PassLok 1.7 manual

by F. Ruiz, 3/19/2014

This version contains few images, but future versions will have more, as I get to it.

Outline:

1) What is PassLok?
   a) What is public-key encryption?
   b) What makes PassLok special?
   c) Keys and Locks
2) Keys and Locks
   a) How to make a strong Key
   b) Generate your matching Lock
   c) Spreading your Lock
   d) Changing Locks
   e) Making a Lock database
   f) The general Lock directory
3) Locking and Unlocking
   a) Anonymous locked messages: msa
   b) Key-locked messages: msk
   c) Short locked messages
4) Signatures
   a) Putting a signature on a text: sig
   b) Verifying someone's signature attached to a text
5) A walkthrough of the interface
   a) Screens
      - Main
      - Help
      - Key
      - Locks
      - Extra
      - Lock directory
   b) Buttons and Checkboxes:
      Main screen:
      - myKey
      - Locks
      - Select, Clear
      - Help
      - Lock/Unlock
      - Sign/Verify
      - More (…)
      - Lock mode selector

- Short mode
- No tags
- Decoy mode

Help screens:

- Back
- Learn mode

Secret Key input:

- Show
- Make Lock

Extra functions screen:

- Mail
- SMS
- Split/Join
- Words
- Spaces
- Cover
- Images
- File I/O

Locks screen:

- Lock directory
- Save
- Delete
- Reset
- List
- All
- Merge
- Move

Lock directory:

- Find
- Save
- Play
- Invite
- Remove

6) Advanced features
   a) Using fake text
   b) Hiding stuff inside images
   c) Signed messages: mss
   d) Perfect Forward Secrecy: msp
   e) Decoy mode
   f) Locking for multiple recipients
   g) Random Keys and Locks
   h) Locking files
   i) Splitting and joining items
   j) Making sure PassLok is genuine
   k) Lock authentication via text or email
7) Appendix: PassLok vs. PGP

What is PassLok?

PassLok is an encryption program, which means that it turns regular readable text into unreadable gibberish, and back. PassLok does several types of encryption, including public-key encryption.

What is public-key encryption?

You encrypt a text using another text string called a key. A key must be supplied during the decryption process to recover the readable text, or the decryption will fail. Typically, the decryption key is the same as the encryption key, and in this case we speak of symmetric encryption, but there are methods in which one key, called the public key, is used for encryption, and a different, private key is the one involved in decryption. The latter is known as asymmetric or public-key encryption.

Public-key encryption has the huge advantage over symmetric encryption that the encryption key does not have to be kept secret and the decryption key does not have to travel, which might expose it to an attacker. When someone wants to send an encrypted message, he/she only needs to get the recipient's public key from the public place where the owner has posted it, and use it to encrypt the message. Only the owner of the private key, which remains secret and has never left the owner's possession, can decrypt that message. Most secure communications between computers use some sort of public-key encryption.

What makes PassLok special?

PassLok is designed to preserve the privacy of ordinary people, who are not secret agents, *after* they find themselves in a surveillance situation. That means pretty much the whole world, as we learned from the 2013 Snowden leaks on NSA surveillance programs. If they had had time to communicate privately with a friend before the surveillance started, they would have been able to establish a shared secret key that would enable them to communicate using symmetric encryption. There are many good programs out there for symmetric encryption, including some that are based on AES (Advanced Encryption Standard), which the US Government chose in 2001 as its official encryption program, after a yearlong contest involving a dozen candidates, but these people can't use them because they don't have a shared secret with their friends.

All public-key encryption programs, such as PGP (Pretty Good Privacy) and a few others, allow people to establish secure communications even when it has become impossible to transmit a secret, since the secret keys never leave their owners, but what if their computers are bugged? This is a real concern today, and PGP and the rest cannot help in that situation since they must be installed. PassLok can still do it.

What allows PassLok to function where the others cannot is its perfect portability. PassLok is not really installed on your computer; it is not running on a server, either. Since nothing secret needs to be stored anywhere, nothing can be compromised. PassLok runs equally well on your Mac or Linux machine, your neighborhood library's public PC, or a passerby's smartphone, and it disappears without a trace after it does its job.

These are the principles guiding the design of PassLok:

- Perfect portability. Runs on any computer or mobile device.
- Completely self-contained so it runs offline. No servers.
- Absolutely nothing should be installed. Nothing forced to be written in the device.
- Highest-level security at every step. No compromises.
- Easy to understand and use by novices. Simple, clean graphical interface. No cryptographic jargon.

