier, which determines which IBS algorithm is used here.)

3. Sign the message $\mathcal{M}$ using $SK_{\texttt{Alice}}$, $MPK_{\texttt{wonderland.com}}$. Optionally, for repudiability, also use $PK_{\texttt{Bob}}$ and $MPK_{\texttt{foo.com}}$ with the Section 10.6.1 compiler. The computed signature is σ.

4. Using the DKIM format for SMTP header signatures, add the `X-LES-Signature` containing σ, $id\_string_{\texttt{Alice}}$, and $id\_string_{\texttt{Bob}}$.

Upon receipt, Bob needs to verify the signature:

1. Obtain the sender's email address, `alice@wonderland.com`, and the corresponding domain name, `wonderland.com`, from the email's `From` field.

2. Obtain $MPK_{\texttt{wonderland.com}}$, using DNS lookup (as specified by DKIM).

3. Ensure that $PK_{\texttt{Alice}}$ is correctly computed from the claimed $id\_string_{\texttt{Alice}}$ and corresponding issuing domain
$MPK_{\texttt{wonderland.com}}$, and that this *id_string* is properly formed (includes Alice's email address exactly as indicated in the `From` field, a valid expiration date, a valid type).

4. Recreate the canonical message $\mathcal{M}$ that was signed, using the declared `From`, `To`, and `Subject` fields, the email body, and the timestamp.

5. If Alice applied an ordinary, non-repudiable signature, verify $\mathcal{M}$, σ, $PK_{\texttt{Alice}}$, $MPK_{\texttt{wonderland.com}}$ to check that Alice's signature is valid.

6. If Alice applied a repudiable signature, Bob **must** check that this signature verifies against both Alice's and his own public key following the proper ring verification algorithm [8]:

   (a) Ensure that $PK_{\texttt{Bob}}$ is correctly computed from the claimed $id\_string_{\texttt{Bob}}$ and the DNS-advertised $MPK_{\texttt{foo.com}}$, and that this *id_string* is properly formed (includes Bob's email address, a valid expiration date, a valid type).

   (b) Verify $\mathcal{M}, \sigma, PK_{\texttt{Alice}}, MPK_{\texttt{wonderland.com}}, PK_{\texttt{Bob}}, MPK_{\texttt{foo.com}}$ to check that this is a valid ring signature for "Alice or Bob".

If all verifications succeed, Bob can be certain that this message came from someone who is authorized to use the address `alice@wonderland.com`. If the `wonderland.com` keyserver is behaving correctly, that person is Alice.

LES vs. Other Approaches The LES architecture provides a number of benefits over alternative approaches to email authentication. We consider three main competitors: SIDF [104] and similar path-based verification mechanisms, S/MIME [155] and similar certificate-based signature schemes, and DKIM, the system upon which LES improves. A comparison chart is provided in table 10.6.2, with detailed explanations as follows:

1. **Logistical Scalability:** When a large organization deploys and maintains an architecture for signing emails, it must consider the logistics of such a deployment, in particular how well the plan scales. With SIDF or DKIM, domain administrators must maintain an inventory of outgoing mail servers and ensure that each is properly configured. This includes having outgoing mail servers properly authenticate individual users to prevent intra-domain spoofing. Meanwhile, with certificate-based signature schemes, domain administrators must provide a mechanism to issue user certificates. By contrast, LES does not require any management of outgoing mail servers or any additional authentication mechanism. LES only requires domains to keep track of which internal email addresses are legitimate, a task that each domain already performs when a user's inbox is created.

   Thus, LES imposes only a small incremental logistical burden, while DKIM, SIDF, and S/MIME all require some new logistical tasks and potentially new authentication mechanisms. Note that it is technically possible to use PGP in a way similar to LES, with email-delivered certificates, though the PGP keyserver then needs to keep track of individual user keys where LES does not.

2. **Deployment Flexibility:** SIDF and DKIM can only be deployed via server-side upgrades, which means individual users must wait for their domain to adopt the technology before their emails become authentic. PGP can only be deployed via client-side upgrades, though one should note that many clients already have PGP or S/MIME support built in. LES can be implemented either at the server, like DKIM, or at the client, like PGP.

