# Challenging the Trustworthiness of PGP: Is the Web-of-Trust Tear-Proof?

Alessandro Barenghi, Alessandro Di Federico, Gerardo Pelosi(✉), and Stefano Sanfilippo

Department of Electronics, Information and Bioengineering – DEIB, Politecnico di Milano, Milano, Italy
{alessandro.barenghi,alessandro.difederico, gerardo.pelosi,stefano.sanfilippo}@polimi.it

**Abstract.** The OpenPGP protocol provides a long time adopted and widespread tool for secure and authenticated asynchronous communications, as well as supplies data integrity and authenticity validation for software distribution. In this work, we analyze the Web-of-Trust on which the OpenPGP public key authentication mechanism is based, and evaluate a threat model where its functionality can be jeopardized. Since the threat model is based on the viability of compromising an OpenPGP keypair, we performed an analysis of the state of health of the global OpenPGP key repository. Despite the detected amount of weak keypairs is rather low, our results show how, under reasonable assumptions, approximately 70 % of the Web-of-Trust *strong set* is potentially affected by the described threat. Finally, we propose viable mitigation strategies to cope with the highlighted threat.

**Keywords:** Web-of-Trust · WoT · OpenPGP · GPG · PGP

## 1 Introduction

The continuous increase in the size of computing systems, and the amount of data processed and exchanged by them calls for a widespread and trustworthy infrastructure for secure communications, encompassing both synchronous data transport and asynchronous messaging. Secure and endpoint-authenticated transport is nowadays provided by the Transport Layer Security (TLS) protocol [6], which is regarded as the most widespread solution when it comes to interactive communications between a server and a client. By contrast, the main workhorse in providing both confidentiality of the contents and sender authenticity, when it comes to secure e-mails, is the Open Pretty Good Privacy (OpenPGP) protocol [3]. The use of OpenPGP has been recently encouraged as a practical countermeasure to dragnet surveillance actions involving e-mail inspection. In particular the Free Software Foundation has promoted a campaign [29] to foster its use even among non technically-savvy users. Finally, the OpenPGP protocol is widely used to ensure data authentication and integrity of


G. Pernul et al. (Eds.): ESORICS 2015, Part I, LNCS 9326, pp. 429–446, 2015.
DOI: 10.1007/978-3-319-24174-6_22

binary packages of both all the Debian and RedHat derived GNU/Linux distributions, and a significant number of other popular ones such as Arch, Slackware and Gentoo. Therefore, the authenticity of the software binaries installed on the overwhelming majority of GNU/Linux systems is provided by OpenPGP signatures.

Since 2010, the official implementation of the OpenPGP protocol is available as a commercial technology by Symantec Corp., even if its source code is publicly available for peer review [28]. In addition to its employment as a solution to provide confidentiality and sender-authentication for e-mails, Symantec's products also employ the same protocol for securing files and documents. The OpenPGP protocol, first defined in the RFC2440 [4] and then amended and extended in the RFC4880 [3] by the Internet Engineering Task Force (IETF), has its best known implementations both in proprietary solutions (e.g., the *Google Chrome* browser extension called `end-to-end` [26], which has also been forked and adopted by *Yahoo! Mail* [32]) and in the free alternative GNU Privacy Guard (GPG) software suite [13].

The security services offered by OpenPGP all hinge on the requirement to perform sound public key authentication. The adopted approach relies on a distributed, asynchronous trust model as an alternative to both the hierarchical Public Key Infrastructure (PKI) [5,7], and the distributed and synchronous approach of *Perspectives* [31]. The mainstay of the OpenPGP protocol is its Web-of-Trust (WoT), which provides a way to establish the binding of a public key to an identity through having a number of peers a specific user trusts acting as certification authorities for it. This is realized through having all the OpenPGP users sign the public key-identity pairs belonging to anyone they could directly verify the identity of (e.g., via meeting in person). This practice, under the "small world assumption", grows a tightly knit network of trust-relationships, which allows anyone to authenticate public key-identity pairs.

**Contribution.** In this work we provide a survey of the state of health of the key material employed by OpenPGP, and globally distributed via a public network of keyservers. Subsequently, we describe a practical threat model, aiming at invalidating the public key authentication mechanism provided by OpenPGP, on the basis of a broken keypair either directly or indirectly authenticated by a trustworthy user. We evaluate the effective applicability of the proposed threat model, as a result of the weak keypairs we detected, reporting the portion of the most trusted subset of the OpenPGP WoT for which the authentication mechanism can be fooled. Finally, we suggest viable countermeasures to mitigate the effect of the described threat, and evaluate their actuation cost.

**Organization of the paper.** Section 2 provides a detailed overview of the inner workings of the OpenPGP protocol, and a survey of the current state of the WoT, Sect. 3 proposes our threat scenario, Sect. 4 reports the state of health of the global key storage, and Sect. 5 evaluates the applicability of the described threat, and proposes mitigation measures.

## 2 OpenPGP Infrastructure

A user in OpenPGP is associated with one or more *user-ID*s, each of which is composed by a text string usually including his real name and e-mail address. Each user generates a bundle of public key/private key pairs. Among the public key/private key pairs, one is denoted as a *primary key* pair, while each one of the others is denoted as a *subkey* pair. Conventionally, the primary key pair is employed only for signing purposes, while subkey pairs are employed to either encrypt or sign messages. The message encryption function employs a hybrid scheme using a combination of symmetric key cryptography for speed, and public key cryptography for ease of secure symmetric-key exchange between the sender and the receiver of the data transfer. In particular, OpenPGP employs a symmetric key cipher, with a randomly generated ephemeral key, to encrypt the message to be transferred. The ephemeral key is sent encrypted with the recipient's public key along with the encrypted message.

Users issue *certificate*s to each other to authenticate the binding between user-IDs and public keys (primary or subkey). This is obtained signing with their primary private keys a subset of certificate data including the user-ID and one public key. A certificate contains one primary public key, and at least a self-signature binding it to the user-ID. In addition, it may contain several signatures verifiable via public keys of other users. The global distribution of OpenPGP certificates is realized via a network of public key directories, known as *keyservers*, which provide a synchronized billboard accessible via either a dedicated interface over HTTP, known as the HKP protocol [25]. We note that the available implementations of the OpenPGP keyserver do not support TLS, although it is possible to add it employing a reverse proxy. The synchronization across keyservers is maintained with a set-reconciliation algorithm [17], which guarantees that the uploading of a certificate on one of the servers will be mirrored by all the others. The servers are not required to perform any integrity or sanity checks.

