ProtonMail Security Features and Infrastructure

Proton Technologies A.G.

8 July 2016

Contents

Introduction

ProtonMail is a secure email system servicing over 1 million customers around the world, ranging from private individuals to large enterprises. It aims to provide a much higher level of security than traditional email services without adversely impacting usability.

To achieve such security, ProtonMail conservatively assumes that all mail servers may eventually be compromised. Thus, ProtonMail uses end-to-end encryption to ensure that plaintext email data is never sent to the server. If a server only contains encrypted messages, then the risks of a central server breach are mitigated.

ProtonMail's security extends beyond just strong encryption. We have seen time and time again that the human factor is the weak link in enterprise security. End user passwords can frequently be compromised by insecure connections, phishing, or malware. ProtonMail takes several additional steps to guard against this. First, ProtonMail uses strong authentication which makes most brute force or dictionary attacks impossible – even if an attacker has compromised the connection between client and server. Second, ProtonMail's encryption protocols ensure that a single compromised account does not endanger other accounts.

We firmly believe that the most secure system is one that users will actually use. Thus, ProtonMail was designed from the ground up with a strong emphasis on usability. To accomplish this, we built the first encrypted email system where the encryption is entirely automatic and invisible to the end user. For usability reasons, we retain compatibility with legacy email protocols such as IMAP and SMTP so ProtonMail accounts can be accessed from existing email clients and can seamlessly communicate with non-ProtonMail email accounts. However, because of the inherent insecurity of IMAP and SMTP, ProtonMail uses a bridge service to maintain encryption and authentication without sacrificing IMAP/SMTP support.

While ProtonMail can be deployed either in the cloud or on an organization's premises, we are firm believers in the cloud as the future of all enterprise software. ProtonMail's cloud offerings provide the best of both worlds. Organizations can benefit from the security and reliability advantages of the cloud, while retaining data control and data privacy due to the end-to-end encryption. Further, the economies of scale of the cloud imply a much lower cost of ownership for email infrastructure. For these reasons, ProtonMail is primarily deployed in the cloud.

The goal of this document is to provide a more detailed look at the technology behind ProtonMail. The first sections cover the technical details for ProtonMail's authentication and encryption technology. The next sections discuss the ProtonMail's extensive administrative tools and how key management is handled within an organization, followed by details of how ProtonMail securely supports legacy email clients. Lastly, an overview of ProtonMail's secure cloud infrastructure is provided, with a discussion of the technologies we utilize to ensure maximum data uptime and availability.

Authentication

ProtonMail users enter a user-chosen password on each login, but while ProtonMail's backend is responsible for validating and resetting the password, the password cannot be derived by either ProtonMail or an attacker with access to the network. This is achieved with the Secure Remote Password protocol, which as detailed below, conveys a zero-knowledge password proof from the user to the server. The security granted by this protocol extends to the user's private keys, which are encrypted with a salted hash of their password before being sent to the server. For additional security, users also have the option of enabling two-factor authentication.

Issues with Traditional Password Authentication

Most online services send the cleartext password or password equivalent to the server on every login. If the server is compromised, whether from malicious code injected onto the server or due to a memory exposure such as in the recent Heartbleed vulnerability, user passwords or password-equivalents can be leaked no matter how they were salted and hashed.

Moreover, if the encrypted TLS layer of the connection to the server is broken, passwords can simply be read from network traffic by any intermediary system between the client and server. This possibility is not as unreasonable or unlikely as it may seem. There have numerous incidents of certificate authorities issuing fraudulent certificates or computers being changed to trust insecure authorities. In 2001, VeriSign issued false Microsoft certificates; in 2011, Comodo and DigiNotar issued false certificates to several websites, including Google and Mozilla; in 2012 it came to light that Trustwave had created a subordinate root certificate capable of attacking a connection to any website; in 2015 it was revealed that Lenovo laptops were shipped with Superfish, software that, among other things, caused the system to trust a root certificate with a publicly known private key. This problem is exacerbated by the certainty that a state actor could force a certificate authority to issue fraudulent certificates.

In contrast, the Secure Remote Password protocol [11] promises theoretically optimal security. When using SRP, even an attacker who can arbitrarily read, modify, delay, destroy, repeat, or fabricate messages between ProtonMail and a legitimate user in an undetectable fashion is limited to checking only a single password guess per login attempt, a task which could be done just by trying to log in directly. Even if a server is compromised and acts maliciously, password-equivalent information is never revealed. This is all done without permanent private keys: all secret information is derived from the user's password.

