An Advanced Introduction to GnuPG

Neal H. Walfield

August 18, 2017

Contents

Part I

Main Matter

Chapter 1

MUA Integration

This chapter contains guidelines on integrating GnuPG into a mail user agent (MUA). Other good sources of information on this topic are existing MUAs, in particular, KMail and Enigmail, which probably have the best GnuPG integration. This is not to say that our recommendations or what KMail and Enigmail implement are optimal. Far from it. A common criticism of GnuPG is how difficult it is to use. We acknowledge these criticisms, and we particularly welcome help in this area. Nevertheless, we suspect that some of the user interactions cannot be significantly simplified without compromising the security of the system, which has traditionally been designed to protect the user from an active adversary.

Most people do not have active adversaries. This is particularly true in democratic countries. People who live in these places primarily turn to a technology like GnuPG to protect their privacy, thwart phishing excursions, or fight mass surveillance. These users do not have the same security requirements as journalists, activists, or lawyers operating in regimes where civil rights are not respected, and a single unencrypted message can result in jail time, or worse.

Given these different classes of users, it is entirely reasonable to simplify some of the proposed interaction patterns for those who are only interested in protecting their privacy by using encryption opportunistically [1]. This is precisely what the Autocrypt project is trying to accomplish. Their hope is that trading protection from active adversaries for increased ease of use will result in greater adoption of encrypted email by people looking to protect their privacy, and fight mass surveillance, but don't want to be bothered with security issues [2].

Simplifying user interactions needs to be done carefully. People currently associate GnuPG and related tools as providing high levels of protection, and may assume that because these new interfaces use GnuPG that they provide the same level of protection. As such, we recommend the MUA make clear to users what level of protection the interface can offer. This could be done using a warning, but text that resembles an EULA is unlikely to be read [3]. Another approach to this problem is to ask the user to choose a profile that best matches their needs (i.e., their threat model), and then adjust defaults accordingly. This is the approach that the Tor Browser Bundle takes. This has the added benefit of causing users to think about risk assessment. The MUA need not support all of the profiles that it shows. Then, if the MUA does not support the user's threat model, the user should be warned.

In the GnuPG context, three profiles appear to be called for:

- **Very Strong Security**: Some users turn to GnuPG, because they fear targeted attacks from a nation state adversary including rubber-hose cryptoanalysis (i.e., the use of torture to recover passwords). These users should almost certainly use a security token, which the MUA should help them configure, HTML should be disabled, and all operations that could leak sensitive information should require explicit confirmation. The MUA should also help these users implement forward secrecy (by regularly rotating subkeys), provide a mechanism to automatically purge old emails, and disable indexing encrypted emails.

- **Strong Security**: Some users need protection from less sophisticated adversaries. For instance, lawyers worry that their communication with their clients may be spied on by criminal groups or corrupt government organizations. Although these users rely on encryption to protect sensitive communication, they also send and receive a lot of unencrypted email, and they don't want to be overly inconvenienced when processing those messages. Consequently, these users should have to confirm sending unencrypted mails when keys appear to be available, and using unverified keys should require confirmation.

- **Privacy Preserving**: Many people, especially those living in functioning democracies, aren't particularly worried about their safety. Instead, they turn to a tool like GnuPG, because they are concerned

about their privacy, and mass surveillance. Other reasons include the need to occasionally send a password by email, and a desire for protection from drive-by phishing expeditions (although since few organizations currently sign their email, this is more wishful thinking than practical protection).

With few exceptions, the MUA should avoid interrupting these users with security questions. One exception is when the user follows up to an encrypted email, but the reply won't be sent in an encrypted manner. Since the sender encrypted the email, it might be for a good reason and, consistent with the do no harm principle, the user should not accidentally endanger her communication partner, or the subject of the mail.

This doesn't mean that the encryption should entirely disappear into the background. The MUA should still help the user understand what is going on, and allow the user to provide input, if desired. For instance, like a web browser, the MUA should indicate whether a message is secure. And, if the user clicks on the icon, she should get more information, and have the option to verify her communication partner's identity. In other words, security should largely be opt-in.

The trade off that these profiles make is straightforward: someone who requires more security is more sensitive to a mistake, and is more willing to interact with the system to ensure this security. For people who have lower security requirements, not only are these interactions annoying, they can actually hurt security elsewhere: showing dialog boxes that are simply clicked away results in habituation [4, 3].

Communication, of course, necessarily involves multiple parties. Thus, if a user with high security requirements communicates with a user with low security requirements, the casual user could accidentally compromise the careful user by forgetting to encrypt an email. Thus, consistent with the do-no-harm principle, it is important that even an implementation designed for users with low security requirements not be too lax.

1.1 Integration

There are two basic ways to add GnuPG support to a MUA: it can be added natively, or it can be added via a plug-in. KMail, and Claws are examples

of MUAs that have native GnuPG support; Enigmail, GPGTools, and gpgol are examples of plug-ins.

One approach isn't necessarily better than the other. But, the development of plug-ins tends to be highly divorced from the actual development of the MUA with the practical result that the needs of the plug-in are often not sufficiently taken into account by the MUA developers. This has been a problem for the Enigmail developers, for instance.

One common problem is controlling how messages are rendered: the GnuPG support code needs a lot of control over this. This control is necessary to prevent mimicry attacks. For instance, it is necessary to not only show when a message is verified, but also prevent an attacker from crafting a message that appears to be verified. One way to accomplish this is to style not only the message, but also the chrome around the message.