3. **Support for Third-Party SMTP Servers:** SIDF and DKIM mandate the use of pre-defined outgoing mail servers. A user connected via a strict ISP may not be able to use all of his email personalities. Incoming-mail forwarding services—e.g., alumni

Table 10.2 LES Compared to Other Approaches for Authenticating Email. [a]: PGP and S/MIME can be adjusted to issue keys from the server, somewhat improving the scalability. [b]: DKIM can support user-level keys by placing each user's public key in the DNS.

Property	**SIDF**	**S/MIME**	**DKIM**	**LES**
Logistical scalability	No	No[a]	No	**Yes**
Deployable with client update only	No	**Yes**	No[b]	**Yes**
Deployable with server update only	**Yes**	No[a]	**Yes**	**Yes**
Supports third-party SMTP servers	No	**Yes**	No	**Yes**
Supports user privacy	**Yes**	No	No	**Yes**
Supports email alias forwarding	No	**Yes**	**Yes**	**Yes**
Supports mailing lists	**Good**	Poor	Acceptable	Acceptable

address forwarding—may not be usable if they do not also provide outgoing mail service. PGP and LES, on the other hand, provide true end-to-end functionality for the sender: each user has a signing key and can send email via any outgoing mail server it chooses, regardless of the `From` email address.

4. **Privacy:** LES takes special care to enable deniable authentication for privacy purposes. SIDF, since it does not provide a cryptographic signature, is also privacy-preserving. However, DKIM and S/MIME provide non-repudiable signatures which may greatly affect the nature of privacy in email conversations. Even a hypothetical LES-S/MIME hybrid, which might use certificates in the place of identity-based signatures, would not provide adequate privacy, as the recipient's current public key would often not be available to the sender without a PKI.

5. **Various Features of Email:** SIDF does not support simple email alias forwarding, while S/MIME, DKIM, and LES all support it easily. SIDF supports mailing lists and other mechanisms that modify the email body, as long as mailing list servers support SIDF, too. On the other hand, S/MIME, DKIM, and LES must specify precise behavior for mailing lists: if the core content or `From` address changes, then the mailing list must re-sign the email, and the recipient must trust the mailing list authority to properly identify the original author of the message. An alternate suggestion for these latter schemes is that the mailing list would append changes to the end of the message, and then clients would discard these changes when verifying signatures and displaying content to the users.

LES provides a combination of advantages that is difficult to obtain from other approaches. Of course, these features come at a certain price: LES suffers from specific vulnerabilities that other systems do not have. We explore these vulnerabilities in Section 10.6.4.

10.6.3 Technology Adoption

The most challenging aspect of cryptographic solutions is their path to adoption and deployment. The deployment features of LES resembles those of DKIM: each domain can adopt it independently, and those who have not yet implemented it will simply not notice the additional header information. However, in LES, individual users can turn to alternate authorities to help sign emails before their own domain has adopted LES.

Alternate Domain Authorities Alice wishes to send an email to Bob. If both domains are LES-enabled, they can proceed as described in Section 10.6.2. What happens, however, if one of these elements is not yet in place?

Getting an Alternate Secret Key Alice may want to sign emails before her domain `wonderland.com` supports it. LES allows Alice to select an alternate master authority domain created specifically for this purpose; e.g., `lightsig.org`. `lightsig.org` must explicitly support the issuance of external keys, i.e., keys corresponding to email addresses at a different domain than that of the issuing keyserver. To obtain such a key, Alice will have to explicitly sign up with `lightsig.org`, most likely via a web interface. Her *id_string* will read:

```
lightsig.org,alice@wonderland.com,2006-12-31,lightsig
```

Note that we do not expect `lightsig.org` to perform any identity verification beyond email-based authentication: the requested secret key is simply emailed to the appropriate address.

A recognized, noncommercial organization would run `lightsig.org`, much like MIT runs the MIT PGP Keyserver: anyone can freely use the service. Alternatively, existing identity services—e.g., Microsoft Passport [103] or Ping Identity [120]—might issue LES keys for a small fee. Where PGP requires large, mostly centralized services like the MIT PGP Keyserver, LES users may choose from any number of keyservers.