The Secure Remote Password Protocol (Version 6a)

The Secure Remote Password (SRP) protocol can be viewed as a variation of the more well-known and widely deployed Diffie-Hellman key exchange. As in Diffie-Hellman, SRP's security in the face of eavesdroppers and other attackers

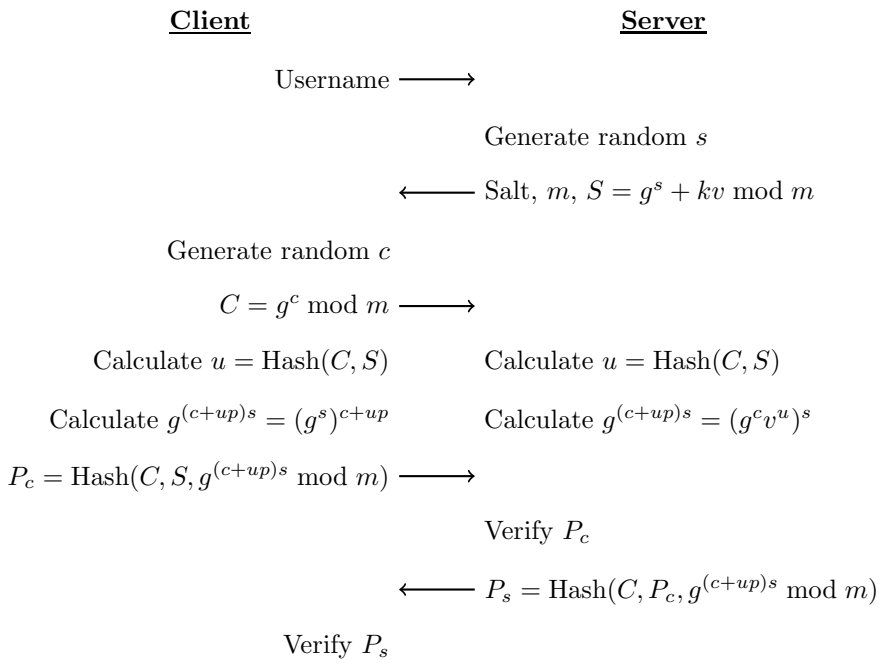

Figure 1: The Secure Remote Password Protocol, as implemented in ProtonMail

relies on the difficulty of the discrete logarithm problem: given a fixed prime number N and g, it is easy to compute $g^x \bmod N$ from x, but not the other way around. Accordingly, for a password p (pre-hashed and salted, both to make dictionary attacks slow and to ensure that there are no weaknesses due to predictability), the server stores the verifier $v \equiv g^p \bmod N$. This verifier can be computed on the client side when setting a password, avoiding the need for the server to see any password-equivalent data. For login, the SRP protocol proceeds in two phases. In the first stage, the client and server generate a shared secret, following the pattern of Diffie-Hellman. In Diffie-Hellman, both parties generate random ephemeral public-private key pairs as a random secret a and $g^a \bmod N$. Then, they can each mix their private key with the other party's public key, producing a shared secret: $(g^a)^b = g^{ab} = (g^b)^a \bmod N$. SRP differs from this by mixing the verifier and the password into the key pairs, thereby causing a mismatch if the password and the verifier do not match.

On the server side, the generation of the ephemeral key pair proceeds normally: the private key is a randomly chosen s, and the public key is $g^s \bmod N$. However, when transmitting the public key to the client, the verifier is mixed in, and $S \equiv kv + g^s \bmod N$ is sent for a random constant k (generated as a hash of N and g). The client then calculates the actual server public key by computing $S - kg^p$.

On the client side, the password is mixed into the private key. Although the

client generates a random c and sends across $C \equiv g^c \bmod N$, the actual private ephemeral key is $c + up$, where u is a mixing parameter derived as a hash of C and S. The client can only calculate this private key by knowing the password p, while the server can calculate the public key from only the verifier as follows:

$$\begin{aligned} Cv^u &= g^c(g^p)^u \\ &= g^c g^{up} \\ &= g^{c+up} \end{aligned}$$

Finally, the client and server generate a shared secret as in standard Diffie-Hellman, finding $g^{s(c+up)} = (g^s)^{c+up} = (g^{c+up})^s$.

The second phase of SRP is the actual authentication phase, in which the client and server prove to each other that they hold the same secret. This only happens when the password held by the client corresponds to the verifier held by the server. Verification is a fairly simple process – the client sends a hash of the shared secret, the server's semi-public ephemeral key ($g^s \bmod N$), and some public data for randomization. In response, the server sends of hash of the shared secret, the user's knowledge proof, and some public data for randomization.

In the first phase, the only sensitive value sent over the network is the verifier mixed into the server's public ephemeral key. However, since s is uniformly random and g is chosen as a generator mod$N$, $g^s \bmod N$ is uniformly distributed (except for 0), and therefore perfectly scrambles the verifier, rendering the message harmless.

In the second phase, assuming the hash function used is secure (in the random oracle model), an attacker cannot figure out anything about the hashed data except via search over possible shared secrets. Since the shared secret is large and randomly distributed, brute-force attacks are infeasible, and generating the shared secret, even from a known password, is assumed to be difficult without knowledge of one of the private keys, which would take discrete logarithms to find. Therefore, an attacker cannot even mount a dictionary attack on a user's password by observing an SRP connection.

Choosing a Modulus

SRP relies crucially upon working modulo an N that makes calculation of discrete logarithms difficult. In particular, when $N-1$ is made up of comparatively small factors, the Pohlig-Hellman algorithm makes it possible to break the problem down into discrete logarithm problems of difficulty proportional only to the size of those factors. Therefore, to minimize this risk, ProtonMail uses safe primes of the form $2p + 1$, where p is another prime number.

