Ghost trace on the wire? Using key evidence for informed decisions

Diana A. Vasile, Martin Kleppmann,
Daniel R. Thomas, and Alastair R. Beresford

Department of Computer Science and Technology
University of Cambridge, UK
{FirstName.LastName}@cl.cam.ac.uk

Abstract. Modern smartphone messaging apps now use end-to-end encryption to provide authenticity, integrity and confidentiality. Consequently, the preferred strategy for wiretapping such apps is to insert a ghost user by compromising the platform's public key infrastructure. The use of warning messages alone is not a good defence against a ghost user attack since users change smartphones, and therefore keys, regularly, leading to a multitude of warning messages which are overwhelmingly false positives. Consequently, these false positives discourage users from viewing warning messages as evidence of a ghost user attack. To address this problem, we propose collecting evidence from a variety of sources, including direct communication between smartphones over local networks and CONIKS, to reduce the number of false positives and increase confidence in key validity. When there is enough confidence to suggest a ghost user attack has taken place, we can then supply the user with evidence to help them make a more informed decision.

Keywords: trust establishment · public key evidence · end-to-end encryption · secure messaging · security usability · informed consent

1 Introduction

Modern messaging apps with end-to-end security, such as Signal, WhatsApp and iMessage, are now regularly used by over 1 billion people [3,21]. These apps use public-key cryptography to encrypt messages on the sending device such that it can only be decrypted by recipient devices; any server infrastructure used to store and forward such messages cannot read or modify message contents. However, modern messaging apps have a common weak point in their security model: knowing whether a user has the right public keys for their communication partners. In the case of Signal, WhatsApp and iMessage, the discovery of public keys is performed using a *key server* or *key directory* operated by the app provider: when a device generates a keypair it sends its public key to the key server, and when a user wishes to communicate with a contact, the app looks up the public keys for the contact's devices using their phone number or email address.

Key servers are an example of a *Public Key Infrastructure* (PKI). In this paper we compare them with *Certificate Authorities* (CAs), a more traditional form of PKI used e.g. in the context of TLS, to provide a mapping between DNS domain names and the public keys of TLS servers. Both key servers and CAs link human-readable names (email addresses, phone numbers, domain names) with public keys, and both require an element of trust in the PKI provider. Both forms of PKI remove the need for users to manually manage keys – a task that has repeatedly been shown to be challenging for users [7,17,18,19,22] – and effectively automate the security decision of whether to trust a particular public key.

Such automation of security decisions and key checking undoubtedly helps, and is one reason why the current generation of messaging apps with end-to-end security have seen widespread adoption, whereas PGP did not. Unfortunately, the security and privacy requirements of a user are not universal. For example, a human rights activist using a messaging app in a country with a repressive regime has a different threat model from a typical user communicating via social media in the United Kingdom. The effectiveness of specific attacks depends on how users behave, what they are trying to protect, and the degree to which they understand the security features of an app. In short, requirements and context matter. Consequently, an app cannot automate all security decisions; some decisions are necessarily deferred to the user.

Deferring security decisions to the user is hard to do safely because we cannot expect users to be cryptographers and security engineers: they do not have the knowledge to reason about and deal with security decisions appropriately.

Moreover, the PKI is a significant weak link in the end-to-end encryption ecosystem as deployed by messaging apps today: a compromised PKI allows an attacker to break end-to-end encryption by adding another "end" (sometimes called a *ghost user*) to the set of public keys that a user has registered with the PKI, allowing the ghost user to read all messages in a conversation. Neither user may be aware that this has happened. This approach has been proposed by GCHQ as the preferred way to provide law enforcement with access to end-to-end encrypted communication without inserting explicit back-doors into the actual encryption protocols [13].

To protect against this kind of attack, WhatsApp and Signal offer users the option of verifying public keys (a.k.a. *safety numbers*) for their contacts. Such verification can be performed manually by comparing long alphanumeric strings, or by scanning QR codes on each others' phones. However, such manual checks take a significant amount of time, and – in the case of the QR code at least – require both users to meet in person. Anecdotal evidence suggests that they are not used much in practice. Even if safety numbers have been checked, they have a tendency to change fairly often, due to a user replacing their device or reinstalling the app; since most changes to safety numbers are due to such benign causes, users are conditioned to treat a change in safety number as harmless, even though this is exactly what a ghost user attack would look like. Apple's iMessage does not even offer the option of checking keys manually.