Therefore, PassLok is **a web page** that can run on any browser, whether computer or mobile. It is written in standard JavaScript language, which all browsers understand. It can be saved and run from file, or downloaded from the source when needed, then thrown away. It is a public file. It leaves no cookies. It contacts no servers. All processing happens in the machine, and it works equally well disconnected from the network. Users can encrypt and decrypt while offline and only connect when their messages are encrypted and the plain messages have securely deleted.

Most other public-key programs rely on long private keys that must be stored somewhere because they are impossible to remember; the public keys are even longer. PassLok allows users to use any text as their secret Key, precisely so they can remember it without having to write it anywhere (observe that Key is now capitalized; we'll follow this convention whenever a word has a special meaning in PassLok). It helps them to make strong Keys and rewards them for this, but it doesn't force users to use a particular sort of Key. Once the user has chosen a secret Key, PassLok applies as many as 6,000 iterations of SCRYPT key stretching, and then 521-bit elliptic curve functions to produce the matching public key (equivalent to 15,000-plus bits for PGP and similar programs), and then combines both to use the 256-bit version of AES, the strongest known today.

It is well known that most users tend to choose bad keys or passwords, which a hacking program such as Hashcat can crack easily. PassLok does not force users to comply to rules in choosing Keys, so that a user's secret Key can indeed be anything he/she likes. Instead, PassLok evaluates Key strength and applies a variable amount to SCRYPT key stretching according to the result. Since a large number of key stretching iterations also cause a large delay to the user, this serves to encourage users to come up with stronger Keys, without forcing them directly. What's best, since bad Keys are still allowed, hackers are forced to run through all of them before they get to the better Keys, wasting large amounts of computer time.

If users decide to use much more secure random Keys, PassLok helps in two ways. First, it will generate a long random Key with the touch of a button. Then, should the user decide to store the key for further use, PassLok can split it into several parts to be stored at different locations, so it is harder to compromise the Key, and then can join the parts back together when the Key is needed.

And then, PassLok sports a very clean interface, where the essential buttons are clearly highlighted. Help pops out if requested or if the user chooses Learn mode, so that every button press brings out a dialog explaining what is about to happen.

Keys and Locks

Consistent with the "no jargon" design principle, PassLok replaces the standard talk about public and private keys with simpler terms drawn from users' experience. Most people are familiar with padlocks. To lock something with a padlock, it is not necessary to have the key, but you need the key to unlock something that has been locked. Therefore, the word consistently used in PassLok to designate a public key is "Lock" (capitalized in this document, as every time a common word has a special meaning in PassLok), and the word used for private key is simply "Key".

There are padlocks that use a numerical or alphabetical combination rather than a key. Those wishing to open one of such locks must have the combination, given to him/her by the lock owner. In PassLok, that combination is called a "shared Key," and corresponds to what in regular cryptographic jargon is called a symmetric key, which is used both to encrypt and to decrypt. When a shared secret is made from a private key and someone else's public key, PassLok talks about combining a Key and a Lock to make a shared Key.

Another piece of jargon that PassLok avoids is "hash". One speaks about "displaying the ID" or a text, rather than of making a hash. Since PassLok stores nothing and involves no servers, there is no need to find replacements for things like key rings and key servers.

Creating Keys and Locks

The first task for a new user would be to choose a personal Key, and then make the matching Lock. Then the Lock has to be exchanged with other users. A user may want to change his/her Key, for a variety of reasons. This section describes all of these things.

How to make a strong Key

Since the cryptographic methods used in PassLok are rather overkill for today's computer power, the weak link in the chain is the insecurity of the personal or shared Key chosen by the user. It has been shown that 80% of users choose bad keys, which a specialized hacking program such as Hashcat can guess in a matter of seconds. The same program running for an hour would guess 99% of passwords made by actual users. PassLok knows the worst 500 Keys and their variations, and won't let users select those as their secret Key, but it doesn't stop here. It knows the strategies followed by hackers and makes them as hard and painful as possible.

Since the Lock made from a user's Key is public, a hacker can try to find the secret Key by making guesses and checking every time whether the Lock deriving from the guess matches the user's published Lock. Once a match occurs, the Key has been revealed and anything encrypted or signed with it can be decrypted or tampered with.

