LEAP: A next-generation client VPN and encrypted email provider

Elijah Sparrow[1], Harry Halpin[2], Kali Kaneko[1], and Ruben Pollan[1]

[1] LEAP Encryption Access Project
PO Box 442 Seattle, WA 98194, USA
`elijah, meskio, kali@leap.se`
[2] INRIA, 2 Simone Iff 75012,
Paris, France
`harry.halpin@inria.fr`

Abstract. As demonstrated by the revelations of Edward Snowden on the extent of pervasive surveillance, one pressing danger is in the vast predominance of unencrypted messages, due to the influence of the centralizing silos such as Microsoft, Facebook, and Google. We present the threat model and architectural design of the LEAP platform and client applications, which currently provisions opportunistic email encryption combined with a VPN tunnel and cross-device synchronization.

Keywords: encryption, email, VPN

1 Introduction

Why in the era of mass surveillance is encrypted email still nearly impossible? Take for example the case of the journalist Glenn Greenwald, who could not properly set-up encrypted email when Edward Snowden contacted him to leak the NSA secrets. This lack of progress in over three decades in securing emails is precisely what allows both content and meta-data analysis of email by agencies such as the NSA to be pervasive and nearly inescapable. Well-understood technologies such as OpenPGP-based email encryption are not used by the vast majority of people for reasons that have been understood for nearly a decade and a half [7]. While there has been considerable progress in the deployment of increased use of TLS and even IP-address level anonymity via the Tor project [1], most people rely on insecure and centralized silos for email. There are few working solutions for encrypted and privacy-preserving email. While Tor provides the best solution for IP-level anonymity, this purpose is defeated when users rely on centralized email systems, where the danger of their communication being intercepted via disclosures by the service provider are considerable [1]. For example, many users simply use Tor to 'anonymize' their access to email services such as Gmail that can simply hand over their data, or even systems such as *riseup.net* that likely have all outgoing and ingoing traffic monitored even if the server itself refuses requests for user data. Although email is often sent over an encrypted network channel via TLS and upgraded from an insecure channel using STARTTLS, typically network traffic is not properly authenticated and even network-level encryption tends to fail during mail transfer [2].

Beyond email, Off-the-Record messaging for chat works well, but requires synchronous chat between two users,[3] while Signal is not an open standard or decentralized. [6]. High-profile efforts such as Mailpile are aimed at essentially replacing the user-experience of Thunderbird and Enigmail, not at actually solving the underlying problems of key management and provisioning encrypted email.[4] Although message security rests entirely on a foundation of authenticity, since without proper validation of encryption keys a user cannot be assured of confidentiality or integrity, current systems of establishing message authenticity are so difficult to use that many users simply ignore this step. To achieve mass adoption of encrypted email, the steps of key provisioning, key validation to determine message authenticity, and managing the server-side must be done automatically so that email is encrypted opportunistically. Opportunistic email encryption also needs to include an excellent client-side user experience, particularly if there are errors that the server cannot resolve.

Our solution to this problem is called LEAP, a recursive acronym for the "LEAP Encryption Access Project." LEAP is still in development, although the core functionality of basic opportunistic encryption email is now available for beta testing.[5] The project source-code on Github is available to all.[6] LEAP infrastructure will be supported by providers such as *riseup.net*.

2 Goals and Requirements

2.1 Goal

The primary goal of LEAP is to provide easy-to-use software for *end-to-end encrypted communication* between individual users. The long-term goals are that the communication services should offer a user experience free of any 'privacy tax' on the user in the form of limited features as well as any additional cognitive load and labor compared to non-encrypted communication. It should be backwards-compatible with existing SMTP (Simple Mail Transfer Protocol) email. Thus, LEAP's primary goal is enabling the use of OpenPGP-enabled SMTP, but in a more secure and user-friendly way than commonly used today by toolsets such as Thunderbird and Enigmail. Thus in addition, we have chosen to prioritize the following secondary goals:

– *Memorable user identifiers*: Users should be able to utilize familiar and memorable user handles such as *username@domain* that are typically already used in email when identifying themselves for purposes of communication.

[3] http://www.cypherpunks.ca/otr/
[4] http://mailpile.is
[5] To try, follow instructions on http://demo.bitmask.net.
[6] https://github.com/leapcode/

– *Resilience*: The communication system as a whole should continue to function even if most of the organizations and infrastructure that constitute the whole system have been eliminated or compromised by a malicious attacker.
– *Untrusted*: A third-party service provider should not have access to the content of a user's communication (via hosting cleartext, decryption keys, or passwords) and minimize the amount of metadata they can access to the amount needed to route the message.