### 2.1 Key Management

Each user keeps his own local key storage, known as *keyring*, containing a number of certificates, plus his own private keys (which are never disclosed). The keyring is complemented with two local maps, stored in the so-called *trust-db*. The first map associates each public key in the keyring to its *trust level*, i.e., the amount of trust the owner of the trust-db has in the actions of the public key owner. The second one binds each public key in the keyring to its *validity level*, i.e., the extent to which the keyring owner deems the key authentic.

**Trust level assignment.** The admissible trust values, according to the default trust model of GPG are: *(i)* `ultimate`, which is reserved for the keyring owner's public keys; *(ii)* `full`, *(iii)* `marginal`, *(iv)* `untrusted`, *(v)* `undefined`, and *(vi)* `unknown`. Trust levels from *(i)* to *(iv)* can only be explicitly assigned by the user through a direct interaction with the OpenPGP client. This is typically done after the user has ascertained the identity and trustworthiness of the public key

owner either meeting her in person, or by any other means he sees fit. The `unknown` trust level is automatically assigned by the OpenPGP client to a new public key whenever it is imported into the keyring. Whenever a public key contained in the keyring is employed to verify a signature on a different public key, the client checks whether its trust level is set to `unknown`, and, if possible, asks the user to provide one. In case it is not possible to obtain an explicit trust level from the user, the client sets the trust level to `undefined`.

**Validity level computation.** The admissible validity values for a public key are: *(i)* `full`, *(ii)* `marginal`, *(iii)* `untrusted`, and *(iv)* `unknown`. The validity level of a public key is computed by the OpenPGP client employing the certificates contained in the keyring, and both the trust and validity values in the trust-db. The public key is deemed authentic if its validity level is `full`. Whenever a new public key is imported into the user keyring, its validity is automatically set to the `unknown` level. The computation of the validity level of a public key takes place every time it needs to be employed, and its validity level in the trust-db is `unknown`. All public keys having an `ultimate` trust level have their validity level set to `full`. Public keys carrying a signature verified with a public key having an `ultimate` trust level are considered to have `full` validity. Thus any piece of key material carrying a signature verifiable by the public key of the keyring owner is considered `fully` valid. Subsequently, all the public keys carrying a signature verified by a `fully` valid, `fully` trusted public key are assigned a `full` validity level. If the signature on a public key is verified by a `fully` valid, but `marginally` trusted public key its validity level is set to `marginal`. Whenever three such signatures are verified on the same public key, its validity level is promoted to `full`. Signatures which can be verified by public keys with an `untrusted` trust level are not taken into account in the computation. If a signature on a public key is verified by a public key having an `undefined` trust level, the signed public key validity is set to `undefined`. The aforementioned validity level computation rules allow the client to assign a value to the validity of a signed public key taking into account the one which verifies the signature. This process effectively creates chains of validity dependence among public keys, where each signed one depends on the one verifying the signature to be validated.

**Revocations management.** The revocation of both public keys and signatures made to certify the binding between an identity and a public key are performed creating a *revocation signature*. Three types of revocation signatures are possible: (i) a primary key revocation, (ii) a subkey revocation, and (iii) a signature revocation, which voids the authenticity of a signature, regardless of whether it is correctly verified by the corresponding public key or not. The OpenPGP revocation management allows to mark a signature as *non revocable*: in this case, all the revocation signatures on it are ignored. Finally, it is possible to indicate an expiration date for both public keys and signatures.

### 2.2 The OpenPGP Web-of-Trust

Each OpenPGP user is endowed with a keyring and a trust-db, which represent the means by which he will authenticate the public keys contained in the former. The most common way to analyze the effectiveness of the authentication mechanism is to examine the certifier-certified relation among the different public keys of the keyring. This certifier-certified relation is commonly represented in terms of a directed graph [30] with public keys as nodes and signatures as directed edges exiting from the certifying node and entering into the certified one. Such a representation implies that the public key contained in the source node can be used to verify a signature on the destination node. The direction of the arcs is a convention chosen for the sake of clarity: we note that the authors of [30] employ arcs in the opposite direction. This graph is known as the Web-of-Trust (WoT) of a keyring, although it is indeed the certifier-certified relationship being represented, instead of the user specified trust.

**Table 1.** Contents of the OpenPGP keyservers as of March 2015, reporting a ≈41 % increase of the number of certificates w.r.t. the figures reported in 2011 by [30]

| | Total | Revoked | Expired |
|---|---|---|---|
| Primary public keys (certificates) | 3,867,397 | 181,833 | 13,754 |
| Public subkeys | 3,597,910 | 27,670 | 2 |
| Signatures | 13,866,817 | 78,976 | 1,828,630 |

Willing to obtain information on the state of all the publicly available certificates, we analyzed the contents of the distributed keyserver network as if it were a single large keyring, and its corresponding WoT. We obtained a snapshot of the whole keyserver contents as of March 2015, of which we report a synoptic overview in Table 1. Note that it is possible for an OpenPGP user to generate a keypair and never upload the corresponding certificate on the keyservers. The number of subkeys is smaller than the number of primary keys: this is caused by the old certificate formats of OpenPGP (Ver. 3 and earlier) not mandating the generation of separate subkeys to relieve the primary keypair from encryption uses. We also note that the amount of revoked keys is comparatively small (≈4.7 %), and the number of expired ones is almost negligible (0.35 %). In particular, we ascertained that 99.6 % of the primary public keys do not have an expiration date set, which may be ascribed to the optional nature of the expiration date field [3]. We report the presence of 3,828,825 unique user-IDs, thus pointing strongly at a one-to-one correspondence between user-IDs and primary public keys for most of the OpenPGP users, although the standard [3] allows for multiple user-IDs. The mean number of identity-public key binding signatures per certificate, including the mandatory self-signature, is 2.08, pointing to the whole WoT as a rather sparse graph. The current global keyring contains 33,136 non revocable signatures.