However, choosing a single safe prime may be insufficient. With algorithms like the number field sieve algorithm, it is possible to do a significant amount of precomputation on an arbitrary modulus to be able to calculate discrete logarithms efficiently in that modulus. While the amount of work necessary is

prohibitive for a one-off calculation, it seems within the reach of state actors to do such a computation on a 1024-bit modulus, and there is evidence that such a computation has already occurred [1]. At ProtonMail, we take a conservative approach towards this threat. First, we use 2048-bit moduli, which ought to be out of reach for even state actors for quite some time. Second, we have opted to not use a single modulus for all users. This greatly reduces the impact of an attack on an SRP modulus, as such an attack would only affect a small fraction of users.

To defend against an MITM (man-in-the-middle) attacker feeding the client a fraudulent, broken modulus, we have two layers of security. First, the modulus is included in the password hash itself, meaning that in the worst case, the attacker would only be able to access information about a different hash of the password than the one used to actually log in. This reduces potential compromise to at worst a dictionary attack. Second, we send the client signed moduli which can be verified to ensure that the modulus actually came from ProtonMail.

Improvements over RFC 5054

A version of the SRP-6a protocol has been standardized by the IETF in RFC 5054 [9] for use in negotiating secure, authenticated TLS connections. Unfortunately, the RFC seems too outdated to be acceptable for use at ProtonMail.

First and foremost, we have deep security concerns around the use of SHA-1 as a hashing algorithm. For password hashing in particular, SHA-1 is highly problematic: In the event of a database breach or the discovery of a weakness in the SRP protocol, attackers would primarily execute dictionary attacks, and so modern password hashes are designed to be slow and memory hungry to impede high-speed, highly-parallel password cracking. SHA-1 is specifically designed to have neither of these two crucial properties. Moreover, SHA-1 is not tunable – there is no clear way to scale up the password hashing cost as computing power increases. In contrast, ProtonMail uses bcrypt, a time-tested, tunably slow hashing algorithm designed for passwords.

Beyond its issues as a password hashing algorithm, SHA-1 is far too short to be used safely in SRP. Many algorithms for computing discrete logarithms, prototypically Pollard's kangaroo algorithm [8], have runtimes that only depend on the range of possible exponents, not the full size of the modulus. In the face of those algorithms, SRP using SHA-1 has security roughly equivalent to using a 180-bit modulus, which is well within the range of breakability.

Additionally, though the bulk of the attacks on SHA-1 are collision attacks that have little bearing on the security of SRP, SHA-1 has recently been showing its age, and it is difficult to be confident that SHA-1 is or will be sufficiently secure. As such, ProtonMail uses MGF-1-SHA-512 [5, B.2.1] both to expand the bcrypt hash to a full 2048 bits and to generate the u and k scrambling parameters.

Second, RFC 5054 is meant as an implementation of authentication for the TLS protocol. While it has its flaws, the more traditional certificate-based TLS

authentication is extremely well tested, studied, supported, and updated. By wrapping our implementation of SRP in a traditional TLS channel, we can leverage the immense body of work that has gone into making existing TLS solutions secure, improve privacy by encrypting usernames, and guard against novel attacks on the less well-tested SRP protocol by preventing even eavesdroppers in the common case.

Two Factor Authentication

Two-factor authentication (2FA) can be optionally enabled for added security. 2FA is a method of confirming identity that requires not only that the user know information (e.g. their login password), but also that the user possess a particular physical device (ex. a phone, computer, or hardware key) configured with their 2FA shared secret. ProtonMail implements the Time-based One-Time Password algorithm (TOTP) [7], which computes a single use passcode from a shared secret key and the current time measured in 30 second intervals. A TOTP passcode is only valid for a limited time, which prevents brute-force and replay attacks.

When 2FA is first enabled for an account, the user is given a shared secret key that they can enter into any TOTP-enabled application or device. Examples include the Google Authenticator, Authy, and 1Password smartphone applications, and Yubico Authenticator, which stores the shared secret on a hardware device called a Yubikey. When a user wants to sign in to their account, the chosen application will use the TOTP algorithm to provide the correct passcode corresponding to the user's secret key. This passcode will need to be entered along with the correct login password in order to access the account. To prevent locking users out of their accounts if they lose their 2FA device, users are also given 16 single use recovery codes when they enable 2FA. A valid recovery code along with the correct login password will also allow users to enter their account, where they can disable 2FA on the lost device and re-enable it on a different device.

Organization administrators are empowered to reset 2FA settings for non-private member users.

Email Encryption

PGP Overview

The PGP protocol utilizes a combination of public key and symmetric cryptography that offers two primary benefits for communications and email: confidentiality, whereby only designated parties can read a particular communication, and authenticity, whereby the recipient can verify the identity of the sender and detect whether the communication has been tampered with in transit.

Public key cryptography utilizes a pair of keys for each party – a public key used to encrypt messages sent to the party that can be widely disseminated, and

a private key used to decrypt messages sent to the party that is known only to that party. Not only should an encrypted message not reveal any information about the message, but a public key should not reveal any information about its associated private key. Both of these properties are achieved by using mathematical problems that are computationally infeasible to solve. For example, in RSA, the default public key cryptosystem provided by PGP, the private key (d) and the public key (exponent e and modulus n) are related such that for all messages m, $(m^e)^d \equiv m \bmod n$. RSA's security relies on two properties: that the message m cannot be derived from the encrypted message $m^e \bmod n$ even with knowledge of the public key, and that the private key d cannot be derived even with knowledge of e, n, and/or m. In order to derive either of these variables, an attacker would have to factor the public key modulus n, the product of two large unknown primes, a problem which is believed to be extremely computationally difficult for sufficient key length.