Of course, when Bob receives a LES email, he must consider his level of trust in the issuing keyserver, especially when it is does not match the `From:` address domain. Most importantly, certain domains—e.g., those of financial institutions—should be able to prevent the issuance of keys for its users by alternate domains altogether. We return to this point shortly.

Bootstrapping Repudiability When Alice wishes to send an email to Bob, she may notice that Bob's domain `foo.com` does not support LES. In order to obtain repudiability, however, she needs to compute a public key for which Bob has at least the capability of obtaining the secret key counterpart. For this purpose, Alice can use the same `lightsig.org` service, with Bob's id_string as follows:

```
lightsig.org,bob@foo.com,2006-07-04,lightsig
```

Note that Bob need not ever actually retrieve a secret key from the `lightsig.org` service. The mere fact that *he could potentially retrieve a secret key at any time* is enough to guarantee repudiability for Alice. Note also that, somewhat unintuitively, it makes sense to have Alice select the LES service to generate Bob's public key: in our setting, Bob's public key serves Alice, not Bob, as it is an avenue for sender repudiability.

Announcing Participation: Domain Policies In an early (April 2005) draft of the LES system, we considered how a domain would announce its use of LES. Since then, DKIM has begun to consider this very same issue [66]. We defer to the DKIM effort for the exact DNS-based policy architecture, and focus on the specific policy parameters that apply to the unique features of LES.

Once an email domain decides to deploy LES, it needs to consider two aspects of its deployment: $InternalPolicy$ and $ExternalPolicy$. $InternalPolicy$ defines how users of `wonderland.com` should behave, in particular whether they are expected to sign their emails, and whether they can use keys issued by other domains. It can also contain information on whether users are permitted to issue repudiable signatures. $ExternalPolicy$ defines what services `wonderland.com` offers to users outside the domain, specifically whether that domain is willing to issue keys to external users.

Thus, one would expect `bigbank.com` to issue strict policies on both fronts: Users of `bigbank.com` must non-repudiably sign all of their emails, and no external authorities are permitted. On the other hand, `lightsig.org` would offer a more relaxed $ExternalPolicy$, offering to issue keys for external users, and a small ISP without the immediate resources to deploy LES may offer an $InternalPolicy$, allowing its users to obtain keys from other services, like `lightsig.org`, and certainly allowing its users to issue repudiable signatures.

Using Domain Policies to Automatically Detect Spoofing When Bob receives an email from Alice, he may be faced with two possibly confounding situations: the message bears no signature, or the message bears a signature signed with a PK for Alice derived from a different master authority than that of Alice's email address. This is where the use of LES domain policies comes into play: Bob must check the policy advertised by Alice's domain `wonderland.com`.

If the message bears no signature, but `wonderland.com` advertises an // *InternalPolicy* that requires signatures, Bob can safely consider the message a spoof and discard it. Similarly, if the message bears a signature authorized by another domain, but `wonderland.com` advertises an *ExternalPolicy* that bans the use of other authorities for its users, Bob can again safely discard the message.

In other cases where the policies are not so strict, there remains a grey area where Bob will have to make a judgment call on whether to trust an unsigned email, or whether to trust the alternate issuing domain. These cases may be solved by reputation systems and personal user preferences, though, of course, one shouldn't expect every case to be decidable with full certainty.

LES at the Server LES can be deployed entirely at the client, as described in the past few pages. Alternatively, LES can be deployed partially or entirely at the server level, mimicking DKIM, if the deploying domain so desires.

DNS MPK Lookups An incoming mail server can easily look up the MPK records of senders when emails are received. This MPK can be easily included in additional SMTP headers before delivering the emails to the user's mailbox. This is particularly useful for mail clients that may be offline when they finally process the downloaded emails.

Signature Verification The incoming mail server can even perform the signature verification, indicating the result by yet another additional SMTP header. The client would be saved the effort of performing the cryptographic operations. This is particularly useful for low-resource clients, like cell phones or PDAs. In cases where the incoming email clashes with the sending domain's *Policy*, as illustrated in Section 10.6.3, the incoming mail server can confidently discard the fraudulent email before the user even downloads it!