Various approaches have been proposed to detect ghost user attacks and compromised PKIs, which we summarise in Section 2. However, these approaches are not perfect, and we discuss false positives and false negatives that can arise in the detection of such attacks in Section 3. We detail failure modes and user expectations in Section 4. Finally, in Section 5 we describe how incentivising users, understanding user context, and collecting and evaluating evidence for key changes could lead to better system designs.

2 Detecting ghost users and key mismatches

Older end-to-end encrypted communication tools such as PGP rely exclusively on manual key fingerprint checking and explicit key signing to build a graph of trusted keys (the *web of trust*). However, PGP has failed to gain traction partly because of the difficulty users faced in performing these operations [22].

PKIs (certificate authorities and key servers) replace these manual processes with a centralised authority; the challenge is then to ensure that this authority remains honest. To this end, Certificate Transparency [12] has introduced the use of public append-only logs. Here, certificate authorities are required to submit a record of all issued certificates to the log infrastructure, providing an externally auditable proof of their existence. While Certificate Transparency does not prevent certificates from being incorrectly issued, it makes it more likely that such behaviour is detected, and thus discourages it. For example, in September 2015 Symantec was found to have mis-issued thousands of certificates; this event was discovered through Certificate Transparency logs, and had consequences for Google Chrome's handling of Symantec-issued certificates [20].

CONIKS [15] and Key Transparency [11] apply the Certificate Transparency approach to key servers. They maintain an append-only log of all public keys submitted to the key server, together with the human-readable username (e.g. email address or phone number) associated with each key. When a client wishes to look up the public keys for a given username, the key server provides a cryptographic proof that its response is consistent with the audit log. CONIKS is able to achieve this while preserving the privacy of phone numbers and email addresses in the log.

In parallel work we propose an alternative solution: to use gossip protocols on local area networks to privately and automatically check the bindings between human-readable names and public keys between contacts without requiring any user intervention. Such a protocol can operate on a local Wi-Fi network with Multicast DNS [4], which enables two devices, such as smartphones, to discover each other and compare their links between human-readable names and public keys directly. If the link between names and keys are the same, this provides additional assurance that there is no ghost user; if the links are not the same, then the key server may have been compromised. Our protocol can also use Private Set Intersection [6] to allow two devices that meet on a local Wi-Fi network to check the links from names to keys of any contacts they both have in their respective address books, without revealing any contact details for individuals they do not

share. The advantage of using gossiping over systems such as CONIKS is that different app PKIs do not need to collaborate and there is no need for the service provider to publish their key directory. Even if contacts only meet on the same Wi-Fi network once a month, this will still provide much better oversight than the current manual verification approach. Some contacts will meet much more frequently (colleagues or geographically close family or friends).

However, while these solutions automate the process of auditing the behaviour of PKIs, there are several cases that need to be resolved by user input. For instance, in CONIKS, users have to reason about errors relating to inconsistent key server summaries, which may be the result of benign clock synchronisation problems between the client and the server, or an actual malicious server publishing different views of its directory [14]. Furthermore, we expect key changes to be the most common errors that CONIKS users and automated gossiping users might have to reason about. While most key changes are caused by the user adding a new device or re-installing the app, it can also be the key server acting maliciously and modifying the user's set of keys.

Section 4 reasons about user perspectives and Section 5.3 explores different levels of evidence that can be used to reduce the number of false positive errors that often overwhelm the user and cause them to ignore security warnings.

3 The imprecision of key change errors

In an ideal world, a communication system would report an error (and refuse further communication) only in those situations where a genuine PKI compromise (e.g. a ghost user attack) by an adversary is taking place, and never otherwise. However, in practice, all PKIs have *false positives* (an error is reported even though no wiretap is taking place) and *false negatives* (a wiretap succeeds without the user being alerted).

User-facing errors related to public keys have been most extensively studied in the context of TLS. It has been shown that most users ignore TLS certificate errors [1,2]; this behaviour is rational, since almost all such errors are false positives (for example, the certificate presented by the server has expired, does not match the given domain name, or is not signed by a certificate authority trusted by the client). Most TLS errors are thus due to client or server misconfiguration rather than a true man-in-the-middle attack, and hence they can be safely ignored [10].

Our goal in designing communication systems should be to reduce the probabilities of both false positives and false negatives as far as possible. False negatives must be minimised because every false negative represents a failure to guarantee the system's required security properties. False positives must be minimised because unnecessary errors amount to "crying wolf", reducing users' confidence in the system, and making it more likely that users ignore true positive errors in the future [2,5,10].