The things that need to be added to a MUA for reasonable GnuPG support is not long: there needs to be a way to create a key, encrypt messages, verify messages, and do some basic key management. But, all of these things have numerous gotchas that can negatively impact both the user experience, and the security of the system. The point of this chapter is to point out these issues to avoid making developers—or worse, their users—rediscover these problems the hard way.

1.2 Key Creation

When a GnuPG-enabled MUA is started, it would seem logical to prompt the user to create or import a key if the user has not already done so. This behavior is reasonable if the user has explicitly enabled GnuPG support by installing a plug-in. However, if the MUA has native GnuPG support, and it is not certain that all users want to use GnuPG, it may be best to wait to avoid overwhelming the user during the initial setup.

If a key is not generated immediately, this doesn't mean that the GnuPG-related functionality should somehow be hidden or disabled. Even without a key, it is still possible to verify signatures, and show unsigned messages as being insecure. Then, if a user clicks on such a security notice, the MUA can explain why the message is considered insecure, and provide an option for the user to configure the GnuPG support. Similarly, it is reasonable to present an option to encrypt a message before a key has been created. If the user selects this option, and there is no key associated with

the sending email address, then the MUA should show the key creation wizard. This significantly improves discoverability.

The key generation wizard should not only allow the user to generate a new key, but also provide an option to import an existing one. When the user enters or selects a user ID, the wizard should look for an existing key with that email address both in the appropriate WKD, and on any configured key servers. If there is a matching key, the wizard should ask the user if she wants to import the key or really create a new one. Importing the key might not be possible if the key is a fake, or if the user lost access to the key, e.g., by formatting the computer, or forgetting the key's passphrase. Both are unfortunately rather common for novice users.

When the key generation wizard starts, the user ID should default to the current identity. For instance, if the user has the email addresses `alice@posteo.de` and `alice@gnupg.net`, and clicks on encrypt while composing an email from `alice@gnupg.net`, the wizard should default to creating a key for `alice@gnupg.net`. If Alice selects a different identity, then the wizard should explain why the key won't be usable for the email she is currently composing.

If the user already has a key, but not one for the current identity, it is reasonable for the key creation wizard to offer to add the identity to the existing key. However, current thinking in the GnuPG project is that users require less training when there is a one-to-one mapping of keys and email addresses than when multiple user IDs are associated with a single key. For instance, if the MUA offers to add the user ID to an existing key, it becomes necessary to explain why this might be undesirable, e.g., most people probably want separate keys for their private, and their work email. And later, if the user retires her email address, it will become necessary to explain the difference between revoking the key and revoking a user ID. Of course, since many users do use keys with multiple user IDs, it is necessary for the MUA to support such keys, and explain their meaning when signing keys, for instance.

The key generation wizard should make key creation as easy as possible by prompting the user for as little information as reasonable. In particular, the user should *not* have the option to enter a comment; adding a comment is almost always inappropriate [5]. Likewise, key generation parameters should not be configurable. But, the user should be allowed to choose whether the key is published on the Internet. This requires an explanation, which can be made by simile: publishing a key on the Inter-

net is like publishing a telephone number in a phone book, and no one is checking the submitted entries.

If it is deemed absolutely necessary that the user be able to tweak key parameters, then the options should be hidden unless the user explicitly enables some sort of expert mode. The reason is simple: for the most part changing these parameters doesn't actually improve the overall security. For instance, using a 2048-bit RSA key is currently considered sufficiently secure by multiple authorities [6]. If more security is really needed, then the user should start by improving their weakest defense, which is almost certainly their opsec and not the cryptography. Bruce Schneier, for instance, argues that the Snowden leaks provide strong evidence that the NSA has not broken strong cryptography. Instead, the NSA appears to get the information they want by compromising infrastructure and endpoints [7]. The easiest and probably most effective measure is to use a smartcard instead of storing the private key material on the computer.

There are also practical reasons for not using an overly large key. Perhaps the most important one is simply based on performance: it does not take twice as long to verify a signature generated with a 4096-bit RSA key than one generated with a 2048-bit RSA key, but about an order of magnitude longer. This performance penalty becomes particularly noticeable for 16,384-bit keys.

1.2.1 Revocation Certificate

After creating a key, the wizard should prompt the user to save the key's revocation certificate, or offer to print it out (or both!). For users with low security requirements, it is also reasonable to send the revocation certificate to the user in an email (along with an explanation of what a revocation certificate is, and how to publish it). This is the easiest way to make sure the revocation certificate is stored in multiple places, but it has the disadvantage that it gives anyone who can access the user's mail the power to revoke her key. This weakness is problematic, but it is not disastrous: that person would be able to perform a denial of service attack (other people would no longer be able to send encrypted messages to the user, and signatures generated by the key would no longer be considered valid), but could not assume the user's identity, or read encrypted messages. And, creating a new key is straightforward. So, the potential damage is limited, and for most users probably represents a net win given the benefits of being able to

revoke a lost or inaccessible key.

1.3 Expiration

When GnuPG 2.1 creates a new key, the default is to set the key to expire in two years. Just because a key expires does not mean that the user needs a new key: the expiration is just an emergency brake if the user loses access to her key, and can't publish a revocation certificate. Consequently, the MUA should support extending a key's expiration date. This can be done when the MUA starts. But, since many users rarely restart their MUA, it may be better to check whenever the key is used.

If the key is about to expire (within, say, three months), the MUA should extend the expiration. Once the expiration is extended, the key needs to be uploaded to the key servers or otherwise distributed to the user's communication partners so that their OpenPGP implementation can take the change into account.

