Submitted to EuroSec 2016.

Practical Encrypted Mailing Lists

Neal H. Walfield
Johns Hopkins University and GnuPG

Abstract

Although email has been one of the most enduring electronic communication mediums and encrypted email has been possible for decades, encrypted mailing lists remain either a usability (and hence security) nightmare or are rather insecure. We propose an extension to OpenPGP that makes encrypted mailing lists both easy to use and secure. Using our extension, a poster encrypts her message to all subscribers. The main difficulty is ensuring that posters have the current list of subscribers. Fortuitously, we can reuse OpenPGP's existing key distribution mechanisms for this without modification. In this paper, we describe how to add encrypted mailing list support to OpenPGP including how to hide the subscriber list, we discuss the work flow for both subscribers and mailing list administrators, and we examine how the mailing list software can improve the user experience and further enhance the system's security.

Categories and Subject Descriptors D.4.6 [*Security and Protection*]: Cryptographic controls

1. Introduction

Just as it is desirable to communicate with someone else securely, it can be desirable to communicate with a group of people securely. Mailing lists are a popular form of group communication for which there is poor support for encrypted communication. The solutions that are available are either insecure by design or have poor usability, which limits their adoption and undermines their security.

We propose an extension to the OpenPGP standard [5] that adds support for encrypted mailing lists. We chose to extend OpenPGP, because it is the preferred standard for secure email. Thus, most people interested in encrypted mailing lists will probably already be using OpenPGP, which significantly lowers the barrier to adoption.

Our OpenPGP extension allows adding a list of encryption keys to an OpenPGP key block. The encryption keys are saved as subkeys, but the parameters are encrypted to hide the subscribers. To create an encrypted mailing list using this scheme, the mailing list administrator simply creates a new key with a special flag and adds each subscriber's public key. To send a message to the mailing list, the poster just selects the mailing list's public key, i.e., the same action as when sending an encrypted mail to an individual. The difference is in how the OpenPGP implementation handles the key: instead of encrypting it using the key's primary encryption key, it encrypts it to all of the listed keys.

In addition to not introducing a new work flow, this approach only requires trusting the mailing list server to relay the message; it doesn't require access to the plaintext like re-encryption gateways [1]. The relay can also not collude with a list member to determine the private key as is the case when using proxy re-encryption [3, 8, 9]. And, unlike when using proxy re-encryption, users can use their usual key. This means there is no need to import a new private key (which conditions users to trust private keys supplied by a third party), it simplifies reading mail on multiple devices, and it allows users to use smartcards. Importantly, propagating updates also doesn't require any new infrastructure: we can use the existing key server infrastructure, which is used to propagate changes (revocations, etc.) to OpenPGP keys.

In this paper, we present our OpenPGP extension for encrypted mailing lists and our implementation for GnuPG. We describe how to modify an OpenPGP key block to include a list of subscribers, how to hide the subscribers, how to efficiently update the list, how to propagate the updates, and how to post a message. We also suggest some checks that the mailing list software can use to improve operational security and usability.

2. Background

Mailing list infrastructure simplifies discussions among a dynamic group of participants. Instead of each poster tracking the set of currently interested and authorized participants, the posters send mail to a list server that forwards it to the subscribers.

An encrypted mailing list has the additional requirement that emails are encrypted to each of the subscribers. This is in conflict with the main purpose of a mailing list: since encryption is done by the sender, the sender now needs the list of subscribers! There are two basic approaches to solve this problem. Either the subscribers' keys are distributed to each poster or a poster encrypts the message to the mailing list's key, which re-encrypts the message. This re-encryption can either be done directly, which exposes the plaintext to the middleware, or by way of proxy re-encryption.

Although we know of several groups that distribute the public keys to each member, we are not aware of any software that simplifies verifying and importing these updates. This significantly decreases the usability of this approach, which in turn seriously harms this system's security. The problem is that failing to verify updates to the subscriber list or not installing updates can allow an attacker to get the plaintext of at least some of the mailing list's traffic.

Schleuder is a popular remailer [1]. Like most remailers, Schleuder has a dedicated key. To post a message, a poster encrypts the message with just the remailer's encryption key and the remailer decrypts the message and re-encrypts it for each subscriber. To add or remove a subscriber, the mailing list's administrator just modifies the mailing list's keyring.

