



# A Protocol for Solving Certificate Poisoning for the OpenPGP Keyserver Network

Gunnar Wolf   [ Posgrado en Ciencia e Ingeniería de la Computación, UNAM | [gwolf@gwolf.org](mailto:gwolf@gwolf.org) ]

Jorge Luis Ortega-Arjona  [ Facultad de Ciencias, UNAM | [jloa@ciencias.unam.mx](mailto:jloa@ciencias.unam.mx) ]

 Circuito de Posgrados. Unidad de Posgrado, Edificio “C” 1er nivel, Ciudad Universitaria, Coyoacán 04510, Ciudad de México, México

Received: 31 October 2023 • Accepted: 16 April 2024 • Published: 23 May 2024

**Abstract** The OpenPGP encryption standard builds on a transitive trust distribution model for identity assertion, using a non-authenticated, distributed keyserver network for key distribution and discovery. An attack termed “certificate poisoning”, surfaced in 2019 and consisting in adding excessive trust signatures from inexistent actors to the victim key so that it is no longer usable, has endangered the continued operation of said keyserver network. In this article, we explore a protocol modification in the key acceptance and synchronization protocol termed *First-party attested third-party certification* that, without requiring the redeployment of updated client software, prevents the ill effects of certificate poisoning without breaking compatibility with the OpenPGP installed base. We also discuss some potential challenges and limitations of this approach, providing recommendations for its adoption.

**Keywords:** OpenPGP, Web of Trust, Transitive trust models, Distributed trust, Certificate poisoning

## 1 Introduction

E-mail encryption is introduced in 1991, with the PGP program [Zimmermann, 1999]. Since then, and although there are other solutions, the OpenPGP standard that derived from it [Callas *et al.*, 2007] remains one of the leading e-mail encryption technologies.

For an encryption mechanism to be useful to more than the handful of users that can personally know each other and exchange respective public keys via a trusted channel (i.e. by meeting in person), a transitive trust model is needed. This is, a network of public keys is created so that, in order for user *A* to securely communicate with user *B*, *A* can find a *trust path* between trusted third parties. Two transitive trust models can be found: the *centralized model* (followed by TLS, found in most browser-based communications and server-to-server protocols) and the *distributed model* (aimed at client-to-client communications, among which OpenPGP’s *Web of Trust* is one of the clearest examples). Both transitive trust models are further explained in Subsection 2.1.

An attack on the OpenPGP ecosystem, termed *certificate poisoning*, surfaced in 2019; this attack (thoroughly explained in Subsection 2.3) causes the boundless growth of a victim’s public key certificate, rendering it useless. This attack becomes impossible to revert due to the nature of the keyserver’s synchronization protocol, and has put the keyserver network for OpenPGP’s *Web of Trust* in existential danger.

This article proposes a protocol for carrying out and distributing key certifications that solves the certificate poisoning attack without giving up, as other strategies presented in Section 3 do, on keeping the whole system free of centralization, with minimal modifications to the keyserver network software, and using already available and deployed facilities in client-side OpenPGP programs.

An experimental verification process is presented in Section

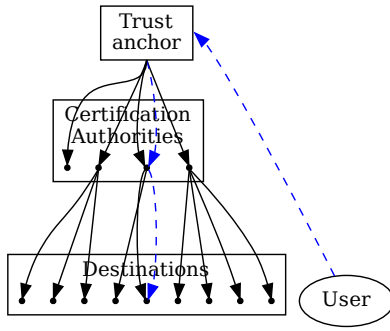
5, with which we show how the proposed protocol effectively stops certificate poisoning attacks. The experiment was carried out using unmodified client software, and with very minor modifications to the server-side software.

## 2 Background

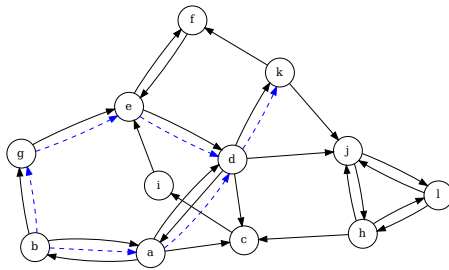
### 2.1 Transitive trust models

While the main goal of encrypted communications mechanism is to protect a given channel or document from eavesdropping or modification by unauthorized parties, this can only be achieved if such mechanism also provides *authentication*: in a public key-based cryptosystem, when the mythical *Alice* wants to communicate with *Bob*, she has to first get hold of his public key,  $k_B$ . But given any party (including evil *Mallory*) can forge certificates in *Bob*’s name, and assuming Internet-scale communications, where it is not feasible for *Alice* to walk to *Bob*’s office to get his key, she has to find a way to locate  $k_B$  and authenticate it. Given *Alice* cannot trust *Bob*’s key is truly  $k_B$ , she trusts a given third party’s declaration stating that *Bob* effectively is in possession of  $k_B$ . This is termed a *Transitive Trust Model* (TTM). The TLS (Transport Layer Security) protocol, which encrypts most communications over the Internet, solves this issue by implementing a *centralized* TTM, illustrated in Figure 1. For this model, users trust a centrally provided set of *Trust Anchors* (TAs) [Rescorla, 2018, p. 45]. This set is usually and controlled by the operating system or browser vendor. Each TA delegates their certification ability to *Certification Authorities* (CAs).

Conveying the server’s public key  $k_S$  to the user interested in establishing a TLS connection is solved by the protocol itself, as being TLS a connection-oriented protocol the server’s



**Figure 1.** Centralized model. User verifies a valid path from a centrally defined Trust Anchor, through a Certification Authority, to the destination they want to reach.



**Figure 2.** Distributed model. User  $b$  builds trust paths towards target  $k$  following the existing edges of a mesh-like graph.

certificate chain  $C_S$  is relayed as part of the session establishment. This certificate chain includes the server’s public key, and includes the *chain of signatures* where trust in their identity flows down from the TA, via the CA, to the particular server.

There is a different TTM, however, based upon a distributed model, named the *Web of Trust* (WoT), illustrated in Figure 2. The set of user-to-user node paths is represented as a mesh, and trust is rooted in the individual actor (node) that wants to trace a trust path to a different one.

In the example depicted, if user *Bob* ( $b$ ) wants to communicate with *Karen* ( $k$ ), he has to build a trust path so that  $b \rightarrow k$  is possible. *Bob* can find two possible paths by acyclically traversing the graph:

$$b \rightarrow a \rightarrow d \rightarrow k$$

$$b \rightarrow g \rightarrow e \rightarrow d \rightarrow k$$

The first path is one *hop* shorter, although given WoT certifications can carry data beyond the mere identity certification (such as trust expiration dates and weights), different models are proposed that reason about *trust levels* provided by different paths could yield different metrics for either of them [Jøsang, 1999].

## 2.2 The keyserver network

In order to search a trust path from  $b$  to  $k$  under a global-scale network, however, there is still one important element missing: a given user does not have the full list of keys that

comprise the WoT. This is where a *key discovery mechanism*, the *keyserver network* comes into play.

Contrary to TLS, OpenPGP is not a protocol where session initiation happens over a live connection. RFC 4880 defines the structure of the *OpenPGP message* [Callas et al., 2007]; the protocol requires the initiator to know the cryptographic identity of the recipient *before transmitting the message*.

Individual users can choose how to publish their keys, so that third parties that want to communicate with them securely can do so. A public, Web-queriable keyserver has been deployed since 1997 [Yamane et al., 2003], with its API being standardized as the OpenPGP HTTP Keyserver Protocol (HKP) [Shaw, 2003]. Independent keyservers complying to HKP are offered by different operators, initially leading to fragmentation and confusion, and highlighting the need for a protocol to synchronize between them.

Synchronization between keyservers is achieved via a set-synchronization protocol. This synchronization is nearly optimal in terms of communication complexity, and tractable in terms of computational complexity, based on a *gossip* or *epidemic* model [Minsky, 2002], which allows for the different keyservers to form a network. This network allows users to upload public keys, as well as their certifications, to any of the participating servers, trusting they are soon synchronized over all of them.