Signature Creation If a user sends email through his regular outgoing mail server, the signature can be applied by that server. This optimization is also particularly useful for inexperienced users and low-resource clients. This server-applied signature certifies the specific `From:` address, not just the email domain.

Transparent LES Internet Service Providers (ISPs) can combine the previous optimizations to provide all functionality at the server, as in DKIM. Home users can get all the benefits of LES without upgrading their mail client or taking any action. This approach is even more appealing—and necessary—for web-based email, particularly web-only email providers like Hotmail, Yahoo, and Gmail. A web client may not provide the necessary framework to perform public-key cryptography. (This is changing rapidly with advanced Javascript capabilities, but it is not yet guaranteed in every browser.) In these cases, the web server is the only system component which can provide signing and verification functionality.

Forwarding, Multiple Recipients, and Mailing Lists Just like DKIM, LES provides end-to-end signatures, which means that whenever the `From` and `To` addresses and other message components are preserved, the signature remains valid. Thus, email alias forwarding is automatically supported by LES, like DKIM. Also like DKIM, a LES email to multiple recipients is signed individually to each recipient.

The handling of signatures for mailing lists is currently under consideration by the DKIM working group: should mailing list servers resign messages, or should signatures attempt to survive mailing lists modifications? We defer to their approach for LES signatures, too. In this work, we only need to consider the case of repudiable LES signatures in the context of mailing lists, which we address in the next Section.

Advanced Repudiation via Evaporating Keys LES offers repudiability because someone in possession of a secret key *other than the message author's* might have signed the message. When the recipient doesn't present a proper avenue for repudiation, an alternate approach is to set up an **evaporating key**. Perrig et al. [118], followed by Borisov et al. [24], proposed evaporating keys in a MAC setting. The same trick can be done in the public key setting.

Evaporating keys in LES are implemented by special keyservers that declare, in their *ExternalPolicy*, whether they issue keys with a special *type* in their *id_string*: `ltaevap`. At regular intervals—e.g., at the end of each day—these servers publish on a website the secret-key component of the evaporating public key. If Alice wishes to use an evaporating key, she does not need to notify the server: she simply computes the appropriate public key against the server's DNS-announced MPK, trusting that the server will evaporate the key at the next given interval.

Even when Alice knows Bob's public key, she can generate a three-party ring signature "Alice or Bob or Evaporating Key," where the key comes from a server that Alice trusts. This provides Alice with total repudiability after the evaporation period. Of course, Bob may always refuse to accept such signatures.

Repudiability for Mailing Lists Evaporating keys can provide repudiation for messages sent to mailing lists. When Alice sends an email to a mailing list, she may not know the eventual recipients of the email. If she signs with her secret key and an evaporating public key, recipients can trust Alice's authorship as much as they trust the evaporating domain, and Alice gains repudiability as soon as the evaporation interval comes to an end. Because email clients are not always aware of the fact that the recipient is a mailing list, one possible option is to *always* create a three-way repudiable signature using Alice's secret key, the recipient's public key, and an evaporating public key.

10.6.4 Vulnerabilities

LES shares enough in architectural design with DKIM that both systems face a number of common vulnerabilities. At the same time, LES is different enough that we must examine the unique vulnerabilities it faces, too.

Vulnerabilities of DKIM and LES Both DKIM and LES distribute a single domain-level key via the DNS. Thus, they share potential vulnerabilities surrounding this particular operation.

1. **DNS Spoofing:** Since DNS is used to distribute domain-level public keys, it may come under increased attack. An attacker might compromise DNS at any number of levels: hijacking the canonical DNS record, all the way down to redirecting a single user's DNS queries. Fortunately, high-level compromises will be quickly noticed

and remedied, though, if an attacker sets a long-enough time-to-live on the hijacked record, the corrupt DNS data may survive on downstream hosts for a few hours or even days. During this period of time, spoofed emails would fool mail clients and servers alike.

Spoofing DNS at a more local level, affecting only a handful of users, is particularly worrisome as it may go undetected for quite some time.