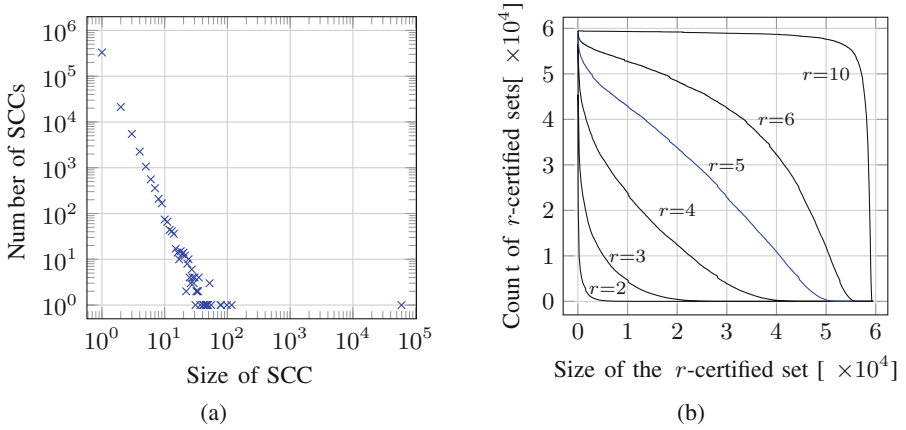


**Fig. 1.** Structural features of the WoT and the strong set as of March 2015. (a) reports the number of distinct SCCs of the WoT and their size. (b) depicts the distribution of the $r$-certified sets sizes over the strong set (GPG default: $r = 5$, in blue) (Color figure online)

The concept of Strongly Connected Components (SCCs) of a directed graph is a key tool to analyze the usefulness of the WoT [30]. A SCC is a maximally connected subgraph where there is at least one path between every node pair. Computing the number of SCCs and their size yields the data reported in Fig. 1(a), taking into account only non-revoked and non-expired public keys and signatures. Examining the sizes of the SCCs, it can be noticed that around 300 k nodes are indeed isolated (top left point in figure), and all but one SCCs have a size smaller than or equal to 117. The largest SCC of the WoT (bottom right point in figure), is composed by 59,466 primary public keys, and is commonly known as the *strong set*. The strong set is significantly less sparse than the rest of the WoT: its nodes have an average of 27.39 signatures on them. However, we note that the distribution of signatures on each certificate of the strong set (incoming arcs in nodes) is rather skewed: in particular, only ≈18 % of the strong set users have more than 27 signatures. The nodes in the strong set are the ones able to better exploit the structure of the WoT to perform public key authentication since, in principle, there is a path between any two of them. However, a bound on the allowed certifier-certified chains length exists, limiting their length to 5 in both in the original PGP and in the GPG trust model; thus, not all the strong set paths are useful. In fact, the maximum length among all such paths, known as the graph diameter, is 27, and only 38.7 % of the distances between pairs of strong set nodes are smaller or equal to 5. To the end of analyzing the effectiveness of the OpenPGP WoT as a public key certifier, we define the concept of $r$-certified set as follows.

**Definition 1 ($r$-certified set).** *Let $n$ be a node of the WoT, and $r$ be an integer $r \geq 1$. The set of nodes reachable from $n$ via a valid certifier-certified chain of length shorter or equal to $r$ is defined as the $r$-certified set of $n$.*

Since each OpenPGP user acts as both user and certification authority, the size of the $r$-certified set of a given node $n$ is a measure of: (i) the extent of the strong set which is actually useful for $n$ to perform public key authentication when acting as a user, and (ii) the usefulness of $n$ as a certification authority.

Figure 1(b) reports the evaluation of the count of $r$-certified sets for the nodes in the strong set, and $r \in \{2, 3, 4, 5, 6, 10\}$. Considering the case of the PGP and GPG default value $r = 5$, highlighted in blue, it can be seen how only a little more than 10 k nodes have a $r$-certified set exceeding 40 k in size (represented by the values on the bottom right corner), out of ≈60 k, while around 15 k nodes have an $r$-certified set not exceeding 10 k in size (values on the top left corner). Lowering the maximum certifier-certified chain length $r$ yields an effective decrease of the usefulness of the strong set, up to the point where, with $r = 2$, no nodes have a $r$-certified set larger than 10 k elements. By contrast, increasing the chain length boosts the certifying capability of the nodes, at the expense of the need for a longer trust chain to be effectively exploited. For instance, for $r = 10$ the overwhelming majority of the nodes have an $r$-certified set exceeding 50 k elements out of ≈60 k, at the expense of the requirement to trust a rather long certifier-certified chain.

## 3 Threats to the WoT Authentication Capabilities

In this section we provide a description of the scenario and the threat model to OpenPGP public key authentication capabilities.

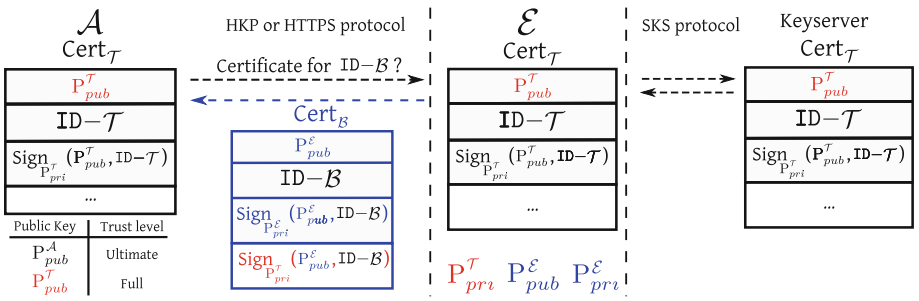


**Fig. 2.** Compromising a fully trusted key scenario. $\mathcal{A}$ queries the keyserver network for $\mathcal{B}$'s certificate, receiving instead one forged by $\mathcal{E}$. Blue portions of the picture are forged by the adversary $\mathcal{E}$, red portions are compromised by $\mathcal{E}$, black portions are genuine (Color figure online)

Assume a user, $\mathcal{A}$, wants to retrieve an OpenPGP certificate containing the public key of whom she wants to communicate with, $\mathcal{B}$. $\mathcal{A}$ will query a keyserver

to retrieve $\mathcal{B}$'s certificate. A malicious keyserver, or an adversary able to act as an active man-in-the-middle, say $\mathcal{E}$, is willing to supply a forged OpenPGP certificate to $\mathcal{A}$, as depicted in Fig. 2. If $\mathcal{B}$'s certificate is considered authentic by $\mathcal{A}$ after running the signature validation procedure, both the confidentiality, and the authenticity of the messages between $\mathcal{A}$ and $\mathcal{B}$ are compromised.

**Compromising a fully trusted key.** Assume $\mathcal{A}$ is trusting a public key $\mathrm{P}^{\mathcal{T}}_{pub}$, with `full` trust level. If the adversary $\mathcal{E}$ is able to compromise $\mathcal{T}$'s keypair, i.e., she is able to obtain $\mathcal{T}$'s private key, she will be able to forge a certificate containing an arbitrary public key $\mathrm{P}^{\mathcal{E}}_{pub}$, generated by herself. She will use this to forge a certificate binding $\mathcal{B}$'s identity `ID`$-\mathcal{B}$ to her public key $\mathrm{P}^{\mathcal{E}}_{pub}$, and perform a self signature on $(\mathrm{P}^{\mathcal{E}}_{pub}, \texttt{ID}-\mathcal{B})$ with $\mathrm{P}^{\mathcal{E}}_{pri}$. Subsequently, $\mathcal{E}$ will compute a signature on $(\mathrm{P}^{\mathcal{E}}_{pub}, \texttt{ID}-\mathcal{B})$ with $\mathcal{T}$'s private key $\mathrm{P}^{\mathcal{T}}_{pri}$, and append it to the forged certificate $\mathrm{Cert}_{\mathcal{B}}$ (depicted in blue in Fig. 2).