Unlike public key cryptography, symmetric key cryptography utilizes only one key, which is used for both encryption and decryption of a message. Symmetric key algorithms tend to be much faster than public key algorithms, but require that both sides of the communication have access to the same key. The requirement that this secret key be securely shared between the sending and receiving parties is the main drawback of symmetric key cryptography.

Thus, in order to take advantage of the secure key distribution of public key cryptography and the speed of symmetric key cryptography, PGP combines a public key algorithm with a much faster symmetric key algorithm. For example, ProtonMail's implementation of PGP uses the public key algorithm RSA and the symmetric key algorithm AES-256.

Message encryption with PGP goes as follows:

1. After the sender creates a message, a random 256-bit number is generated that will only be used during this transaction. This is called a "session key".
2. PGP symmetrically encrypts (AES-256) the message using this session key.
3. The session key is encrypted with the chosen public key algorithm for each recipient using their public key. Because the session key is only 256 bits long, the relative slowness of the public key algorithm does not significantly increase the overall computation time.
4. These encrypted session keys are prepended to the encrypted message and are sent together to all desired recipients.

To read a PGP-encrypted message, the recipient will:

1. Decrypt the encrypted session key (with their private key) meant for them, yielding the original session key
2. Use the session key to decrypt the encrypted message, yielding the original message

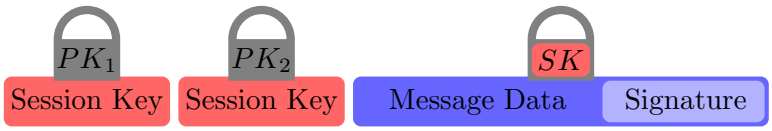

Figure 2: The makeup of a PGP message with two recipients with public keys PK_1 and PK_2. The session key SK is separately encrypted with the two public keys, and the message data is symmetrically encrypted with the session key. The signature is optional.

Because no other party has the ability to decrypt any of the encrypted session keys, and thus the ability to decrypt the message, the message will only be available to the sender and intended recipients.

In addition to encrypting the message, the sender can optionally choose to provide a digital signature, which will allow the recipients to verify that the message is indeed from the sender and has not been tampered with in transit.

To digitally sign a communication, the sender will:

1. Produce a hash of the message using a hash function that must be both one-way (the hash value does not reveal any information about the message) and deterministic (the function always produces the same result on the same input)

2. Use their private key to sign this hash value using a digital signature algorithm. This signed hash value is called the digital signature

3. Append the digital signature to the message before encryption and sending (which will proceed as detailed previously).

In order to verify the signature, the recipient will:

1. Decrypt the message as explained previously, yielding the digital signature and the original message

2. Use a signature verification algorithm corresponding to the signing algorithm used by the sender, along with the sender's public key, to produce the hash value from the digital signature

3. Calculate the hash value of the decrypted message using the same hash function as the sender.

4. Check whether the two hash values from steps 2 and 3 are equal. If they are equal, the signature is verified.

If the signature is verified, then the recipient knows that the message was signed with the sender's private key and has not been changed since.

Implementation of the OpenPGP Standard

ProtonMail implements OpenPGP [2], the most widely-supported PGP standard, using OpenPGP.js on the web client and a native code implementation on the mobile applications and IMAP/SMTP bridge. OpenPGP.js is an open-source Javascript library that is actively used and vetted by the security community, including two independent security audits by the German cybersecurity firm Cure53. The native implementation uses cryptographic primitives from OpenSSL, one of the most ubiquitous and popular implementations of TLS/SSL. Both of these libraries provide key management functions as well as encryption, decryption, signing, and signature verification of messages/attachments. ProtonMail's interface to each of these functions is detailed in the following subsections.

Key Distribution and Management

Upon creating a ProtonMail account, every user generates a RSA public key/private key pair. These keys are generated client-side with a user ID (the user's ProtonMail email address) and a passphrase (the user's mailbox password, known only to the user) as inputs. The private key is symmetrically encrypted with the mailbox password using AES-256. The public key and encrypted private key are then stored on the ProtonMail server along with the user's other account information and retrieved whenever a user logs in successfully. The encrypted private key is decrypted on successful mailbox password entry on the user's local device and can be used to read and sign messages during that session. Because the private key is stored encrypted by the mailbox password, and the mailbox password is known only to the user, ProtonMail cannot read the user's messages nor impersonate the user.

User accounts may have multiple email addresses associated with them, and each address will have one or more sets of public/private keys. The primary set is used for encryption/signing, while the secondary (usually older) sets are used for decryption to ensure readability of older messages.