It turns out that there are only so many reasons why Keys can be bad. Hackers know them, and this allow them to guess the keys very quickly. If the user avoids making those mistakes, the Key likely won't be cracked. So here's how a typical key-cracking program proceeds, and how to thwart it:

1. The program first tries every possible combination of numbers, letters (lowercase and capitals) and special characters comprising four or five characters. This is called a "brute force" attack, which can only be overcome by making the key longer than those few characters. Six is typically thought to be enough, but PassLok won't be happy until the Key has seven or more characters.
2. Since going beyond this involves an exponentially longer computing time, the program then switches to a dictionary of ready-made words for its guesses, to which it may append or prepend a short number sequence. The dictionary contains all common words in one or more languages (hackers know what languages you speak), including variations like common capitalization (first letter) and misspellings, and letters replaced by numbers or symbols, as in: appl3, b@n@n@, and so forth. The programs also know keyboard patterns like qwerty and qazwsxedc, so these are not secure at all. Finally, any personal information that might be known to others (a birthday, spouse's nickname, an address) is sure to have been added to the dictionary.
3. Having tried all single words, the program will now try two words at a time, beginning with pairs that make grammatical sense: hitme, iwin, etc., then three

or more. It may try conventional phrases at this point: I love you, correct horse battery staple, and so forth.

4. Beyond this point, the hacker's Zen may take him/her/it along different paths, but it will always be following patterns and a dictionary. The user still has a chance to thwart the Zen.

So how does a user make a real strong Key? First of all, recognize how a hacker will try to guess it and don't facilitate his/her/its job. That means:

1. The Key must have sufficient length. PassLok marks as Terrible any Key having fewer than eight characters. PassLok will still take it, but execution becomes painfully slow, especially on a smartphone.
2. Don't use common words, which can be found in a dictionary, even if numbers replace some letters or are capitalized, or are misspelled in a common way.
3. Don't use (exclusively or in conjunction with common words) personal data that others might also know.
4. Adding numbers at the end or at the beginning doesn't add much security.
5. If you have more than one word, make sure the set does not make grammatical sense. This would still be quite easy to remember.
6. Do add a mix of lowercase, capitals, numbers, and special symbols. This will force the cracker to sift through a much larger pool of possibilities, in case he/she/it has the ability to brute force a long Key. This is why websites often force this criterion, at least partially. PassLok does not force you, but you'll be penalized with longer computing times if you ignore this advice.

Examples of Very Good keys:

- Idw2g2s.Thm! (made with the initials, with a couple numerical switches, of "I don't want to go to school. They hate me!)
- 1+1+1+1+1=Five (yes, a math formula; math formulas are full of special symbols; 1+1+1+1=Five is even better, though it is shorter, because it is incorrect)
- c0rrect,H0rse.st@ple;b@ttery (even though correcthorsebatterystaple is well known, and therefore bad as a Key, a different permutation will work, especially if it contains numbers, uppercase, and symbols; there's just too many permutations to keep them all in the dictionary)
- BN32892-3782-GBa (that's the serial number written at the bottom of my old laptop; it's long and random-looking and full of different kinds of characters, and no one else knows it; it will work so long as I'm near enough to read it)
- toSerOderNão2être (funky take on "To be or not to be," combining English, Spanish, German, Portuguese, and French; a hacker might have dictionaries for all those languages, but he/she doesn't know in which order they were used and there are just too many permutations)

Even if you follow all these rules, there is still a chance that, given a lot of computing time and oodles of storage, a powerful attacker might have pre-computed the Locks for a huge database of possible Keys, what is known as a "rainbow table." But it is very unlikely that this database is personalized to defeat you in particular. Thus, appending

some personal data (not necessarily secret) to an already good Key will defeat the rainbow table.

How can you tell if your key is any good? Just write it into the Key box of PassLok, and a text will appear giving you the score. Now, PassLok does not include a dictionary so it cannot tell whether what you type is a real word or not; that's up to you to decide. But PassLok will tell you if it's too short or the keyspace could have been made bigger by adding other kinds of characters. It will not let you get away with things like adding a number at the start or the end. If you choose a known bad Key, it will refuse to budge until you change it.

Generate your matching Lock

After you type your secret Key into the key box, you are ready to make its matching Lock. To do this, just click the Make Lock button. The Lock will appear in the main box, replacing whatever was there before.