This approach avoids the key distribution problem, but the mailing list server must be trusted, since it handles the plaintext. One can argue that the mailing list server is just one more subscriber and thus giving it access to the plaintext only results in a marginal decrease in the system's security. We are convinced, however, that the mailing list server is potentially more sensitive than individual subscribers, because mailing lists tend to be concentrated. Hosting facilities, such as SourceForge and GitHub, manage not just to a few mailing list, but a huge number. Thus, the scale of a potential compromise is much larger. Further, we know from Snowden's revelations and the Lavabit fiasco that companies readily cooperate (willingly or not) with spying agencies.

Another alternative is to use proxy re-encryption, which allows the mailing list server to re-encrypt a message without access to its plaintext [3]. This is the approach taken in PSELS [8, 9]. Using proxy re-encryption, each subscriber is supplied with a private key that is a random increment of a master key. To re-encrypt a message, the list server doesn't need access to the mailing list's private key, it simply adds the appropriate increment to the ciphertext.

The main problem with re-encryption algorithms is that a subscriber must use a new secret key. This encourages bad security practices by conditioning the user to trust secret keys provided by third parties (in PSELS, they are sent by email). It means the user can't use a smartcard. It makes it harder to read mail on multiple devices. And, users must manage many secret keys (one for each mailing list). Another problem is that the mailing list server and a subscriber can collude to recover the mailing list's secret key [8].

2.1 Goals and Requirements

Our primary goals are to provide encrypted mailing list users with a similar level of security as OpenPGP provides for normal email, and the same work flow. Concretely, only subscribers should be able to access a message's plaintext, and they should not have to install any more software than they normally do to use OpenPGP, or do anything more than what they usually do to send an encrypted email. Further, only users who post to the mailing list should be required to have an OpenPGP implementation that supports our extension.

The subscriber list should also not be public (but, we don't want to hide the subscribers from each other since the messages partially reveal this information anyway). Using SMTP, it is impossible to protect email addresses in transit [6]. But, the difference between the resources required to downgrade TLS connections and passively observe SMTP traffic, and the resources to casually traverse some publicly and permanently stored data years later is huge.

2.2 OpenPGP

OpenPGP is defined by RFC 4880 [5] and is both a message format for storing messages as well as a collection of algorithms that define how to encrypt, sign and encode data.

An OpenPGP message consists of a number of packets, which logically form a nested structure. The most important packets are: symmetrically encrypted data (SED) packets, which contain ciphertext encrypted with a symmetric key; public-key and symmetric-key encrypted session key (SK-ESK and PK-ESK, respectively) packets, which contain a session key for decrypting an SED packet and are encrypted using a public or symmetric key; signature packets, which contain a digital signature over some other packet; and, public key and user id packets, which respectively contain public keys and human-readable identities.

For a public key packet or user id packet to be considered valid, it must be followed by a signature packet whose signature was generated by the primary key. Signature packets also include metadata relevant to the signed packet. This includes cipher and hash preferences, supported features, an expiration time, and notations. Notations are key-value pairs. They can be used for extensions and to make assertions. If an implementation doesn't understand some notation, it simply ignores it unless the notation's critical bit is set. In this case, the implementation must conservatively refuse to do any operations with the key. Most of this information can be updated by generating a new self-signed data packet and sending the new key block to any communication partners.

This key distribution problem is solved in OpenPGP using key servers: after modifying a key, the user uploads it to a key server and communication partners check for updates. OpenPGP treats the key block as an append-only log. This preserves a record of changes to a key's expiry and prevents an attacker from revalidating a revoked key. For most properties, however, only the newest self-signature is relevant.

$$\begin{tikzcd} {} \arrow[r] & \overset{E_{\text{key}_n}}{\boxed{\text{sub } x}} \arrow[r] & \overset{E_{\text{key}_n}}{\boxed{\text{unsub } a}} \arrow[r] & \overset{E_{\text{key}_n}}{\boxed{E_{\text{CP-ABE},\sim a}\ \boxed{\text{key}_{n+1}}}} \arrow[r] & \overset{E_{\text{key}_{n+1}}}{\boxed{\text{sub } y}} \arrow[r] & {} \end{tikzcd}$$

Figure 1. When a unsubscribes, the CP-ABE policy prevents her from reading the new symmetric key.