If these types of DNS attacks become more popular, it will become imperative to implement more secure DNS options, such DNSSEC [45], or, as a stop-gap solution before a secure DNS alternative is deployed, to provide certified keys in the DNS TXT records.

2. **Domain Key Compromise:** An attacker might compromise the domain's secret key and then successfully sign emails from any address in that domain. If the attacker is careful to use this compromised key with moderation, he may go unnoticed for quite some time. Once such a compromise is discovered, of course, the remedy is relatively simple: Generate a new key and update the DNS record.

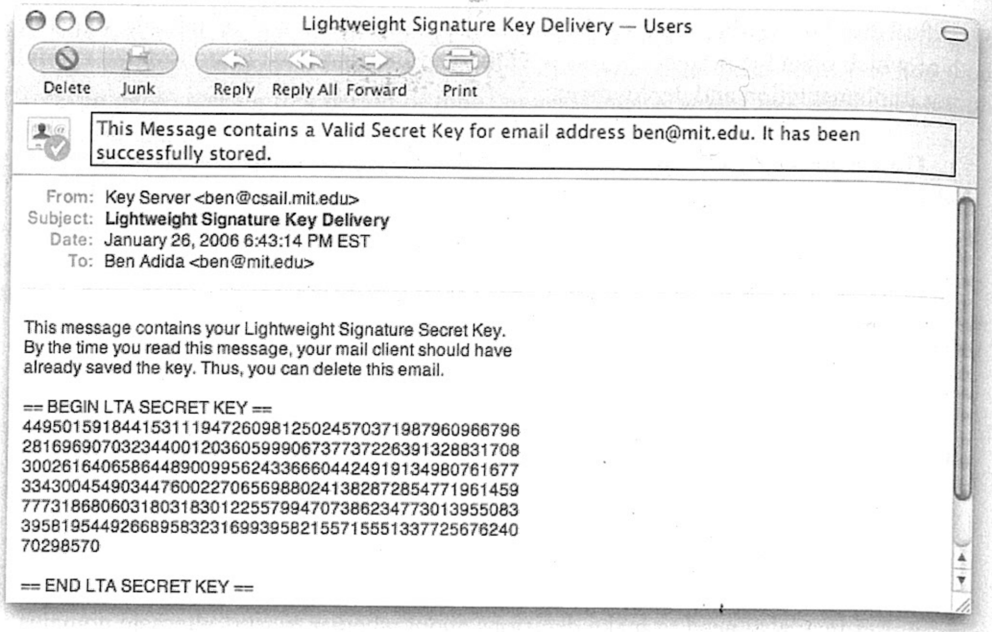
   As phishing attacks become more targeted and sophisticated, it will become important to strongly secure the secret domain key and possibly to renew it fairly regularly.

3. **Zombie User Machine:** an attacker might gain control of a user's machine via one of the numerous security exploits that turn broadband-connected machines into "zombies". In this scenario, the attacker can clearly send signed email exactly like the legitimate user. If an attacker makes moderate use of this ability, it may take quite some time to detect the problem.

   There is, of course, no complete solution against zombie user machines given that they can act exactly like legitimate users. However, with DKIM or LES signatures, illegitimate behavior can eventually be detected and traced to a single user account. Once this abuse has been detected, DKIM and LES domain servers can take rapid action to shut down that user's ability to send emails.

4. **User Confusion:** When a user receives a validly signed email, the email may still be a phishing attack. A valid signature should not be interpreted as a complete reason to trust the sender, though of course it should provide greater accountability for criminal actions. One notes that, as of August 2005, 83 percent of all domains with SIDF records were spammers [89].

DKIM Vulnerabilities Minimized by LES With LES, the threat of a domain key compromise can be significantly reduced compared to DKIM. In particular, whereas DKIM's domain secret key must be available to an online server—the outgoing mail server—the generation of individual LES user keys can be performed by an offline machine that only acts as a mail client. Even in the LES setting, where all cryptographic operations are done at the server level, the domain could give the outgoing mailserver only user-level secret keys with short expiration dates, instead of the master secret key. In that case, a compromise of the outgoing mail server would only yield the ability to forge for a very short period of time, and the domain could recover from this compromise without needing to update its DNS entry. For other benefits of LES over DKIM, see Section 10.6.2.



