An Analysis of the ProtonMail Cryptographic Architecture

Nadim Kobeissi
Symbolic Software
nadim@symbolic.software

November 24, 2018

Abstract

ProtonMail is an online email service that claims to offer end-to-end encryption such that *"even [ProtonMail] cannot read and decrypt [user] emails."* The service, based in Switzerland, offers email access via webmail and smartphone applications to over five million users as of November 2018. In this work, we provide the first independent analysis of ProtonMail's cryptographic architecture. We find that for the majority of ProtonMail users, no end-to-end encryption guarantees have ever been provided by the ProtonMail service and that the *"Zero-Knowledge Password Proofs"* are negated by the service itself. We also find and document weaknesses in ProtonMail's *"Encrypt-to-Outside"* feature. We justify our findings against well-defined security goals and conclude with recommendations.

1 Introduction

ProtonMail[1] is an email service founded in 2014 and based in Geneva, Switzerland. By promoting its claims of end-to-end encryption features, the service was able to receive initial support through a crowdfunding campaign and now supports itself through paid plans being offered adjacent to its free-of-charge service. As of January 2017, ProtonMail claimed to have over 2 million users [1]. This number grew to over 5 million users by September 2018 [2]. ProtonMail is primarily accessed through its webmail interface, with iOS and Android applications launching in 2016, two years after the service was opened to the public.

ProtonMail has published a technical specification [3] detailing its *"security features and infrastructure"* in July 2016. Despite the strong security claims made in this paper and despite ProtonMail's success, no independent formal analysis of ProtonMail's security claims has been published.

In this work, we provide the first independent analysis into ProtonMail's cryptographic design.

[1] ProtonMail is accessible at https://protonmail.com.

- We begin by defining a set of security goals in §2 which are based on ProtonMail's own claims as well as on cryptographic definitions for standard security goals.
- In §3, we establish the cryptographic primitives relevant to our analysis. This allows us to describe ProtonMail's protocol flows in §4.
- Based on our comparison of ProtonMail's security goals against its protocol flows, we present a security analysis §5 in which we find serious shortcomings with regards to ProtonMail's effective security guarantees. we provide recommendations in our conclusion (§6).

In carrying out this analysis, we evaluated ProtonMail's security claims against well-defined security goals. We found that ProtonMail's cryptographic architecture ultimately does not guarantee end-to-end encryption for the majority of users and that Zero-Knowledge Password Proof security goals are effectively not provided for any ProtonMail user. Furthermore, we also uncover weaknesses in ProtonMail's *"Encrypt-to-Outside"* functionality which is intended to allow ProtonMail users to send end-to-end encrypted email to recipients that do not use ProtonMail.

2 Security Definitions

> *"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 Security Features and Architecture Specification [3]

We begin with security definitions for each of ProtonMail's security claims. Each security definition is motivated by claims made on ProtonMail's website and is defined in such a way as to be consistent with how such security goals are understood in modern cryptographic literature.

2.1 Security Assumptions

Our security definitions concern three **clients**: ProtonMail user $\mathcal{A}$, ProtonMail user $\mathcal{B}$ and Microsoft Outlook[2] user $\mathcal{S}$. We also consider two **servers**: ProtonMail webmail server $\mathcal{P}$ and Microsoft Outlook webmail server $\mathcal{M}$. These principals operate under the following network assumptions:

- **Transport Layer Security.** We assume that all communications between all principals occur over an authenticated TLS link.

[2] For simplicity, we use Microsoft Outlook as a running example of any third-party run-of-the-mill email service that is not ProtonMail. This could also potentially be Apple iCloud Mail, Gmail, FastMail, etc.

- **No Client State Compromise.** We assume that none of the clients $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{S}$ ever suffer a local state compromise.
- **Untrusted Server.** We assume that $\mathcal{P}$ is untrusted and could act with the intent to recover encrypted communications between clients $\mathcal{A}$, $\mathcal{B}$, $\mathcal{S}$. We treat $\mathcal{M}$ as controlled by an adversary.

The Untrusted Server assumption is directly informed by the above-mentioned quotes from ProtonMail.

We are therefore assuming a relatively safe threat model where transport layer communications are always encrypted and where local state compromise never occurs.