3. Design

To allow users to use their own keys and to ensure that the mail server does not have access to the plaintext, a poster needs to directly encrypt her message to each of the list's subscribers. This means that we need to distribute the subscribers' keys to each authorized poster. To do this, we can include the subscribers' details in the mailing list's key block and take advantage of the existing key distribution infrastructure to ensure that all posters transparently and quickly receive updates to the subscriber list.

The main open design question is then how to store the subscriber list in the key block. There are two primary constraints. The first is since key blocks are effectively append-only logs, we need to be careful to not allow them to become too large. (Currently, GnuPG won't upload key blocks that are larger than 20 MB to a key server, for instance.) Second, since the key block is public, we need to encrypt the list of subscribers so that only authorized posters can read it.

A naïve implementation might encrypt the list of subscribers to the list of authorized posters each time the subscriber list is updated. Although this protects the subscriber list, it makes inefficient use of the available storage: if n users (all posters) subscribe to a new list, there will be n updates, which will consume $O(n^2)$ space.

A better solution is one that stores just the changes. That is, when a user subscribes to or unsubscribes from the list, we don't write out the whole subscriber list, but just a short record indicating what user was added or removed.

The question now is how to encrypt these records. A simple solution is to encrypt the records using a symmetric key that is only available to authorized posters. Such a key can be generated when the list is created. Then, when a new poster is added to the list, this key is encrypted using the poster's public key and added to the mailing list's key block. This construct ensures that the subscriber list can only be read by posters. Further, since each update consumes $O(1)$ space, n updates require just $O(n)$ space!

This approach has the disadvantage that new posters find out who unsubscribed, and removed posters can continue to decrypt records added after they were removed from the list.

We can fix the latter problem by rotating the symmetric key when a poster is removed. This can be done efficiently using a ciphertext-policy attribute-based encryption (CP-ABE) scheme [2, 7] that supports non-monotonic (negative) access policies [10]. Using CP-ABE, keys are associated with a set of attributes and an access policy is associated with each ciphertext. To decrypt a ciphertext, we need the right key with a set of attributes that satisfies the policy.

We can use CP-ABE to efficiently rekey when a poster is removed. When creating a list, we generate a new CP-ABE key. Then, when a poster is added to the list, a secret key is derived from the master CP-ABE key with a unique attribute, the key is encrypted using the new poster's public key, and the result is included in the mailing list's key block. To rotate the symmetric key, we encrypt the new symmetric key using the CP-ABE key with the access policy $\sim X$, where X is the attribute of the poster being removed, and then we encrypt the result using the current symmetric key. See Figure 1.

The access policy prevents X from accessing the new symmetric key and, as such, from decrypting subsequent records. Encrypting with the current session key means that we only have to exclude X and not all posters who have been removed in the past. This is essential since the storage requirements of the access policy are $O(n)$ where n is the size of the formula. Since our scheme only has a single condition, the size of the ciphertext is $O(1)$!

This scheme does not protect against collusion. By construction, when Alice is unsubscribed from the mailing list, she can decrypt the symmetric encryption, but not the CP-ABE encryption protecting the new session key. She can, however, provide the CP-ABE ciphertext to another unsubscribed user, who can use his CP-ABE key to decrypt it. Excluding every unsubscribed user in access policy would cause the size of the cipher text to be $O(n)$, which is what we were trying to avoid. If this attack is a real threat, then a simple solution is to simply rotate the mailing list's key.

Although this scheme prevents removed posters from reading future events, new posters can still determine all past subscribers even if they are no longer subscribed. This is necessary, because, by construction, a poster traverses all events to determine the current list of subscribers. We consider this a minor security problem: in practice, subscribers are often given access to the mailing list's archive, which allows them to largely reconstruct the list of past subscribers anyway. If this is a serious problem, the mailing list's key can be periodically rotated. In this case, only those subscribers who were removed since the last key rotation are exposed.

To rotate a mailing list's key, we simply revoke the mailing list's key and issue a new one in the usual way. By indicating the new key in the old key's revocation certificate, the rotation can be fairly painless. In the future, this should be entirely transparent: we have submitted a proposal for the next version of the OpenPGP specification that provides a standard, machine-readable way to indicate the new key.

4. Implementation

We now consider how to integrate our design into OpenPGP. The main issues are: creating a list; adding a subscriber; removing a subscriber; and, posting a message.