As the OpenPGP ecosystem follows a culture with strong interests for decentralization and censorship resistance, answering to a threat model including national governments’ interference, the *gossip*-based protocol allows for information to spread reliably, but *does not provide any way* for information to be removed; once key material has entered the keyserver network, and given that there are enough nodes synchronizing any missing material from each other, it is unfeasible to remove information from it.

Given that (a) data submission to keyservers is unauthenticated (any user can create a set of OpenPGP keys and upload them to the network), and that (b) once information has been relayed across several nodes it cannot be realistically removed, and that (c) the OpenPGP key format allows for arbitrary data to be input as part of a key, the keyserver network can be seen as an *append-only, public-access, distributed database*. This conflicts with privacy laws such as the European Union’s General Data Protection Regulation (GDPR), and it is the cause for several keyserver operators to stop offering their servers [Pramberger, 2010]. Several attacks are documented as means to force operators into switching to different practices, where control over the distributed material can be effectively exercised [Yakamo, 2018].

## 2.3 OpenPGP certificate poisoning

Without downplaying the factors highlighted by Yakamo [2018], the authors stand by the importance of a fully decentralized network. This work is centered on preventing the ill effects of a specific attack surfaced shortly afterwards, termed *certificate poisoning*, and addressed as *CVE-2019-13050* in the Common Vulnerability Exposure (CVE) database.

The OpenPGP key format specifies data can be appended to an existing key as extra *packets* [Callas et al., 2007]. Let

us consider that user *Alice* controls the public key  $k_A$ , and wants to start encrypted communication with previously unknown (so not yet validated by her) user *Bob*, who controls the public key  $k_B$ . *Alice* requests a keyserver corresponding to `bob@example.org`, obtaining the certificate chain  $C_{k_B}$ . OpenPGP keys are specified to be *self-certified*, indicating information such as their validity periods. Thus, the minimal  $C_{k_B}$  contains  $k_B, c_{k_B \rightarrow k_B}$ . To ensure the key she received truly belongs to *Bob* and is not an attempt by evil *Mallory* to perform a *Man-in-the-Middle attack* (MitM) [Conti *et al.*, 2016], *Alice* learns that their mutual friends *Charly* and *Diana* certify *Bob*'s key. This is, *Alice* received from the keyservers a certificate chain that includes:

$$C_{k_B} = k_B, c_{k_B \rightarrow k_B}, c_{k_C \rightarrow k_B}, c_{k_D \rightarrow k_B}$$

However, in her attempts to disrupt communication, *Mallory* creates thousands of throwaway keys. Those keys might not even have a valid identity string, and these are in no way traceable back to *Mallory*. It must be noted that this can be achieved in few minutes using modern-day hardware. She now controls:

$$k_{M_1}, k_{M_2}, k_{M_3}, \dots, k_{M_{9999}}, k_{M_{10000}}$$

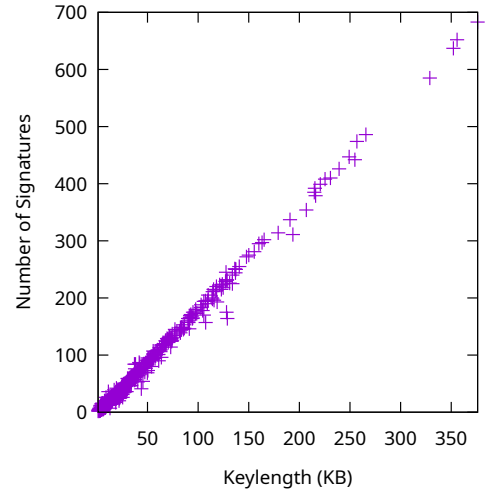
*Mallory* proceeds to certify  $k_B$  with all of her generated keys and uploads the result to a keyserver, so that a request for  $k_B$  now yields a much larger result:

$$C_{k_B} = \begin{cases} k_B, c_{k_B \rightarrow k_B}, c_{k_C \rightarrow k_B}, c_{k_D \rightarrow k_B}, \\ c_{k_{M_1}} \rightarrow k_B, c_{k_{M_2}} \rightarrow k_B, c_{k_{M_3}} \rightarrow k_B, \\ \dots \\ c_{k_{M_{9999}}} \rightarrow k_B, c_{k_{M_{10000}}} \rightarrow k_B \end{cases}$$

In normal use, OpenPGP public keys are usually a few kilobytes long. Most frequent users connect to a couple dozen other users [Wolf and Quiroga, 2018], increasing their size roughly by 500 bytes per signature (depending on the cryptographic algorithm each of the keys uses); Figure 3 shows the correlation between the number of signatures for each of the 909 keys in the Debian project's curated keyring [Wolf and Gallegos, 2017] to its length in kilobytes. This figure shows not only the clear linear correlation between the number of signatures in a key and its length; keys range between 1 194 bytes (2 signatures) and 376 041 bytes (683 signatures), but the regular combined size of the key and certificates of an important subset of the global keyring.

After the attack, *Bob*'s key would be bloated by *Mallory*'s attack by over 5 MB; were she to devote more resources to it, the resulting key would grow linearly. *Bob*'s key is, thus, said to be *poisoned* by *Mallory*. When *Alice* fetches *Bob*'s key from the keyservers, it becomes unusable. *Alice*'s OpenPGP client faces orders of magnitude more information than it's supposed to handle; observed failures from this attack range from program freezes to corruption of *Alice*'s local keystore.

*Alice* is thus thwarted from securely communicating with *Bob*. However, the real attack is on *Bob*: His identity, that



**Figure 3.** Relation between the number of signatures and the size of certificate chains in the Debian project's curated keyring. Data from <https://salsa.debian.org/debian-keyring/keyring/-/tree/master/debian-keyring-gpg> at commit 2b34757d

should convey the trust of his peers, is no longer usable; *Bob* has to migrate to a new identity,  $k'_B$ . But this new  $k'_B$  is not linked to the WoT, as it does not have any certifications; *Alice* has no reason to trust  $k'_B$  over any other key claiming to belong to *Bob*, and they cannot engage in secure communications. Furthermore, even if *Bob* meets again with *Charly* and *Diana* and builds a certificate chain again, *Mallory* can trivially repeat her attack.

While few certificate poisoning attacks are documented on keys actually used in production [Kahn Gillmor, 2019b], the threat on the OpenPGP community is inherent to the keyserver network model.

### 3 Related work

Naturally, since certificate poisoning attacks are public (and particularly since the discussion following [Kahn Gillmor, 2019a]), other researchers and practitioners have put efforts in countering and mitigating the described attack, as well as other weaknesses in the keyserver network.

A comparison between the strategies mentioned in this section can be found succinctly in Table 1; this information is summarized in Subsection 3.5.

#### 3.1 DANE and WKD

As explained in Subsection 2.2, the keyserver network serves two fundamental roles: *key distribution* (finding the public keys necessary for communication) and *WoT distribution* (relaying said key's certificate chains and allowing users to traverse the resulting WoT).

DNS-Based Authentication of Named Entities (DANE) [Wouters, 2016] and the Web Key Directory (WKD) [Koch, 2021] are very different protocols, based respectively on DNS and HTTP, that transfer part of the trust traditionally linked to the WoT with domain control: If *Alice*'s mail address is `alice@example.org` and she wants her key to be made widely available, she can publish it as a RRtype record of the `example.org` DNS zone (for DANE) or as a specially named

**Table 1.** Comparison between other implementations that counter the poisoned certificates attack.

	Domain adm. independent	OpenPGP user tools compatible	WoT support	Key distribution	UID address verification	Decentralized operation
WKD and DANE	○	●	●	●	◐	○
TOFU: Autocrypt	●	●	◐	◐	○	●
Verif: PGP Glob.Dir.	●	●	●	●	●	○
Verif: Mailvelope	●	●	○	●	●	○
Verif: Hagrid	●	●	●	●	●	○
OTR (various IM)	◐	○	○	◐	●	○
Current (SKS)	●	●	●	●	○	●
1PA3PC	●	●	●	●	◐	●

Web page under the `https://openpgpkey.example.org/` domain (for WKD).