Since extending a key's expiry requires making a self-signature, the user will need to unlock the secret key. This interaction can be hidden by piggy backing the operation onto some other operation that requires the user to unlock the key.

For security sensitive users, it may make sense to ask the user if this is desired. For very high risk users, there should also be an option to rotate the keys.

1.4 Sending Mail

The mail composition window should have a toggle to "secure" or "encrypt" the current message. When active, this toggle should actually cause the message to be encrypted *and* signed. There should *not* be a separate toggle for signing the message. As explained previously in Section `sec:option-pgp-encryption`, most users assume that encrypting includes signing, and don't understand signing at all.

The button may have a menu that becomes visible after, for instance, a long press, which allows the user to select between "Encrypt and Sign", "Sign-only", "Encrypt-only" and "No protection." However this menu is activated, it should be reserved for advanced users, which justifies the poor

discoverability of this feature: needing to only encrypt or only sign a message is relatively specialized, and these users can be expected to have had training; normal users should only have to choose between a secure, and an insecure option.

The Mailpile MUA always signs messages, even if they are not encrypted. To avoid confusing users who do not have an OpenPGP capable MUA, Mailpile uses inline signatures when possible, because, with the exception of one line, the signature shows up at the bottom of the message, and users have learned to ignore mumbo jumbo at the end of messages. Anecdotal evidence suggests that this approach doesn't impose any cognative load on users whose MUAs don't support OpenPGP. When an inline signature can't be used, Mailpile exports the signature as an ASCII-armored blob, adds a description explaining the purpose of the signature, and names the attachment `signature.asc.html`. The naming is essential: if a recipient open the attachment, she sees the explanation, and knows that she can ignore it. Anecdotal evidence suggests that this also significantly reduces the amount of confusion that signatures typically cause.

For users with high security requirements, it makes sense to always enable encryption by default, and then require that the user explicitly disable it if encryption is not desired. This avoids mistakenly sending a message unencrypted when it should have been encrypted. However, this default can be annoying for users who do not normally encrypt their mail.

As mentioned earlier, a MUA can deal with this dilemma by setting appropriate defaults for the user's threat model. But even for low security users, there are cases in which it is clear that encryption should be enabled by default. For instance, if the user is replying to an encrypted message, then encryption should be enabled. In fact, if the user tries to disable encryption, it is reasonable to show a warning of the form: "you are replying to an encrypted message, do you really want to disable encryption for your reply?" Similarly, if a recipient consistently sends encrypted mail, or there is a verified key available, then encryption should probably be turned on. Although it is appealing to encrypt whenever possible, encryption can sometimes decrease usability. This is particularly the case for users who process email on multiple devices, but only a subset of them are able to decrypt the messages.

An appropriate default can be more difficult to find when there are multiple recipients. For instance, when a user replies to an encrypted message, she might not have keys for all of the recipients. But, the application can

help the user find the keys, and, in this case, finding appropriate keys is actually straightforward: due to the way that OpenPGP encrypts data, the long key ID of the sender and any recipients will normally be embedded in the message (specifically, in the PK-ESK packets). Unfortunately, the key IDs are subject to tampering, but since this requires a more determined adversary, they are almost certainly much more reliable than simply searching a key server for keys with a particular email address. It is also possible to try and find the key using WKD, which provides a basic verification check. Another reason to avoid key servers is that using a key found on a key server may cause more problems than it solves: the message may be encrypted, but because it is the wrong key, the intended recipient can't decrypt it. Making decryption unreliable is a sure way to discourage the use of encryption. Key discovery is covered in more detail in Section 1.6.1.

Sometimes mails include keys as attachments, or references to them. In such cases, the MUA should either import them automatically or provide a button to allow the user to import them. But, the keys should always be imported if they are already available locally: the keys might contain updates, such as new subkeys, an extended expiration, or a revocation certificate. This topic is discussed further in Section 1.6.1.

1.4.1 Encryption Keys

To make it clear whether there is a key for a particular recipient, the MUA should add a small icon, e.g., a padlock, next to each email address. As usual, to improve discoverability, and provide a reminder to encrypt, this should always be done, even if encryption for a draft has not yet been enabled. In that case, the padlock should also be crossed out. The coloring and the icon should vary according to the degree to which the key is verified. (It is important to not only change the color to support colorblind users.)

We recommend that the UI distinguish between the different degrees of verification. The web of trust provides three verification levels: a key can either by fully verified, marginally verified or not verified. (Note: for historical reasons, GnuPG uses the term "trusted" here instead of "verified." To reduce confusion in this document, we reserve the term trusted for when a key is not just verified to be controlled by the stated entity, but may act as an introducer. MUAs should do the same.) And, the TOFU trust model provides even finer grained verification levels. These distinctions are impor-

tant for security conscious users, and, as a rule of thumb, marginally verified keys should **not** be shown as having the same level of security as fully verified keys. Instead, fully verified keys should be shown in, say, green, and partially verified keys should be shown in, say, yellow. If it is somehow desirable that marginally verified keys have the same security level as fully verified keys, then the user should explicitly set the `marginals-needed` option in her `gpg.conf` file to `1`. In the very least, the UI should distinguish between fully verified keys, and not fully verified keys, i.e., if the UI only shows two states, it should show marginally verified keys the same way it shows completely unverified keys.