This Message contains a Valid Secret Key for email address ben@mit.edu. It has been successfully stored.

From: Key Server <ben@csail.mit.edu>
Subject: **Lightweight Signature Key Delivery**
Date: January 26, 2006 6:43:14 PM EST
To: Ben Adida <ben@mit.edu>

```
This message contains your Lightweight Signature Secret Key.
By the time you read this message, your mail client should have
already saved the key. Thus, you can delete this email.

== BEGIN LTA SECRET KEY ==
449501591844153111947260981250245703719879609667 96
281696907032344001203605999067377372263913288317 08
300261640658644890099562433666044249191349807616 77
334300454903447600227065698802413828728547719614 59
777318680603180318301225579947073862347730139550 83
395819544926689583231699395821557155513377256762 40
70298570

== END LTA SECRET KEY ==
```

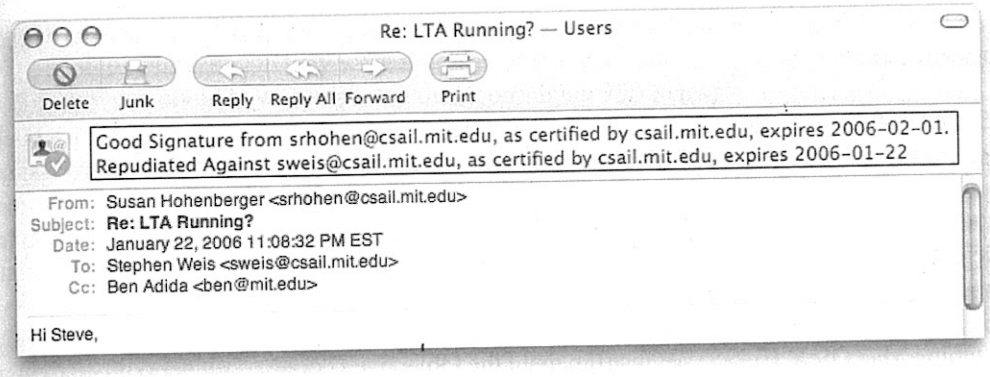
Figure 10.18 A screenshot of the secret key delivery email that was automatically processed by the LES extension in the user's AppleMail client.

extensions are related to the underlying cryptography, we believe they will not impact performance and usability.

We set up the client-side software to check every domain for a MPK, and to default to `csail.mit.edu` when one wasn't found. Emails from domains that **do** distribute a MPK were expected to be signed by default (i.e., for our test, the presence of a MPK was interpreted as a strict $InternalPolicy$.) Thus, an email from `alice@wonderland.com` was not expected to be signed, though any signature on it was obviously checked. On the other hand, emails from `csail.mit.edu` **were** expected to be signed at all times, and a lack of signature was actively signalled to the recipient.

User Testing of Prototype A group of nine users was assembled to test our prototype during a one week period from January 20 to 27, 2006. Each participant used a Mac running OS X 10.4 and AppleMail 2.0. After reading a statement on the purpose of the test, users were asked to sign an electronic consent form, double-click on an automatic installation, and then go to a website to do a one-time key request for their email address. This request took the place of a domain *automatically* sending the user a key by email. Requests were automatically processed and, usually within one minute, a user received a secret key via email, as shown in Figure 10.18. The secret key was automatically processed by the user's mail client running our LES extension.