In both schemes, *Alice* remains in control of the contents of the resulting certificate chain, and is expected to select and publish what often amounts to a small subset of the signatures on her key; DANE Subsection 2.1.2 explicitly mentions that “the user can also elect to add a few third-party certifications, which they believe would be helpful for validation in the traditional ‘web of trust’”, due to the size of the resulting RRtype record, and gives several examples on *reducing the transferable public key size*.

These schemes advocate for key distribution models other than general keyservers, and require control over specific content held at given servers. Although several organizations identified as strong users of the OpenPGP keyservers, such as free software development projects, are known to support both DANE and WKD for their domains, it remains fundamentally impossible for users of arbitrary e-mail services to use these schemes for their personal addresses (as *gmail.com*, *hotmail.com*, or hundreds of other massive e-mail providers don’t offer DANE or WKD facilities).

The aforementioned protocols also are heavily centralized: it is trivial for the owner or operator of a domain to control which entries are hosted in their systems, so it is very easy to centrally prevent a key from being distributed at all. Thus, while a key owner does regain control over the precise contents of the version of their key presented to other users if switching from the keyserver network to a DANE or WKD, the decentralization offered by keyservers is lost. Also, with a vast majority of e-mail users using large-scale infrastructure providers (often not amenable to both of these protocols), the ability to distribute their keys is clearly hampered.

### 3.2 Trust On First Use

The keyserver distribution model is not the only way to convey keys and to express trust relationships. A different model, achievable using the same client tools as OpenPGP’s WoT, is TOFU (short for *Trust On First Use*).

TOFU questions the WoT’s validity, as it recognizes as “generally unreasonable to set someone whom you have never met as a trusted introducer” [Walfield and Koch, 2016]. The authors reason that the WoT’s validity cannot be meaningfully extended beyond two hops (that is, accepting as authentic the identity of *friends of friends*).

The TOFU security module, also known in literature as Weak Authentication (WA) or Leap of Faith (LoF), is a security

model where a user, upon a first connection to a yet unauthenticated endpoint, instead of finding a trusted third party to confirm the new peer’s identity, proceeds to *trust the presented identity to be valid* the first time it is encountered, and locally stores it. For any future interactions, though, the communication channel is authenticated with the preexisting, stored identity [Pham and Aura, 2011]. The model is, from the onset, known to be weaker than both the centralized and distributed TTMs, but it is widely used due to it not requiring any additional infrastructure (registration effort, cost and scalability). Quoting from Pham and Aura [2011]:

The security of LoF relies on an important assumption that the attacker is unlikely to be present during the first communication (...) In the classical computer security model, LoF is unquestionably insecure. In practice, it has been applied in several context. The most prominent one is SSH. (...) Also, when a user downloads and installs a web browser or an operating system, a list of root certificates (...) is configured on that user’s machine. Most users do not bother to verify offline the correctness of these certificates, and thus the LoF mechanism is applied.

Even given all its limitations, TOFU adoption has grown over the years. SSH, mentioned in the above quote, is a very specific case where a systems administrator generates keys in several systems and trusts them because their ability to immediately verify the functionality. Bluetooth establishes cryptographic secure channels between discrete devices with no proper MitM protection because it is a distance-limited protocol, and usually runs in devices with interfaces too limited to perform deeper checking. The multimedia-oriented Session Initiation Protocol (SIP) needs to blend in with preexisting communications networks where any authentication other than TOFU would be extremely impractical, as well as disrupting to the expected communication experience [Arkko and Nikander, 2004]. However, since the mid 2010s, TOFU has also grown into a very important niche, becoming the dominant trust distribution model: instant messaging applications, such as *WhatsApp*, *Telegram* or *Signal*, provide an opportunistic end-to-end secure messaging mode based on TOFU [Herzberg and Leibowitz, 2016].

Instant messaging is a very particular case: Given its wide adoption, with implementations reaching over a billion users, and the usability hurdles of encryption [Whitten and Tygar, 1999] and the low awareness of its importance [Renaud *et al.*,

2014], notifications regarding mismatching keys have to be kept as non-intrusive as possible, but provide a means for out-of-band verification by tech savvy users [Johansen *et al.*, 2017, 23–48]. TOFU has proven to be a good compromise, even though the overall identity assurance of the system is lower than with the other mentioned trust distribution models.

The Autocrypt project, pursuing furthering usability in e-mail encryption in order for more users to be able to adopt it, while building over PGP (which uses the WoT), acknowledges the advantages of TOFU and provides a hybrid trust mechanism. Quoting Krekel *et al.* [2018],

That’s why we trust on first use and distribute public keys in the email header. It is hidden, but decentralized, and leaves the users in control of their keys, without them necessarily knowing it. And if they want to do an out-of-band verification with their associates, there will always be user-friendly options, e.g. with a QR code comparison.

Technically, Autocrypt is not much more than a set of some reasonable configuration decisions. But together, the decisions made by Autocrypt can streamline the complex PGP system to be usable for encrypted communication between everyone. What encrypted communication needs is simple, measured steps of improvement. That’s the only way to bring people together while maintaining the original intent of the architecture.

Thus, TOFU provides a convenient scheme for key distribution, a topic that is presented next, and for minimal involvement cases are as a simple trust conveying mechanism, although for stronger security guarantees it still must be combined with a proper trust distribution model. And this is where a comparison with the solution this work proposes can be found: while TOFU is a solution to the certificate poisoning problem, it is not deemed to cover the needs for many of the use cases covered by WoT users.

### 3.3 Verifying keyservers

Several other high profile weaknesses and vulnerabilities are known since before the appearance of certificate poisoning attacks presented in Subsection 2.3 that prompted this work. While certificate poisoning is seen as many, as Subsection 3.4 shows, as *the nail in the coffin* for the decentralized *gossip*-based synchronized keyserver network, said prior attacks clearly accelerate the certainty within the OpenPGP community that the keyserver network based on *gossip* synchronization needs a strong overhaul.

The idea behind *verifying keyservers* is to add requisites *before the acceptance of* given key data into the keyring set. The work presented here presents a *halfway verifying* keyserver: New keys can be added with no requisites whatsoever, but signature packet additions are verified. Our modified protocol continues to run, however, over a *gossip* network.

Verifying keyservers are not a novel idea. Already by 2004, the PGP Coproration releases the PGP Global Directory [PGP Global Directory, 2004], a proprietary product now

owned by Symantec. It verifies the uploader of key material controls the email addresses used as its UserID attributes, and allows for key modification and removal — but operates in a purely centralized way. It failed to gain traction among the OpenPGP users, probably due to the social traits earlier discussed regarding centralized and company-provided solutions. The *Mailvelope keyserver*, presented in 2012 [Sharma *et al.*, 2021], is another implementation that attempts to replace the aging and then-dominant SKS keyserver software implementation with a much simpler and more modern code base, written using a widely understood technological base (Javascript, instead of OCaml) and meant to be used as a browser extension, nevertheless doing away with the WoT altogether, mainly due to the justification that it does not scale in terms of usability, and pointing to placing more trust in the service provider that hosts a keyserver, switching to an operation mode closer to TOFU. Mailvelope suffered from the onset of a difficult user interface, with which users struggled to perform tasks supposed to be basic [Mauriés *et al.*, 2017].