There are many other possible goals that end-to-end encrypted communications system wish to provide. There are a number of possible goals that we are explicitly not addressed by LEAP at this time:

– *Device protection*: If a user's client device is subject to an ongoing compromise while the device is powered on, then LEAP does not offers security benefits as the private key is stored on the device. Possible mitigations are under investigation.
– *Anonymity*: The LEAP system does not offer anonymous communications at this time as users and service providers are given stable identifiers. However, LEAP may be used in conjunction with IP anonymization such as Tor and LEAP is currently exploring the feasibility of using mix networking for anonymous messaging.

2.2 Threat Model

In our threat model, we are considering two distinct types of attackers, an *active server attacker* that focuses on decrypting messages on the server, and a *global passive adversary* that simply copies all messages in transit between servers (encrypted or not). For attackers, the goal is both (1) to gain access to the content of the encrypted messages and (2) to determine the social graph of who is communicating to whom. For the former goal of decrypting messages, attacking a single server with many clients makes more sense than attacking many clients for most attackers. For this section, we will consider only the first attacker, as the second requires advanced approaches such as mix-networking.

The active server attacker uses either technical attacks or legal means to force a server to hand over the private keys of its users so the attacker can decrypt the encrypted messages. To prevent this, the private key material must not reach or remain in cleartext form on any server. It should be that an attacker cannot decrypt the encrypted message by compromising the server or placing the server under compulsion. An example would be Lavabit, which had a single point of failure in the form of the system administrator himself: Ladar Levinson had access to the key material for all his users, defeating the purpose of having end-to-end encrypted email.[7] Strangely enough, other services such as Protonmail[8] seem to be repeating this flawed model for encrypted messaging. Lastly,

[7] https://www.thoughtcrime.org/blog/lavabit-critique/
[8] https://protonmail.ch

this is trivially true (as shown by the NSA Prism programme) for centralized messaging services such as GMail that do not store the content of messages encrypted. Server seizures are a threat in the USA that legally resist backdoors, such as recent seizures against a Mixminion anonymous re-mailer on *riseup.net*.[9] In terms of the second *global passive attacker* who is aiming at collecting metadata, most systems today offer no protection. Given the difficulty of defending against this attacker, our current system does not currently aim provide metadata anonymity from their perspective. That being said, LEAP aims not reveal the social graph of a user via techniques for key validation such as the OpenPGP keyservers that display the 'web of trust' of users to the public.

2.3 Requirements

When a system claims to offer security for a user's communication data, typically the focus is on confidentiality and integrity. Although confidentiality and integrity are certainly preconditions for any secure system, in order to achieve high usability a public-key communication system should additionally focus on these requirements:

- *High data availability*: Users expect to be able to access their data across multiple devices with little delay and have the data backed up to redundant cloud storage.
- *Automatic public key authenticity*: If key authentication is difficult, then there is low effective confidentiality for any user who might be subject to an active attack. Since existing systems of public key authentication for messages are either very difficult for users or require a central authority, the confidentiality of existing messaging system is often low in practice.

The LEAP architecture is designed around a federated model, like traditional SMTP-based email or XMPP, where each user registers an account with a service provider (that consists of one or more servers) of their choice and runs their own client on a local device to connect to the provider in order to retrieve encrypted e-mail. Both distributed (peer-to-peer) and centralized architectures were considered, but both fell short of our requirements. A detailed analysis of our approach in comparison to others such as is maintained online.[10] In contrast to LEAP's design, competing encrypted e-mail services rely on centralized key escrows or a web-browser that are vulnerable to an active server attacker. On a high level, LEAP's requirements are met in the following manner:

- *High data availability:* A user's message data is client encrypted and synchronized with redundant federated cloud servers and with a user's other devices. Their data quickly downloaded when needed and so not lost if a user's device is destroyed.

[9] http://www.infosecisland.com/blogview/21186-FBI-Overreaches-with-May-First-Riseup-Server-Seizure.html

[10] https://leap.se/en/docs/tech/secure-email

- *Automatic key authenticity:* With the assistance of a network of federated servers containing the latest public key information of their users, the user's client intelligently manages public keys automatically by following a series of rules that embody best practices and so validate public keys to the greatest extent possible. As the public key information of LEAP-enabled users is kept redundantly by a number of different servers, the client can audit the validation of a key without relying on a single trusted server.
- *Unmappability:* As much metadata is possible is stored so that the provider has no access to this information. Key validation is done via Nicknym (as described in Section 3.4) as to not reveal the social graph of users to unnecessary third-parties, unlike OpenPGP's 'web of trust' keyservers. In future work, LEAP will extend this to the metadata of messages in transit (including size and timing information), by incorporating mix networking into the delivery of messages and a CONIKS-style architecture for key validation [4].