You can tell it is a Lock because it begins with a PL**lok tag (where ** is the version number), followed by an equal sign, exactly 87 base64 alphanumeric characters (numbers and lower- and upper-case letters, plus + or / symbols), then another equal sign, and ends with a final PL**lok tag. Despite its great security, it is short enough to be shared within a text message (160 character limit) or posted on Tweeter (140 characters).

Your Lock **is not a secret**; your Key is. There is a one-to-one correspondence between a Lock and its Key, but it is nearly impossible to retrieve the Key from its Lock. To achieve this, other than by guessing the Key until the correct Lock is made, would involve finding a computationally inexpensive solution for the "discrete logarithm" problem over an elliptic curve. No such solution has been found to date. Whoever achieves this is pretty sure to get the Fields medal (equivalent to a Nobel Prize in mathematics), so it's not for lack of smart people trying.

Spreading your Lock

People will need your Lock in order to lock messages that only you can unlock using your Key, so it's imperative that your Lock be publicly known. Think of it as a phone number and treat it as such. You give it to those who you wish to call you back. You can give someone else's Lock to a common friend, too, and nobody gets upset.

Now, a Lock is a bit long and complicated for people to read it off, let alone memorize it, but there are other ways to give it to someone else. As mentioned above, you can text it and you can Tweet it. You can send it by email by clicking the Email button on PassLok, which will open a pre-formatted email with only the recipient's address and the title left to be filled. Pressing the SMS button will open the default texting program if you are running PassLok on a mobile device. You must select and copy the Lock first, however, because standard JavaScript code does not have direct access to the clipboard.

If you don't have connectivity, you can display it as a QR code by copying it to the clipboard and pasting it into a QR code making app (there are many good free ones, such as Red Laser). People now only need to use a QR code reading app in their smartphones in order to retrieve your Lock.

Other ways of spreading your Lock are:

- Add it to your email signature. That way anyone who receives an email from you will also get your Lock. Unlike PGP public keys, which also get spread this way, PassLok Locks don't add much bulk to your signature, barely one line.
- Write it on your business card. Better yet, add a QR code version of it to your business card. People will be able to retrieve your Lock long after you met, and they will have a reasonable assurance that it is authentic (more on this later).
- Post it on social media, as part of your public profile. People will find it easily and use it if they have something confidential to tell you.
- Websites, directories, you name it. The more places have your Lock, the easier it will be for others to find it, and the harder for an attacker to switch it with a counterfeit Lock.
- Upload it to PassLok's official Lock directory (more on this later).

Unless you hand your Lock to someone in person, there's always the chance that an attacker might intercept the communication and replace your genuine Lock with a counterfeit one, to which he/she has the Key. Then he/she will be able to read secret messages meant for you, and then maybe pass on to you something else that suits his/her evil purposes. You'll never know that this person has become the "man in the middle." How can people know that a Lock is genuinely yours, in such case?

This is why PassLok displays IDs. The ID of an item is displayed automatically as soon as you enter the extra functions screen. If the item on the main screen is a Lock, then you display the unique ID of that Lock, which you can read over the phone or within a video call. If the ID shown on the other person's device is the same shown on your end, then everything's good. You could have read the actual Lock to each other, but likely the ID is a lot easier and quicker to read. The ID won't work as a Lock, though, and you cannot get the Lock from the ID. To be able to send you a locked message, the other person must have your actual Lock in his/her possession.

After you have made a Lock that you are going to distribute, it is a good idea to make a video of yourself reading the ID, the Lock itself, or a portion or either one, and post that somewhere online where people can get it. Then you can put the URL of that video in the places where you have posted your Lock, even attached to the Lock itself as an inseparable unit, since it won't affect the functionality of the Lock. Have music playing in the background as you record the video. This way, a sophisticated hacker will have a tough time chopping your original video into pieces and putting it back together so it appears that you are reading a different ID.

If it is impossible to use a rich channel such as voice or video to authenticate a Lock, there are still ways to detect by text or email if a man-in-the-middle is intruding into your conversation. It is explained in the Advanced section.

Changing Locks

Let's say that, despite your great care to keep your Key secret, never writing it down anywhere and keeping it masked in PassLok when using it, you fear it has been compromised. Or maybe you want to make a stronger Key, or simply got tired of the old one. How does one handle this situation? Programs like PGP have a complex system of expiration dates, revocation certificates, etc. What does PassLok have to let people know that a given Key, and therefore the Lock derived from it, is no longer good?