Sending Encrypted and Signed Messages and Attachments

A message and its attachments can be sent encrypted from a ProtonMail account to any email address with an available corresponding PGP public key. If the email address is associated within ProtonMail, the recipient's public key will be automatically retrieved. If the email address is not associated with a ProtonMail account, the corresponding public key must be imported by the user and saved in a contact. Message and attachment encryption in these two cases are handled differently, as detailed below.

Internal Emails (ProtonMail to ProtonMail) In this case, messages and attachments are encrypted and signed separately. Signing is optional for attachments because it requires re-downloading the encrypted attachments from the

server on message send, which can be slow, and signatures often go unchecked. Sending without signing only requires the encrypted session key packets to be downloaded and re-encrypted with the recipients' public keys. Messages are always signed because there is no performance penalty. Keeping messages and attachments separate helps facilitate responsive webmail and IMAP interfaces, as large attachments do not need to be downloaded from the server in order to read the associated message. This also enables fast forwarding of messages, as attachments can be copied server-side.

External Emails (ProtonMail to another PGP email client) There are two PGP formats available for external PGP encryption – PGP/Inline and PGP/MIME. Sending an external PGP/Inline email works exactly the same way as internal encryption – messages and attachments are separately encrypted and signed as detailed above. The downside to Inline PGP from the user's perspective is that external clients often do not support HTML emails with PGP/Inline, but they are much more suited to webmail than PGP/MIME. PGP/MIME combines the message body and attachments in a multipart MIME form, and then encrypts and signs them together. This method boasts full support for HTML emails but can be slower if the message included large attachments, as the attachment must be downloaded and the large encrypted body re-uploaded to send.

Decryption and Signature Verification

ProtonMail can decrypt and verify internal emails (from another ProtonMail account) as well as any other emails PGP-encrypted with the user's public key as long as the sender's public key is in the user's contacts.

Internal Emails If an email is sent from another ProtonMail account, then the message and attachments have been encrypted separately. In this case, the message is decrypted and verified as it is read. The attachments are decrypted and verified only if downloaded.

External Emails If an encrypted email sent from outside ProtonMail is in the PGP/Inline format, then decryption and verification are the same as internal emails, with the exception that the sender's public key may not be known to the user. In this case, the message and attachments will be decrypted but the signatures cannot be verified.

If the encrypted email is in the PGP/MIME format, then the messages and attachments have been combined and encrypted as a whole. In this case, the combined form is decrypted and the signature optionally verified immediately.

The user will be alerted to any irregularities with the signature verification.

Password-Protected Messages

ProtonMail's Encrypt-to-Outside feature (EO) allows a user to communicate with someone without PGP keys outside of the ProtonMail platform in a fully end-to-end encrypted manner. EO encryption takes place on the client device and is available in both web and native mobile application formats. Using EO encryption requires a shared secret (password) known to both parties communicated to eachother via other means (e.g. phone). An optional password hint and/or expiration time for the message can also be set. By defult, all messages expire after 28 days if an earlier expiration date is not specified.

The encryption password is used to encrypt (with AES256) both the message itself as well as a randomly-generated token. The encrypted message, encrypted token, and the plaintext token are then sent to the ProtonMail server. It's important to note that the encryption password is never saved or sent to ProtonMail at any time which maintains a zero-knowledge result.

The ProtonMail server then sends the designated recipient a generated email notifying them that they have a message waiting. This generated email contains a very long, unique link to access it. These links are very long and rate-limited to prevent brute-force guessing. The notification message can be white-labeled/customized.

Upon navigating to the unique link in a web browser the visitor is prompted to decrypt the random token with the password. If successful, the visitor can use the token to authenticate to the server and retrieve the message, attachments, and associated metadata. Message decryption is then performed with the password.

The authenticated visitor can also reply to the message. These replies are encrypted with the ProtonMail user's public key and also end-to-end encrypted.

A maximum of 5 replies can be sent per EO message. This limit is to prevent abuse by the recipient.

Moreover, sending and replying fully support end-to-end encrypted attachments.

Administration

ProtonMail for Enterprise combines maximum protection for corporate data against unauthorized access with the tools necessary for successful organization management and regulatory compliance.

The Organization

Each ProtonMail for Enterprise client has an organization defined within the ProtonMail Cloud. An organization is a collection of users with at least one administrator which share email domains, billing, and cloud resources. Organization resources, such as storage space, can be provisioned to individual users by organization administrators. Organizations also have an associated public/private key pair. This key pair is shared among the administrators and can

be used to access and read mail in non-private member accounts as well as sign organization data.

Roles

There are two possible user roles within an organization. The basic role is that of the organization member, who can read and send email but cannot administer the organization, assign or remove addresses, modify their storage space, etc. The other user role is that of administrator. Administrators can add and remove users, domains, addresses, storage space, and modify the organization's name and other characteristics. They have access to and can modify organization billing and subscription information and they can also access non-private user accounts and perform actions as the subordinate user. As administrators have absolute power within the organization, it is recommended that administrator accounts not be used for regular mail purposes and instead only be used for administration.

Domains and Addresses

Domains (i.e. mycorporation.com) are added and managed by organization administrations. Once the domain is correctly configured and verified, addresses associated with the domain can be provisioned to members of the organization. Users can have multiple addresses, but mail is organized separately by address. All addresses in the ProtonMail system ignore case, hyphens, periods, and underscores when routing mail, which reduces misrouted mail. Additionally, ProtonMail-hosted addresses support '+'-style subaddresses for routing (e.g. mail sent to test+abc@example.com is routed to test@example.com, as is mail sent to test+def@example.com).