2.4 General Design

The design of LEAP tackles each of the requirements for high data availability, automatic key authenticity, and unmappability. The primary new contribution of LEAP is tackling the problem of high data availability while defending against active server attackers: How can we keep the key material from being inaccessible to the server and at the same time having the keys and data available for synchronization across devices?

The problem can be broken down into a number of distinct components: Server-side infrastructure, usable client software, and the fundamental protocols needed to communicate between the server and the client. What is necessary is to have the client and server actively work together in order to encrypt the message, as to prevent the situation where private key materials stored only on the server are only defended by weak defenses such as passwords. Simply storing the private key on a single device of the user, as done by most encrypted mail programs, is not enough as users need to access their email through multiple devices and keep the state of their inbox synchronized. Thus the main problem facing such a system is safely getting the correct keys onto users' devices, a problem known as *key synchronization*. This becomes an even more important problem if best practices such as frequent key rotation are to be employed.

LEAP solves the problem of key synchronization through the installation of a multi-purpose LEAP client application called Bitmask, that appears to the user mainly as an OpenVPN client.[11] However, there is more to Bitmask than just a VPN. Inside of the Bitmask client are the routines for generating, validating, and discovering keys as well as synchronizing keys and related material (such as the status of messages being "read" across multiple devices). The LEAP client appears to be a VPN as many users likely would install a VPN (but not special 'key manager' software) and the VPN provides additional security benefits by

[11] http://openvpn.net/

creating an authenticated and encrypted channel for all traffic between the LEAP client and server.

When a user installs a LEAP client, the LEAP client asks the user for a username and master passphrase, and to select a LEAP-enabled provider. A 'recovery code' is made for the device (currently a 22 digit code) that is used to derive a device key. The first time that an user authenticates a new device against that LEAP-enabled provider after installing a LEAP client, the keymanager on the LEAP client will attempt to perform a Soledad synchronization (Soledad is described in Section 3.2). If the user has created a new account and so no valid keypair is found, the LEAP client will generate a public-private OpenPGP keypair on the user's device. After such generation is completed, the keypair will be symmetrically encrypted with a key derived from the master passphrase and the wrapped key will be uploaded to the remote Soledad replica in the server, in order to let the user add new devices and synchronize them with Soledad. For incoming email, messages are received by the service provider's MX servers, encrypted to the current user's public key, and stored in the user's database in an incoming message queue. The LEAP client then fetches the incoming message queue as part of a periodic Soledad synchronization, decrypting each message and saving it in the user's inbox, stored in the local Soledad database. The mail module exposes the stored messages through a local IMAP server, so that the messages can be accessed using any standard MUA.

Since email is distributed to the client and stored via the Soledad API, any changes to the mailbox will eventually be synchronized to all devices. The mutable parts of the messages and the attachments are kept in separate documents, so that the sync overhead is kept low. Soledad allows for selective synchronization so that header documents can be synchronized first, leaving the ability to download attachments on the background or under demand, which will be specially interesting for mobile.

For outgoing email, the LEAP client runs a thin SMTP proxy on the user's device, bound to *localhost*, and the mail user agent (MUA)[12] is configured to bind outgoing SMTP to *localhost*. When this SMTP proxy receives an email from the MUA, it issues queries to a local keymanager (Nicknym agent) for the user's private key and public keys of all recipients. The message is then signed, and encrypted to each recipient. If a recipient's key is missing, email goes out in cleartext (unless user has configured the LEAP client to send only encrypted email). Finally, the message is relayed to provider's SMTP relay. The LEAP approach outlined is similar to the approach taken by Garfinkle [3] and Symantec,[13] although these systems do not include key discovery, key validation, encryption of incoming messages, secure storage, or synchronization of email among devices.