Upon receiving the forged certificate, $\mathcal{A}$ will verify both signatures and, trusting the actions of $\mathcal{T}$ fully, she will consider $\mathrm{P}^{\mathcal{E}}_{pub}$ `fully` valid, according to the key authentication mechanism described in Sect. 2.

A noteworthy point is the fact that $\mathcal{E}$ generates the forged certificate for $\mathcal{B}$ from scratch. In case a certificate for $\mathcal{B}$ is already present on the keyservers, $\mathcal{E}$ simply refrains from synchronizing the forged one, effectively presenting to $\mathcal{A}$ a different view on the state of the distributed key storage with respect to the other keyservers. This strategy is viable as the SKS synchronization protocol between keyservers allows each member to choose which certificates should be included in the synchronization, without any enforcement on the inclusion of all of them. In case a certificate for $\mathcal{B}$ is not present, $\mathcal{E}$ can refrain from performing an active man-in-the-middle attack, and simply upload the forged certificate on the global directory, leaving the delivery to $\mathcal{A}$ up to the keyserver network.

**Compromising a key verified by a fully trusted one.** Consider the alternate scenario where $\mathcal{A}$ is trusting $\mathcal{I}$ `fully`, and has $\mathcal{I}$'s certificate in her own keyring, as depicted in Fig. 3(a). $\mathcal{E}$ has compromised $\mathcal{T}$'s keypair and has generated a keypair $\mathrm{P}^{\mathcal{E}}_{pri}, \mathrm{P}^{\mathcal{E}}_{pub}$ which she desires to substitute to the legitimate one for $\mathcal{B}$. $\mathcal{A}$ gets to know that $\mathcal{T}$ is on a certifier-certified chain leading to $\mathcal{B}$ (for instance using an online tool [23]), and fetches $\mathrm{Cert}_{\mathcal{T}}$ from the keyserver with the intent of verifying the signature made with $\mathrm{P}^{\mathcal{T}}_{pri}$ present in $\mathrm{Cert}_{\mathcal{B}}$. The resulting state of $\mathcal{A}$'s keyring is depicted in Fig. 3(a), and the trust level of $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$ is set to `unknown`. Subsequently, $\mathcal{A}$ attempts to compute the validity of $\mathrm{P}^{\mathcal{E}}_{pub}$ contained in $\mathrm{Cert}_{\mathcal{B}}$. To this end $\mathcal{A}$ requires a definite validity level for $\mathrm{P}^{\mathcal{T}}_{pub}$, which is computed to be `full`, through verifying both $\mathcal{I}$'s and $\mathcal{T}$'s signatures on $\mathrm{Cert}_{\mathcal{T}}$ and knowing that $\mathrm{P}^{\mathcal{I}}_{\mathrm{pub}}$ is `fully` trusted. $\mathcal{A}$'s client will ask $\mathcal{A}$ to set a trust value for $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$ in order to proceed with the validation of $\mathrm{P}^{\mathcal{E}}_{\mathrm{pub}}$. Assume $\mathcal{A}$ sets the trust level of $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$, to `full` on the basis that $\mathcal{I}$ and $\mathcal{T}$ cross-signed their certificates, thus providing reasonable evidence for the presence of a mutual trust relationship among them. This assumption is reasonable especially whenever both $\mathcal{T}$ and $\mathcal{I}$ are members of the strong set, and thus highly regarded in terms of reliability in the use of OpenPGP. We note that, in case $\mathcal{A}$ decides against setting the trust

level to `full`, she will be forfeiting the usefulness of this WoT path as a mean for authenticating $\mathcal{B}$'s public key, effectively decreasing the WoT usefulness as a public key authenticator from $\mathcal{A}$'s point of view. Once the trust level for $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$ is set to `full`, $\mathcal{A}$ will set the validity level of $\mathrm{P}^{\mathcal{E}}_{\mathrm{pub}}$ to `full` as it is correctly signed by $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$, thus effectively believing $\mathcal{E}$'s forgery.

By induction on the length of the certifier-certified chains of the aforementioned scenario, $\mathcal{E}$ will be able to forge an arbitrary certificate whenever $\mathcal{A}$ `fully` trusts a key containing in its $(r-1)$-certified set the compromised key $\mathcal{T}$ (recall that $r \geq 1$). Figure 3(b) reports an example of WoT, including the forged $\mathrm{P}^{\mathcal{E}}_{\mathrm{pub}}$, and the compromised $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$, drawn, together with their signatures, in blue and red respectively. The portion of the graph (both keys and signatures) drawn in black is genuine and non compromised. The red-filled nodes are the ones having in the $(r-1)$-certified set the compromised public key $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$ considering the GPG default value $r = 5$. If $\mathcal{A}$ trusts any one of the red-filled nodes, and extends the trust to the ones which have mutual signatures with respect to it, $\mathcal{E}$ will be able to forge a certificate for an arbitrary identity and get it validated by $\mathcal{A}$. In this respect, we note that, if $\mathcal{A}$ requires mutual signatures between a trusted node and one with an `unknown` trust level to extend her trust, it is possible for $\mathcal{E}$ to forge $\mathcal{T}$'s signature on $\mathrm{P}^{\mathcal{I}}_{\mathrm{pub}}$ (the red arc from $\mathrm{P}^{\mathcal{T}}_{\mathrm{pub}}$ to $\mathrm{P}^{\mathcal{I}}_{\mathrm{pub}}$ in Fig. 3(b), should it be missing. We can thus state our attacker model as follows.