In June 2019, the *Hagrid* keyserver project, as well as the `keys.openpgp.org` service, are announced [Walfield, 2019]. Hagrid, written in Rust and based on the Sequoia OpenPGP implementation library, addresses shortcomings not only in SKS, but also in the *gossip* protocol, and even in the design goals of a design goal of the keyserver network [Hansen, 2019]. In 2020, the first *VerifyThis Collaborative Long Term Challenge* deductive program verification challenge is carried out over Hagrid, assessing its truth to specification and code quality [Huisman *et al.*, 2020]. Walfield reasons that the keyserver network append-only data set is no longer feasible in today’s Internet, for reasons very much aligned to what we have so far described as well as other attacks:

Another problem with SKS is that it is effectively a garbage pile. This is by design: the key servers manage a *de facto* append-only log, and anyone can upload a key in anybody’s name. This would be fine if people used the key servers as intended. That is, as a repository for finding key updates (...) Instead, people assume that the key servers are a directory, like a telephone book, and that the published keys have received some minimal vetting. Unfortunately, the key server interface invites this interpretation: historically it makes looking up keys by email address easy, and up until a few years ago, this actually worked surprisingly well. But since vandalism on the key server infrastructure (...) has become common, the key servers have become increasingly polluted. This understandably confuses many users, and even advanced users and trainers blame the key server operators for not doing something to improve the situation.

*Hagrid* verifies email addresses before publishing user IDs, and by linking each key to its user ID, allows for users to control what information is stored. However, the project does not foresee a fully decentralized (*federated*, as referred to in their writeup) operation mode to be compatible with their aims. Some of the data they carry (particularly, key updat-

ing) might be made to operate over a *gossip*-like network in the future [Walfield, 2019]:

The SKS key server network is a federated network. But, a verifying key server requires a central authority to manage the authenticated directory. This would appear to preclude federation. In fact, it appears to preclude mirroring, because User IDs should not be wholesale exported due to privacy and GDPR concerns.

Happily, these concerns only apply to the email directory: the rest of the data can be synchronized. Since it is good practice to regularly check for key updates, we expect that most of the load will be directed to this part of the service anyway. Thus, a Hagrid-based key server network can function in a similar way to the SKS key server network, and it can use a similar trust model.

(...)

Currently, Hagrid does not provide a synchronization mechanism. If the OpenPGP community adopts Hagrid, then we plan to add support for this. One issue is that the protocol used by SKS, although peer reviewed, is not well documented.

*Hagrid* is not the only keyserver implementing verification. Few months after its announcement, the *onak* keyserver added a verification phase, accepting only *reachable keys* — keys with trust paths to and from the set of already known keys.

### 3.4 Beyond OpenPGP

Tools also have natural lifetimes, and considering that OpenPGP as a technology might be reaching its natural end-of-life should also be considered. This concern is cited as early as in Borisov *et al.* [2004], in which the authors present the *Off-The-Record* secure messaging protocol, based on the pursuit a different set of security properties from that of OpenPGP. Most particularly, it includes *deniability* (OTR messages are authenticated to come from the intended third party, but an interceptor cannot prove their provenance) and *perfect forward secrecy* (previous conversations are not considered compromised even in the event of the loss or leakage of private key material). OTR user authentication is based upon TOFU (described in Subsection 3.2).

This is achieved by very short-lived encryption keys and message authentication codes (MACs), automatically derived and discarded *at each communication cycle* between communication parties. OTR includes an ingenious step where communication parties *publish their previous private keys* in order to ensure past messages (that should be destroyed after receipt) explicitly *lose their non-repudiability*, as would happen with in-person communications where nobody can prove their counterpart said something: Once a message is acknowledged to be received, the session keys used to send it are rotated, and the MAC keys used to encrypt and sign previous messages are sent *in the clear*, ensuring all messages could have been forged.

In instant messaging, both the widely used *Signal* and the closely related *Whatsapp* end-to-end encryption protocols are based on OTR [Marlinspike, 2016], meaning the presented ideas are used by over a billion active users. It is undeniable, then, that OTR has been instrumental to bring usable encryption to its massification — and quite probably, that a large portion of users in the world are better served by other encryption and authentication means than OpenPGP. We hold, however, that there is still important value in preserving and promoting the decentralized, strong authentication properties of the WoT as provided by OpenPGP.

### 3.5 Related works summarization

This subsection presents a quick round-up of the above presented alternative strategies, and follows the summarization presented in Table 1; the two last rows of the table present the current working of the keyserver network, and the proposal put forward by this work (detailed next, in Section 4) is illustrated in the last row of the table.

The first column presents whether a tool is independent from the domain administration. This is important so that users with a UID located within a domain name can distribute and authenticate their keys using a given strategy; clearly, WKD and DANE require the domain administrators to grant users control of either a given (“well-known”) web page or DNS record. Users of large-scale mail providers (i.e. gmail.com, hotmail.com or the like) are not able to distribute their public keys or certificates in this way. The OTR protocol cannot by itself be clearly subject to this criteria, as it depends on the messaging service that builds upon it; while OTR can be used over fully decentralized protocols such as XMPP or IRC, its most common use is over centralized instant messaging networks, such as *Signal* or *Whatsapp*, but said platforms don’t provide for a way to interact with users in third-party platforms.

The second column addresses whether the alternatives can currently interface with the OpenPGP tooling already deployed to users. All of the tools defined within the OpenPGP ecosystem have taken care to build upon. Naturally, the OTR protocol defines a completely different ecosystem, and its program are not compatible with OpenPGP’s.

Next, the table addresses whether said implementation considers WoT support and its usage provides for using it. WKD and DANE do provide the necessary information for a user to build a WoT, but given the fact that many users are hampered from using said schemes as they might use a provider not supporting it, the support can only be considered partial. We consider that TOFU-based implementations to be partial as well, as they are based in a completely different trust model, do not fully support the WoT: public keys (also known as certificate chains) are distributed as part of the initial mail, and no way to locate participants of a WoT is provided. Mailvelope is built around the OpenPGP standard, and uses several OpenPGP tools, but its developers have chosen not to present any information related to transitive trust in order to keep the tool simpler

for their target users. Finally, the OTR protocol does not present a transitive trust model.

The following column details whether the strategy in question supports key distribution, this is, whether it conveys the needed keys to the communicating parties. All tools provide a means for conveying the keys for both endpoints of a conversation, so all tools have at least a *half-mark*, however, OTR does not implement a *Web of trust*, and TOFU schemes distribute only the key for the immediate counterpart: building a WoT trust path is impossible without resorting to a keyserver.

The fifth column, *UID address verification*, compares whether each of the solutions implements some sort of UID address verification. WKD and DANE provide address verification for the UID that falls under each of the domains that has a relevant service defined, but cannot verify identities for any outside keys. TOFU schemes do not provide a UID verification. As discussed earlier, this is one of the factors that allowed for the emergence of the certificate poisoning attack: throwaway identities can easily be created and used to create meaningless signatures. And this is precisely the point enforced by our 1PA3PC proposal; as mentioned in Section 3.3, the 1PA3PC protocol presents a *halfway verifying* keyserver, where new keys can be added without verification, but signature packet additions to them need to be certified.

The final column shows whether said scheme can support fully decentralized operation. Given most of the presented solutions work by claiming back control over the *current version* of the distributed keys, it is no surprise that none of the alternatives that present a single key distribution point fail to provide decentralized operation; among the presented alternatives, only TOFU (and both the current and proposed keyserver network protocols) provide true decentralized operation.

## 4 First-party attested third-party key certification (1PA3PC) protocol

As mentioned in Section 2.3, the described attack is based upon the fact that it is trivial for an attacker to append extra packets to an existing OpenPGP key: the current protocol for *Bob* to sign *Alice*'s key is as simple as Figure 4 illustrates; it is expected that *Bob* obtained *Alice*'s key fingerprint directly from her, but it is not enforced in any way. There is no validation from the keyserver that *Alice* agrees in any way with *Bob*'s addition to her certificate chain.