Primary key:	
mailing-list	Version information. (Notation.)
subscriber-list-session-key	Initial encryption key for subkey data. Encrypted with the CP-ABE key. (Notation.)
User IDs:	
comment field: *mailing list*	Human readable indicator.
Subkeys:	
public-key	The public key parameters and the key's creation time encrypted with the current *subscriber-list-session-key* key. (Notation)
public-key-encrypted-with	Index of the key used to encrypt the public parameters. (Notation.)
subscriber-list-key	The subscriber's CP-ABE key encrypted with the subkey. Only set if the subscriber is an authorized poster. (Notation.)
subscriber-list-session-key	A new session key. Encrypted with the CP-ABE key and the current session key. (Notation.)
subscriber-list-session-key-encrypted-with	Index of the key used to encrypt the *subscriber-list-session-key* notation. (Notation.)

Table 1. Summary of our OpenPGP extensions to support encrypted mailing lists.

4.1 Mailing List Creation

To create a mailing list, we start by generating a new OpenPGP key in the usual way.

To indicate that the key corresponds to a mailing list, we set the *mailing-list* notation[1] in the primary key's self-signed data or the primary user id's self-signed data (although more appropriate, the former is rarely used in practice). Further, since notations are not normally shown, we set the user id's comment to *mailing list*. The notation's critical bit doesn't need to be set if the mailing list server can recognize that a mail was only encrypted to the mailing list, which it usually can by checking the key ids stored in any PK-ESK packets. If this happens, it can forward the message to the list's owner who can re-encrypt it. This clearly introduces some latency and an additional burden on the mailing list's owner, however, it provides some additional backwards compatibility.

To allow an easy upgrade path, the value of the *mailing-list* notation could either be a version identifier or a list of required or desired features.

We store the initial symmetric key used to encrypt the subscriber list (key 0) in the primary user id's self-signed data under the *subscriber-list-session-key* notation. This notation contains a PK-ESK packet that is encrypted using the CP-ABE key with an unrestricted access list. This allows any poster to access it, and, by extension, the list of subscribers.

In addition to the address for the mailing list's exploder, mailing lists typically also have an alias for reaching the mailing list's owner. Since the mailing list's owner controls the list's key, we can make it easier for subscribers to securely reach the mailing list's owner by adding an appropriate user id. To make its purpose clear, the comment should be set to a standard string, perhaps *mailing list: owner*.

Related addresses, such as one for an email accessible interface, shouldn't be directly added to the key: they need use a different secret key. To make them accessible, they can be specified using some standardized notations.

When creating the key, the mailing list owner should choose reasonable preferences (preferred cipher, hash, etc.). When a key is added to the list, the OpenPGP implication should check that the key supports the chosen preferences. This avoids multiple subscribers with incompatible preferences forcing a downgrade to weak defaults.

4.2 Adding a Subscriber

To add a new subscriber to the mailing list, we need to add the user's encryption key to the list of subscribers. The key needs to be encrypted so that only posters can read it. And, if the subscriber is authorized to post to the list, we need to derive a CP-ABE key for her.

To add a subscriber's key, we simply store it in a new subkey packet. (If a subkey already exists with the specified public key, then we don't create a duplicate. This happens when a user unsubscribes and then later resubscribes.)

It is not possible to fully store an OpenPGP key in a subkey packet: an OpenPGP key consists of a primary key, subkeys, user ids, preferences, signatures, etc. However, the only data that we need to store is the data required for a poster to encrypt a message to the subscriber; everything else is irrelevant. This data consists of the user's encryption key and a bit of meta-data, specifically, the key's creation time. This is needed to compute the key's id, which is stored in the PK-ESK packet to make it easy to find the right decryption key. This data fits perfectly in the existing subkey structure and its corresponding self-signature.

Because an OpenPGP key may have multiple valid encryption keys, the OpenPGP implication needs to choose one if the subscriber did not specify the one to use. Although OpenPGP does not make a recommendation of how to choose among multiple valid encryption keys, in GnuPG, the newest valid encryption-capable subkey is used and we recommend this approach here as well.

[1] Actually, *mailing-list@gnupg.org*. Unstandardized notations must include the vendor's domain name, but we exclude it here due to lack of space.

4.2.1 Privacy

Because we want to protect the user's identity, we encrypt the user's public key parameters with the the current symmetric key. We store the encrypted parameters in an SED packet under the *public-key* notation on the subkey. We also store the index of the symmetric key used to encrypt them in the *public-key-encrypted-with* notation. Note: we just use a simple encrypted packet and not one that is integrity protected, because notations are already signed.