**Definition 2 (Threat model).** *Consider the OpenPGP public key authentication scheme based on the PGP/GPG trust model with a certifier-certified chain bound $r = 5$ and the WoT signature verification infrastructure. Assume an adver-*

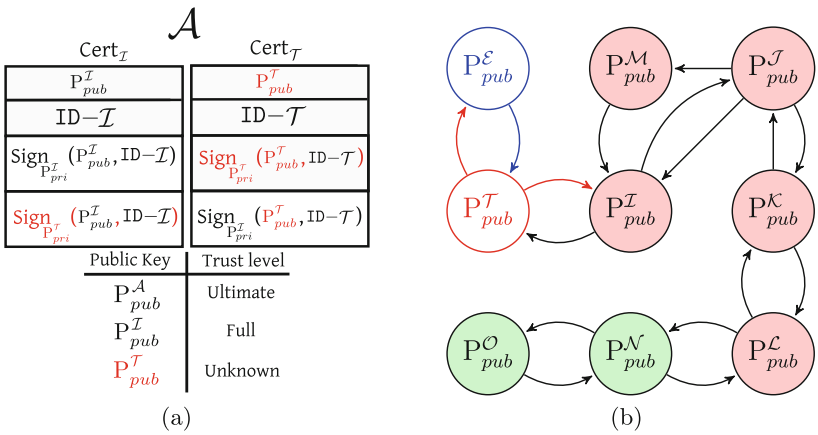


**Fig. 3.** Compromising a key verified by a fully trusted one scenario. Subfigure (a) represents the state of $\mathcal{A}$'s keyring and trust-db after fetching the certificate for the compromised key Cert$_{\mathcal{T}}$. Subfigure (b) depicts a sample WoT, highlighting the extent to which the attack is successful (red filled nodes), and the immune portion (green filled ones). Items drawn in blue are forged by $\mathcal{E}$, the ones in red are compromised by $\mathcal{E}$, the black ones are genuine (Color figure online)

*sary* $\mathcal{E}$ *able to compromise the keypair of a user* $\mathcal{T}$, $\left(\mathrm{P}^{\mathcal{T}}_{pri}, \mathrm{P}^{\mathcal{T}}_{pub}\right)$. $\mathcal{E}$ *is also able to either act as a keyserver or perform active man-in-the-middle between a targeted user* $\mathcal{A}$ *and* $\mathcal{A}$*'s keyserver of choice. Whenever* $\mathcal{A}$ *tries to fetch the certificate of another user* $\mathcal{B}$*, the adversary* $\mathcal{E}$ *is able to forge a valid* $\mathrm{Cert}_{\mathcal{B}}$*, provided: (i) the public key of the compromised keypair,* $\mathrm{P}^{\mathcal{T}}_{pub}$*, is within* $r-1$ *certifier-certified steps from a public key deemed* `fully` *trusted and* `fully` *valid by* $\mathcal{A}$ *and (ii)* $\mathcal{A}$ *extends her trust to the intermediate certifier public keys.*

Consequentially, compromising a well connected public key in the WoT will yield a potentially larger attack surface against the target user $\mathcal{A}$. In particular, compromising nodes in the strong set has the maximum potential for certificate forgery, both due to the strongly connected nature of the strong set (each of its nodes has an average of 27.39 signatures in contrast with the 2.08 average of the global WoT as mentioned in Sect. 2.2), and to the potential willingness of $\mathcal{A}$ to trust them as certifiers. Note that the capability of $\mathcal{E}$ of performing signatures on behalf of $\mathcal{T}$ allows her to connect $\mathrm{P}^{\mathcal{T}}_{pub}$ to the strong set as long as $\mathcal{T}$'s certificate contains a signature from a strong set member. This observation allows to extend the effectiveness of the described threat model to keys which are not members of the strong set, as we will highlight in Sect. 5.

The practical impact of the described attacker model is thus dependent on the robustness of the generated keypairs, and the location of their public keys within the OpenPGP WoT. In the following we will examine the factors allowing $\mathcal{E}$ to compromise a keypair present in the current OpenPGP global keyring, thus meeting the former requirement.

## 4 State of Health of the OpenPGP Global Keyring

In the following, we report, for each one of the asymmetric key cryptosystems employed to perform signatures in OpenPGP, the possible issues on the key material which may lead to compromise a keypair. A similar analysis, tackling keypairs employed in the SSL/TLS and SSH protocols was performed in [11]. The asymmetric key ciphers available for signature purposes in OpenPGP are RSA [24], DSA [19], ElGamal [8] and ECDSA [9]. ECDSA signatures have been recently introduced in OpenPGP and currently account for a negligible portion ($<0.01\,\%$) of the total keypairs, and were thus ignored in our analysis. Table 2 reports, for each examined issue, the number of affected keypairs, and the number of signatures performed by strong set members on their certificate.

Note that the expiration of a compromised key does not prevent an adversary from performing the certificate forgery. In fact the expiration date of a key can be updated by performing a new self-signature which is feasible by the adversary, as he owns the corresponding private key. Moreover, in the man-in-the-middle scenario, we note that the adversary is also able to drop revocation signatures, thus practically voiding their effect. However, willing to take into account the scenario where the adversary uploads a forged certificate for a non-existent recipient, and does not need to perform an active man-in-the-middle attack, we will assume that revoked keys are unusable.

**Table 2.** Report on the state of health of the global OpenPGP keyring, highlighting which issues are present, how many keypairs are affected, and how many of them were signed by strong set members

| Cryptosystem | Keypair issue | Public keys | | |
|---|---|---|---|---|
| | | Affected | Not revoked | Signed by strong set |
| RSA | $\lceil \log_2 N \rceil < 600$ | 8,260 | 89.48 % | 1.79 % |
| | $600 \leq \lceil \log_2 N \rceil \leq 800$ | 11,205 | 91.79 % | 2.03 % |
| | Prime modulus | 1 | 100.00 % | 100.00 % |
| | Common primes | 4 | 100.00 % | 0.00 % |
| DSA | None | – | – | – |
| ElGamal | Use as a signing key | 1,383 | 89.73 % | 2.60 % |
| Any | MD5 Hash function | 155,760 | 89.79 % | 3.78 % |

### 4.1 RSA Cryptosystem

The RSA algorithm is a very popular choice in the OpenPGP ecosystem both for signing and encryption purposes. In particular, it gained popularity in 2009, when the first version of GnuPG using it as default algorithm for the key generation process was released (i.e., Ver. 1.4.10). An RSA public key is constituted of a pair of integers $(e, N)$ where $N$ is obtained as the product of two large, randomly chosen primes $p$ and $q$, having substantially the same bit-size. The RSA private key $d$ is computed as $d = e^{-1} \bmod \varphi(N)$, where Euler's Totient function $\varphi(\cdot)$ can be evaluated only by the keypair owner who knows the factorization of the modulus $N$, since $\varphi(N) = (p-1)(q-1)$. The values of $p, q$ and $\varphi(N)$ should be kept as secret as the value $d$. The security margin of the RSA cryptosystem hinges on the difficulty of factoring $N$, provided nothing else is known on the form of the two factors: $p$ and $q$ [14]. In the following, we present the aspect on which we focused our attention during the analysis of the RSA public keys contained in the WoT.