One word: nothing. Because, if you stop thinking about it, nothing is needed, really. What do you do when you change your phone number? Does your phone number have an expiration date? Do you need a certificate to let people know that it has changed? No? Same with PassLok Locks.

When you change your phone number, you notify first those who are most likely to use it: family, friends, co-workers, merchants, authorities. Then you *might* post the new number in a public place, such as a website or a social network profile. But not necessarily. Very likely, knowledge of the new number will spread little by little, as you and your friends tell others. Those who never find out are probably best left in the dark. Same with a Lock.

Now, it may be that someone sends you a message locked with an old Lock. You will know when you are unable to open the message with the new Key. But, unlike physical locks that can no longer be opened when the key is thrown away, you can always unlock those messages using the old Key, provided you still remember it. If you don't and are still curious about what the message says, you can always send a message back to the sender, giving him/her the new Lock.

Making a Lock database

As you collect more and more Locks for people whom you wish to send private messages, it can get unwieldy. What is a good way to keep all those Locks straight?

At the risk of repeating myself once too many, let me say it again: a Lock is a lot like a phone number. Whatever works for a phone number will probably work for a Lock, too. You can store Locks in the same online database that you are using right now for phone numbers, email addresses, etc. For instance, the Gmail Contacts database has the ability to add custom fields. Why not add a "PassLok" field for each person whose Lock you obtain, and put the Lock there? Locks are not secret information, so it doesn't matter very much that someone might be able to read it at some point. What matters is that the Locks be free from tampering by third parties, which most web mail providers swear is impossible in their systems.

If you need a Lock to lock something for somebody, a trick that I found works quite well is to keep your webmail contacts database open in one browser tab while PassLok is open in another. Navigate the address book to the individual in question, and copy the Lock into the clipboard. Then go to the tab containing PassLok and paste the Lock in the key box. You're set to go.

If this weren't easy enough, beginning with version 1.6 PassLok can keep its own Locks database within the program itself, and it doesn't get deleted when the program is closed or even if the browser's cache is flushed. When you want to store somebody's Lock, you only need to click the **Locks** button on the main screen, type a unique name in the smallest of the two boxes that appear, then paste the Lock into the larger box, and click the **Save** button. If you have previously entered your secret Key through the **myKey** button, the Lock will be permanently saved, encrypted for good measure.

To retrieve that Lock so it's ready to be used, you only need to get back to that screen via the **Locks** button, and start typing the name you gave that Lock until something appears in the large box. The full name should also appear above the small box. So long as your secret Key has been entered, the Lock will be decrypted and be ready to use for locking messages or verifying signatures.

Your Locks database doesn't need to stay with the particular device where you entered the Locks. PassLok has a nifty **Move** button which will take the entire database, encrypt it, and display it on the main screen, ready to be copied, emailed, or whatnot. PassLok will also offer to delete the database from the device at this point, should you want to cover your tracks on that device.

Even the most user-friendly PGP-based programs, such as Mailvelope, take more steps in order to do the equivalent thing, because PGP public keys are so large. What's worse, you don't see what's being done, so you have to trust that whatever they use for storage is safe and free from intrusion.

The general Lock directory

Starting with version 1.7.03, PassLok has an official web-based directory where people can upload their Locks for others to download. It is accessed from the Locks screen, by means of a big button at the top, labeled **Lock directory**.

Let's say you want to lock a message for your friend Alice to read, but you don't have Alice's Lock. You head to the Lock directory and type Alice's email address on the top box (alice@wonderland.org), and then you type Enter or click the **Find** button on that screen. If Alice has uploaded her Lock to the directory, it will appear in the lower box in the blink of an eye. Then you can select it and copy it, so you can paste the Lock in the Locks screen and then you can lock that message for Alice.
Now, you may wonder whether this is really Alice's Lock, or maybe someone else put that Lock in the directory, in which case I may be locking messages for that someone

else to read. Not good. The Lock directory has two features that help prevent this scenario:

1. Uploading a Lock to the directory involves replying to an email containing a special link, without which the upload is not complete. Someone who wants to upload a fake Lock for Alice would also have to intercept the email that goes to alice@wonderland.org, in order to click on that link.
2. The directory has space for adding an authenticating video to each Lock, instructions on how to make this video, and a button to view it. If Alice has added that video, you can just click the **Play** button and watch Alice reading a portion of her Lock, or of her Lock's ID (explained elsewhere in this manual).