It is worth reminding that a certificate chain is a set of RFC4880-compliant packets [Callas *et al.*, 2007], starting with the holder's public key, to which arbitrary packets can be appended; each key certification, in which this work is focused, is one such packet.

This work proposes adopting a *first-party attested third-party certification* (1PA3PC) protocol for the keyserver network, as suggested in Kahn Gillmor [2019a] for an abuse-resistant keystore; this protocol is illustrated in Figure 5. The improve-

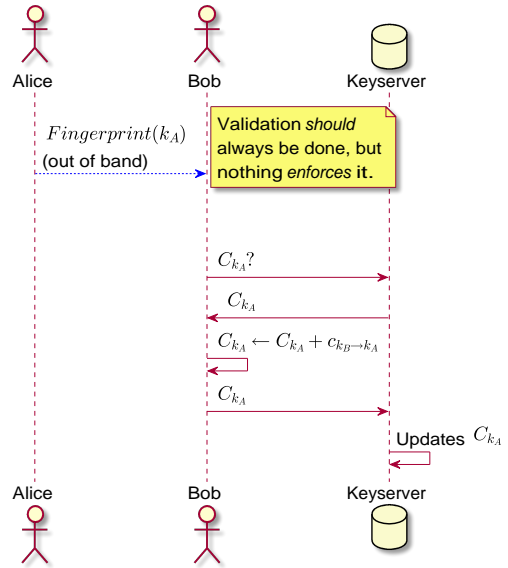


Figure 4. Current protocol for adding *Bob*'s signature to *Alice*'s certificate chain in the keyserver network.

---

### Algorithm 1: *Bob* intends to sign *Alice*'s certificate chain

---

**Input:** *Bob* has verified *Alice* as the owner of  $k_A$  out-of-band, has  $k_A$ 's fingerprint  $f_A$ .

**Output:** *Alice*'s certificate chain  $C_{k_A}$  including *Bob*'s signature,  $c_{k_B \rightarrow k_A}$ .

**Result:** *Alice* receives  $C_{k_B \rightarrow k_A}$  to act upon.

```

1 begin
2    $C \leftarrow \text{search}(\text{keyserver}, \text{Alice});$ 
3   if  $\text{found}(C)$  and  $\text{fingerprint}(C) == f_A$  then
4      $C \leftarrow \text{sign}(C, k_B);$ 
5     send(Alice,  $C$ );
6   end
7 end
```

---

ment this protocol presents centers upon the fact that no actor (*Bob*) should be able to append certification packets to a certificate chain unless said packets are *attested* by the certificate chain's owner (*Alice*).

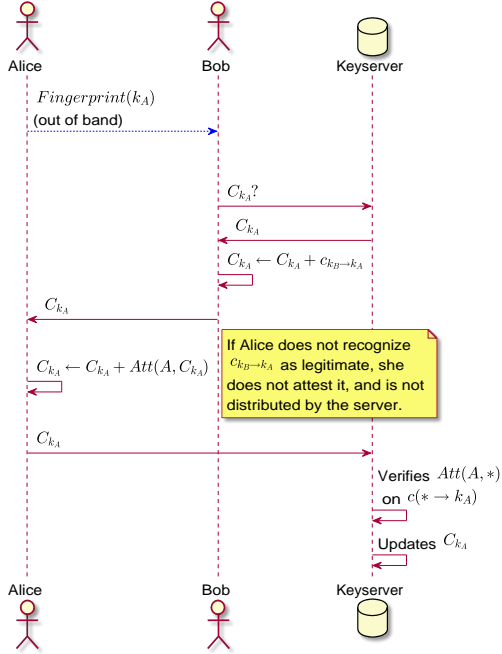
### 4.1 Protocol walk-through

As figure 5 shows, our proposed protocol involves the same three actors: Two public key owners: *Alice*, who owns (controls both the public and the private components) for  $k_A$ , and *Bob*, who owns  $k_B$  and an automated keyserver.

After meeting in person and exchanging identity information so that *Bob* trusts *Alice* is the true owner of  $k_A$ , she gives him the key's *fingerprint*. *Bob* wants to publicly certify *Alice*'s key, so he follows Algorithm 1:

*Bob* requests the keyserver for the certificate chain corresponding to *Alice*'s identity  $C$  and, after verifying the fingerprint matches and can be asserted to be  $C_{k_A}$ , creates a signature packet  $\text{sign}(C, k_B)$  and appends it to his local copy of  $C$ . *Bob* then sends  $C$  including his signature to *Alice*.

Upon receiving  $C$  from *Bob*, *Alice* proceeds with Algorithm 2: *Alice* traverses the list of packets in  $C$ . When she finds a packet  $c$  that contains *Bob*'s certification of her key ( $k_B \rightarrow k_A$ ), given she remembers having recently met and



**Figure 5.** Proposed protocol for adding *Bob's* signature to *Alice's* certificate chain in the keyserver network, requiring IPA3PC.

exchanged key fingerprints with *Bob*, acknowledges it is a legitimate certification. Of course, if *Alice* does not recognize  $c$  to be a legitimate packet, she discards it.

**Algorithm 2:** *Alice's* actions to acknowledge *Bob's* signature by attesting it.

**Input:** Certificate chain  $C$  received from *Bob*.  
**Output:** Certificate chain  $C_{k_A}$ .  
**Result:** *Alice* sends the keyserver  $c_{k_B \rightarrow k_A}$  and the matching attestation  $Att(k_A, C_{k_A})$ .

```

1 begin
2   foreach packet  $c$  in  $C$  do
3     if  $c = k_B \rightarrow k_A$  and  $A.acknowledge(c)$  then
4        $C_{k_A} \leftarrow c$ ;
5        $C_{k_A} \leftarrow Att(k_A, C_{k_A})$ ;
6       send(keyserver,  $C_{k_A}$ );
7     else
8       discard( $c_{k_A}$ );
9     end
10  end
11 end

```

She then appends  $c$  to her certificate chain  $C_{k_A}$ , creates an attestation  $Att(k_A, C_{k_A})$  acknowledging the addition of this signature to her certificate chain, and appends this attestation to her certificate chain as well. Having done that, *Alice* uploads her updated certificate chain  $C_{k_A}$  to the keyserver.

This triggers the keyserver to start Algorithm 3: The keyserver receives a set of RFC4880-compliant packets,  $P$ .  $P$  happens to include data augmenting *Alice's* certificate chain,  $C_{k_A}$ . Do note that the keyserver does not validate who is the originator for: it might have been submitted by an individual user such as *Alice*, or it could have been imported after synchronizing with a different keyserver; Algorithm 3 is ex-

ecuted nonetheless.

**Algorithm 3:** The keyserver receives *Alice's* certificate chain, which might carry new packets in it

**Input:** Keyserver has received cryptographic material including any packets to be appended to  $C_{k_A}$   
**Result:** Keyserver's database is updated, appending to *Alice's* certificate chain  $C_{k_A}$  the new signature  $c_{k_B \rightarrow k_A}$  and its attestation by *Alice*,  $Att(k_A, C_{k_A})$ .

```

1 begin
2    $Att_P \leftarrow attestations\_index(P)$ 
3   foreach packet  $c$  in  $P$  do
4     if  $c(* \rightarrow k_A)$  then
5       if  $Att_P$  includes  $Att(k_A, c)$  then
6          $k_A \leftarrow c$ ;
7          $k_A \leftarrow Att(k_A, c)$ ;
8         add_to_database( $k_A$ );
9       else
10        discard( $c$ );
11      end
12    end
13  end
14 end

```

For the purposes of this work, we focus on the case where the received data includes packets to be appended to  $C_{k_A}$ . The keyserver builds an index of attestations  $Att_P$  present in  $P$  and starts processing all packets in  $P$ . If a given packet  $c$  is a certification on  $k_A$ , it searches whether  $Att_P$  includes a matching attestation  $Att(k_A, c)$ . If a matching attestation is found, the keyserver adds both  $c$  and  $Att(k_A, c)$  to its database, otherwise, it discards  $c$ .