3.1 Reducing false negatives

With traditional PKIs, if the adversary is able to compromise the PKI (for example by stealing a certificate authority's secret key for signing certificates), it can perform interception without being detected by clients. Framed this way, we can see that CONIKS, Certificate Transparency and Key Transparency are mechanisms for reducing false negatives: by employing a verifiable append-only log, they make it very difficult for a key server or CA to return different public keys to different clients depending on who is asking, without being detected.

Transparency logs address one particular weakness through which false negatives can occur, namely the compromise of a CA or key server, but they do not fully eliminate false negatives: for example, an adversary may be able to block communication between clients and the transparency log (thus preventing the log integrity from being checked), it may block OCSP requests (thus allowing an attacker to continue using a stolen private key with a revoked certificate), or it may tamper with NTP server responses (thus setting the client clock incorrectly). Adversary control over client clocks may allow interference with time-based actions, such as preventing a daily consistency check of keys from occurring by making the client believe that less than 24 hours have elapsed since the last check. Further work is needed to address these additional sources of false negatives.

In Section 2 we described the use of gossip protocols to exchange public keys directly between clients on a local network; this approach also reduces false negatives, since it detects when two users have different public keys for the same contact, which may indicate that a wiretap key has been inserted for that contact. However, this approach also carries the risk of increasing false positives: if a client does not correctly follow the protocol, either by accident or by malice, it may report incorrect public keys for a contact and thus trigger warnings, even though no actual attack is taking place.

3.2 Reducing false positives

A false positive occurs when a user is shown a warning or error that might indicate an attack, when in fact no attack is taking place. For example, the WhatsApp and Signal messaging apps show the user a warning message whenever the public key for one of their contacts changes (see Figure 1), even though in the vast majority of cases the cause of this key change is benign (e.g. the user installed the app on a new device, or deleted and re-installed it on an existing device). The explanatory message rightly downplays the significance of this key change, so users are conditioned to ignore these warnings.

The most straightforward way of reducing false positives is to report fewer warnings and errors. For example, iMessage has no manual key verification feature and no transparency log, i.e. key server responses are assumed to be fully trusted, and it does not report key changes to users. This means that there are no key mismatch errors, reducing false positives; of course, this approach also increases false negatives, since a compromise of the key server remains undetected.

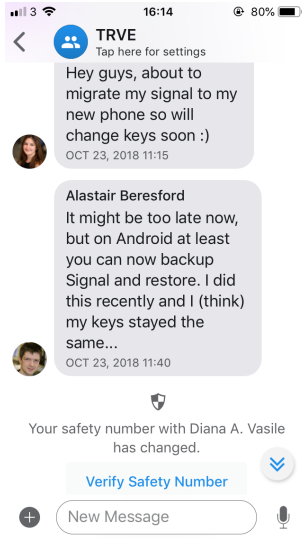

Fig. 1. Warning message displayed by Signal when a user's key changes.

Reducing false positives without increasing false negatives is also possible. For example, in the context of TLS, many false positive certificate errors are due to server misconfiguration (e.g. serving a certificate for the wrong hostname, or forgetting to renew a certificate before its expiry), client misconfiguration (e.g. client clock is set incorrectly, making the client believe it is outside of the certificate's validity period), or 'benign' network interference (e.g. antivirus software or captive portals on public Wi-Fi networks) [1].

Various measures can be taken to reduce these false positives: for example, about one third of HTTPS errors on Windows are due to misconfigured client clocks [1]; to reduce this source of errors, the use of authenticated time services has been proposed [8]. Key servers, such as those used by Signal and WhatsApp, are less susceptible to clock errors than certificate-based PKIs, since the key server's response is assumed to be valid immediately, and usually does not include an explicit validity period.[1] Also, smartphones typically sync to cellular network time (which is derived from GPS) and so (mostly) avoid clock synchronisation issues. Other devices may use NTP, but in all cases malicious action may cause problems.

3.3 Authenticating key changes

To reduce false positive warnings about key changes, systems could offer an authenticated key change: a user's old private key can be used to sign a statement

[1] Communication between client and key server occurs over TLS, with the key server usually authenticated with a certificate, so there is still some residual dependence on clocks in this case.

saying which public keys the user will be using henceforth, and other users will not be shown a warning if the key change is correctly authenticated in this way.[2] However, this approach is only possible if the old key is still accessible; if the user bought a new device because their old device was lost or irrecoverably destroyed, the user may need to recover their account without being able to access their old key.