User and Key Management

Administrators are empowered to create new organization members and assign them addresses, a role, and a storage quota. Regardless of role, each user can be either a private or non-private user. For compliance reasons, most if not all corporate users are non-private. The difference between private and non-private users is who has ultimate control over the user's encryption keys. For private users, this is the user herself. She generates her own keys, and no one but her can read his correspondence. For non-private users, an administrator generates the encryption keys used for the account and saves a copy for administrative use. This allows the administrator both to read the user's mail if necessary and also restore account access in the event the user forgets her mailbox password. Administrators can also reset the login passwords and 2FA information of non-private users.

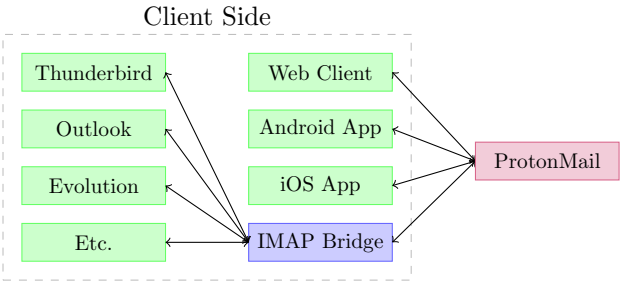

Figure 3: The different ways to connect to ProtonMail

Import/Export

Data import is achieved via a downloadable, cross-platform tool which seamlessly transfers data from the previous provider to ProtonMail. All mail is encrypted within the tool before sending the encrypted payload to ProtonMail. This ensures that the ProtonMail servers never see the unencrypted data. Export is achieved via the same tool run in reverse.

Data Retention

Data retention periods for non-private users are enforced by disabling full message deletion for messages newer than a configured cutoff age, drafts excepted.

Email Client Compatibility

ProtonMail is compatible with existing email clients such as Thunderbird, Outlook, and others via a client-side IMAP/SMTP bridge that facilitates all encryption and decryption operations and communicates with the ProtonMail API.

The IMAP and SMTP protocols [3][6] are the commonly implemented standard for interaction between email clients and servers. Unfortunately, few IMAP/SMTP client implementations provide end-to-end encryption. To retain compatibility with these clients while still offering end-to-end encryption, our bridge acts as a local intermediary that encrypts emails after they leave the client but before they leave the user's computer.

This bridge runs on Linux, OS X, and Windows and is downloadable from the ProtonMail website. Once installed, it acts as a proxy server for the ProtonMail API. All communications between the bridge and the cloud API are authenticated using the SRP protocol and secured using TLS/SSL. Additionally, because the bridge is a static application installed locally, external malicious code changes are prevented. Moreover, by keeping the decryption software separate from the actual email client, vulnerabilities in clients cannot allow an attacker to steal encryption keys. Using the bridge is simple and only requires

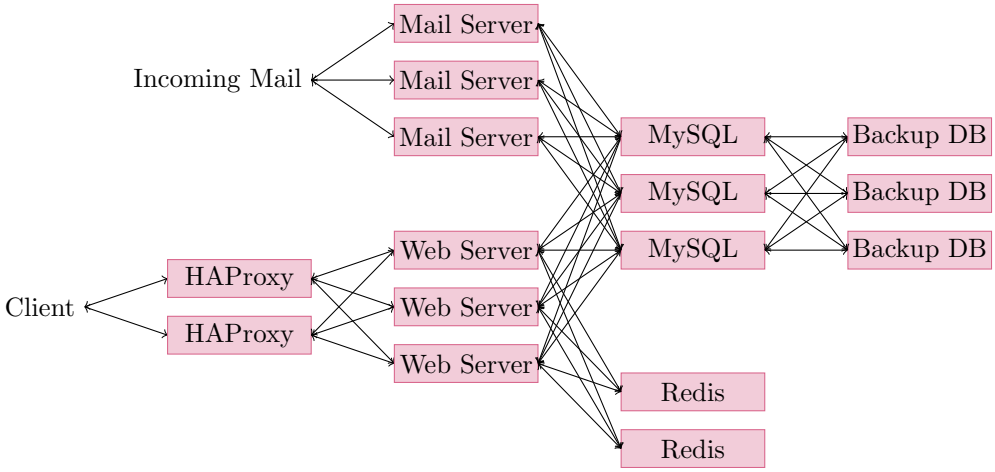

Figure 4: ProtonMail's infrastructure

entering "localhost" and custom port numbers as the IMAP and SMTP servers in an existing email client.

Sending email through the IMAP/SMTP bridge is seamless from the user's perspective. When email is sent, the bridge reads the recipients and determines the appropriate encryption to use (ProtonMail internal, EO/PGP if configured in the ProtonMail contacts, or unencrypted for external recipients). If a recipient is a ProtonMail user, the recipients' public keys are fetched using the ProtonMail API and used to encrypt the message using standard PGP encryption. EO and PGP recipients are handled similarly except that the password/keys are retrieved and decrypted from the user's encrypted contacts. The message is only sent to the ProtonMail server post-encryption, which preserves end-to-end security.