A user's client then began automatically signing **all** outgoing messages and checking **all** incoming messages for valid signatures using LES. The two-party repudiability option was turned permanently **on** for all users. A display at the top of each message let the user know whether a message was properly signed or not; we provide examples in Figures 10.19 and

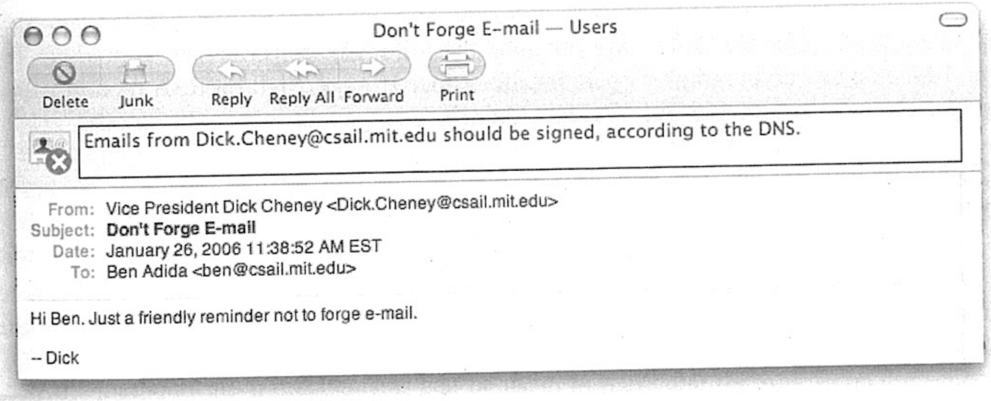


Good Signature from srhohen@csail.mit.edu, as certified by csail.mit.edu, expires 2006-02-01.
Repudiated Against sweis@csail.mit.edu, as certified by csail.mit.edu, expires 2006-01-22

From: Susan Hohenberger <srhohen@csail.mit.edu>
Subject: **Re: LTA Running?**
Date: January 22, 2006 11:08:32 PM EST
To: Stephen Weis <sweis@csail.mit.edu>
Cc: Ben Adida <ben@mit.edu>

Hi Steve,

Figure 10.19 An example of a well-signed email.



Emails from Dick.Cheney@csail.mit.edu should be signed, according to the DNS.

From: Vice President Dick Cheney <Dick.Cheney@csail.mit.edu>
Subject: **Don't Forge E-mail**
Date: January 26, 2006 11:38:52 AM EST
To: Ben Adida <ben@csail.mit.edu>

Hi Ben. Just a friendly reminder not to forge e-mail.

-- Dick

Figure 10.20 An example of a forged intra-domain email from a domain with a strict *InternalPolicy*, where, for the purposes of our test, all emails should be signed.

10.20. Users were asked to use their mail clients as normal and to report any feedback. At the week's end, they were asked to uninstall the software.

Apart from one user whose client-side configuration triggered an installer bug, users managed to install, use, and uninstall LES without difficulty and reported no noticeable performance difference. Though our group was not large enough to provide statistically significant usability results, our experiment leads us to believe that this approach is, at the very least, practical enough to pursue further.

User Feedback We learned from our user test that our unoptimized signature headers of approximately 1500 bytes triggered errors with certain mail servers. A future version of LES will remedy this situation by splitting the signature into multiple headers. Other users noticed that signatures on emails with attachments and multiple recipients were not handled appropriately, though we knew this in advance. Overall, users *did* notice when their client warned them that an email from `csail.mit.edu` should be signed but wasn't.

Results In addition to user tests, we measured the cryptographic performance of the system.

Experimental Setup We ran server benchmarks on a single-processor, 3.2 Ghz Intel Pentium 4 with 2 Gigs of RAM and 512 MB of L2 cache, running Fedora Core Linux with kernel v2.6.9. We used Python v2.3.3. We instrumented the Python code using the standard, built-in `timeit` module, running each operation 1000 times to obtain an average performance rating. We did not make any overzealous attempt to cut down the number of normally-running background processes.

We ran client benchmarks on a 1.5 Ghz Apple Powerbook G4 with 1.5 Gigs of RAM, running Mac OS X 10.4.4. We instrumented the Objective C code using the built-in Cocoa call to `Microseconds()`, which returns the number of microseconds since CPU boot. We ran each operation 1000 times to obtain an average running time. Though we were not actively using other applications on the Powerbook during the test, we also made no attempt to reduce the typically running background processes and other applications running in a normal Mac OS X session.

Cryptographic Benchmarks We obtained the following performance numbers in Table 10.6.5 on raw cryptographic operations for either 1024 or 2048-bit RSA moduli with a public exponent size of 160 bits (Guillou–Quisquater exponents cannot be small).