[12] Such as Thunderbird, Evolution, or Outlook.
[13] http://www.symantec.com/desktop-email-encryption

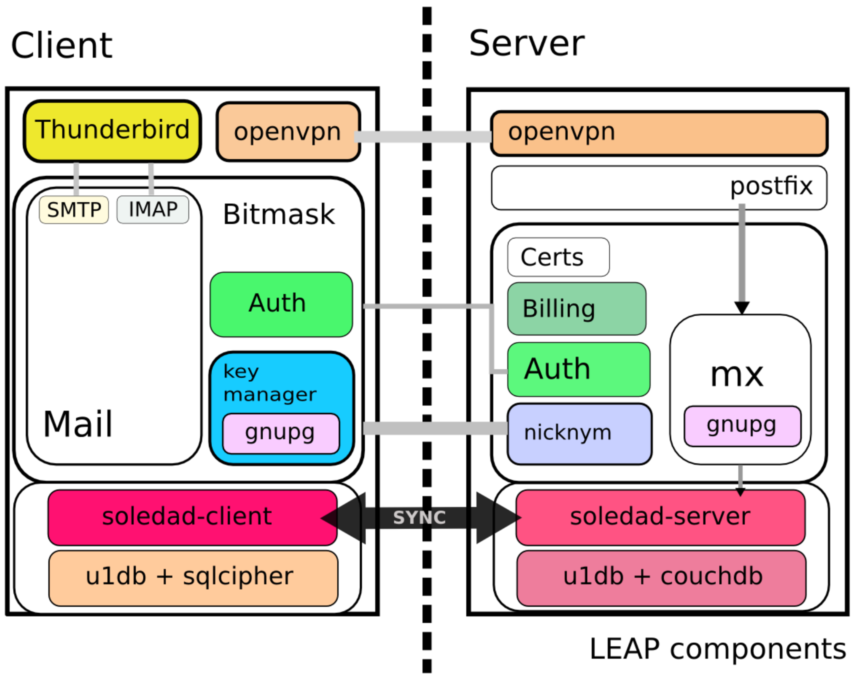

Fig. 1. Components of LEAP Email Architecture

3 The LEAP Architecture

In detail, the LEAP federated architecture consists of three-components: (1) a server-side platform automation system; (2) an easy-to-use client application; and (3) new protocols such as Soledad and Nicknym that allow the user to place minimal trust in the provider, as well as well-known and standardized protocols such as IMAP. These components are illustrated in Diagram 1.[14] The cryptographic details are also subject to change (in particular, migrating from large RSA keys to Curve 25519 when possible) and are maintained online.[15]

The LEAP platform offers a set of automation tools to allow an organization to deploy and manage a complete infrastructure for providing user communication services in the servers controlled by them. The LEAP client is an application that runs on the user's local device and is tightly bound to the server components of the LEAP platform. The client is cross-platform, auto-configuring, and auto-updating, with the initial configuration and updates verified via The Up-

[14] Note that parts of Section 3 are modified versions of material available on the LEAP wiki at http://leap.se/en/docs.

[15] https://bitmask.net/en/features/cryptography

date Framework[16] in order to prevent a compromised server from forcing new key material or accessing the existing client key material via a compromised update.

3.1 LEAP Platform

The "provider instance" is a self-contained encapsulation of everything about an organization's server infrastructure (except for actual user data). The LEAP platform consists of a command line tool and a set of complementary puppet recipes allow an organization to easily operate one or more clusters of servers to provision LEAP-enabled services. With the LEAP command line tool, a system administrator can rapidly deploy a large number of servers, each automatically configured with the proper daemons, firewall, encrypted tunnels, and certificates.

LEAP Data Storage One design goal of the LEAP platform is for a service provider to act as an 'untrusted cloud' where data are encrypted by the client before being sent to the server, and we push as much of the communication logic to the client as possible. There are a few cases where the server must have knowledge about a user's information, such as when resolving email aliases or when processing support requests. Every user has a personal database for storing client encrypted documents, like email and chat messages. In the current implementation, data storage is handled by CouchDB although this may be changed in future versions.

The unencrypted information stored on the server needed to resolve email, including the database for routing incoming and outgoing email, is similar to any traditional email provider, with the one exception that user accounts don't have traditional passwords. Mail is received via a Soledad synchronization session (detailed in Section 3.2) and authenticated using Secure Remote Password (SRP), a standardized password-authenticated key agreement (PAKE) mechanism that can be strengthened via device keys.[17] Mail is sent using SMTP with standard SASL (Simple Authentication and Security Layer) authentication using client certificates.[18] In detail, the unencrypted database of user information maintained by the service provider includes:

- *username:* The login name for the user. This is not necessarily the user portion of 'user@domain.'
- *SRP verifier:* Akin to a hashed password, but used in SRP 'zero-knowledge' dual pass mutual authentication between client and server.
- *uuid:* Random internal identifier for internal usage.
- *identities:* One or more 'user@domain' identities, with corresponding public keys and separate authentication credentials for SMTP (stored as a fingerprint to an x.509 client certificate). Each identity has its own authentication

[16] http://theupdateframework.com/
[17] https://tools.ietf.org/html/rfc2945
[18] https://tools.ietf.org/html/rfc4422

credentials so that the email headers show that the user authenticated with the SMTP server using their identity username, not their real username. Each identity includes delivery information, either to a uuid or to a third party email address that messages should be forwarded to. One or more devices are tied to an identity.