**Outdated key sizes.** The OpenPGP system has a long history, which means that a good share of the key-pairs were generated in an era when security margins were significantly lower, and were never revoked. Nowadays RSA keys using a modulo $N$ smaller than 768 bits are considered weak, as it was proven the practical feasibility of factoring one in [12], while factoring 512-bit RSA moduli was proven to be feasible in about 10 h of computation time on Amazon EC2 for a cost around 100 USD [10]. Moreover, as shown in [2], the cost of factoring multiple RSA moduli with the same size increases less than linearly with their number. As a consequence, we consider all the 8,260 RSA moduli smaller than 600 bits to be compromised, and the 11,205 ones using a modulus between 600 and 800 bits-long as nearly compromised. As reported in Table 2 only $\approx$10 % of the keys were revoked, and none of them are part of the strong set. However, there are more than 1,000 signatures from the strong set vouching for their

authenticity, thus meeting the requirement of Sect. 3 for 1.79 % keys smaller than 600 bits, and 2.03 % of the ones between 600 and 800 bits.

**Prime modulus.** If the modulus $N$ is prime, $\varphi(N)$ can be trivially computed as $\varphi(N) = N - 1$, thus allowing an adversary to compute the private exponent $d$. We found a single instance of this issue, running a primality test on all the moduli of the RSA public keys available. This compromised key does not belong to the strong set, but was signed by one of its members.

**Common primes.** Given two moduli $n = p \cdot q$ and $n' = p' \cdot q'$, if they share a factor (e.g., $p = p'$), then it is possible to efficiently factor both. In fact, computing the greatest common divisor (using Euclid's algorithm), it is possible to obtain the common factor $p = p'$, and, via trivial division operations, the $q$ and $q'$ too. Looking for common primes is a technique which has been proven successful in discovering flaws in the TLS certificate pool [11]. The causes for the repeated use of the same primes were either a low entropy availability on the generating system, in particular on embedded devices or during the boot process, or simply faulty PRNG implementations. The results of our survey point to substantially different results: only two pairs of public key shared a common prime, thus providing good evidence of the soundness of the prime generators employed in OpenPGP implementations. We note that none of the public keys sharing primes were in the strong set, nor they were signed by one of its members, and thus are not exploitable in the described threat model. We also report the presence of 253 RSA moduli which are not the product of two large primes. They share a large amount of small factors and the self signatures on the corresponding public keys are not valid, nor there are other valid signatures on them.

### 4.2 Digital Signature Algorithm

The examination of keypairs generated to perform signatures with the DSA algorithm were found to be sound, and passed all the tests mandated by the [19] standard (prime generation methodology, check on the order of the generator). The only keys found not to be passing the tests were belonging to corrupted certificates where neither the public key was respecting the constraints of [19], nor the signatures made by others on it were verified correctly.

### 4.3 ElGamal Cryptosystem

The ElGamal cryptosystem is formed by two primitives, signature and encryption, based on the Discrete Logarithm Problem [1,8] over a multiplicative cyclic subgroup of order $q$ of $\mathbb{Z}_p^*$, the set of equivalence classes of signed integers modulo $p$, with both $p, q$ primes and $p \geq 2^{1024}, q \geq 2^{160}$. The ElGamal signature algorithm produces an output longer than DSA, and, for this reason, it has been historically discouraged. Thus, ElGamal keys are assigned two separate algorithm identifiers depending on whether they are only employed to perform encryptions, or if they are also used for signatures. In 2004, Nguyen et al. [20]

discovered a significant flaw in how ElGamal signatures were implemented in GPG. The ElGamal signature requires the generation of an unpredictable random integer $l$ of the same bit-size of $p$. The issue with the GPG implementation of the ElGamal signature is that $l$ is generated as an unpredictable random integer $q$ bits long, with $q \ll p$, for efficiency reasons. This allows to recover the ElGamal private key exploiting the material of a single signature. Since all the ElGamal primary keys must have signed their own certificate, we consider them to be all compromised. It is interesting to note how the WoT still contains $1,241$ unrevoked keys ElGamal public keys allowed to perform signatures, and bearing signatures from members of the strong set.

### 4.4 MD5 Based Signatures

In digital signatures, the choice of the cryptographic hash to be employed to reduce the signed content size to a fixed length is crucial. The MD5 hash algorithm has been proven vulnerable to a specific collision attack, where it is possible for an attacker to choose the prefix of the colliding messages. In particular, Stevens et al. [27] exploited the aforementioned issue to construct a rogue X.509 certificate, splicing out a valid signature, and forging a set of certificate contents colliding with the signed hash. This was possible especially as the X.509 standard allows an arbitrary comment field to be placed as the suffix of the material to be hashed [5]. The same attack can be performed also on a OpenPGP certificate signature, since the RFC4880 [3] allows to add arbitrary subpackets at the end of the data to be signed. Despite, RFC4880 explicitly discourages the use of MD5, signatures made with it are still quite widespread. In fact, our analysis shows that ≈115 k unrevoked keys performed at least a MD5 signature, and 3.78 % of them were signed by a strong set member as reported in Table 2.

## 5 Vulnerability Evaluation

In this section we provide concrete evidence of the extent of applicability of the attack scenario described in Sect. 3 against the public OpenPGP keyring. To this end, all the certificates in the public keyring were parsed and stored in a database, from which the relevant data was extracted and further processed.

In order to avoid GPG-specific parsing behaviors, we adopted a parsing library [15] independently built on RFC4880 OpenPGP format specifications. The library was modified to extract the RSA and DSA/ElGamal signature material for subsequent analysis steps and to make the parsing process more robust towards recoverable errors caused by corrupted entries in packets.

We point out that parsing the dumps available for keyserver bootstrapping, split in chunks of 10 k–50 k certificates, results in a few non recoverable errors, due to corrupted packet metadata, which in turn cause the parser to go out-of-sync with the underlying format. The issue was made worse by the lack of synchronization points, as an OpenPGP certificate bundle has no recognizable trailer to skip to. In fact, when such an error is encountered, the parsing process

cannot proceed further and the rest of the chunk is discarded. Such an issue was encountered in [30], and prevented the parsing of around 50 k certificates. However, we observed that the main implementation of the SKS server [16] reported no errors in importing the certificate dumps, while the GnuPG client fails in parsing them in the same way our parser does. Combining the observations above, we decided to bootstrap our own instance of the SKS keyserver with the provided dump and re-export the dataset as a separate file for each certificate bundle. By doing so, we were able to limit the impact of unrecoverable parsing errors to the single certificate in which they were located. All the certificates in the public keyring dump were correctly parsed.