Incoming email is also handled through the bridge. When the bridge receives a new message from the ProtonMail API, the user's private key is used to decrypt the message on the fly. Because the decryption only happens locally the integrity of the end-to-end encrypted communication is maintained.

Other IMAP actions are translated to ProtonMail API commands as appropriate. For example, copying a message to a user-defined folder is interpreted as labeling the message, and deleting a message from a user-defined folder is interpreted as removing the corresponding label. Only deletions from the Trash folder are interpreted as actual deletions.

Infrastructure

The ProtonMail server infrastructure is heavily distributed and redundant to ensure high reliability and performance. The main parts of the system are shown in Figure 4.

Mail Servers

ProtonMail use MX DNS to load balance incoming SMTP traffic to multiple mail servers. This redundant setup ensures that if a mail server is taken offline, incoming mail will be automatically rerouted to the other servers. The MTA used for receiving and sending emails is Postfix, which is open source, secure, and widely used. In order to encrypt SMTP traffic, TLS 1.0 or higher and perfect forward secrecy using Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange are supported while vulnerable protocols such as SSL 3.0 and weak cipher suites are disallowed to prevent downgrade attacks. To filter mail, we deploy OpenDKIM (verifies and signs DKIM), OpenDMARC (verifies DMARC), ClamAV (rejects viruses), and SpamAssassin (assigns a score to incoming email according to how spam-like the message is). These "milters" funnel clean emails to the inbox and spoofed or spam emails to the spam folder. Postfix passes processed incoming mail to the application layer, which does further processing (including custom filtering) and encrypts the email with the user's public key. The encrypted email is then stored and cannot be decrypted by anyone except the recipient user.

Web Servers

ProtonMail's web servers, powered by Apache, serve API requests originating from the browser-based web application and the native iOS/Android mobile apps. HSTS ensures that all traffic uses HTTPS and HPKP to prevent man-in-the-mittle attacks. As with incoming mail, data in transit is encrypted using a recent version of TLS with strong cipher suites – as of July 2016, these measures earn ProtonMail domains A+ ratings on Qualys SSL Labs' test. Our servers run hardened versions of CentOS and are load balanced by redundant HAProxy servers, which automatically reroute traffic if any web server goes offline. With our own range of IPs, we ensure that all outgoing emails are sent from servers with good IP reputation and won't be summarily rejected by spam filters.

Database Servers

All of our critical user data resides on MySQL databases, which are widely used and battle-tested by Internet giants like Facebook and Dropbox. While there are newer and fancier database solutions available, none are mature or reliable enough to meet our standards, and none can match MySQL when it comes to existing community and support.

To improve performance, data is horizontally sharded across multiple database servers. Each shard unit consists of a master that handles both reads and writes and replicates its data to two slaves. The primary slave is in the same datacenter as the master and is ready to take over if the master goes down, while the secondary slave is in a separate datacenter in case of datacenter-wide failures. Backups are regularly made for disaster recovery, including to cold storage. All shard units are maintained and operated with identical tools so that adding

more shards does not actually add more complexity. By knowing exactly where each piece of data is and having full control of the replication topology, we ensure that we can scale up without sacrificing reliability.

To maximize performance, Redis servers act as a cache layer above MySQL for frequent reads and transient data. The Redis servers are also deployed in a fully redundant configuration to eliminate single points of failure.

Network and Facilities

ProtonMail's cloud infrastructure runs on a dedicated network for both security and reliability reasons. Our network spans three datacenters in Switzerland, two of which are ISO 27001 certified. Our datacenter providers include Deltalis SA and Equinix (NASDAQ: EQIX). Our data storage facilities are spread out across both Western and Eastern Switzerland for geographic diversification and directly connected to major Internet Points of Presence (PoP) in both Zurich and Geneva.

We use an unique mix of facilities that have a special emphasis on security due to the special requirements of our customer base. Our primary datacenter in Attinghausen is located in the former Swiss air force command and control bunker under 1000 meters of solid rock and linked to the outside world through a ultra reliable dark fiber link maintained by the Swiss Federal Railways for signaling purposes along the Gothard tunnel. Being connected to critical Swiss national infrastructure helps to ensure the highest possible uptime on our uplinks.

Proton Technologies AG is also a Local Internet Registry (LIR). Since 2014, we have been a member of RIPE NCC (Réseaux IP Europens Network Coordination Centre) with a dedicated allocation of IPv4 and IPv6 addresses. This allows us to control the IP reputation of our own subnet for optimal email deliverability and isolate our network infrastructure away from other entities which may be prone to failure. This also serves to protect our network from unauthorized tapping or other network based attacks because we maintain full control over our network up until the main Swiss Internet PoP in Zurich.

Denial of Service Resistance

Proton Technologies AG has in place sophisticated monitoring on all levels of our network to detect intrusions and other malicious activity on our network. We have also partnered with Radware (NASDAQ: RDWR) to protect our network against external threats such as Distributed Denial of Service (DDoS) attacks, in which attackers flood a network with requests, attempting to make it unavailable to its users. Such protection is increasingly necessary because DDoS attacks have now become the most common form of cyberattack, a trend which is expected to continue (see Figure 5).