For email delivery, the receiving MX (Mail exchanger record) servers do not have access to the entire database. They only have read-only access to 'identities.' This allows the implementation of the Nicknym protocol (described in Section 3.4) for resolving pseudonyms. Additionally, there are several non-encrypted databases containing the minimal information needed to connect user accounts to optional support tickets and even billing details. The LEAP platform includes a web application for user and administrator access to these non-encrypted databases, although future research will hopefully be able to minimize if not eliminate this information.

3.2 Soledad

Soledad ("Synchronization Of Locally Encrypted Data Among Devices") is responsible for client-encrypting user data, keeping the data synchronized with the copy on the server and on all the other devices of each user, and for providing local applications with a simple API for document storage, indexing, and search that is akin to CouchDB and related document-centric NoSQL databases. The document that is saved and synchronized with Soledad can be any structured JSON document, with binary attachments. Soledad is implemented on the LEAP client to store email messages, the user's public and private OpenPGP keys, and a contact database of validated public keys. Soledad is based on U1DB, but modified to support the encryption of both the local database replica and every document before it is synchronized with the server.[19] Local database encryption is provided by a block-encrypted SQLite database[20] via SQLcipher.[21] Documents synchronized with the server are individually block encrypted using a key produced via an HMAC of the unique document id and a long storage secret. In order to prevent the server from sending forged or old documents, each document record stored on the server includes an additional client-computed MAC derived from the document id, the document revision number, and the encrypted content. The server time-stamps each update of the database, so that Soledad's MAC and HMAC keys used to encrypt the client database can only send the server new databases. Each time the LEAP client is online (both after re-connecting with the LEAP platform and after each pre-set time interval), the client re-synchronizes the messages and key material.

In addition to synchronizing public-private key materials and a contact list of validated keys, Soledad is used to encrypt and synchronize email. This has the

[19] https://one.ubuntu.com/developer/data/u1db/
[20] https://sqlite.org/
[21] http://sqlcipher.net/

benefit of not storing sensitive metadata on the server and allowing for searchable locally-encrypted database of messages on the client side. For efficiency, a single email is stored in several different documents (for example, for headers, for attachments, and for the nested MIME structures). While in transit between LEAP-enabled SMTP servers and the LEAP client, there are three different forms of encryption that a single message is subject to:

1. Encrypted by sender, or on arrival by recipient's service provider using OpenPGP.
2. Decrypted from and re-encrypted in an SQLcipher database using AES block encryption.
3. Individually re-encrypted for storage on a service provider that supports the LEAP platform using block encryption with a nonce.

3.3 LEAP Client

The *LEAP client* (also known as Bitmask[22]) is a cross-platform application that runs on a user's own device and is responsible for all encryption of user data. It currently includes the following components: *Bitmask VPN*, *Soledad* (multi-device user data synchronization), *Key Manager* (Nicknym agent and contact database), and *email proxy* (opportunistic email encryption). The client must be installed in the user's device before they can access any LEAP services (except for user support via the web application). Written in Python (with QT, Twisted, OpenVPN, SQLcipher), the LEAP client currently runs on Linux and Android, with Windows and Mac being under development.[23] When a user installs a LEAP client, a *first-run* wizard walks the user through the simple process of authenticating or registering a new account with the LEAP provider of their choice, using the Secure Remote Password (SRP) protocol so that a cleartext copy of the password never reaches the server. The SRP encoded password can also be strengthened by wrapping it with a device key and so stored multiple times, once per device key. The password's strength is assessed via *zxcvbn*.[24] Note that when a user authenticates with the client, via a username and password, these credentials as provided are used to both authenticate with the service provider (via SRP) and also to unlock locally encrypted secrets (via Soledad).

One threat would be that an active server attacker would compel a LEAP-enabled server to push a malicious update to the clients to compromise their keys. This threat applies equally to any browser or plug-in based approach, and in fact to the installation of *any* software. LEAP employs mitigation strategies to prevent this attack. When distributed through the self-contained bundles, the client has auto-updating capabilities, using The Update Framework (TUF) to update LEAP code and other library dependencies as needed using the same Thandy library as deployed by Tor [5].[25] Unlike other update systems, TUF

[22] https://bitmask.net/

[23] The Android version tends to lag behind development compared to the Linux version due to the design having to be re-coded in Java.