For high-risk users it may be appropriate to disallow any unauthenticated key changes, requiring the user to back up their keys, and accepting that the user account would be irrecoverable if all keys are lost. However, for most users such a strict policy on lost keys is likely to be unacceptable: estimates indicate that approximately 20% of all Bitcoin ever mined, valued at billions of dollars, have been irretrievably lost due to the loss of the corresponding wallet private keys [16]. If so many keys with an immediate financial value are lost, we expect that keys for communication apps (with no direct financial value) are even more likely to be lost. From this statistic we conclude that for most users, a key change without signature from a previous key will sometimes be necessary.

Even if the system were restricted to only allow authenticated key changes, false negatives are possible: a malicious key server could return the attacker's public key the first time a key is requested for a user (for which there is no authentication, since there is no prior key), or withhold a key change to revoke a key that has been compromised. Detecting those attacks will still require either manual checking of public keys, a transparency log, or a gossip approach as discussed in Section 3.1.

4 Message visibility and key changes

There are three main operations that will alter a user's set of keys:

Addition: a key is added to the set of current valid keys for a user;
Revocation: a key is revoked from the set of current valid keys for a user;
Replacement: an old key is replaced with a new key (revocation + addition).

Each one of these operations triggers warnings or errors in secure messaging apps, such as Signal and WhatsApp. For instance, a key replacement often happens when a user gets a new device or re-installs the app on their current device. While this is not an attack, it looks very similar to key server misbehaviour by changing the user's set of keys without authorisation because it triggers a key mismatch.

Most users, however, do not directly care about or understand key operations. Instead they care about the authenticity and confidentiality of the messages they send and receive.

[2] To avoid key changes, Signal allows the user to save a backup of the secret key on the old device, and restore it on the new device. However, this process is poorly documented and difficult to perform correctly. It requires the use of third-party apps to transfer and set up the backup before installing Signal on the new device.

For example, *read receipt* is a concept that serves the purpose of telling senders whether the intended user received their message and, if enabled, whether they read it or not. This could be further augmented by ensuring a full list of message readers is provided, for instance, if the recipient has two devices registered with the system, user-provided aliases may be used to identify to the sender that both the phone and the tablet received the message.

Message deletion can happen either at one client (deleting the messages only on one user's device) or, with application support, on all clients (deleting the messages from every communication participant). This is another important piece of information users would care about, but it may be difficult to explain the difference between the two to the average user until they experience someone deleting a message they had received.

Aside from the issues discussed above, end-to-end encrypted group messaging poses further difficulty. For instance, it is non-trivial to explain to users what happens to their messages when a new user joins their group, and different apps have taken different approaches. For instance, users may expect that a new joiner in the group would have access to the historical messages from the inception of the group, but this is not always the case.

5 Future directions

We now explore possible solutions which ensure systems only display warning messages where necessary and that the content of such messages help users make informed decisions.

5.1 Incentivising users

Anecdotal evidence suggests that a user treats a new app as a toy, exploring its features more in the first few sessions than later. This could be leveraged for the purpose of achieving better security.

For instance, an app can provide proactive manual checking as a feature, which can make it easier to detect if something has gone wrong. Users can be awarded points, levels, or badges to encourage such checks. The relative confidence in the correctness of the keys for a particular user can be displayed, e.g. with Bronze, Silver, Gold medals or a red-to-green scale with levels automatically assigned based on observed cryptographic evidence, as further explained in Section 5.3.

Such gamification is employed by many apps to incentivise user behaviour. However, gamification might incentivise users to spend more time on security than is rational given the benefits to them, so care is required to ensure the design is ethical.

5.2 User context

Context matters and users have different threat models. Thus, allowing users to customise the display of key change notifications is important. This enables

the system to vary what it displays based on the user's context, so a user who suspects they are of interest to an intelligence agency can be shown warnings which probably include false positives, while a more typical user would only be shown warnings more likely to be true positives. Alternatively, an initial setup phase to infer a user's threat model could then be used to customise defaults intelligently.

5.3 Evidence for key validity

To determine what notifications, if any, the messaging software should show to a user, we propose collecting evidence on the validity of a contact's keys. Rather than unconditionally trusting a signature from a PKI or a response from a key server, we can combine several types of evidence with prior expectations on the probability of a compromise to estimate the likelihood of a false-positive or a true-positive. This information can then, in turn, help users make an informed security decision.

To establish a binding between people (identified by a human-readable name, such as an email address or phone number) and the public keys of their devices, we propose collecting evidence in the following categories:

Trusted The key for the user's currently in-use device is completely trusted.