We imported the information contained in the global keyring into a `MySQL` 5.5.41 database with `MyISAM` backend. The parsing stage lasted approximately 3 h and resulted in 10 GB of database files being stored on disk. The underlying schema was modeled on the content of the OpenPGP packets composing a certificate bundle, with a table for each packet type. With said database in place, the various datasets used for the analysis presented in this paper were gathered as SQL queries and prepared for further processing steps. In particular, each analysis step was coded as a `Python` script with specified dependencies onto the results of previous steps. The whole set of analysis was then orchestrated by a `makefile`, making the whole process fully automatic and therefore easily reproducible. The WoT was exported as a graph, with arcs being trust signatures and keys as nodes, and all connectivity and reach-ability measures and the related graph processing were carried on using `graph-tool` [22], a comprehensive `Python` toolkit for graph analysis. The `fastgcd` tool [11] was used to find common RSA primes in an efficient way. With the aforementioned infrastructure in place, and after performing the analyses which produced the results shown in Table 2 and described in Sect. 4, we proceeded to compute the portion of the WoT affected by the threat model described in Sect. 3.

Figure 4 reports the amount of public keys which allow an adversary to forge an arbitrary certificate, should one of them be trusted by an end-user, and should the end-user trust the certifier-certified chain up to the compromised key. Full grey bars take into account only public keys belonging to the *strong set*, while thatched bars represent keys not in the *strong set*. In particular, Fig. 4(a) reports the amount of certifiers which pose a risk in being trusted, considering as compromised all the short and mis-generated RSA, Elgamal keypairs: in both cases compromising the keypair (i.e., computing the private key) is feasible with limited resources, and further signatures made with the compromised keypairs will be accepted by any client as valid. We note that such a key compromise is performed fully offline, and thus is most likely not alerting the legitimate keypair owner. Figure 4(b) complements the previous information reporting the extent of the risk whenever spliced MD5-based signatures can be reused, and RSA keypairs with a modulus size between 600 and 800 bits are compromised. These two cases will either require a significant amount of computational effort to compromise the keypair, or yield signatures which may be discarded by modern

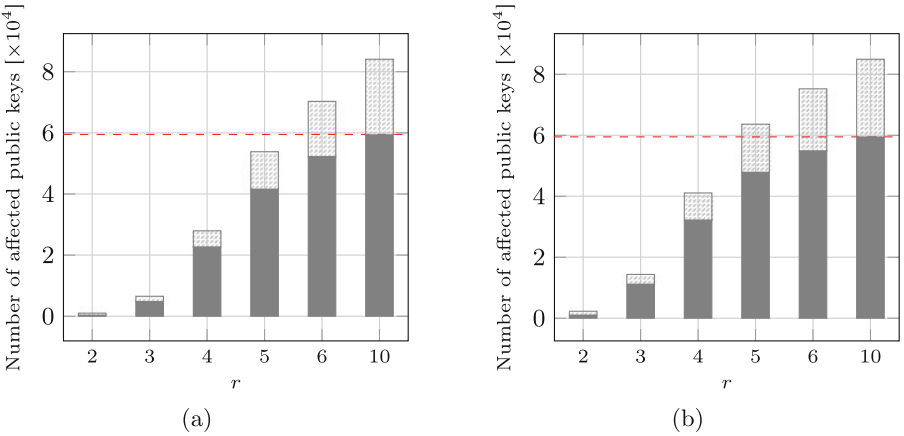


**Fig. 4.** Number of public keys affected by the attack described in Sect. 3 as a function of $r$. Subfigure (a) reports the amount of public keys which allow an adversary to forge a certificate considering short ($\leq$600 bits) RSA moduli, mis-generated RSA keypairs and ElGamal keypairs employed for signature purposes. Subfigure (b) provides the same results for RSA keys with insufficient sized moduli (between 600 and 800 bit in length) and keys which performed at least an MD5 based signature. The red dashed line marks the size of the strong set

clients (GPG stopped taking into account MD5-based signatures starting from June 2014, with GnuPG 2.0.23).

Considering the default bound on the length of certifier-certified trust chains, $r = 5$, the reported results show that around 70 % of the keys in the *strong set* of the WoT are affected by the described threat scenario if all the keypairs breakable with limited resources are effectively broken. This significant amount of keys is cut down to 37 % of the strong set, if the user is willing to cap the length of the maximum chain on the WoT to $r = 4$ (or, equivalently, not to extend her trust), and is further cut down to 8 % in case $r = 3$. However, we note that such a restriction also reduces the effectiveness of the strong set as an authenticator (see Fig. 1(b)), and thus comes at a usability expense. In particular, this link between the effectiveness of the threat and the size of the longest certifier chain $r$ would allow a threat coverage of above 87 % in case the default chain length were one step longer, i.e., $r = 6$. Moreover, in the extreme case where it is desired for the overwhelming majority of the users to be able to use the entire strong set as a certifier, setting $r = 10$ (as shown in Fig. 1(b) and described in Sect. 2), substantially the whole strong set (99.6 % of the keys) are affected by the described threat. The effects of either compromising moderately sized RSA moduli, or splicing MD5 signatures yields similar effects to the aforementioned ones, as reported in Fig. 4(b), although providing a slightly higher coverage of the strong set (83 % of the keys for $r = 5$).

**Mitigation strategies.** We now propose three viable mitigation strategies for the described threat scenario. A first approach to counteract the actions of the adversary, while preserving usefulness of WoT, is to employ redundant chains of authentication to validate a public key. However, this approach requires two, or more, disjoint paths on the WoT to be present in order to effectively thwart the certificate forgery. To evaluate the practical feasibility of such an approach, we computed how many public keys in the strong set are the unique way to reach a portion of it (i.e., removing one of such keys would split a portion of the strong set out). There are 8,804 such nodes (14.8 % of the strong set), thus pointing to a lack of practical viability of such an approach, especially for certification chains of lengths up to 5. An alternate approach is to prevent the attacker from successfully impersonating a keyserver, fetching the certificates from multiple ones. In this case, even if a single malicious keyserver tries to present a "split world"-view to a targeted user, fetching certificates from multiple servers and comparing them will allow to detect the malicious intent. This approach is rather feasible in practical terms, although it may be a concern in rare cases where the user is trusting only a single keyserver to be safe.