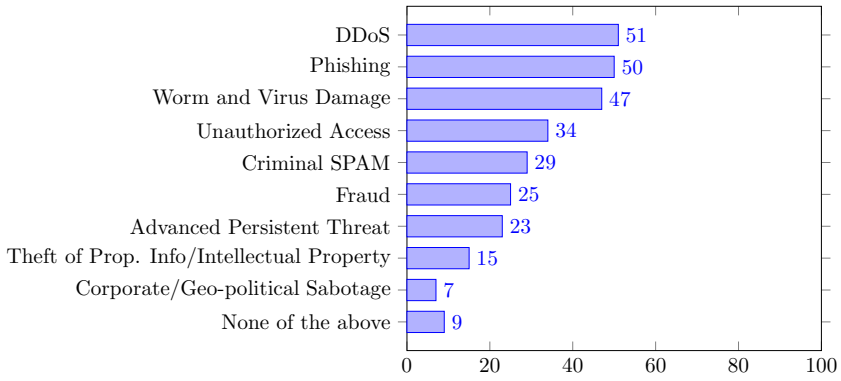

Figure 5: Percentage of companies experiencing various cyberattacks [10]

Together with Radware and other network infrastructure partners, we operate a 24/7 Network Operations Center with a 24/7 Emergency Response Team to swiftly respond to any network based attacks and ensure maximum uptime. ProtonMail uses a DDoS protection system based off of Border Gateway Protocol redirection. When an attack is detected, all network traffic is immediately diverted to dedicated scrubbing centers that remove attack traffic. During an attack, we usually divert traffic to DE-CIX in Frankfurt because ProtonMail has previously experienced attacks that would seriously stress the national network capacity of Switzerland.

After the removal of attack traffic, clean traffic is delivered to our network edge via redundant GRE tunnels [4]. Thus, attack traffic is prevented from hitting our border routers and overwhelming their capacity. The system that has been implemented in partnership with Radware is capable of withstanding up to 2 Tbps of attack traffic and has successfully defended against attacks reaching up to 120 Gbps (for comparison, the largest DDoS attack ever launched was 500 Gbps). Due to the frequency with which our network sees attacks, DDoS protection is usually maintained in an "active" state which allows our network to automatically respond within 18 seconds of an attack with no impact to customers.

Conclusion

As a whole, ProtonMail's cloud based encrypted email services provide a strong combination of security, usability, and reliability that is essential for any enterprise. Consistent with our company values of transparency and peer review, most of ProtonMail's code is open source and available for review by the security community. As part of that community, we are also actively engaged in both public and private efforts to improve the state-of-the-art in encryption technology. Thus, specifications and implementation details discussed in this whitepaper are always subject to change as improvements become avail-

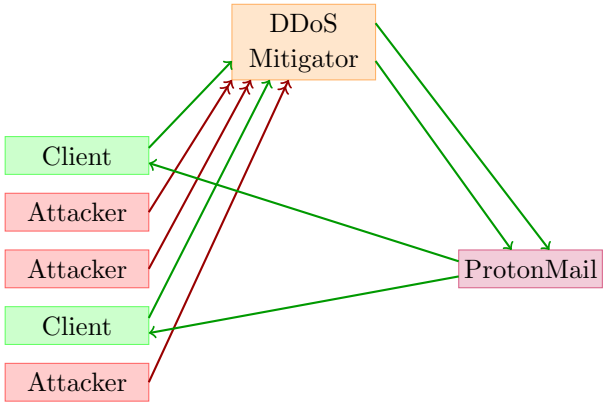

Figure 6: The network configuration when ProtonMail is under attack

able through continued research and development. For the most up to date whitepaper, please contact your account manager.

References

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, October 2015.

[2] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP message format. RFC 4880, RFC Editor, November 2007. `http://www.rfc-editor.org/rfc/rfc4880.txt`.

[3] M. Crispin. Internet message access protocol - version 4rev1. RFC 3501, RFC Editor, March 2003. `http://www.rfc-editor.org/rfc/rfc3501.txt`.

[4] Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. Generic Routing Encapsulation (GRE). RFC 2784, RFC Editor, March 2000. `http://www.rfc-editor.org/rfc/rfc2784.txt`.

[5] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA cryptography specifications version 2.1. RFC 3447, RFC Editor, February 2003. `http://www.rfc-editor.org/rfc/rfc3447.txt`.

[6] J. Klensin. Simple Mail Transfer Protocol. RFC 5321, RFC Editor, October 2008. `http://www.rfc-editor.org/rfc/rfc5321.txt`.

[7] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-based One-Time Password algorithm. RFC 6238, RFC Editor, May 2011. `http://www.rfc-editor.org/rfc/rfc6238.txt`.

[8] John M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32(143):918–924, July 1978.

[9] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) protocol for TLS authentication. RFC 5054, RFC Editor, November 2007.

[10] Radware Emergency Response Team. Global application & network security report. Technical report, Radware, 2015-2016. Figure 4.

[11] Thomas Wu. SRP-6: Improvements and refinements to the Secure Remote Password protocol. Submission to IEEE P1363.2 working group, October 2002.