At this point, it should become clear to the reader that the proposed IPA3PC protocol effectively counters the flaws in the current scheme that make the certificate poisoning possible (refer back to Figure 4): by making any packet addition to certificate chain  $C_{k_A}$  depend on having been approved by its owner *Alice* by the means of an attestation  $Att(k_A, c)$ , a legitimate user  $B$  (*Bob*) is able to certify  $C_{k_A}$ , but given each certification needs *Alice* to attest it, a malicious actor *Mallory* is no longer able to attack  $C_{k_A}$  with certificate poisoning; at most, she can create an army of fake identities to saturate *Alice's* mailbox with unwanted attestation requests — but such an attack constitutes only a temporary annoyance (*Alice* will be able to purge her mailbox of said spam), and does not carry any longer lasting implications.

## 4.2 Interaction with the deployed OpenPGP ecosystem

It is worth noting the protocol our work presents is aligned with the uses recommended in Daniel Kahn Gillmor's document, currently presented as an early IETF draft [Kahn Gillmor, 2023], roughly following the *Reasonable Workflows* outlined in its section 4.1, *Third-party Certification and Attestation Workflow*. Given its status as an early-stage draft,



our implementation does not yet follow the other workflows therein suggested.

Attestations are akin to certifications on individual OpenPGP packets, or sets thereof, signed by the user controlling the private key material corresponding to a certificate chain.

Our proposed protocol requires all signature packets affecting a given certificate chain  $C_{k_A}$  to be attested by  $k_A$ ; the keyserver performs a verification on each of them, dropping all those that fail the validation.

Our protocol modifies only the logic at the keyserver, not requiring any modifications in the installed base client-side OpenPGP software for key querying; while key attestation is not part of the published OpenPGP message format [Callas *et al.*, 2007], standardization efforts are under way and in their final stages of approval by the IETF [Koch *et al.*, 2022], and the *Sequoia* client software already supports it [Azul *et al.*, 2021].

As a beneficial side effect of adopting this protocol, and as can be seen in Figure 5, the owner of a key is no longer cut off from any action performed over it: if *Bob* is willing to certify *Alice*, this information is made public only after (and whether) *Alice* attests it. This allows *Alice* to control the amount of social interactions her key displays. It also thwarts careless users who certify the wrong key without thoroughly verifying its full fingerprint. It should be noted that mailing signed keys to the signee for them to control publication has been a *best practice* recommendation by keysigning aide tools for decades [Palfrader *et al.*, 2020], but cannot be mandated if the keyservers do not require it as part of their protocol.

Although 1PA3PC is suggested as a strategy for implementing an abuse-resistant OpenPGP keystore [Kahn Gillmor, 2019a], no other implementations are published following it. It should be clear to understand why 1PA3PC makes certificate poisoning impossible: Following again the interaction presented as example in Subsection 2.3, even if *Bob* is not careful and attests some of *Mallory*'s spurious certificates, he does not accept the tens of thousands; at most, *Bob* suffers a Denial of Service (DoS) attack on his mailbox, but any possible damage is transient, and does not impact any issues for the future use of *Bob*'s certificate chain. *Bob* does not attest tens of thousands of attestation requests, and as a consequence, the keyservers never distribute most of  $C_{k_{M1}}$  through  $C_{k_{M10000}} \cdot C_{k_B}$  thus remains sane and usable.

## 5 Experimental verification

This section presents the experimental model built and used to support the above presented claims, namely, that a 1PA3PC protocol can be applied, with minimal modifications to keyserver software and with no modifications needed for user-side tools, that completely prevents the ill effects of certificate poisoning. We support our claims via the following experiment:

We set up a network of five keyservers based on the *Hockeypuck* software [Marshall, 2015]. 500 OpenPGP keys are generated using the *Sequoia* client [Azul *et al.*, 2021], and a laid out WoT on it, with a signature between each of two

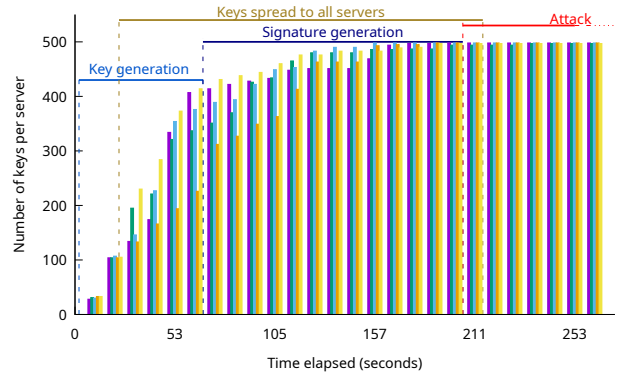


Figure 6. Number of keys present in the keyserver network over the lifetime of the attack simulation.

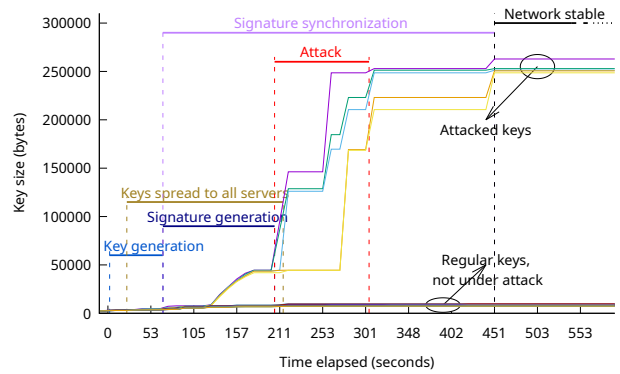


Figure 7. Size of the 20 largest certificate chains, as seen by one of the keyservers in the network, during the lifetime of the attack simulation, using the traditional protocol.

keys at random being made with  $p = 0.01$ . We do not believe specific WoT properties are relevant for the experiment, so it was not attempted to replicate the internal structure of the real OpenPGP WoT; the reason for creating the signatures is to introduce the expected variance in the number of packets that constitute each of the keys, thus driving their size to realistic values. The 500 keys are uploaded randomly to each of the five keyservers. Figure 6 shows the upload and synchronization progress to the five keyservers until they converge on the same keyset. As Figures 6, 7 and 8 show, key generation and upload happens since the beginning of the experiment and approximately until the end of its first minute ( $t = 60$ ).

The keyservers connect with each other for synchronization several times per minute, and this can be seen with the spread of keys through them, starting approximately at

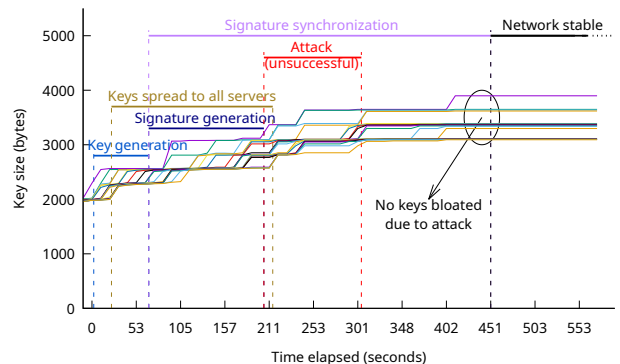


Figure 8. Size of the 20 largest certificate chains, as seen by one of the keyservers in the network, during the lifetime of the attack simulation, using our 1PA3PC-enforcing protocol.

$t = 15$  and until the number of keys per server stabilizes at 500, in Figure 6, at  $t = 215$ . Signature packets continue to be appended to the keys, but their number remains stable with 500 keys from this point on: all of the additional information received consists on signature packets added to existing keys.