2.2 End-to-End Encryption

> *"ProtonMail's zero access architecture means that your data is encrypted in a way that makes it inaccessible to us. Data is encrypted on the client side using an encryption key that we do not have access to. This means we don't have the technical ability to decrypt your messages, and as a result, we are unable to hand your data over to third parties. With ProtonMail, privacy isn't just a promise, it is mathematically ensured. For this reason, we are also unable to do data recovery. If you forget your password, we cannot recover your data."* — ProtonMail Security Details Page [4]

End-to-end encryption denotes a collection of properties inherent to a cryptographic protocol, almost always a secure channel protocol. These properties, known as *confidentiality*, *integrity* and *authenticity*, first became available to the general public through the introduction of the *Pretty Good Privacy* (PGP) email encryption protocol in 1991. *Off-the-Record Messaging* (OTR), a communications system that meant to supersede PGP [5] built upon these properties and expanded them with new ones: *forward secrecy* and *deniability*. A decade later, rigorous formal analysis [6, 7, 8] of the Signal Protocol [9, 10], which itself was in turn inspired by OTR and which currently encrypts all messages sent through WhatsApp and other applications, further formalized the security properties of modern end-to-end encryption systems. Tangentially, similarly rigorous formal analysis [11, 12, 13] of TLS, the protocol underlying the transport encryption of almost all web traffic, further helped provide strong applied security definitions for the properties underlying end-to-end encryption.

Given that ProtonMail uses PGP to provide end-to-end encryption, we consider the following security properties as being the components that achieve end-to-end encryption in the context of ProtonMail[3]. Given that this is a practical analysis, we colloquialize the understanding of the security properties provided by the cited work into the following definitions:

- **Confidentiality.** An email sent from any client to any other client can only be decrypted by the recipient and, optionally, the sender.

[3] We note that forward secrecy, post-compromise security [14] and deniability are not considered as relevant properties in this context.

- **Authenticity.** If a client receives a message that appears to be from another client, then this apparent sender must have sent the email to the recipient. Note that this definition of authenticity also encapsulates the standard definition of integrity in production end-to-end encryption systems.

2.3 Zero-Knowledge Password Proof

> *"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.* `[...]` *The Secure Remote Password protocol [15] 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."* — ProtonMail Security Features and Architecture Specification [3]

Zero-Knowledge Password Proofs (ZKPP) result from mechanisms intended to protect against dictionary attacks [16]. By employing ZKPP, a client can prove to a server that it knows the value of some password, without revealing any information about the password itself. This is different from more traditional design where the server stores a hash of the client's password. In the latter design:

- Passwords are vulnerable to brute force attacks in the event that the adversary obtains the hash.
- The server can distinguish a valid password from an invalid password entry by comparing it to the stored hash.

However, the ProtonMail Security Features and Architecture Specification, quoted above, states that no information is stored on the ProtonMail server that allows the password to be derived by *"either ProtonMail or an attacker with access to the network."*

We therefore define the ZKPP security goal thus: a server $\mathcal{P}$ should at no point in time possess any information that would allow them to confirm brute force or dictionary attack attempts for the password of a client $\mathcal{A}$ (also known as an "offline attack"). ProtonMail's own SRP implementation is illustrated in Fig. 1.

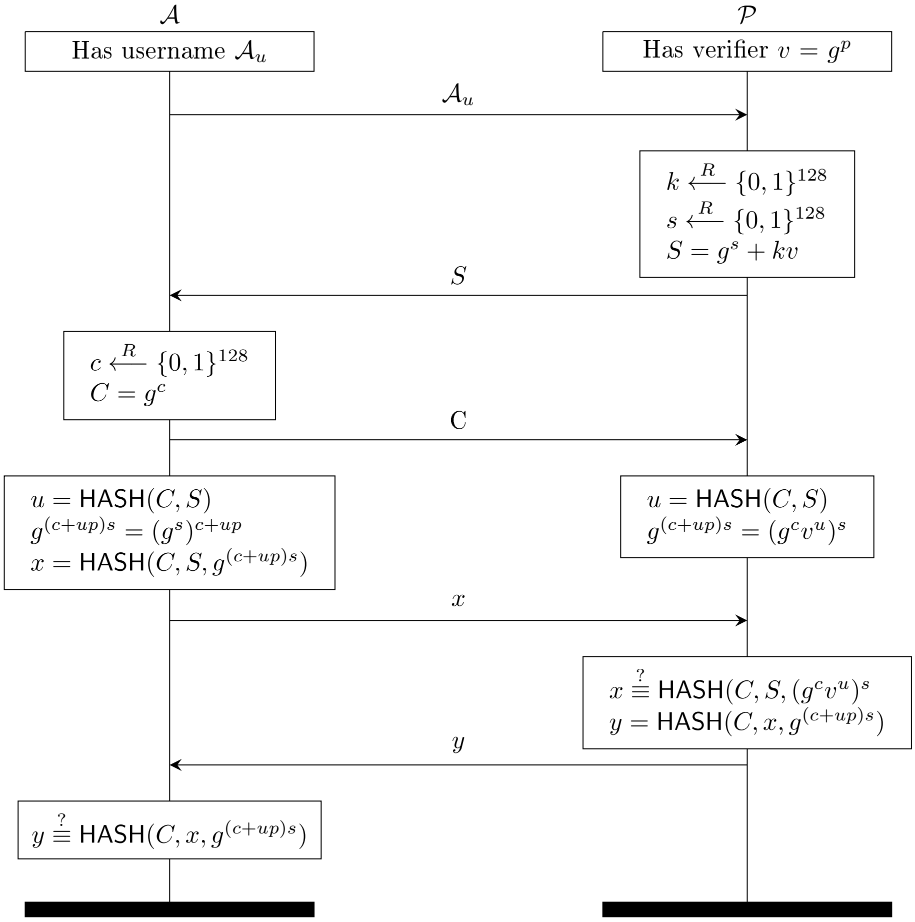

$\mathcal{A}$
$\mathcal{P}$
Has username $\mathcal{A}_u$
Has verifier $v = g^p$
$\mathcal{A}_u$
$k \xleftarrow{R} \{0,1\}^{128}$
$s \xleftarrow{R} \{0,1\}^{128}$
$S = g^s + kv$
S
$c \xleftarrow{R} \{0,1\}^{128}$
$C = g^c$
C
$u = \mathsf{HASH}(C, S)$
$g^{(c+up)s} = (g^s)^{c+up}$
$x = \mathsf{HASH}(C, S, g^{(c+up)s})$
$u = \mathsf{HASH}(C, S)$
$g^{(c+up)s} = (g^c v^u)^s$
x
$x \overset{?}{\equiv} \mathsf{HASH}(C, S, (g^c v^u)^s$
$y = \mathsf{HASH}(C, x, g^{(c+up)s})$
y
$y \overset{?}{\equiv} \mathsf{HASH}(C, x, g^{(c+up)s})$

Figure 1: Authentication via SRP, as implemented in ProtonMail [3]. All arithmetic operations are modulo some safe 2048-bit Sophie-Germain prime.

3 Cryptographic Primitives

We define the following cryptographic primitives:

- **Hashing.** $\mathsf{HASH}(x) \longrightarrow y$. A standard one-way cryptographic hash function.
- **Encryption and Decryption.**
  - **Encryption.** $\mathsf{ENC}(ek, p, a) \longrightarrow (\mathsf{E}_{ek,p,n,a,t}, n, t)$. This is an authenticated encryption with associated data (AEAD) construction where ek is an encryption key, n is a randomly generated nonce, a is associated data and t is an authentication tag.
  - **Decryption.** $\mathsf{DEC}(ek, \mathsf{E}_{ek,p,n,a,t}, n, a, t) \longrightarrow \{p, \perp\}$.

PGP Interface. We also describe the following simplified PGP API:

- **Key Generation.** $\mathsf{PGPGEN}(c) \longrightarrow (sk, pk)$.
- **Encryption.** $\mathsf{PGPSEND}(c, m, pk) \longrightarrow \mathsf{PGP}_{m,pk}$.
- **Decryption.** $\mathsf{PGPREAD}(c, \mathsf{PGP}_{m,pk}, sk) \longrightarrow \{m, \perp\}$.

In the operations described above, c indicates the application code that the client is using in order to perform PGP operations, generally assumed to contain a correct implementation of the OpenPGP protocol. The relevance of c is such that if c does not contain a *correct* implementation of OpenPGP, all output generated by PGPGEN, PGPSEND and PGPREAD could be arbitrarily affected. $\mathsf{PGP}_{m,pk}$ is a ciphertext of m encrypted towards public key pk. PGPREAD decrypts this ciphertext if the corresponding secret key sk is provided for pk. Otherwise, it returns $\perp$.

4 Protocol Flows

In this section, we provide high-level descriptions of two distinct protocol flows:

- **ProtonMail-to-ProtonMail.** $\mathcal{A}$ sends a PGP-encrypted message m to $\mathcal{B}$ through $\mathcal{P}$.
- **"Encrypt-to-Outside".** $\mathcal{A}$ sends a symmetrically encrypted email to $\mathcal{S}$ through $\mathcal{P}$ which relays the email to $\mathcal{S}$ through $\mathcal{M}$. $\mathcal{S}$ sends a PGP-encrypted reply r to $\mathcal{A}$ using the web interface J and $\mathcal{A}$'s PGP public key $\mathcal{A}_{pk}$, both provided by $\mathcal{P}$.

Both protocol flows are pre-empted by a ZKPP password mechanism using SRP, described in Fig 1.

4.1 Effects of Client Application Choice on Protocol Flows

Users have the option of accessing ProtonMail either through a webmail client or through a smartphone application.[4] How $\mathcal{A}$ accesses their ProtonMail inbox has substantial implications on protocol flows and ultimately on the security properties that their communications obtain. The reason for this has to do with the authenticity properties provided by the code delivery methods, which differ between applications.

4.1.1 ProtonMail Webmail

ProtonMail's webmail application appears to be the primary ProtonMail product and how ProtonMail is most often accessed by its users.[5] When visiting `https://protonmail.com`, $\mathcal{A}$'s web browser is served with JavaScript code representing the ProtonMail web application [17] and its underlying OpenPGP implementation, also written in JavaScript [18].

Since communication between all ProtonMail users (including $\mathcal{A}$ and $\mathcal{B}$) to $\mathcal{P}$ is assumed to be encrypted using TLS (§2.1), delivery of the ProtonMail web application is assumed to be safe against a network attacker. However, we note that a malicious $\mathcal{P}$ (also an assumption in §2.1) would be able to arbitrarily serve compromised webmail clients to $\mathcal{A}$ or any other ProtonMail user without this being detectable and that, conversely, correct delivery of webmail/OpenPGP client code is not verifiable.

Therefore, for $\mathcal{A}$, providing input to ProtonMail's webmail application provided by $\mathcal{P}$ is directly equivalent to providing input to $\mathcal{P}$ since, in effect, $\mathcal{P}$ acts as a man-in-the-middle between the user and the webmail application code. No existing mechanism, including TLS, is sufficient to offset $\mathcal{P}$'s middle-man authority in this application scenario.

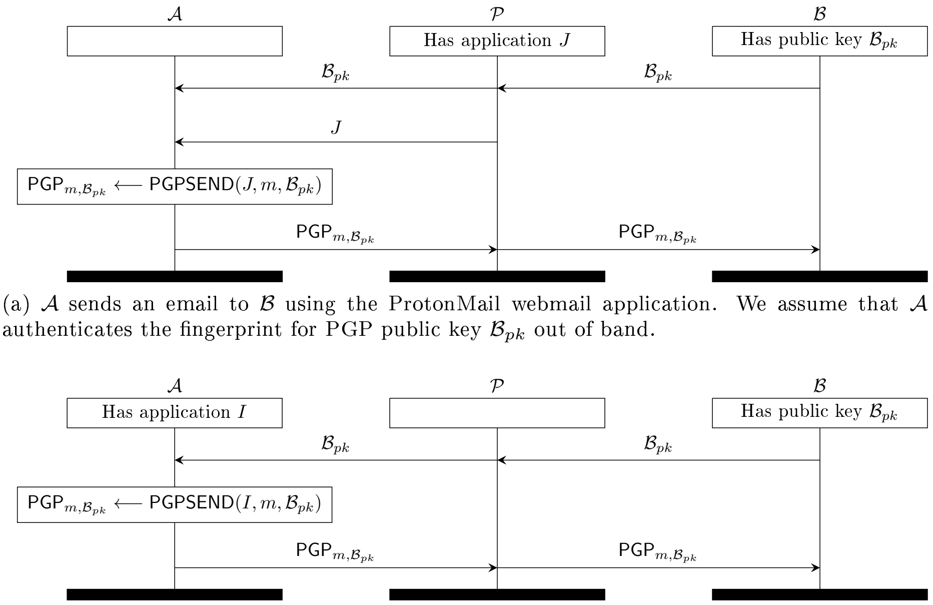
4.1.2 ProtonMail Smartphone Application

Mainstream smartphone applications on modern mobile platforms are delivered through a significantly different process to ProtonMail's webmail application. When a new ProtonMail smartphone application version is released for the Apple App Store or the Google Play Store, its author must increment a version number and release timestamp. The application binary and its manifest are both cryptographically signed with a key owned by the author and again by Apple or Google (depending on the platform) upon publication.

Note that in this application scenario, releases are cryptographically authenticated and tracked through an incremental version number, and that the delivery of client code is restricted purely to software update instances. Users are therefore able to audit whether they received the same binary for some version of the application as everyone else. Furthermore, the application distributor (Apple, Google) adds a second layer of authentication and its separate

[4] A desktop "bridge" application is also offered to paying customers. For simplicity, we group it with smartphone applications.

[5] This observation is made based on anecdotal evidence. ProtonMail does not publish platform usage statistics.


$\mathcal{A}$
$\mathcal{P}$
$\mathcal{B}$
Has application J
Has public key $\mathcal{B}_{pk}$
$\mathcal{B}_{pk}$
$\mathcal{B}_{pk}$
J
$\mathsf{PGP}_{m,\mathcal{B}_{pk}} \longleftarrow \mathsf{PGPSEND}(J, m, \mathcal{B}_{pk})$
$\mathsf{PGP}_{m,\mathcal{B}_{pk}}$
$\mathsf{PGP}_{m,\mathcal{B}_{pk}}$
$\mathcal{A}$
$\mathcal{P}$
$\mathcal{B}$
Has application I
Has public key $\mathcal{B}_{pk}$
$\mathcal{B}_{pk}$
$\mathcal{B}_{pk}$
$\mathsf{PGP}_{m,\mathcal{B}_{pk}} \longleftarrow \mathsf{PGPSEND}(I, m, \mathcal{B}_{pk})$
$\mathsf{PGP}_{m,\mathcal{B}_{pk}}$
$\mathsf{PGP}_{m,\mathcal{B}_{pk}}$

(a) $\mathcal{A}$ sends an email to $\mathcal{B}$ using the ProtonMail webmail application. We assume that $\mathcal{A}$ authenticates the fingerprint for PGP public key $\mathcal{B}_{pk}$ out of band.

(b) $\mathcal{A}$ sends an email to $\mathcal{B}$ using the ProtonMail smartphone application. We assume that $\mathcal{A}$ authenticates the fingerprint for PGP public key $\mathcal{B}_{pk}$ out of band.

Figure 2: $\mathcal{A}$ sends an email to $\mathcal{B}$ using the ProtonMail webmail application (Fig. 2a) and using the ProtonMail smartphone application (Fig. 2b).

governance from ProtonMail renders targeted delivery of malicious code even more difficult for a malicious $\mathcal{P}$.

4.2 ProtonMail-to-ProtonMail

In Fig. 2, we see a ProtonMail user $\mathcal{A}$ sending a PGP-encrypted email to another ProtonMail user $\mathcal{B}$. Notice that there is a significant difference in the protocol flow depending on whether Alice uses the webmail application (Fig. 2a) or the smartphone application (Fig. 2b). In the former case, $\mathcal{A}$ is forced to obtain an unauthenticated copy of J from $\mathcal{P}$ every time before sending an encrypted email to $\mathcal{B}$. In the latter case, $\mathcal{A}$ can simply rely on her local authenticated binary residing in her smartphone.

4.3 "Encrypt-to-Outside"

In Fig. 3, we see a ProtonMail user $\mathcal{A}$ sending a symmetrically encrypted email to a Microsoft Outlook user $\mathcal{S}$ using ProtonMail's *"Encrypt-to-Outside"* (ETO) feature. Both the sender and the recipient are expected to have advanced knowledge of some symmetric encryption key psk. $\mathcal{A}$ sends her symmetrically encrypted message m through $\mathcal{P}$ mail servers, which add a URI to J and a copy of Alice's PGP public key $\mathcal{A}_{pk}$ into the payload and in turn relay it to the Microsoft Outlook mail servers at $\mathcal{M}$ which then relay it to $\mathcal{S}$. $\mathcal{S}$ follows the URI

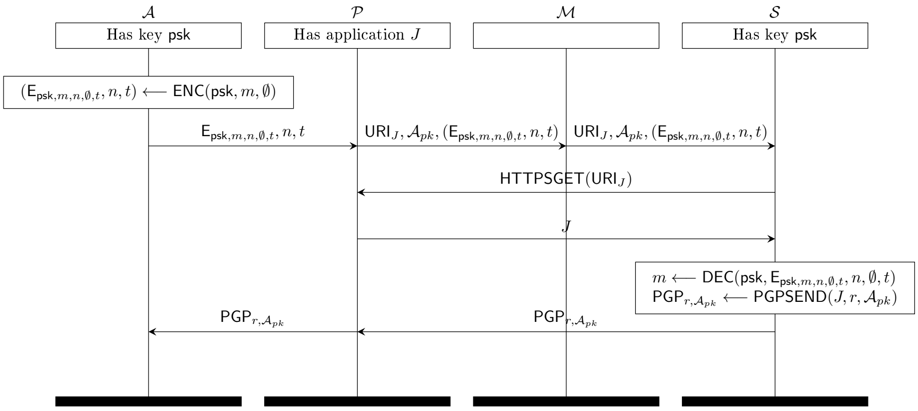

$\mathcal{A}$
$\mathcal{P}$
$\mathcal{M}$
$\mathcal{S}$
Has key psk
Has application J
Has key psk
$(\mathsf{E}_{\mathsf{psk},m,n,\emptyset,t}, n, t) \longleftarrow \mathsf{ENC}(\mathsf{psk}, m, \emptyset)$
$\mathsf{E}_{\mathsf{psk},m,n,\emptyset,t}, n, t$
$\mathsf{URI}_J, \mathcal{A}_{pk}, (\mathsf{E}_{\mathsf{psk},m,n,\emptyset,t}, n, t)$
$\mathsf{URI}_J, \mathcal{A}_{pk}, (\mathsf{E}_{\mathsf{psk},m,n,\emptyset,t}, n, t)$
$\mathsf{HTTPSGET}(\mathsf{URI}_J)$
J
$m \longleftarrow \mathsf{DEC}(\mathsf{psk}, \mathsf{E}_{\mathsf{psk},m,n,\emptyset,t}, n, \emptyset, t)$
$\mathsf{PGP}_{r,\mathcal{A}_{pk}} \longleftarrow \mathsf{PGPSEND}(J, r, \mathcal{A}_{pk})$
$\mathsf{PGP}_{r,\mathcal{A}_{pk}}$
$\mathsf{PGP}_{r,\mathcal{A}_{pk}}$

Figure 3: $\mathcal{A}$ sends an email containing message m to Microsoft Outlook user $\mathcal{S}$ symmetrically encrypted using a pre-shared key psk. $\mathcal{S}$ responds through the webmail interface provided by $\mathcal{P}$, encrypting his reply r using PGP to $\mathcal{A}_{pk}$.

to J using HTTPS, whereupon he enters psk in order to obtain m. Using J, $\mathcal{S}$ may also send a reply r which is PGP-encrypted to public key $\mathcal{A}_{pk}$.

5 Security Analysis

In this section, we provide the results of our examination as to whether the protocol flows described in §4 achieve the end-to-end encryption security goals as defined in §2.2 and the ZKPP security goals as defined in §2.3 while operating under the assumptions defined in §2.1.

5.1 On End-to-End Encryption

We present three findings showing that ProtonMail has, since its inception, never achieved end-to-end encryption security guarantees for the majority of its users.

5.1.1 ProtonMail Webmail Does Not Provide End-to-End Encryption

A crucial security assumption, based on ProtonMail's self-professed security goals in its specification documents (§2.1), is that the ProtonMail server $\mathcal{P}$ is untrusted. In Fig. 2a, we see that this untrusted server $\mathcal{P}$ *must* serve an authentic OpenPGP implementation J every time $\mathcal{A}$ logs into ProtonMail or, in some cases, multiple times in between $\mathcal{A}$ same single ProtonMail session. Since $\mathcal{P}$ is untrusted and since no authentication mechanism is implemented to check for the correctness of J, $\mathcal{P}$ can arbitrarily and untraceably compromise any information that $\mathcal{A}$ sends as part of her ProtonMail session. This includes $\mathcal{A}$'s PGP secret key and any emails she has sent and received.

The ProtonMail smartphone applications are unaffected by this issue. However, even if $\mathcal{A}$ has used the ProtonMail smartphone applications for the entire lifetime of her ProtonMail account thus far and then logs into ProtonMail via the web application *just once*, $\mathcal{P}$ still obtains $\mathcal{A}$'s PGP secret key and is therefore able to not only impersonate $\mathcal{A}$ going forward but also to retroactively decrypt all of $\mathcal{A}$'s previous communications.

In effect, this means that any ProtonMail webmail user has never obtained end-to-end encryption guarantees under ProtonMail's own security model and security goal definitions.

5.1.2 "Encrypt-to-Outside" Allows Mail Servers to Recover Pre-Shared Key and Reply Plaintext

In Fig. 3, we see that $\mathcal{P}$ relays the URI for ProtonMail web application code J to $\mathcal{S}$ through third-party mail server $\mathcal{M}$. This provides both $\mathcal{P}$ and $\mathcal{M}$ with the ability to stage significant Man-in-the-Middle attacks:

- $\mathcal{P}$ is free to arbitrarily replace J with an incorrect OpenPGP implementation or to replace $\mathcal{A}_{pk}$ with a PGP public key corresponding to a secret key that $\mathcal{P}$ itself controls. The former would allow $\mathcal{P}$ to recover psk and m, while the later would allow $\mathcal{P}$ to recover r.
- $\mathcal{M}$ is free to arbitrarily replace URI_J with any other arbitary URI to a web application that it controls, which could in effect pretend to be ProtonMail. This would allow $\mathcal{M}$ to harvest psk as well as r. $\mathcal{M}$ can then obtain $\mathcal{A}$'s legitimate ciphertext and decrypt it using J and psk, thereby obtaining m, and also encrypt r to $\mathcal{A}$ using $\mathcal{A}_{pk}$ thereby performing an undetected man-in-the-middle attack.

5.1.3 Mailbox Keys, PGP Secret Keys and "Encrypt-to-Outside" Pre-Shared Keys Vulnerable to Dictionary Attacks

In our testing[6] of the ProtonMail applications, we were able to set both user mailbox passwords and "Encrypt-to-Outside" pre-shared key passwords that were exceptionally weak and vulnerable to simple guessing attacks. These passwords included "`1`", "`iloveyou`" and "`password`" and were used to derive encryption keys for PGP secret keys that were later stored on ProtonMail servers as well as for "Encrypt-to-Outside" symmetric encryption.

As we will see in §5.2, the ProtonMail servers do indeed possess a password oracle that renders dictionary and brute force attacks possible. Allowing extremely weak passwords only further exacerbates the issue and could potentially allow for the easy obtention of a user's PGP secret key. Furthermore, an attacker that obtains access to ProtonMail's database of millions of encrypted user PGP secret keys will likely find that a significant portion of them can be brute-forced using dictionary attacks that simply run through the top 100,000 most common passwords, a relatively small amount of guesses.

[6] Our testing occurred on November 17, 2018.

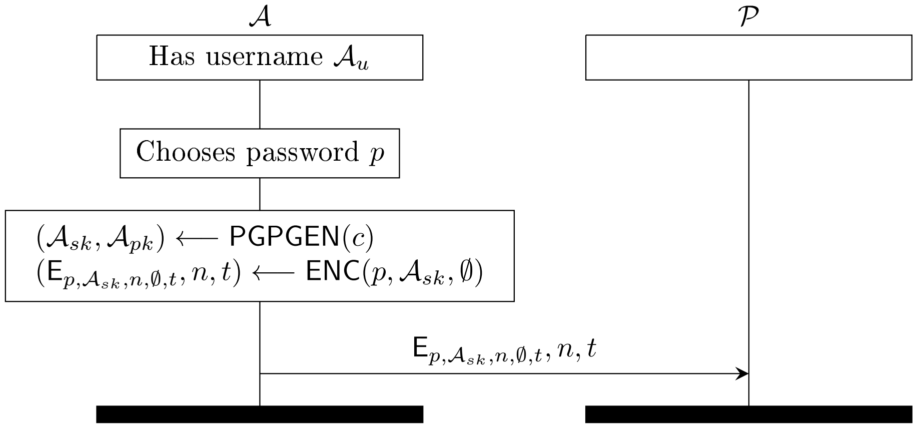

$\mathcal{A}$
$\mathcal{P}$
Has username $\mathcal{A}_u$
Chooses password p
$(\mathcal{A}_{sk}, \mathcal{A}_{pk}) \longleftarrow \mathsf{PGPGEN}(c)$
$(\mathsf{E}_{p,\mathcal{A}_{sk},n,\emptyset,t}, n, t) \longleftarrow \mathsf{ENC}(p, \mathcal{A}_{sk}, \emptyset)$
$\mathsf{E}_{p,\mathcal{A}_{sk},n,\emptyset,t}, n, t$

Figure 4: $\mathcal{A}$ sends an encrypted copy of her PGP secret key to $\mathcal{P}$ during account creation. In this flow, the application code c provided to PGP operations is irrelevant.

5.2 On User Authentication

While SRP authentication, as described in Fig. 1, does indeed provide ZKPP-based authentication, the ProtonMail Security Features and Architecture specification also states the following:

> *"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 retreived whenever a user logs in succesfully. 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."* — ProtonMail Security Features and Architecture Specification [3]

Indeed, we document this functionality in Fig. 4. During account creation, $\mathcal{A}$ will send $\mathcal{P}$ a copy of her PGP secret key $\mathcal{A}_{sk}$ encrypted under her mailbox password p. This encrypted secret key acts as an oracle for p. Therefore, $\mathcal{P}$ posesses an oracle for $\mathcal{A}$'s mailbox password. This oracle can be used for brute force or dictionary "offline" attack attempts, thereby invalidating ProtonMail's ZKPP claims [3] and breaking the Zero-Knowledge Password Proof security goal as defined in §2.3.[7]

[7] ProtonMail uses the `bcrypt` [19] password hash which slows down dictionary attacks. However, ProtonMail restricts the number of `bcrypt` rounds to a relatively small number of 2^{10} [17] which, especially when coupled with recent advances in `bcrypt` computation [20], renders dictionary attacks feasible once more.

6 Recommendations and Conclusion

Our findings, presented in §5, constitute serious shortcomings in ProtonMail's cryptographic architecture that we believe should be urgently remedied. As it stands, ProtonMail does not meet its self-professed security goals when these are subjected to analysis.

While this paper presents the first formal argument justifying these findings, some of them have already been documented: the results discussed in §5.1.1 seem to have been known informally since at least 2015 [21, 22].