If the TOFU trust model is enabled, the number of days on which a message has been encrypted to the key plus the number of days on which a message signed by the key has been verified should be shown next to the icon. This can be shown in a small bubble subscripting the icon, which is similar to what Twitter does for showing counts. For large numbers, it is reasonable to show approximate numbers (e.g., rounding `1132` to `1.1k`).

Showing these statistics is important to help users to detect mimicry attacks, which are often employed by phishers. For instance, if a bank normally signs their emails, then users hopefully become used to seeing the count slowly increase. Then, if they get an email that appears to be from their bank, but the count is `0`, they will hopefully become suspicious.

If the user hovers the mouse over the padlock icon or clicks on it, the MUA should show a short, tweet-length message explaining why the key is considered verified (or not). If the key is not fully verified, an option to start a key verification wizard should be provided. If there is a TOFU conflict, there should be an option to start a TOFU conflict resolution wizard. And, if there is no key associated with the email address, there should be an option to start a key discovery wizard. (The wizards are described in Section 1.6.)

1.4.2 BCC Recipients

When sending a mail, if there are any `bcc` recipients, the MUA should create a separate mail for each `bcc` recipient, and one for the rest. This avoids having the OpenPGP implementation leak the `bcc` recipients to the other recipients. Although it is possible to hide a recipient's key ID in a message by using a speculative key ID (e.g., using `gpg`'s `--throw-keyids` option), this still reveals to the recipients that the message was probably encrypted to other people. Using separate emails avoids this leak.

1.4.3 Saving Drafts

In general, when a draft—whether it has been marked to be encrypted *or* not—is saved on the IMAP server, it should be encrypted to the user. It should not be encrypted to any recipients; they should only be able to decrypt the final version.

It is important to encrypt all drafts even if they that have not been marked for encryption, because the user's intent is only known once the mail has been sent. It may be reasonable to relax this requirement in cases where it is clear that the user is only using the encryption for privacy purposes. But a safer way to avoid using the private key to decrypt the drafts is to *also* either save the session key or an unencrypted copy locally.

1.4.4 Sent Mails

When sending a mail, it is important to also encrypt the mail to the user. Given the near universal prevalence of a sent folder in MUAs, most users clearly expect to occasionally be able to later read the mails that they send. This can be done using `gpg` 's `encrypt-to` option, or, when encrypting an email, the sender can be specified explicitly.

1.4.5 Attaching Keys

To make it easier for a recipient to reply to a message in an encrypted manner, the MUA should provide an option to attach all public keys she would need to do so.

Receiving a key can be surprising to users who don't use or know about GnuPG. But, if you are encrypting, this is not a concern: you know the recipient's MUA understands OpenPGP messages. As such, in these cases, the keys can be attached automatically.

When attaching a key, it is reasonable to just include a minimal version of the key. In particular, it doesn't need to include any certifications, because once the recipient has the key, it is easy to get the rest of the data from a key server. A minimal key can be created by providing the option `--export-options export-minimal` when exporting a key using `gpg`.

The user's key should also always be specified in the OpenPGP header [8]. This is the case whether the mail is encrypted or not. This provides a strong hint to recipients that the user can work with OpenPGP messages.

1.5 Reading Mail

When the user opens an email message, it is necessary to identify if the message is encrypted or signed and to act accordingly. This is relatively straightforward, but does require a robust MIME parser to to handle all email. In particular, emails that have been transformed during transport can be problematic. The more challenging issue is making sure the user understands whether a message has been transferred securely. As a general rule of thumb, it is better to be conservative, and indicate that a message has been transferred insecurely than to incorrectly claim that a message has been transferred securely when that might not be the case. For instance, instead of attempting to interpret all possible structures, it is better to white list acceptable structures, and treat deviations as being insecure. Other issues include avoiding unnecessary passphrase prompts, and searching encrypted email.

1.5.1 Verifying Messages

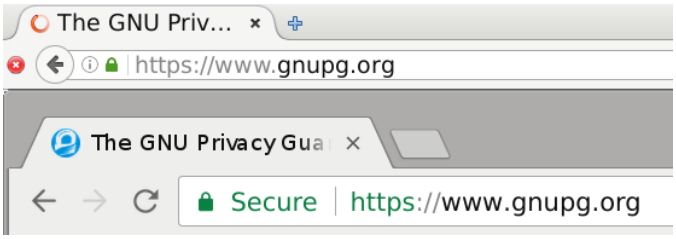

Figure 1.1: Padlock icons shown by Firefox and Chromium when a website is transferred securely.

When a user views an email, it is important to communicate whether the contents were transferred in a secure fashion. In web browsers, this type of information is usually shown using a small padlock icon in the address bar.

Firefox, for instance, shows a green padlock if it transferred the website in an encrypted manner, *and* it could authenticate the end-point. It uses a gray padlock with a yellow warning triangle if some—but not all—of the content was encrypted, and eavesdropping was possible, or if the website used a self-signed certificate. It uses a gray padlock with red strikethrough if a man-in-the-middle attack was possible. And it just shows a neutral,

"more information" icon if TLS was not used at all [9]. There are two important issues with this scheme.

The first issue is that this scheme conflates encryption and authentication. Although it might be reasonable to demand that websites that use authentication also use encryption to be considered secure—it simplifies user training, and doesn't impose a significant deployment cost—this argument doesn't apply in an email setting. Consider, for instance, a company that wants to sign all of its outgoing emails to help mitigate phishing. In this scenario, encryption is more of a hindrance than a help: requiring encryption would mean that the company would have to somehow find the right encryption key for each of its correspondents. When only providing an authentication mechanism, not only are the customers' keys not required, the customers don't even need to have a key: they just need the ability to validate the signature.