After the “valid” signatures are generated, and as the “legitimate” WoT gets populated, signatures are still spreading through the keyserver network, an attack is simulated: Starting around  $t = 200$ , 1 000 further keys are created (but not uploaded to the keyserver network). Five victim keys are selected and signed with the 1 000 attacker keys, with the signatures uploaded randomly to each of the keyserver. The attack is quite efficient time-wise: It takes only close to 100 seconds to perform. As a result, as Figure 7 shows, by  $t = 300$ , the five attacked keys are *bloated* to almost 300KB, while the rest of the keys remain under to 5KB in size. The keyserver synchronization protocol is left running so the information spreads to the whole network, and by  $t = 450$ , information flow over the keyserver network stabilizes.

In contrast, using our proposed 1PA3PC protocol, the same experiment is carried out. It becomes clear that the attack does not succeed, as Figure 8 shows: The 20 largest keys in the keyring remain all within the same range and, while they continue to successfully receive certifications, an ill-intentioned *Mallory* is no longer able to disrupt *Bob* and his position in the WoT.

Summing up the reported times, the experiment consists of the following approximate intervals (in seconds):

- 0 – 60** Key generation
- 15 – 215** Keys spread to all servers
- 60 – 200** Signature generation
- 200 – 300** Attack
- 60 – 450** Signature synchronization
- 450 –** Network is stable

As for the results, while the attack led to five keys being disproportionately larger than the rest of the keys present in the keyserver network in Figure 7, showing evidence of a successful attack, Figure 8 shows all keys within the same range, as expected from the random signature distribution explained earlier in this section. It is worth keeping in mind that the attack simulated in this experiment consisted of 1 000 throwaway, hostile keys only; observed certificate poisoning attacks have been close to a hundred times bigger.

## 6 Discussion

The above section shows how the attack presented in Subsection 2.3 is effectively countered. Adopting our 1PA3PC protocol makes it impossible to generate an attack with the characteristics described, as any modifications on  $C_{k_B}$  (any packets appended to  $k_B$ ) have to be attested by  $k_B$ . If such attestations appear, it would be a clear indication that, if anything, the corresponding private key has been leaked to the attacker, and  $k_B$  should be revoked; safely protected keys being bloated beyond a level of usability are now completely prevented.

## 6.1 Threats to validity and applicability

Given this work proposes a protocol change for the keyserver network, a scope for its adoption must be set.

This proposed protocol should ideally be adopted by the totality of nodes in a keyserver network, not a subset of them. Given the validation is carried out when *accepting new packets*, the whole network, as it currently stands, with its over 6 million keys, can be taken as a basis. All keys and certifications currently part of the WoT network can be kept operative. However, a keyserver using our 1PA3PC protocol *should no longer peer* with servers not using it: given the logic of operation in the *gossip* protocol, delivery of new packets including non-attested signatures would continue to be attempted by old-version keyserver, only to be analyzed and dropped by 1PA3PC ones, leading to an ever-growing delta that can end up disrupting synchronization.

The new logic, validating every new packet “kicks in” at the moment any key’s packets are considered for inclusion in any of the keyserver network nodes; this is, *each of the keyserver* will validate *every new certification packet* to have been attested by its corresponding key. Of course, if this new protocol is to be deployed over the preexisting keyserver network, the keys and their certificates before the protocol change would remain valid.

## 6.2 Death by kindness

The OpenPGP ecosystem has long been criticized because, probably due to the complex interactions it allows for, its usability is seen as dismal by many of its users [Whitten and Tygar, 1999][Sheng *et al.*, 2006][Woo, 2006].

In personal communication with Neal H. Walfield, from the Sequoia project, he points out that, while the present protocol modification proposal does address certificate poisoning, it can lead to *death by kindness*: OpenPGP users are relatively very few among computer users that actually pursue tools to protect the privacy of their communications. From them, the subset that takes part in the WoT TTM is even smaller. Walfield succinctly points out that adding requirements of any kind such as transitioning from the protocol illustrated by Figure 4 to one carrying more communicational complexity, such as our proposal in Figure 5, risks further shrinking the user base beyond a usable threshold.

## 7 Conclusions and future work

Through this paper, we show how it is possible to create a protocol that makes the *certificate poisoning* attack unfeasible, requiring a very small change to the currently leading keyserver software’s logic and without requiring client-side software changes beyond what the OpenPGP standard already specifies.

Our solution preserves the distributed properties of the preexisting *gossip*-based keyserver synchronization network. This protocol introduces a much better level of control on the distribution of key certification material than what has been historically available for the distributed transitive trust model

known as the *OpenPGP Web of Trust* (presented in Subsection 2.1), particularly solving the problem presented in Subsection 2.3 *ascertificate poisoning*.

Section 5 shows the experimental set up that the proposed protocol effectively protects a keyserver network from the specific threat our work is set to counter. Section 6 explains some scenarios, however, of why adopting this protocol is not *just* a matter of reconfiguring keysevers — what kind of justified non-technical resistances can be found for this protocol to be widely adopted. Section 3 covered a review of other ways cryptographic keys’ trust, discovery and distribution can be tackled.

Being this a security-oriented protocol proposal, we acknowledge the importance of presenting a formal verification of the protocol’s soundness. Part of the necessary work we intend to do following this article is modelling and verifying the interaction to ensure, beyond the limits an experimental verification process allow, all intended security properties are effectively met and upheld.

We acknowledge there are still important problems to be solved if we are to expect the keyserver network to recover from the fall it suffered, such as the *right of erasure* required by the European GDPR and comparable privacy-preserving laws throughout the world. Even though they have contributed to the reported shrinking base of the keyserver network, they are outside the scope of this work. Finding a way to allow for secure deletion of personally-identifying information while preserving decentralization could constitute the focus of future work. However, given our approach bases the ability to control any modification to the certification packets that make up a given key on them being cryptographically attested, it is not expected this line of work can solve the issue of a court order or simple user request not backed by a cryptographic proof of ownership; if a given user loses access or control of their private key material, neither ours nor any other protocol-level modifications will present a viable solution.

## Declarations

### Acknowledgements

The authors want to thank Jonathan McDowell and Justus Winters for their valuable input during the conceptualization and development of this project. Heiko Schaefer provided fundamental guidance towards understanding and modifying bits Rust code to provide the needed hooks for the proposed protocol.

### Authors’ Contributions

All authors contributed to the writing of this article, read and approved the final manuscript.

### Competing interests

The authors declare they have no competing interests.

## Availability of data and materials

The data generated for this study is randomly generated, as explained in Section 5. Source code for the keyserver modifications, as well as for driving the data generation and study, are available at <https://1pa3pc.gwo1f.org/>.

## References

- Arkko, J. and Nikander, P. (2004). Weak authentication: How to authenticate unknown principals without trusted parties. In Christianson, B., Crispo, B., Malcolm, J. A., and Roe, M., editors, *Security Protocols*, pages 5–19, Berlin, Heidelberg. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-39871-4\_3.
- Azul, Matuszewski, I., Winter, J., Michaelis, K., Walfield, N., Widdecke, N., and Kwapisiewicz, W. (2021). User manual: sq - a command-line frontend for sequoia, an implementation of openpgp. Available at: <https://docs.rs/crate/sequoia-sq/0.26.0>.
- Borisov, N., Goldberg, I., and Brewer, E. (2004). Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. Association for Computing Machinery. DOI: 10.1145/1029179.1029200.
- Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and Thayer, R. (2007). Openpgp message format. *Internet Engineering Task Force (IETF)*, (4880). Available at: <https://www.rfc-editor.org/info/rfc4880>.
- Conti, M., Dragoni, N., and Lesyk, V. (2016). A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 18(3):2027–2051. DOI: 10.1109/COMST.2016.2548426.
- Hansen, R. J. (2019). Sks keyserver network under attack. Available at: <https://gist.github.com/rjhansen/67ab921ffb4084c865b3618d6955275f>.
- Herzberg, A. and Leibowitz, H. (2016). Can johnny finally encrypt? evaluating e2e-encryption in popular im applications. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust*, pages 17–28. DOI: 10.1145/3046055.3046059.
- Huisman, M., Monti, R., Ulbrich, M., and Weigl, A. (2020). The verifythis collaborative long term challenge. In Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., and Ulbrich, M., editors, *Deductive Software Verification: Future Perspectives: Reflections on the Occasion of 20 Years of KeY*, volume 12345 of *Lecture Notes in Computer Science*, chapter 10, pages 246–260. Springer. DOI: 10.1007/978-3-030-64354-6\_10.
- Johansen, C., Mujaj, A., Arshad, H., and Noll, J. (2017). Comparing implementations of secure messaging protocols. Technical Report ISBN 97-82-7368-440-0, Universitet i Oslo. Available at: <https://www.duo.uio.no/handle/10852/60949>.
- Jøsang, A. (1999). An algebra for assessing trust in certification chains. In *Proc. Network and Distributed Systems Security Symposium, 1999 (NDSS’99)*. The Internet Society. Available at: <https://api.semanticscholar.org/CorpusID:2119340>.