Optimizations As this was a proof-of-concept, our implementation omitted a number of optimizations that a real deployment would surely include:

1. User secret key generation involves an effective RSA exponentiation by the private exponent d. This can usually be sped up by a factor of 4 using the standard Chinese Remainder Theorem optimization.

2. In our prototype, Guillou–Quisquater signatures are represented as a triple (t, c, s), though technically only (c, s) is required, as t can be recomputed from c and s. This optimization would shorten the signature by about 40%, without altering performance (as is, we verify that t is correct by recomputing it from c and s).

Table 10.3 Performance Estimates for an Average of 1000 Runs in milliseconds. The sizes in bytes do not include encoding overhead. The symbol * indicates the number includes an estimated 50 bytes for the identity string of the user.

		1024-bit modulus		**2048-bit modulus**	
Operation	**Machine**	**Time**	**Size**	**Time**	**Size**
Master Keypair Generation	server	143	200	1440	300
User Secret Key Computation	server	167	178*	1209	316*
User Public Key Computation	client	0.03	178*	0.03	316*
Ring Signature of 100K message	client	37	575*	210	1134*
Ring Verification of 100K message	client	37	N/A	211	N/A

3. All of our user secret keys were encoded in decimal during the email distribution step (which meant they were roughly 1380 bytes) rather than alphanumeric encoding (which could have compacted them to roughly 750 bytes).

Next Steps The results of our experiment show that LES is practical enough to compete with existing signature schemes in a realistic user setting, even when the two-party ring signature is used. However, as our user base was small further investigation is needed. An in-depth user study could help define exactly how repudiable signatures fit into the picture, how users interpret signature notifications in their email client, and whether using LES actually does, in practice, reduce the probability that a user will fall victim to an email-based phishing attack.

Summary

The plethora of proposed solutions to the email spoofing problem reveals a clear demand for trustworthy email. DKIM is one of the most promising approaches yet, with a simple deployment plan, and reasonable end-to-end support via the use of cryptographic signatures.

We have proposed Lightweight Email Signatures (LES), an extension to DKIM which conserves its deployment properties while addressing a number of its limitations. LES allows users to sign their own emails and, thus, to use any outgoing mail server they choose. This helps to preserve a number of current uses of email that DKIM would jeopardize: choosing from multiple email personalities with a single outgoing mail server because of ISP restrictions, or using special mail forwarding services (e.g., university alumni email forwarding) that do not provide an outgoing mail server.

LES also offers better privacy protection for users. Each individual email address is associated with a public key, which anyone can compute using only the domain's master public key available via DNS. With the recipient's public key available, any number of deniable authentication mechanisms can be used, in particular the ring signature scheme we propose.

Our prototype implementation shows that LES is practical. It can be quickly implemented using well-understood cryptographic algorithms that rely on the same hardness assumptions as typical RSA signatures.

We are hopeful that proposals like DKIM and LES can provide the basic authentication foundation for email that is so sorely lacking today. These cryptographic proposals are not complete solutions, however, much like viewing an SSL-enabled website is not a reason to fully trust the site. Reputation systems and "smart" user interfaces will likely be built on the foundation that DKIM and LES provide. Without DKIM or LES, however, such reputation systems would be nearly impossible.

REFERENCES

1. Bochs Emulator Web Page. `http://bochs.sourceforge.net/`.
2. Bypassing Windows Personal FW's. `http://www.phrack.org/show.php?p=62&a=13`.
3. Introduction to Side Channel Attacks. Technical report, Discretix Technologies Ltd.
4. Platform for privacy preferences (P3P) project. World Wide Web Consortium (W3C). Referenced 2005 at `http://www.w3.org/p3p`.
5. VMWare Inc. `http://www.vmware.com/`.

Phishing and Countermeasures

Understanding the Increasing Problem of Electronic Identity Theft

Edited by

Markus Jakobsson
Indiana University
Bloomington, Indiana

Steven Myers
Indiana University
Bloomington, Indiana

BICENTENNIAL 1807 WILEY 2007 BICENTENNIAL

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION