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

Login and Mailbox Passwords

ProtonMail's novel authentication implementation protects data against current and future attacks. Even if new cryptanalytic breakthroughs or implementation flaws completely subvert the authentication system, ProtonMail and remote attackers should never be able to read a user's email. To achieve this, users create two passwords. The mailbox password, which is used to encrypt a user's private key, has a very simple security guarantee: it is never sent to the ProtonMail server in any form, except in that the server stores the mailbox password-protected private key. The login password, which ProtonMail uses for authentication, is also never sent over the wire, but since the ProtonMail backend is responsible for validating and resetting the login password, its security, deriving from the Secure Remote Password protocol, is significantly more complex.

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.

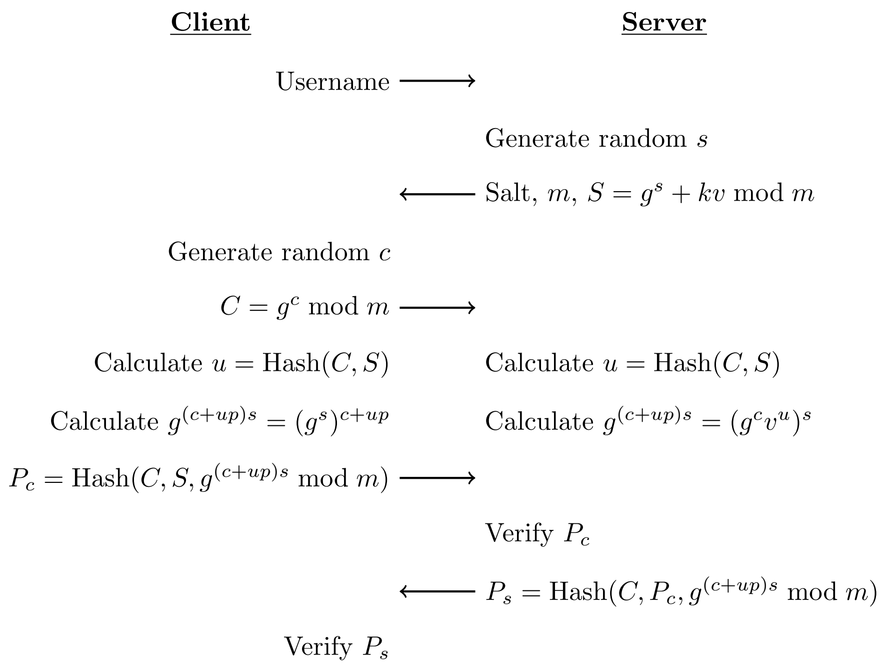
Client
Server
Username $\longrightarrow$
Generate random s
$\longleftarrow$ Salt, m, $S = g^s + kv \bmod m$
Generate random c
$C = g^c \bmod m$ $\longrightarrow$
Calculate $u = \mathrm{Hash}(C, S)$
Calculate $u = \mathrm{Hash}(C, S)$
Calculate $g^{(c+up)s} = (g^s)^{c+up}$
Calculate $g^{(c+up)s} = (g^c v^u)^s$
$P_c = \mathrm{Hash}(C, S, g^{(c+up)s} \bmod m)$ $\longrightarrow$
Verify P_c
$\longleftarrow$ $P_s = \mathrm{Hash}(C, P_c, g^{(c+up)s} \bmod m)$
Verify P_s

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

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 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. login and mailbox passwords), 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 and mailbox passwords 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 and mailbox passwords 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.