We replace the original public key parameters with a small fixed integer (specifically, the number 2). We chose this instead of using a random number, because generating good keys is expensive and generating bad keys makes analysis of valid keys (e.g., [4]) more difficult. Further, this provides a cheap check (for both machines and humans) to determine whether the key is a mailing list subscriber key.

We replace the key's creation time with the current time. The most important thing here is to make sure that the selected time is unique among the subscribers. The issue is that the key id is computed from the key parameters and the creation time. Since the key parameters are now constant, the key id is entirely determined by the creation time. Using our scheme, this can result in duplicate key ids when rapidly adding subscribers to a list. Duplicate key ids can confuse OpenPGP implementations, because signatures reference keys within the same key block using just the key id. If we detect a duplicate, we simply increment the time by one second and recheck. If another method is used to chose the creation time, it is also important to avoid dates from the future as this can result in gratuitous warnings.

If the user is a poster, then we also set the notation *subscriber-list-key* to a CP-ABE secret key with a unique attribute. Concretely, we store the CP-ABE key in a secret key packet encapsulated by a PK-ESK packet that encrypts the data using the poster's public key.

Since PK-ESK packets normally include the key id needed to decrypt them and we want to protect the poster's identity, we set the key id to 0. This is a well understood GnuPG extension to hide the key id. Unfortunately, this means that for a poster to find her CP-ABE key, she needs to try to decrypt all of the *subscriber-list-key* notations. Further, at least GnuPG will only try to decrypt PK-ESK's with hidden recipients if explicitly configured to do so.

To overcome these problems, we propose a new scheme called partially hidden key ids. Using this feature we expose, say, 8 bits of the user's key id and clear the other 56 bits. Unlike a 64-bit id, which provides a very good indicator of the likely key, given millions of potential keys, 8-bits reveals very little about the actual key, but it significantly reduces the number of PK-ESK packets that the user has to try to decrypt (most encrypted mailing lists are unlikely to have more than a few hundred subscribers), and will often uniquely identify the required decryption key (since most users won't have more than a few secret keys). Having to try just a single key is important as it reduces gratuitous passphrase prompts and smartcard swapping. Further, some information about the key is leaked by the ciphertext anyway. For instance, messages encrypted with RSA reveal some information about the public key: an encrypted packet contains a random number chosen uniformly between 0 and the public exponent minus one. With enough messages, it is possible to recover the most significant bits.

4.3 Removing a Subscriber

To remove a subscriber, the mailing list administrator simply expires the relevant subkey in the usual fashion.

If the subscriber was also a poster, then we also set the notation *subscriber-list-session-key* to a new symmetric key, which will be used to encrypted future events. As previously described, this key is encrypted with the CP-ABE key and the current symmetric key whose index we also store in the *subscriber-list-session-key-encrypted-with* notation.

We'd like to use an SK-ESK packet to store the new symmetric key. But, despite the name, SK-ESK packets are not directly encrypted with session keys. Instead, they are encrypted with passphrases that are turned into session keys using the S2K key derivation function [5]. Instead of introducing a new extension, we simply convert the symmetric key to hexadecimal and use it as the passphrase.

4.4 Sending a Message

To send a message, a poster needs to first get the current list of subscribers (or rather, their keys). If the mailing list key hasn't been refreshed recently, the OpenPGP implementation should first do this or, in the very least, print a warning that the mail might not reach all current subscribers.

To get the list of subscriber keys, we just need to iterate over the subkey self-signatures. The ordering is important, because we rotate the symmetric key used to encrypt the subscriber data when a poster is removed. As already noted, the first symmetric key is in the primary key's self-signed data. To find the subscribers added before the first key rotation, we find all subkeys that were encrypted by that key, which we can easily do by finding all self-signatures whose *public-key-encrypted-with* notation is 0. Note: if any of the subkeys have expired, then the user has unsubscribed and should not be included in the subscriber list.

To get the next symmetric key, we find the valid self-signature that contains the *subscriber-list-session-key* notation encrypted using the current session key (again using the *subscriber-list-session-key-encrypted-with* notation).

If there are any unprocessed self-signatures, we repeat the above steps with the new index. Otherwise, we are done.

5. Increasing Usability and Security

Because keys may be updated and revoked, it is essential that the mailing list owner periodically refresh the subscribers' keys to make sure that they are still valid and that the best encryption key is used. (This should, of course, be automated.)