With regards to the ProtonMail web application, features such as Subresource Integrity (SRI) [23] could arguably provide some increased authenticity to the code delivery mechanism. However, these features are deemed insufficient for ProtonMail to meet its security goals, and it is our conclusion that no webmail-style application could. Other messaging services, such as WhatsApp and Signal, provide downloadable web applications which run locally on the user's device and therefore accomplish the same code integrity as ProtonMail's smartphone applications (as described in Fig. 2b).

Failing the removal of ProtonMail's web application, ProtonMail simply should not claim end-to-end encryption except for use cases where both senders and recipients restrict themselves to ProtonMail's mobile applications.

We also recommend that ProtonMail never store user PGP secret keys on its servers. As shown in §5 and in contradiction with ProtonMail's claims, these keys are indeed vulnerable to "offline" dictionary attacks and moreso with the allowance of exceptionally weak passwords. Instead, we recommend a system in which the device from which the user is signing up to ProtonMail from (iOS app, Android app, desktop app) locally derive a PGP identity and that this identity be communicated from device to device using QR codes. This method is already implemented by other popular secure messaging applications with success.

With regards to user passwords, we recommend that ProtonMail at the very least impose a minimum character requirement and to use a password strength measurement library such as zxcvbn [24]. A stronger way to address the issue, however, could be for ProtonMail to encourage the use of passphrases instead of passwords, which seems warranted given that PGP does not provide forward secrecy and that the atomic compromise of a PGP secret key has indefinite consequences for the lifetime of that PGP key pair.

Finally, we would like to pre-empt ProtonMail's potential response, which could indicate that the findings in this analysis were already known to ProtonMail. If this is the case, then ProtonMail must radically overhaul its existing specifications, documentation and product presentation materials ([3, 4, 17]) to remove all mentions of end-to-end encryption and Zero-Knowledge Password Proof authentication, as any continued claim of achieving these properties would be misleading for most users and therefore indefensible.

Acknowledgements

This paper is dedicated to music composer Toby Fox.

References

[1] Proton Technologies A.G. Fighting Censorship with ProtonMail Encrypted Email over Tor, Jan 2017. https://protonmail.com/blog/tor-encrypted-email/. 1

[2] Nick Lucchesi. ProtonMail Hits 5 Million Accounts and Wants Users to Ditch Google by 2021, Sep 2018. https://www.inverse.com/article/49041-protonmail-ceo-andy-yen-interview. 1

[3] Proton Technologies A.G. ProtonMail Security Features and Infrastructure, Jul 2016. https://protonmail.com/docs/business-whitepaper.pdf. 1, 2, 4, 5, 11, 12

[4] Proton Technologies A.G. ProtonMail Security Details Page, Nov 2018. https://protonmail.com/security-details. 3, 12

[5] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84, 2004. 3

[6] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017. 3

[7] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. 2018. 3

[8] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated Verification for Secure Messaging Protocols and their Implementations: A Symbolic and Computational Approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017. 3

[9] Trevor Perrin and Moxie Marlinspike. The X3DH Key Agreement Protocol, 2016. https://signal.org/docs/specifications/x3dh/. 3

[10] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm, 2016. https://signal.org/docs/specifications/doubleratchet/. 3

[11] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A Comprehensive Symbolic Analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1773–1788, New York, NY, USA, 2017. ACM. 3

[12] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1197–1210, 2015. 3

[13] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 483–502. IEEE, 2017. 3

[14] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On Post-Compromise Security. In *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*, pages 164–178. IEEE, 2016. 3

[15] Thomas Wu. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol, Oct 2002. 4

[16] Steven M Bellovin and Michael Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 72–84. IEEE, 1992. 4

[17] Proton Technologes A.G. Official AngularJS web client for the ProtonMail secure email service. `https://github.com/ProtonMail/WebClient`. 7, 11, 12

[18] Proton Technologes A.G. OpenPGP.js: OpenPGP Implementation for JavaScript. `https://github.com/openpgpjs/openpgpjs`. 7

[19] Niels Provos and David Mazieres. Bcrypt algorithm. USENIX, 1999. 11

[20] Katja Malvoni, Designer Solar, and Josip Knezović. Are your passwords safe: Energy-efficient bcrypt cracking with low-cost parallel hardware. In *WOOT'14 8th Usenix Workshop on Offensive Technologies Proceedings 23rd USENIX Security Symposium*, 2014. 11

[21] 12

[22] 12

[23] Devdatta Akhawe, Francois Marier, Frederik Braun, and Joel Weinberger. Subresource Integrity. *W3C working draft, W3C, July*, 2015. 12

[24] Daniel Lowe Wheeler. zxcvbn: Low-Budget Password Strength Estimation. In *USENIX Security Symposium*, pages 157–173, 2016. 12