Signed trusted A key can be signed by this device's key as being one of the user's keys on another device because, at some point in the past, they authorised it and optionally performed some mutual verification (QR-codes, local network gossiping, password authenticated key exchange [9]).

PKI signature This key-name mapping was supplied and signed by the PKI. So the PKI believed it to be correct at that time based on sending a text message/email or other verification process; this can also happen if the PKI is manipulating keys because of a warrant or a rogue employee.

Auditable PKI signature The signature on the key-name mapping by the PKI can be audited as it was published using CONIKS or Certificate Transparency. Therefore, the key-name mapping can be checked by the owner of the name and misbehaviour by the PKI will be detected.

Manually verified The user has verified the key-name mapping using QR-codes or some other out of band mechanism (confidence provided will vary with mechanism). This provides a strong guarantee that at a particular time the key-name mapping was correct.

Other communication channel The user exchanged key material with the contact via a partially trusted communication channel, such as email, SMS, phone call, or another messaging app. Although this channel may not provide strong confidentiality guarantees, it may be difficult for the adversary to tamper with this communication.

Signed by a key for the same name Another key for the same name has signed this key-name mapping and it has its own evidence as to its validity. The evidence this provides depends recursively on the evidence for the validity of the signing key. In the context of the device for the key which

did the signing this would be **signed trusted** but the evidence provided is lesser when this signature is from a key for a contact rather than your own device.

Gossiped directly The key-name mapping was directly gossiped with the device for that mapping and so was on the same network as the user's device at that time. Being able to display the location and time to the user provides greater confidence that either the key-name mapping is correct or the network they were on was compromised.

Gossiped indirectly The key-name mapping was gossiped via a mutual contact. Therefore the PKI has supplied the same key-name mapping to other contacts making detection of misbehaviour more likely.

Freshness from gossiping In the gossip protocol, devices use mDNS to publish advertisements proving that the device has control of the key at a particular point in time. These advertisements can be stored and provide evidence of the freshness of the key in a particular location. Since timestamps are chosen by the device producing the advertisement, the receiving device must verify it is within sensible bounds (advertisements time out).

Indirect freshness Mutual contacts can pass on advertisements they have observed for a key and so provide evidence of freshness during gossiping. Since timestamps cannot be verified by the device, this evidence is not absolute, but tampering requires collusion by the contact.

Revoked by self A message indicating that a key has been revoked, signed by the key being revoked, shows that someone in control of the key has tried to revoke it (not necessarily deliberately).

Revoked by a signed trusted key A key for the same name as the key being revoked has signed the revocation message and it was signed as trusted by the key being revoked. This means that a key on another device belonging to the user has been used to revoke the key (but it could have been compromised or this might be accidental).

Revoked by a key for the same name A key for the same name as the key being revoked has signed the revocation message and has some evidence as to its validity for that name.

Mutual revocation Two (or more) keys have mutually revoked each other (but not themselves) indicating that one of the keys has been compromised and which key under the control of the legitimate user is disputed. In this scenario relevant evidence for affected keys should be discarded and evidence built up again.

Expired The key-name mapping has a validity period and the device's local time is outside of this period. This is probably accidental, as in the case of TLS certificates.

Most evidence is associated with a timestamp, such as the time manual verification occurred or the freshness timestamp in the gossip key-check message, and in most cases older evidence gives us less confidence than newer evidence. For example, we might have manually verified the key two years ago, which gave the user high confidence at the time, but the contact might have changed that device

in the meantime. Hence, time-based discounting is required for the confidence derived from pieces of evidence. An exponential decay with a two-year half-life might be suitable as a conservative estimate of device replacement frequency. However, a contact maintaining the same key for an extended period of time gives greater confidence as a wiretap would have to have been maintained across the whole period to avoid detection. A log of gossip messages or timestamps when signed-session-keys for messaging were established could provide evidence of an extended period of continuous usage.

By leveraging such cryptographic evidence we can estimate confidence in key authenticity. This measurement can be used to decide whether to trigger a user warning, while also taking into account the user's notification settings as set up in Section 5.2.

6 Conclusions

Users of end-to-end encrypted messaging are not interested in key management: they are interested in who can read their messages and the authenticity and confidentiality of the messages they receive. The insertion of ghost users into end-to-end encrypted chats by unauthorised parties causes the same warning as a routine key change. Most of these warnings are false positives. After analysing existing approaches, we suggested how warning messages can be shown only when it is useful. We proposed deriving the prior probability of true positives from the user's context and combining it with evidence of key validity. In this way we aim to equip the user with the capacity to make informed decisions. We propose collecting this evidence from different sources, such as key directories, gossiping, or manual verification, and supplement this by incentivising the user to generate further evidence. Difficult trade-offs arise, as some measures to reduce false negatives may also increase false positives, or vice versa.