The second problem is that a TLS connection that can't be authenticated is shown to be worse than a connection that is completely insecure. For instance, until the recent introduction of *Let's Encrypt*, website operators who wanted to offer an encrypted connection to their website, but didn't want to pay for a certificate could use a self-signed certificate. Although data protected by such certificates is not secure in the sense that the end point can't be authenticated without user intervention, such certificates enable encryption, which does protect users from passive surveillance. In other words, self-signed certificates provide more protection than nothing at all, but websites that use self-signed certificates are shown as being less secure than sites that use no protection at all! (Although encrypting is better than not encrypting, we nevertheless recommend that MUAs show encrypted and unsigned mails in the same way that they show unencrypted and unsigned mails to avoid confusing users.)

Happily, at least the Chrome browser does not make this distinction. And, like Chrome, we strongly recommend that whatever mechanism is used to show that a mail can't be authenticated be used for *both* unsigned mails, and mails with a signature that can't be verified. Specifically, we recommend considering an unencrypted and unsigned email to be the baseline, and that an email is never displayed in such a way that the user would consider it to be less secure than the baseline, unless there is strong evidence of an attack.

It is reasonable to show unverified messages, and unsigned messages in a neutral manner, and to show verified messages in a positive man-

ner. However, it may also be reasonable to show unverified messages, and unsigned messages in a negative manner. This is how MS Outlook behaves when S/MIME is enabled. This has the added advantage that it may prompt the user to learn why the MUA showed the message as being unsafe.

The first step to checking whether a message is authentic is to check whether the signing key is verified according to some trust model, e.g., the web of trust. When verifying an email, another step is required: it is also necessary to make sure the key is controlled by the sender. This can be done by checking that the email address in the email's `From` header actually appears in one of the key's verified user IDs. This is necessary to prevent an attacker from reusing a message in a different context. For instance, assuming Romeo trusts his father, his father could write an email that appears to come from Juliet, but sign it with his own key. If the MUA doesn't check that the `From` header and the signer field agree, then the MUA would show Romeo that the message is verified. Unfortunately, some mailing lists rewrite the `From` header, which will cause this test to gratuitously fail. One reason for doing this is to improve DMARC compatibility.

Just checking that the sender matches a verified user ID is not actually enough to prevent all replay and mimicry attacks. It is also necessary to make sure the embedded timestamp is similar to (i.e., within a few hours of) the email's timestamp. If the timestamp in the email is years later than the one embedded in the signature, then the email may be part of an attempted replay attack. Similarly, it is possible to change the recipient. For instance, Juliet might send the following signed message to Paris: "Go away, I do not love you!" But, Paris, realizing that Romeo and Juliet are in love, and hoping to trick Romeo, might simply send a copy of the message to him with the `From` header set to Juliet. These types of attacks can be mitigated by also verifying the mail headers. The Memory Hole project was started to do exactly this [10]. Unfortunately, the standard isn't finished, and work on it appears to have stalled. Nevertheless, there is enough information to understand the intent, and several mail clients including Enigmail and Mailpile implement it.

Sometimes, a message may include multiple signatures. Any signatures from keys that match the email address in the `From` header should be used for verification purposes. Other keys may be listed when showing the verification details.

If the TOFU trust model is enabled, then the TOFU statistics should be

shown as in the encryption case.

1.5.2 Multi-part Emails

Thanks to inline signatures, it is trivial to make a message that is only partially verifiable.

For simplicity's sake—we don't want to confuse the user—it is tempting to treat such messages as insecure like web browsers do. However, some companies, and some mailing lists automatically append a footer to all messages. This modification would change a message that is otherwise completely verifiable to one that contains a part that isn't signed. Thus, messages coming from these sources would never show up as secure.

A straightforward *technical* solution is to show each section individually. This can be done using a frame. The frame should be part of the MUAs chrome and not the message to avoid mimicry attacks. Further, each part should have an icon, e.g., a padlock, that shows information about the part's verification (the degree to which it is verified, and the key's TOFU statistics), and that, when clicked, shows a menu allowing the user to get more information, and find the key if it is missing, verify the key if it is present, or resolve a TOFU conflict, as appropriate.

To further distinguish between verified and unverified parts, a special background can be used. Ideally, the background should be unique for each user to further frustrate any attempt at a mimicry attack.

Unfortunately, this information rich technical solution may overwhelm many users. An alternative is to show a single icon at the top that shows the minimum security level of the individual parts. Since some corporations and mailing lists attach a small footer to all mails, this should be excluded from the calculation, but it should not be shown as verified.

The issues raised so far are manageable. Unfortunately, MIME makes things much more complicated: MIME can not only encode multi-part documents, but it can also encode rich content that logically consists of multiple MIME parts only some of which are signed, such as an HTML document and images that it references.

If a message includes at least one verified part, then the MUA should only show those parts that are verified, and warn the user that the message contained unverified content that is hidden. It is reasonable for the warning to include an option to show the unverified parts anyway. At that point the message should be displayed as insecure.

This suggestion conflicts with our earlier suggestion of showing unsigned messages in the same way as unverified messages. The best suggestion we have is to show a warning along the lines of "this message is unverified, show anyway" for unsigned messages. But, since most users will primarily deal with unsigned mail, this warning will very quickly get annoying, and lose its value. If security profiles are supported, this option should only be enabled for users who have very high security requirements.