The video is actually uploaded to one of many video-hosting websites, such as YouTube or Vimeo, so what is attached to the Lock is a short web address, which becomes a part of the Lock and doesn't affect its function.

Let's say now you want to add your Lock to the directory so that others can find it. You open the Lock directory page, write your email address in the top box, and the Lock in the bottom box; then you click the **Save** button. A message tells you that an email has been sent containing a link that you must click in order to complete the process. So you open your email client, which should be getting that link any time now. Sometimes email clients are overzealous filtering spam, so you may want to check the spam folder if the email takes to long to show up. When you get the link, you click on it and another page tells you that the process is complete.

The video can be added at the same time, by placing its URL on the line below the Lock, or later on. You can update your Lock as many times as you want, and in each case the update won't happen until you click on the confirmation link sent by email. There is also a **Remove** button that you can click after filling in your email address if you wish to remove your email and its associated Lock from the directory (email confirmation required).

Finally, if somebody's Lock is not yet in the directory you can remind that person that people would like it to be listed there. There is an **Invite** button that produces a ready-made email with instructions on how to upload a Lock. Unlike the other emails, this one comes from your own account, so the invited person can feel better reassured.

The Lock directory looks a lot like the rest of PassLok, except for a different color scheme. This is to remind the user that he/she is browsing a different page, and that the result of the search does not get automatically transmitted to the encryption code. Copy and paste is necessary. The reason for the separation is to keep the encryption code without any communication with a server, since the Lock directory page must necessarily remain connected to the server where the general Lock database resides. It's a small inconvenience to put up with in exchange for airtight security.

Locking and Unlocking

Having covered the essentials of what PassLok is and how to take care of the most critical ingredient to maintaining security, we go now to the nuts and bolts of actually locking and unlocking messages. Despite its tiny size, PassLok is one complex piece of software, comprising four different locking modes, with one variant and a secondary mode for each of them. We'll start with the one most likely to be used, and then introduce the others.

Anonymous locked messages: msa

When a message is locked with the recipient's Lock, so that only he/she is able to unlock it with his/her secret Key, the result is an anonymous locked message. It looks like a piece of gibberish, bracketed by PL**msa tags, where ** is the version number.

This is what you do to lock a message in this mode:

1. Fetch the recipient's Lock and paste it in the large box accessed with the **Locks** button. He/she would have sent that Lock accompanying an email, or in and SMS text message, or posted it publicly, or displayed it as a QR code, which you scanned. You need this Lock before you can do anything else. It is okay if the tags up to the "=" signs are missing, or extra spaces, carriage returns, or special characters other than = + or / have been added. If you have that Lock in your permanent database, start typing its name in the small box until the (encrypted) Lock appears in the large box. It will be decrypted, ready to use, provided you have entered your secret Key through the **myKey** button.

2. Write or paste your message in the main box. Click the **Lock/Unlock** button. The locked message will appear in the main box, replacing the original message. It will normally include identifying tags at both ends, unless the **no Tags** checkbox has been checked prior to locking. Copy it and paste it into your communications program or click **Mail** to open your default email. If you click the **Images** button, it will be displayed as a cellphone-scannable QR code in a new screen.

If you have received a locked message with PL**msa tags, here is what you do to unlock it:

1. Write your secret Key into the key box, which is accessed by pressing the **myKey** button. It is masked by default, so if you want to display it, check the **Show** checkbox.

2. Paste the locked message in the main box. It is okay if it is broken up by spaces, carriage returns, and special characters other than = + or / or is missing its tags. Then click the **Lock/Unlock** button. The unlocked message will appear in the main box, replacing the locked message.

If the unlocking fails, this is likely because the locked message has been corrupted (make sure the tags are intact) or because the Lock used to lock it does not match your secret Key. Remember one more thing, though: since the Lock is public, you won't necessarily know for sure who has sent the message. Proceed accordingly. In any case, this is likely to be the most common mode used because it does not require those sending a message to have any secret in common with those receiving it.

Key-locked messages

In PassLok, a shared Key is a code that both the sender and the recipient know. This means that it must have been shared beforehand for the exchange to work. If you lock something with your secret Key, for instance, the recipient won't be able to unlock the message because he/she does not have your Key (remember that you should *never* give your secret Key to anyone, or even write it down). So don't use your secret Key, but a different Key that you can afford to share with someone else. All that was said above about making good secret Keys applies to making shared Keys.