[24] https://github.com/dropbox/zxcvbn

[25] https://gitweb.torproject.org/thandy.git.

updates are controlled by a timestamp file that is signed each day. This ensures that the client will not miss an important update and cannot be pushed an old or compromised update by an attacker. Updates to the LEAP client via TUF require signatures from multiple keys, held by LEAP developers in different jurisdictions. Lastly, LEAP has started work on a system of reproducible builds, which is working in an automated fashion for Android and in the future should apply to all other platforms.[26]

VPN The goal with LEAP's VPN service is to provide an automatic, always on, trouble-free way to encrypt a user's network traffic. The VPN service encrypts all of a user's traffic and works hard to prevent data leakage from DNS, IPv6, and other common client misconfigurations that are not tackled by OpenVPN via a strict egress firewall. Currently OpenVPN is used for the transport. OpenVPN was selected because it is fast, open source, and cross-platform. In the future, LEAP plans to add support for Tor as an alternate transport. We believe LEAP is the only VPN that autoconfigures and auto-restarts when connectivity is lost. When started, the LEAP client discovers the LEAP-enabled service provider's proxy gateways, fetches a short-lived X.509 client certificate from the provider if necessary, and probes the network to attempt to connect. If there are problems connecting, the LEAP VPN client will try different protocol and port combinations to bypass common ISP firewall settings since VPN access is typically blocked crudely by simple port and protocol rules rather than deep packet inspection. In terms of deep packet inspection, obfsproxy[27] integration is under development to hide the VPN connection to an observer. By default, the LEAP client will auto-connect the VPN service the next time a user starts the computer if the encrypted proxy was switched on when the user the client quit or the machine was shutdown. If network connectivity is lost while the proxy is active, the LEAP client will automatically attempt to reconnect when the network is again available. A firewall is also activated before launching the VPN service, providing a fail-close mechanism that limits the unprotected access to the network in case of client malfunction or crashes. Due to its stringent security requirements, the LEAP VPN does not work when the user is behind a captive network portal.

3.4 Nicknym Key Management

One of the main features of the LEAP system is to provide strong authentication of public keys in a way that is easy for users. To do this, LEAP relies on a protocol called Nicknym in the form of *username@domain* (similar to an email address). Nicknym maps user nicknames to public keys. With Nicknym, the user is able to think solely in terms of nicknames, while still being able to communicate with a high degree of security (confidentiality, integrity, and authenticity). Another goal

[26] See work by Debian on reproducible builds that LEAP is applying to its code: https://wiki.debian.org/ReproducibleBuilds.

[27] https://www.torproject.org/projects/obfsproxy.html.en

of Nicknym is to, unlike the OpenPGP 'Web of Trust' mechanism, not reveal the social graph of the user to the public.[28]

Although various new key validation infrastructure schemes have been recently proposed, most of the new opportunistic encrypted email projects have proposed starting with some sort of "Trust On First Use," (TOFU) but the term itself is undefined. LEAP specifies generic rules for automatic key management that can form a basis for defining a version of TOFU and to transition from TOFU to more advanced forms of key validation. In particular, the rules try to define when a user agent should use one public key over another. This section is written from the point of view of Alice, a user who wants to send an encrypted email to Bob, although she does not yet have his public key.

LEAP assumes the goal is to automate the process of binding an email address to a public key. Alice knows Bob's email address, but not his public key and either Alice might be initiating contact with Bob or he might be initiating contact with her. Likewise, Bob might use an email provider that facilitates key discovery and/or validation in some way, or he might not. Unless otherwise specified, *key* in this text always means *public key.* A *key directory* is an online service that stores public keys and allows clients to search for keys by address or fingerprint. A key directory does not make any assertions regarding the validity of an address and key binding. A *key validation level* is the level of confidence the key manager has that it has the right key for a particular address, where *key registration* is when a key manager assigns a validation level, being somewhat analogous to adding a key to a user's keyring. A *key endorser* is an organization such as a LEAP provider that makes assertions regarding the binding of *username@domain* address to public key, typically by signing public keys. When supported, all such endorsement signatures must apply only to the uid corresponding to the address being endorsed. *Binding information* is evidence that the key manager uses to make an educated guess regarding what key to associate with what email address. This information could come from the headers in an email, a DNS lookup, a key endorser, and so on. A *verified key transition* is a process where a key owner generates a new public/private key pair and signs the new key with a prior key. An *endorsement key* is the public/private key pair that a service provider or third party endorser uses to sign user keys. Currently, LEAP implements these rules when encountering new keys or finding keys from other providers.