Finally, we point out that the nature of the OpenPGP protocol makes the phase out of an outdated hash algorithm trickier than in a common hierarchical PKI infrastructure. In fact, cases such as the current one with MD5, need to be tackled through dropping altogether the support for it. However, such a solution may potentially endanger the soundness of the WoT in case many signatures are removed from it, as it could happen dropping the support for SHA-1 based signatures (which currently constitute 88.9 % of the entire WoT arcs). To the end of preventing a significant alteration in the WoT structure, it is advisable to exploit the nature of OpenPGP certificates, which allow for more than one signature coming from the same issuer to be appended. To allow a graceful phase out of SHA-1, or any other hash which should be phased out, it is thus sufficient to modify the OpenPGP clients so that signatures on the same key material signed with SHA-1 are made, exploiting a better algorithm (e.g., the OpenPGP message standard supports SHA-2-256). As it is already happening in the public web PKI [18,21], this operation should be performed while it is not yet possible to find meaningful collisions on SHA-1. This allows a graceful, and transparent deployment of a stronger hash algorithm, allowing safe disposal of SHA-1 signatures, should the need come anywhen in the future.

## 6 Conclusion

In this paper we proposed a threat scenario to the authentication capability of the OpenPGP WoT, relying on the possibility for an adversary to perform a man in the middle attack, trying to forge a requested certificate. Under the described threat scenario, the adversary needs to compromise a keypair and get the target user to be trusting the compromised keypair certifiers to be able to forge a certificate for an arbitrary identity. Willing to provide an evaluation of the impact of the threat, we performed a survey of the current state of health

of the OpenPGP WoT, both in structural terms, and concerning the security of the individual keypairs. The results show how, in a context where public key authentication can be performed indirectly exploiting verification made by trusted users, even a limited amount of broken or outdated keys can have a dramatic impact on the security of the whole system. Another relevant aspect that emerged from our analysis is the impact of the recent decision to reject MD5-based signatures by some OpenPGP clients, namely GnuPG. A unilateral decision to disable MD5 for public key authentication effectively removed more than 432 k signatures for the WoT, without any preemptive measures being taken to compensate for the loss. For this reasons we suggest a strategy to perform a graceful phase-out of SHA-1, which is currently used in the vast majority of OpenPGP signatures, through a signature refreshment strategy amenable to automation, performed by the clients, using a more modern algorithm.

## References


1. Barenghi, A., Beretta, M., Di Federico, A., Pelosi, G.: Snake: an end-to-end encrypted online social network. In: Bourgeois, J., Magoulès, F. (eds.) 2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, 20–22 August 2014. IEEE (2014)
2. Bernstein, D.J., Lange, T.: Batch NFS. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 38–58. Springer, Heidelberg (2014)
3. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP Message Format. RFC 4880, updated by RFC 5581 (2007)
4. Callas, J., Donnerhacke, L., Finney, H., Thayer, R.: OpenPGP Message Format. Internet RFC 2440 (1998)
5. Chokhani, S., Ford, W.: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 2527, obsoleted by RFC 3647 (1999)
6. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, updated by RFCs 5746, 5878, 6176 (2008)
7. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Crypt. **2**(2), 107–125 (1992)
8. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
9. Hall, T.A., Keller, S.S.: The FIPS 186–4 Elliptic Curve Digital Signature Algorithm Validation System. NIST (2014). http://csrc.nist.gov/groups/STM/cavp/documents/dss2/ecdsa2vs.pdf
10. Heininger, N.: Factoring as a Service. CRYPTO 2013 Rump session (2013). https://www.cis.upenn.edu/nadiah/projects/faas/
11. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: detection of widespread weak keys in network devices. In: Kohno, T. (ed.) Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, 8–10 August 2012, pp. 205–220. USENIX Association (2012)
12. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA modulus. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 333–350. Springer, Heidelberg (2010)

13. Koch, W.: The GNU Privacy Guard (2015). https://www.gnupg.org
14. Lenstra, A.K.: Integer factoring. Des. Codes Crypt. **19**(2/3), 101–128 (2000)
15. McGee, D.: PGP Packet Parser Library (2015). https://github.com/toofishes/python-pgpdump
16. Minsky, Y., Clizbe, J., Fiskerstrand, K.: Synchronizing Key Server (SKS) Software Package (2015). https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Home
17. Minsky, Y., Trachtenberg, A., Zippel, R.: Set reconciliation with nearly optimal communication complexity. IEEE Trans. Inf. Theory **49**(9), 2213–2218 (2003)
18. Mozilla Security Engineering Team: Phasing Out Certificates with SHA-1 based Signature Algorithms (2014). https://blog.mozilla.org/security/2014/09/23/phasing-out-certificates-with-sha-1-based-signature-algorithms/
19. National Institute of Standards and Technology: Digital Signature Standard (DSS). Federal Information Processing Standards Publication (FIPS) 186-4. U.S. Department of Commerce (2013). http://dx.doi.org/10.6028/NIST.FIPS.186-4
20. Nguyên, P.Q.: Can we trust cryptographic software? Cryptographic flaws in GNU privacy guard v1.2.3. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 555–570. Springer, Heidelberg (2004)
21. Palmer, C., Sleevi, R.: Gradually Sunsetting SHA-1 (2014). http://blog.chromium.org/2014/09/gradually-sunsetting-sha-1.html
22. Peixoto, T.P.: The Graph-tool Python Library (2014). http://figshare.com/articles/graph_tool/1164194
23. Penning, H.P.: PGP Pathfinder and Key Statistics (2015). http://pgp.cs.uu.nl/
24. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
25. Shaw, D.: OpenPGP HTTP Keyserver Protocol (HKP). Expired Internet-Draft (2013). http://tools.ietf.org/html/draft-shaw-openpgp-hkp-00
26. Somogyi, S.: End-to-End Chrome Browser Extension (2015). https://github.com/google/end-to-end/wiki
27. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)
28. Symantec Corp.: Symantec Encryption (PGP) Docs. Article Tech202483 (2015)
29. The Free Software Foundation: Email Self-Defense Campaign (2015). https://emailselfdefense.fsf.org/
30. Ulrich, A., Holz, R., Hauck, P., Carle, G.: Investigating the OpenPGP web of trust. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 489–507. Springer, Heidelberg (2011)
31. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: improving SSH-style host authentication with multi-path probing. In: Isaacs, R., Zhou, Y. (eds.) 2008 USENIX Annual Technical Conference, Boston, MA, USA, 22–27 June 2008, pp. 321–334. USENIX Association (2008)
32. Zhu, Y., et al.: End-to-End for Yahoo! Mail (2015). https://github.com/yahoo/end-to-end