References

1. Acer, M.E., Stark, E., Felt, A.P., Fahl, S., Bhargava, R., Dev, B., Braithwaite, M., Sleevi, R., Tabriz, P.: Where the wild warnings are: Root causes of Chrome HTTPS certificate errors. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. pp. 1407–1420. CCS '17, ACM (2017). https://doi.org/10.1145/3133956.3134007
2. Akhawe, D., Amann, B., Vallentin, M., Sommer, R.: Here's my cert, so trust me, maybe?: Understanding TLS errors on the Web. In: Proceedings of the 22nd International Conference on World Wide Web. pp. 59–70. WWW '13, ACM (2013). https://doi.org/10.1145/2488388.2488395
3. Apple Inc.: Apple reports first quarter results (Feb 2018), https://www.apple.com/newsroom/2018/02/apple-reports-first-quarter-results, https://perma.cc/M6WV-Q4HK
4. Cheshire, S., Krochmal, M.: Multicast DNS. RFC 6762 (2013)
5. Clark, J., van Oorschot, P.C.: SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. IEEE Symposium on Security and Privacy pp. 511–525 (2013). https://doi.org/10.1109/SP.2013.41

6. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear computational and bandwidth complexity. IACR Cryptology ePrint Archive **2009/491** (2009)
7. Garfinkel, S.L., Miller, R.C.: Johnny 2: A user test of Key Continuity Management with S/MIME and Outlook Express. In: Proceedings of the Symposium on Usable Privacy and Security. pp. 13–24. SOUPS '05, ACM (2005). https://doi.org/10.1145/1073001.1073003
8. Google, Inc.: Roughtime (2016), https://roughtime.googlesource.com/roughtime
9. Hao, F., Ryan, P.Y.: Password authenticated key exchange by juggling. Security Protocols **LNCS 6615**, 159–171 (2011)
10. Herley, C.: So long, and no thanks for the externalities: The rational rejection of security advice by users. In: Proceedings of the New Security Paradigms Workshop. pp. 133–144. NSPW, ACM (2009). https://doi.org/10.1145/1719030.1719050
11. Hurst, R., Belvin, G.: Security through transparency (Jan 2017), https://security.googleblog.com/2017/01/security-through-transparency.html
12. Laurie, B.: Certificate transparency. ACM Queue **12**(8), 10 (2014). https://doi.org/10.1145/2668152.2668154
13. Levy, I., Robinson, C.: Principles for a more informed exceptional access debate (Nov 2018), https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate, https://perma.cc/7RJK-FM32
14. Melara, M.: Why making Johnny's key management transparent is so challenging (Mar 2016), https://freedom-to-tinker.com/2016/03/31/why-making-johnnys-key-management-transparent-is-so-challenging/, https://perma.cc/RX2S-MZQH
15. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing key transparency to end users. In: USENIX Security Symposium. pp. 383–398 (2015)
16. Roberts, J.J., Rapp, N.: Nearly 4 million Bitcoins lost forever, new study says (Nov 2017), http://fortune.com/2017/11/25/lost-bitcoins/
17. Ruoti, S., Andersen, J., Zappala, D., Seamons, K.: Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. arXiv (2015), http://arxiv.org/abs/1510.08555
18. Ruoti, S., Kim, N., Burgon, B., van der Horst, T., Seamons, K.: Confused Johnny: When automatic encryption leads to confusion and mistakes. In: Proceedings of the Ninth Symposium on Usable Privacy and Security. pp. 5:1–5:12. SOUPS '13, ACM (2013). https://doi.org/10.1145/2501604.2501609
19. Sheng, S., Broderick, L., Hyland, J.J., Koranda, C.A.: Why Johnny still can't encrypt: Evaluating the usability of email encryption software. In: Symposium On Usable Privacy and Security (SOUPS). pp. 3–4 (2006)
20. Sleevi, R.: Sustaining digital certificate security (Oct 2015), https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html
21. WhatsApp Inc.: Connecting one billion users every day (Jul 2017), https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day, https://perma.cc/8WZJ-Y5UT
22. Whitten, A., Tygar, J.D.: Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In: USENIX Security Symposium. pp. 169–184 (1999)