1.5.3 Unencrypted Cache

The OpenPGP email workflow assumes that messages are stored on an untrusted server, and thus continue to need protection even after the mail has been delivered. Supporting this type of workflow is one of the primary reasons that OpenPGP doesn't provide forward secrecy. There are two major consequences of this workflow.

First, every time a message is accessed, it needs to be decrypted. This can lead to many passphrase prompts. These can be largely mitigated by increasing the amount of time `gpg-agent` caches passphrases, or by using a password manager. But, it is also annoying for smartcard users who need to basically always leave their smartcard inserted, which effectively nullifies a nice security property of smartcards: the user can observe operations, because they can only be done when the card is inserted.

Second, it is not possible to search encrypted mails. This is a major usability problem, particularly when the subject line is also obscured as it should be to avoid accidentally leaking the message's contents.

Both of these issues can be largely mitigated by caching the unencrypted version of each message locally. This assumes, of course, that the local device is secure. At a minimum, the user should have the mail stored on an encrypted partition.

1.6 Key Management

There are three main aspects to key management: key discovery, key verification, and key organization.

1.6.1 Key Discovery

The first requirement for encrypting or verifying a message is having the appropriate key. There are several ways to find the right key. Unfortunately, most of them make no guarantee that the key that is returned is the correct key. But some are significantly more difficult for an adversary to corrupt than others making them at least appropriate for opportunistic encryption.

Exchanging Fingerprints in Person

The most secure way to find a person's key is to get it from that person directly. If a physical meeting is possible, this can be done by exchanging fingerprints in person. At least in the business world, the cost of this exchange can be driven to zero: because exchanging business cards is a common practice in this world, adding your fingerprint to your business card makes securely exchange fingerprints a free byproduct of a well-established ritual.

Having a fingerprint on a business card is not quite enough to use it: it still needs to be entered into the system. The key discovery wizard can make this process easier by suggesting possible matches based on what the user has entered so far. (Possible matches can be found by querying a key server.)

We recommend having the user enter at least 64-bits worth of the fingerprint before enabling auto completion to ensure that the user checked a minimal amount of the fingerprint. For instance, it is possible to create a key with a specific 32-bit key ID in just a few *seconds* on modern desktop computers [11].

If the email address is known (and it is probably reasonable to first ask the user to specify a contact if this is not clear from the context), and there is at least one matching key, an alternative approach is to show a series of buttons with fragments of the matching fingerprints, and have the user select the matching fragments. This idea is illustrated below:

```
[ 8F17 ] [ 5200 ] [ 18A3 ]
[ 3DDA ] [ 6396 ] [ 8723 ]
[ AACB ] [ 6388 ] [ 0BAD ]

[ None of the above ]
```

The "none of the above" option is useful if the right key is not on the key servers, for whatever reason.

A more user-friendly technique could use a webcam and OCR to read in the fingerprint. From an implementation perspective, this is more demanding than scanning a QR code, for instance, but there are many fewer people who add a QR code containing their fingerprint to their business card than those who add their fingerprint. But, providing an option to display a public key using a QR code on screen can be helpful: someone could scan it.

Picking up the Phone

Exchanging keys in person requires that people actually meet face to face. This is often not practical. The next best alternative is to pick up the phone. This approach is appropriate for all but those people who have the highest security concerns—those whose threat model includes a real-time voice imitator. Although this has been technically feasible for years. It requires precise timing that only a nation-state adversary could afford.

Again, assuming the email address is known, the button grid can be used to facilitate transcription of the fingerprint.

Searching a Website

Calling someone is not always possible or desirable. In this case, it is sometimes possible to find the person's key on her website. The caveats are that even a relatively unsophisticated attacker can often own a website or spoof it, and because there hasn't traditionally been a standard place to publish keys, the MUA can't actually help the user find it.

In 2016, the GnuPG project published a new key discovery protocol called the web key directory (WKD). WKD automates, and hardens this key discovery process. The basic idea is that to find `romeo@posteo.de` 's key, Juliet looks for the key associated with `romeo@posteo.de` in a database on `posteo.de` [12]. This protocol relies on the security of TLS, and the mail provider. The mail provider can currently be held in check by periodically auditing the database, e.g., periodically fetching your own key via Tor and making sure it hasn't been replaced. Eventually, something like certificate transparency [13] could be added to catch abuse or detect things like national security letters. The reliance on TLS and its centralized infras-

tructure goes against the philosophy of OpenPGP, but it is acceptable for people whose threat model is limited to privacy violations, and phishing excursions.

Currently, the only commercial mail provider that supports WKD is Posteo, but, as of 2017, there are discussions underway with other mail providers to add support for this feature.

Searching Key Servers

A seemingly convenient way to find someone's key is to search for it using that person's email address on a public key server. Unfortunately, this method has very bad security properties: anyone can upload a key to a key server with any user ID. It is trivial to forge a user ID. In fact, in 2014, all known keys were cloned with identical short key IDs [11]. But, even if you are only interested in the privacy aspects of encryption, the key servers are a bad place to look for keys. Because many people forget their passphrase or forget to migrate their key to a new computer, the key servers are littered with seemingly valid keys that are practically unusable. Since someone searching the key servers doesn't know what key is correct, these people often get emails that they can't decrypt. This is annoying, and causes people to avoid encryption. Consequently, if a MUA decides to provide support for looking up keys by their user ID, we strongly advise adding a prominent warning about the possible problems. Further, if this approach must be used, it is better to encrypt to all matching keys. When the recipient replies, it is then possible to narrow down the set of potential keys based on the signature or the PK-ESK packets—assuming there was no man in the middle attack.