If this is not the case, then either the offending subkey should be expired or rotated, respectively.

When the mailing list software receives a mail, it should first check that the set of apparent recipients (as determined by the key id in the PK-ESK packets) matches its view of the subscriber list. (The mailing list owner needs to provide this directly to the mailing list software or provide it with the CP-ABE key so that it can decrypt the subscriber list. It should obviously not be added as a subscriber as then it will be able to read the plaintext.) If some subscribers are excluded or some unsubscribed keys are included and the recipients are not explicitly listed in the mail's to or cc header, the mail should be held and the poster informed that her version of the mailing list key is probably not up to date.

If the message is not encrypted at all, then the mailing list software should warn the user. It can also refuse to post the message and send a note to the mailing list owner to make her aware of the subscriber's poor opsec practices.

Before forwarding a mail, the mailing list server can sign the message. This can't be done using the mailing list's key, since it is not available. Instead, a special subkey could be used. To improve integration with existing applications, the encrypted part should not be encapsulated in a literal packet, but the original OpenPGP message should be modified to include another signature outside of the encrypted part.

Since our extension only impacts key management, it is entirely possible to implement it in an external application, which the OpenPGP calls when it detects the extension.

6. Evaluation

To evaluate our extension, we modified GnuPG to support encrypted mailing lists. Our implementation doesn't support CP-ABE cryptography, which we leave for future work. Thus, the initial session key is stored in each *subscriber-list-key* notation instead of a CP-ABE key. In this model, rotating keys is not strictly necessary, but for completeness, we keep this functionality and just encrypt the contents of the *subscriber-list-session-key* notations with the current symmetric key and not also the CP-ABE key.

Our implementation is available in the *neal/encrypted-mailing-list* branch of the GnuPG git repository and consists of about 2000 lines of changes.

In our prototype, a new 2048-bit mailing list key without any subscribers requires about 1.5 KB of storage. (A normal 2048-bit key initially uses 1.2 KB.) Adding a subscriber who is authorized to post to the mailing list adds 1 KB to the key's size and removing a subscriber adds another 400 bytes. In a full implementation, these values will be slightly larger since we will also have the CP-ABE key. Nevertheless, it appears that we can easily absorb ten thousand events before we have to rekey to due to a too large key size (which, as we noted, is about 20 MB). In practice, we'd probably want to rekey long before this point, due having to process all records to determine the current set of subscribers.

7. Conclusions and Future Directions

We presented the design and implementation of encrypted mailing lists for OpenPGP and demonstrated its feasibility by adding support for it to GnuPG. Unlike existing solutions, our design doesn't require the mailing list software to re-encrypt the messages nor does it require users to have new secret keys. Instead, we make use of OpenPGP's existing key distribution infrastructure to distribute the list of subscribers to the mailing list's posters. This makes our implementation more secure and more usable.

Our proposed solution is as secure as OpenPGP and as usable as any of the OpenPGP implementations. First, sending a mail to an encrypted mailing list is no different from sending an encrypted mail to some individual. Second, since we publish the list of subscribers in the mailing lists key block, we also encrypt the list of subscribers so that only posters can read it. This prevents casual post hoc analysis of the subscriber list, which provides a similar amount of privacy as OpenPGP encrypted email.

We are currently working to integrate our proposal into the next version of the OpenPGP specification or to publish it as a standalone RFC. If the OpenPGP community agrees that the extension is worthwhile, then we will work to complete our GnuPG support and integrate it upstream.

References

[1] Schleuder. `https://schleuder2.nadir.org/`.

[2] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[3] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology—EUROCRYPT'98*, pages 127–144. Springer, 1998.

[4] Hanno Böck. A look at the PGP ecosystem through the key server data. Cryptology ePrint Archive, Report 2015/262, 2015. `http://eprint.iacr.org/`.

[5] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.

[6] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor MITM...: An empirical analysis of email delivery security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 27–39. ACM, 2015.

[7] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[8] Himanshu Khurana, Jin Heo, and Meenal Pant. From proxy encryption primitives to a deployable secure-mailing-list so-

lution. In *Information and Communications Security*, pages 260–281. Springer, 2006.

[9] Himanshu Khurana, Adam Slagell, and Rafael Bonilla. SELS: a secure e-mail list service. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 306–313. ACM, 2005.

[10] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. In *Public-Key Cryptography–PKC 2014*, pages 275–292. Springer, 2014.