LEAP Key manager rules

1. First contact: When a new key is first discovered for a particular address, the key's the highest validation level is registered.
2. Regular refresh: All keys are regularly refreshed to check for modified expirations, or new subkeys, or new keys signed by old keys.
   (a) This refresh should happens via an anonymizing mechanism (currently Tor) in order to prevent targeted attacks on particular servers.

[28] Details at https://leap.se/nicknym.

   (b) The key should not be revoked before the expiration date of the key, unless it can be proved that there is a new version of the key or the key has been compromised.
3. Key replacement: A registered key must be replaced by a new key in one of the following situations, and only these situations:
   (a) Verified key transitions (when the new key is signed by the previously registered key for same address).
   (b) If the user manually verifies the fingerprint of the new key (using an out-of-band authenticated channel).
   (c) If the registered key is expired or revoked and the new key is of equal or higher validation level.
   (d) If the registered key has never been successfully used and the new key has a higher validation level.
   (e) If the registered key has no expiration date.

Previously registered keys must be retained by the key manager for the purpose of signature authentication. However, these old keys are never used for sending messages. A public key for Bob is considered successfully used by Alice if and only if Alice has sent a message encrypted to the key and received an authenticated response.

Validation levels A number of validation levels are described, from lowest to highest validation level.

Weak chain: Bob's key is obtained by Alice from a non-auditable source via a weak chain. The chain of custody for 'binding information' is broken as at some point the binding information was transmitted over a connection that was not authenticated.

Provider trust: Alice obtains binding information for Bob's key from Bob's service provider via a non-auditable source over a strong chain. By strong chain, we mean that every connection in the chain used to determine the 'binding information' from Bob's provider to Alice is done over an authenticated channel. To subvert this 'provider-trust' validation, an attacker must compromise Bob's service provider or a certificate authority (or parent zones when using DNSSEC), so this level of validation places a high degree of trust on service providers and CAs.

Provider endorsement: Alice is able to ask Bob's service provider for the key bound to Bob's email address and Bob is able to audit these endorsements. Rather than simple transport level authenticity, these endorsements are time stamped signatures of Bob's key for a particular email address. These signatures are made using the provider's 'endorsement key.' Alice must obtained and register the provider's endorsement key with validation level at 'provider-trust' or higher. An auditable endorsing provider must follow certain rules:

– The keys a service provider endorses must be regularly audited by its users. Alice has no idea if Bob's key manager has actually audited Bob's provider, but Alice can know if the provider is written in such a way that the same

client libraries that allow for submitting keys to a provider allow auditing of these keys. Otherwise, it is considered to be the same as 'provider-trust' validation.
– Neither Alice nor Bob should contact Bob's service provider directly. Provider endorsements should be queried through an anonymizing transport like Tor [1]. Without this, it is easy for provider to prevent Bob from auditing its endorsements, and the validation level is the same as 'provider-trust.'

Note that with provider endorsement, a service provider may summarily publish bogus keys for a user. Even if a user's key manager detects this, the damage may already be done. However, if a provider is suspected of being caught 'cheating,' the evidence should be submitted to a third-party endorser (ideally with an audit log) so that can Bob (and other providers) can query the third-party, as should happen during third-party endorsement and provider consensus. Regardless, a provider endorsement is a higher level of validation than 'provider trust' because there is a good chance that the provider would get caught if they issue bogus keys, raising the cost for doing so.

Third-party endorsement: Alice asks a third-party key endorsing service for binding information, using either an email address of key fingerprint as the search term. This could involve asking a key endorser directly, via a proxy, or asking a key directory that includes endorsement information from a key endorser. A third-party key endorser must follow certain rules:

– The third-party key endorser must be regularly audited by the key manager of users. If there are conflicts, the key should be reduced to 'provider trust' validation until the conflict is resolved.
– The key endorser must either require verified key transitions or require that old keys expire before a new key is endorsed for an existing email address. This is to give a key manager time to prevent the user's service provider from obtaining endorsements for bogus keys. If a key endorsement system is not written in this way, Alice's key manager must consider it to have the same level of validation as 'provider-endorsement.'
– Key endorsers should provide information about key endorsements to key owners to the user through an authenticated channel not controlled by the provider (such as Tor) so the user can detect possible 'cheating' by their own provider.

Third-party consensus: This is the same as third-party endorsement, but Alice's user agent has queried a quorum of third party endorsers and all their endorsements for a particular user address agree.