In this mode, the locked gibberish ends up bracketed with tags reading PL**ms*, where ** is the version number and the final character tells you the type of Lock-locking used, if any. The tags don't tell you if a shared Key was used for locking the message, but hopefully you can figure that out, knowing the sender, because you share a secret with that person (the shared Key). PassLok will omit the tags if the **no Tags** checkbox is checked prior to locking. Here's what you do to lock a message with a shared Key:

1. Write or paste the shared Key in the large box that appears when you click the **Locks** button. If the shared Key is on your database, start typing its name into the small box until something appears in the large box and the correct full name is displayed above everything. Make sure your secret Key has been entered via the **myKey** button so the shared Key can be decrypted and can be used.

2. Write or paste the message in the main box. Click the **Lock/Unlock** button. The locked message will appear in the main box, replacing the original text. Copy it and paste it into your communications program or click **Mail** to open your default email. You can also do the QR trick described above.

If you have received a message that you suspect is locked in this mode, here's what you do to unlock it:

1. Write or paste the shared Key in the **Locks** large box. If it is on the database, start typing its name as described above; you'll also need to enter your secret Key with the **myKey** button.

2. Paste the locked message in the main box. It is okay if it is broken up by spaces, carriage returns, and special characters other than = + or / or is missing its tags. Then click the **Lock/Unlock** button. The unlocked message will appear in the main box, replacing the locked message.

Unlike the previous mode, you would know who sent you a Key-locked message, because only people knowing the shared Key, which you also know, are able to lock the message to begin with. If the unlocking process fails, this is usually because the locked message has been corrupted (make sure the tags are intact), or because the Key used for locking is not the same you tried for unlocking. Unfortunately, this mode is hampered by the need for a previously shared secret between sender and recipient (or more people, if that should the case). One way to get around this quandary is to use the anonymous locking mode to exchange a new shared Key between the parties, and use the Key-locked mode from then on. Once you receive a shared Key, you can store it within PassLok itself in the same way as you would store a Lock. Just click the **Locks** button, supply a name for the item in the small box, paste the shared Key in the large

box, and click **Save**. To retrieve a shared Key from the database, just start typing the name associated with it in the small box until the full name appears above it.

Short locked messages

Both anonymous and Key-locked messages are at least 200 characters long, which wouldn't fit in a cellular text message (SMS), which is limited to 160 characters. PassLok has a special mode for situations when you want the locked message to fit within an SMS, which is invoked by clicking the **Short mode** checkbox on the main screen. Because of the length limitation, the size of the text that can be locked is also limited.

In order to fit a message as long as possible, the locked message will have no tags. You can tell what it is, however, because the resulting gibberish is exactly 160 characters long (159 in PFS mode). Here's what you do to lock a message in this mode:

1. Check the **Short mode** checkbox at the bottom of the main screen.

2. Write or paste your message in the main box. Message length is limited to 58 ASCII characters if locking with a shared Key, 38 if locking with a Lock in anonymous mode. Non-ASCII characters use 6 spaces each, so avoid them if you can. Any text beyond the limit will be lost.

3. You will need to have entered a shared Key or the recipient's Lock in the Locks screen, which is accessed by clicking the **Locks** button. After this is done, you can return to the main screen and click the Lock/Unlock button. The locked message will appear in the main box, replacing the original message. Copy it and paste it into your communications program.

4. If you are using a smartphone, the view will switch to the extra screen, with the locked text selected and ready to be copied to the clipboard. If you want to send it using the texting app, copy it, and then click the SMS button above the text box, which will open the texting app. Now you only have to type in the recipient, and paste in the locked message.

Once you receive a 160-character long piece of gibberish, which likely is a short locked message, here's what you do to unlock it:

1. Paste the locked message into the main box. It is okay if it is broken up by spaces, carriage returns, and special characters. As for regular unlocking, you also need to have entered your secret Key in the key box or, if it was locked with a shared Key, to have that Key displayed in the Locks screen.

2. Then click the Lock/Unlock button. The unlocked message will appear in the main box, replacing the locked message.

Be aware that the length of the message is very limited in this mode. If you want to send something long by SMS, one way to get around this limitation is to store it as a file in one of many anonymous cloud storage services, get the short URL to the file, and send that, locked, by SMS.