Note: these problems don't mean that key servers are completely useless. Far from it. The problem with key servers is that user IDs are not authoritative. But, if you have already verified someone's key, then key servers are the perfect place to get any updates (e.g., new signatures, revocation certificates, etc.), because cryptography can be used to determine whether the information really belongs to the key in question.

Exploiting Context and Hints

There are two main places where context can be used to discover potentially useful keys: a signed message indicates what key was used to sign

it, and an encrypted message usually includes the key IDs of the sender and other recipients in the PK-ESK packets. Emails also sometimes include hints about the right keys to use. For instance, some people attach either their key to the emails that they send (pEp does this by default), or the keys of all recipients in order to make it easier for people to reply in a multi-party discuss. Another hint can sometimes be found among a mail's headers: the `OpenPGP` header allows the sender to advertise a key [8].

In theory, there is no reason to not import these keys. Simply importing a key will not cause it to be considered verified: whether a key is considered to be verified, is determined exclusively by the trust model, not whether it happens to be available locally. But, having what is probably the right key available locally is useful for opportunistic encryption. And, used in conjunction with the TOFU trust model, it is even possible to bootstrap some trust over time.

Unfortunately, in practice there are two important issues with harvesting keys.

The first issue is that automatically fetching keys via the network can be used as a back channel. A sophisticated attacker could create a new key for each message. When a user fetches the key, the attacker can potentially learn not only that the user opened the message, but also the user's IP address. This attack can be mitigated by routing this type of traffic via Tor (to do this, Tor must be installed and GnuPG configured to use it by adding `use-tor` to `dirmngr.conf`). Using Tor not only hides the user's IP address, but also requires the attacker to actually control the user's preferred key servers to observe the fetch. This is only feasible by an adversary with a lot of resources.

Even if automatically fetching keys is disabled, the MUA can still harvest this information, and save it in a local database. Then when the user explicitly searches for the key associated with an email address, say, the hints can be exploited.

The second issue is that GnuPG doesn't handle very large key rings (those with thousands of keys) very well. This manifests itself in two ways. It shows up as longer random access times: `gpg` does a linear scan of the key ring the first time it is accessed. Also, GnuPG's trust calculations are done on demand when `gpg` starts. These calculations can take minutes on large key rings. And, they are done whenever a new key or signature is imported, or a key expires or is revoked. When harvesting keys, this can happen very often. Happily, the trust calculations can

be deferred by setting `no-auto-check-trustdb` in `gpg.conf` and then running `gpg --check-trustdb` periodically, e.g., from something like `cron`. But obviously, this means the trust model may not be completely up to date. However, the only long-term fix is to improve the way that keys are stored on disk.

Note: `gpg` can automatically fetch keys needed for verifying signatures by setting the `auto-key-retrieve` option in `gpg.conf`, and for encrypting messages by setting the `auto-key-locate` option. These options have the disadvantage that they can potentially block the `gpg` process for a relatively long time. Consequently, it is often more appropriate to attempt to fetch the key in the background. In the verification case, the message can be rerendered if the key becomes available. And, in the encryption case, a key should be located in the background when the recipient is added, not when the message is sent.

Taking Advantage of Trusted Introducers

Designating someone as a trusted introducer means that the user trusts that person to correctly verify others. Since friends of friends are likely to be friends as well, it makes sense to proactively fetch any keys that trusted introducers have signed.

`gpg` does not do this itself. And, unfortunately, the key servers do not provide a mechanism to find all keys signed by a particular key. But, since verification is usually mutual, it is possible to approximate this by fetching all keys that signed a trusted introducer's key. The MUA can do this periodically in the background.

1.6.2 Key Verification

Key verification is essential to the security of the system. Although people who are primarily interested in preserving their privacy will not spend much time on this task, it is essential that the key verification support is robust for those who depend on it for its security properties.

This requirement first means that it should be easy to start the key verification wizard in appropriate contexts. For instance, when the user adds a recipient to an email, as explained above, an icon should be displayed showing whether there is a key associated with the contact, and, if so, the degree to which the key is considered verified. Clicking on the icon should

allow the user to verify the key.

When the key verification wizard is started, it should not just prompt the user to check the fingerprint, but actually guide the user through the different ways to obtain a fingerprint. For instance, the following is a bad idea:

```
Certify this key?

  8F17 7771 18A3 3DDA 9BA4  8E62 AACB 3243 6300 52D9

                                  [ Ok ]  [ Cancel ]
```

Instead, the key verification wizard should ask the user how she wants to confirm the key: using a business card or other printout, or via phone. This approach educates the user without being patronizing: the user learns how to verify a fingerprint, and that it is not okay to just click verify without actually verifying the key.

To prevent the user from simply clicking okay without checking the fingerprint, we recommend requiring that the user enter at least part of the fingerprint. This can be done by using the buttons with the fingerprint fragments, as described above.

Ownertrust

It is strongly recommended that an option to set a key's `ownertrust` be well hidden relative to the key verification option. In fact, it should only be possible to set the `ownertrust` if the key in question is already fully verified (e.g., directly signed). Also, even though there are a few rare cases where it makes sense, it shouldn't be possible to set a key to be ultimately trusted if no secret key material is available.