Auditing: This works similar to third-party endorsement, but with better ability to audit key endorsements. With historical auditing, a key endorser must publish an append-only log of all their endorsements. Independent agents can watch these logs to ensure new entries are always appended to old entries. The benefit of this approach is that an endorser is not able to temporarily endorse and publish a bogus key and then remove this key before Alice's key manager is able to check what key has been endorsed. The endorser could try to publish an entire bogus log in order to endorse a bogus key, but this is very likely to be eventually detected. As with other endorsement models, the endorsement key must be bootstrapped using a validation level of provider trust or higher.

Fingerprint Verification: Alice has manually confirmed the validity of the key by inspecting the full fingerprint or by using a short authentication string with a limited time frame. For established endorsers like LEAP providers, this authenticated key has to be then hard-coded as known by the software.

As currently written, Nicknym relies on an approach based on network perspectives as enabled by Tor (i.e. retrieving the key from multiple network locations given by Tor exit nodes) to detect endorser equivocation, which allows for the possibility that the endorser could publish a bogus key for a short period of time in order to evade detection although eventually a discrepancy would be detected by other key endorsers (via third-party consensus) and the history of endorsed keys via a CONIKS-style approach would also detect this attack [4]. Any endorser equivocation should be widely reported and proven via audit logs so that the users of a malicious endorser can detect the attack. To mitigate the problem of stolen keys (either by a malicious or compromised provider or a third-party adversary), LEAP is working on a system where a user can contact their service provider (revealing their identity) and prove their identity via a one-time passphrase generated at installation of the LEAP client on a device in order to revoke verified key transitions.

4 Current State and Future Work

As of September 2016, the current LEAP architecture provides a VPN service and end-to-end encrypted e-mail service available via *bitmask.net*. The LEAP platform, Soledad, Nicknym, and the basic Key Manager are currently complete. However, there is still ongoing work on greater scalability and reliability for the LEAP platform's encrypted data-storage. On the side of the LEAP client, LEAP is pursuing greater compatibility with existing IMAP clients, improved usability, and better network probing for the VPN. In co-operation with Thoughtworks, we are working on a custom user-interface called Pixelated[29] to be bundled with LEAP client for users looking for alternatives to existing e-mail clients. Immediate goals also include porting LEAP from Android and Unix-based environments (Linux and MacOS) to iOS and Windows environments. Work is ongoing to improve the key validation rules (including key verification revocation) and support validation with multiple network perspectives. In terms of research, LEAP plans to add both Tor and mix-networking for messaging in transit both in between LEAP providers and clients to prevent metadata collection by passive global attackers, support for CONIKS for key validation [4], the use of the Signal protocol or simplified variant between LEAP-enabled providers as a higher-security alternative to SMTP with perfect forward secrecy [6], increased security for key

[29] The source code for Pixelated is available here: https://github.com/pixelated-project/.

material on the server and back-ups for stolen key material using threshold password authenticated key exchange (TPAKE) and secret sharing (TPASS), two-factor authentication for users to strengthen passwords for user authentication, and deploying reproducible builds. In the future, LEAP may expand its basic federated infrastructure to also provide hosting for other end-to-end encrypted and traffic-analysis resistant services needed by users, such as chat and voice-over-IP.

At this moment, email providers such as *riseup.net* provide centralized email providers to tens of thousands of highly sensitive users such as activists that are likely targets of surveillance. Likewise, many ordinary users and organizations want to migrate from centralized silos that are easily compromised by programs such as PRISM. Therefore, it is critical that technical solutions be provided that work today with existing heavily-used protocols such as SMTP to combat surveillance. The LEAP solution, by providing an integrated client and server that stores all SMTP email encrypted without having the server have access to the keys and that automates key management for the user, presents the first open architecture for enabling widespread open federated end-to-end encryption for email.

Acknowledgements: The authors are supported by NEXTLEAP (EU H2020 ref: 688722) and the Open Technology Fund. A final version was published in Cryptology and Network Security (CANS 2016). The final publication is available at Springer via *http://dx.doi.org/10.1007/978-3-319-48965-0_11*.

References

1. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *Proceedings of the 13th USENIX Security Symposium*, 2, 2004.
2. Ian D Foster, Jon Larson, Max Masich, Alex C Snoeren, Stefan Savage, and Kirill Levchenko. Security by any other name: On the effectiveness of provider based email security. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 450–464. ACM, 2015.
3. Simson L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. In *Proceedings of the 2003 annual national conference on Digital government research*, dg.o '03, pages 1–4. Digital Government Society of North America, 2003.
4. Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015.
5. Justin Samuel, Nick Mathewson, Justin Cappos, and Roger Dingledine. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 61–72, New York, NY, USA, 2010. ACM.
6. Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE, 2015.

7. Alma Whitten and J. D. Tygar. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.