- Kahn Gillmor, D. (2019a). Abuse-resistant openpgp keystores. Internet-Draft draft-dkg-openpgp-abuse-resistant-keystore-04, Internet Engineering Task Force. Available at: <https://datatracker.ietf.org/doc/html/draft-dkg-openpgp-abuse-resistant-keystore-04>.
- Kahn Gillmor, D. (2019b). Openpgp certificate flooding. Available at: <https://dkg.fifthhorseman.net/blog/openpgp-certificate-flooding.html>.
- Kahn Gillmor, D. (2023). First-party attested third-party certifications in openpgp. Internet-draft, Internet Engineering Task Force. Available at: <https://datatracker.ietf.org/doc/draft-dkg-openpgp-1pa3pc/>.
- Koch, W. (2021). OpenPGP Web Key Directory. Internet-Draft draft-koch-openpgp-webkey-service-12, Internet Engineering Task Force. Available at: <https://datatracker.ietf.org/doc/html/draft-koch-openpgp-webkey-service-12>.
- Koch, W., Huigens, D., Winter, J., and Niibe, Y. (2022). Openpgp message format. Internet-Draft draft-ietf-openpgp-crypto-refresh-06, Internet Engineering Task Force. Available at: <https://datatracker.ietf.org/doc/pdf/draft-ietf-openpgp-crypto-refresh-06>.
- Krekel, H., McKelvey, K., and Lefherz, E. (2018). How to fix email: Making communication encrypted and decentralized with autocrypt. *XRDS: Crossroads, The ACM Magazine for Students*, 24(4):37–39. DOI: 10.1145/3220565.
- Marlinspike, M. (2016). Whatsapp’s signal protocol integration is now complete. Available at: <https://signal.org/blog/whatsapp-complete/>.
- Marshall, C. (2015). Hockey puck. Available at: <https://hockeypuck.io/>.
- Mauriés, J. R. P., Krol, K., Parkin, S., Abu-Salma, R., and Sasse, M. A. (2017). Dead on arrival: Recovering from fatal flaws in email encryption tools. In *The LASER Workshop: Learning from Authoritative Security Experiment Results (LASER 2017)*, pages 49–57. USENIX Association. Available at: <https://www.usenix.org/conference/laser2017/presentation/mauries>.
- Minsky, Y. M. (2002). *Spreading rumors cheaply, quickly, and reliably*. PhD thesis, Cornell University. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.5620&rep=rep1&type=pdf>.
- Palfrader, P., Berg, C., and Moulin, G. (2020). Caff: Certificate authority fire and forget. Available at: <https://manpages.debian.org/bookworm/signing-party/caff.1.en.html>.
- PGP Global Directory (2004). Pgp global directory verified key service. Available at: <https://keyservers.gpg.com/>.
- Pham, V. and Aura, T. (2011). Security analysis of leap-of-faith protocols. In *International Conference on Security and Privacy in Communication Systems*, pages 337–355. Springer. DOI: 10.1007/978-3-642-31909-9\_19.
- Pramberger, P. (2010). Keyserver.pramberger.at terminating. Available at: <https://lists.nongnu.org/archive/html/sks-devel/2010-09/msg00009.html>.
- Renaud, K., Volkamer, M., and Renkema-Padmos, A. (2014). Why doesn’t jane protect her privacy? In De Cristofaro, E. and Murdoch, S. J., editors, *Privacy Enhancing Technologies PETS 2014*, volume 8555 of *Lecture Notes in Computer Science*, pages 244–262. Springer International Publishing. DOI: 10.1007/978-3-319-08506-7\_13.
- Rescorla, E. (2018). The transport layer security (tls) protocol, version 1.3. *Internet Engineering Task Force (IETF)*, (8446). Available at: <https://www.rfc-editor.org/info/rfc8446>.
- Sharma, R., Dangi, S., and Mishra, P. (2021). A comprehensive review on encryption based open source cyber security tools. In *2021 6th International Conference on Signal Processing, Computing and Control (ISPCC)*, pages 614–619. DOI: 10.1109/ISPCC53510.2021.9609369.
- Shaw, D. (2003). The openpgp http keyserver protocol (hkp). Internet-Draft draft-shaw-openpgp-hkp-00, Internet Engineering Task Force. Available at: <https://datatracker.ietf.org/doc/html/draft-shaw-openpgp-hkp-00>.
- Sheng, S., Broderick, L., Koranda, C. A., and Hyland, J. J. (2006). Why johnny still can’t encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4. ACM. Available at: [https://cups.cs.cmu.edu/soups/2006/posters/sheng-poster\\_abstract.pdf](https://cups.cs.cmu.edu/soups/2006/posters/sheng-poster_abstract.pdf).
- Walfield, N. H. (2019). Hagrid: A new verifying key server built on sequoia. Available at: <https://sequoia-pgp.org/blog/2019/06/14/20190614-hagrid/>.
- Walfield, N. H. and Koch, W. (2016). Tofu for openpgp. In *EuroSec’16: Proceedings of the 9th European Workshop on System Security*, pages 1–6. DOI: 10.1145/2905760.2905761.
- Whitten, A. and Tygar, J. D. (1999). Why johnny can’t encrypt: A usability evaluation of pgp 5.0. In *USENIX Security Symposium*, volume 348, pages 169–184. Available at: [https://www.usenix.org/legacy/events/sec99/full\\_papers/whitten/whitten.ps](https://www.usenix.org/legacy/events/sec99/full_papers/whitten/whitten.ps).
- Wolf, G. and Gallegos, G. (2017). Strengthening a curated web of trust in a geographically distributed project. *Cryptologia*, 41(5):459–475. DOI: 10.1080/01611194.2016.1238421.
- Wolf, G. and Quiroga, V. G. (2018). Insights on the large-scale deployment of a curated web-of-trust: the debian project’s cryptographic keyring. *Journal of Internet Services and Applications*, 9(1):1–12. DOI: 10.1186/s13174-018-0082-7.
- Woo, W. K. (2006). *How to exchange email securely with Johnny who still can’t encrypt*. PhD thesis, University of British Columbia. Available at: <https://open.library.ubc.ca/cIRcle/collections/ubctheses/831/items/1.0064951>.
- Wouters, P. (2016). Dns-based authentication of named entities (dane) bindings for openpgp. *Internet Engineering Task Force (IETF)*, (7929). Available at: <https://www.rfc-editor.org/info/rfc7929>.
- Yakamo, K. (2018). Are sks key servers safe? do we need them? Available at: <https://medium.com/>

@mdrahony/are-sks-keyservers-safe-do-we-need-them-7056b495101c.

Yamane, S., Wang, J., Suzuki, H., Segawa, N., and Murayama, Y. (2003). Rethinking openpgp PKI and openpgp public keyserver. *CoRR*, cs.CY/0308015. DOI: 10.48550/arXiv.cs/0308015.

Zimmermann, P. (1999). Why i wrote pgp. Available at: <https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html>.