When the `ownertrust` option is shown, it must be well explained that this option is not only about trusting the person, but also trusting how she verifies keys. For instance, I might trust my best friend when he introduces me to people in the physical world, but without understanding how he verifies keys (does he just click on yes to make the padlock green?), I probably should not set him as a trusted introducer. In practice, the latter is generally much more difficult for people to judge, because they don't understand the process very well themselves, and, given how hard it is to get people

to exchange fingerprints, it is unlikely that we will ever convince them to discuss their security practices.

Publishing Signatures

The key verification wizard should provide an option to publish the signature. This should be accompanied by an explanation of what this means and why this is useful (people who trust you won't need to manually verify this fingerprint).

It is also reasonable to provide an option to make a trusted signature instead of a simple certification. Again, this requires an explanation. This option should probably only be hidden unless expert mode is enabled.

1.6.3 TOFU Conflict Resolution

Like the web of trust, TOFU is a trust model. The major difference between the two is that the web of trust provides strong guarantees, but requires a lot of upfront verification work whereas TOFU builds up trust slowly over time and is only secure in an asymptotic sense, but requires little user support. The `tofu+pgp` trust model should be the default for users with low security requirements. For backwards compatibility reasons, TOFU has not been made the default in GnuPG.

Normally, the user only needs to interact with the TOFU trust model to resolve conflicts—when multiple valid keys have the same email address. A conflict is a strong sign that a man-in-the-middle attack is underway. But, it can also just be because the user replaced a key that she could no longer access or revoke. The only way to resolve this is by asking the user to verify the key. (When creating a new key, a conflict can be avoided by either promptly revoking the old one or cross signing the two keys.)

When the user starts the conflict resolution wizard, the wizard should explain what a conflict is, show the conflicting keys and their statistics, and explain how to resolve the problem (ideally, the user should call the contact to verify the fingerprint). Because the user might not be able to resolve the conflict immediately, it is better to provide a resolve later option, which is the default, rather than have the user simply accept the key without validating it.

Note: just because a key has a lot of past usage does not mean that it is the right key: the man in the middle might just have failed to intercept

the most recent message. Likewise, the new key is not necessarily the right one: the man in the middle might just have started the attack.

1.6.4 Address Book Integration

A key ring is effectively a backwards address book: instead of names being the primary keys, and OpenPGP keys being associated with contacts, a key ring reverses this. This unusual arrangement can cause novice users significant confusion. As such, it is better to avoid the key ring as much as possible, and instead directly integrate keys into the user's address book.

If the address book supports identities with multiple email addresses, then it should be possible to associate each email address with a different key. Also, it should be possible to force messages sent to a particular contact to be encrypted to multiple keys. This is useful in the case where an email address acts as an exploder.

It is also useful to keep track of users who appear to use GnuPG. A recent encrypted or signed email is the best indicator, but the presence of the OpenPGP mail header is also an excellent hint. The presence of a key with the user's email address is, however, not sufficient proof that the user can use GnuPG. This functionality can also be exposed as an option: "always encrypt to this user."

Bibliography

[1] V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. RFC 7435, RFC Editor, December 2014. https://www.rfc-editor.org/rfc/rfc7435.txt.

[2] Holger P. Krekel, Danial Kahn Gillmor, et al. Autocrypt level 1. https://autocrypt.readthedocs.io/en/latest/.

[3] Rainer Böhme and Stefan Köpsell. Trained to accept?: A field experiment on consent dialogs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2403–2406, New York, NY, USA, 2010. ACM.

[4] Rainer Böhme and Jens Grossklags. The security cost of cheap user interaction. In *Proceedings of the 2011 Workshop on New Security Paradigms Workshop*, NSPW '11, pages 67–82, New York, NY, USA, 2011. ACM.

[5] Daniel Kahn Gillmor. Openpgp user id comments considered harmful. https://debian-administration.org/users/dkg/weblog/97, May 2013.

[6] Damien Giry. Keylength - cryptographic key length recommendation. https://www.keylength.com/. Last accessed: July 25, 2017.

[7] Bruce Schneier. NSA surveillance: A guide to staying secure. https://www.theguardian.com/world/2013/sep/05/nsa-how-to-remain-secure-surveillance, September 2013.

[8] Atom Smasher and Simon Josefsson. The "OpenPGP" mail and news header field. Internet-Draft draft-josefsson-openpgp-mailnews-header-07, IETF Secretariat, August 2014. https://tools.ietf.org/html/draft-josefsson-openpgp-mailnews-header-07.

[9] mozilla support. How do i tell if my connection to a website is secure? https://support.mozilla.org/en-US/kb/how-do-i-tell-if-my-connection-is-secure. Last accessed: July 23, 2017.

[10] Daniel Kahn Gillmor et al. Memory hole. http://modernpgp.org/memoryhole/,https://github.com/ModernPGP/memoryhole. Last accessed: July 23, 2017.

[11] Richard Klafter and Eric Swanson. Evil 32: Check your gpg fingerprints. https://evil32.com/, https://www.defcon.org/html/defcon-22/dc-22-speakers.html#Klafter, August 2014. Last accessed: July 28, 2017.

[12] Werner Koch. OpenPGP Web Key Service. Internet-Draft draft-koch-openpgp-webkey-service-02, IETF Secretariat, October 2016. https://tools.ietf.org/id/draft-koch-openpgp-webkey-service-02.txt.

[13] Ben Laurie, Adan Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, RFC Editor, June 2013. https://www.rfc-editor.org/rfc/rfc6962.txt, https://www.certificate-transparency.org/.