Signatures

Digital signatures are the other use of Keys and Locks (asymmetric keys), which is rather backward from locking. In locking, a message is locked with a Lock and unlocked with the matching Key. On the other hand, a message is signed with a Key, and then that signature is verified with the matching Lock.

Because one needs the Key, which is secret, to make a signature, anyone verifying the sign can ascertain that the text was indeed seen, and somehow attested to, by the owner of the Key. It's like signing the text, which is why this is called "digital signature" in specialized jargon. The tool used to verify a signature must be something that is not secret, so that anyone can have access to it, and which can distinguish the right Key from all the rest. That tool is the Lock made from that Key.

One thing is different in signing/verifying from locking/unlocking, and this is that the signed text does not turn into gibberish. Rather, the signature is a separate piece of gibberish, exactly 250 characters long plus PL**sig tags at either end (again, ** is the version number), which PassLok adds at the bottom of the signed text. The text itself remains perfectly readable. If one wants to lock it, it can always be done after that, either with a Lock or a shared Key. PassLok will omit the tags if the **no Tags** checkbox is checked prior to locking.

To verify that the signed text has indeed been signed with someone's secret Key, it is necessary to maintain the original text without any modifications, as well as the signature. Any changes made to the text, even if it is only a space somewhere, will cause the verification to fail. The signature itself is a little more resilient and admits to be broken up and lose its tags, but obviously if some of it is missing or corrupted the verification will fail, too.

Putting a signature on a text

Unlike in locking/unlocking, there is only one way to make and verify signatures.

Here is the process for making a signature:
1. Write or paste your secret Key in the key box. It is masked by default, so if you want to display it, check the **Show** checkbox.

2. Write or paste the text to be signed in the main box. Click the **Sign/Verify** button. A signature matching the text and the key will be appended at the end of the text in the main box. Copy it and use it as appropriate. If you click **Mail**, the text with its signature will be placed into an email using the default program. It is okay to strip the tags up to the "=" sign, but not recommended. PassLok will omit the tags if the **no Tags** checkbox is checked prior to signing. It is also okay to split the sign with spaces, and punctuation other than line returns or = + or /.

Verifying a signature

The process for verifying a signature affixed at the end of a text is this:
1. Paste the Lock of the person who allegedly made the signature into the Locks screen. It is okay if the tags up to the "=" signs are missing, or extra spaces, carriage returns, or special characters other than = + or / have been added. If you have that Lock in your stored database, you can retrieve it by typing its name into the small box in the Locks screen.

2. Write or paste the text with its signature appended at the end in the main box. It is okay if the signature is broken up by spaces and special characters other than = + or / or is missing its tags up to the "=", but it should not be broken by carriage returns. Then click the **Sign/Verify** button. A popup message will say whether or not the signature for that text has been verified.

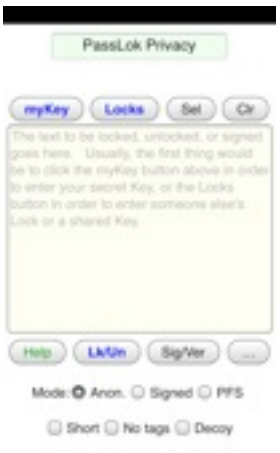
In addition to serving as a digital sort of signature, this process can be useful for assuring the recipient of who locked a message in anonymous mode. This is a very common use of digital signatures in programs like PGP, where the text is first signed, then encrypted. PassLok, however, has a better way to identify the sender, which is explained in the Advanced section.

A walkthrough of the interface

We now describe all the objects on the PassLok screens and what they do.

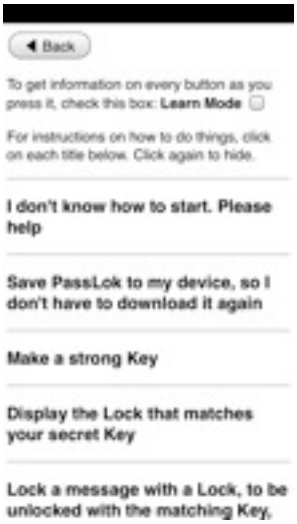
Screens

Main



This is the screen containing the "PassLok Privacy" title, at the top, plus a large box and a few buttons. If the background of the PassLok title is green, everything is OK and the computations are able to run. If not, the background will be red and a warning message appears instead of the title. PassLok is not operational if this is displayed. Probably JavaScript support is turned off in the browser (go to its settings to fix this), or the browser or the device itself are just not capable (use a different browser or a different device).

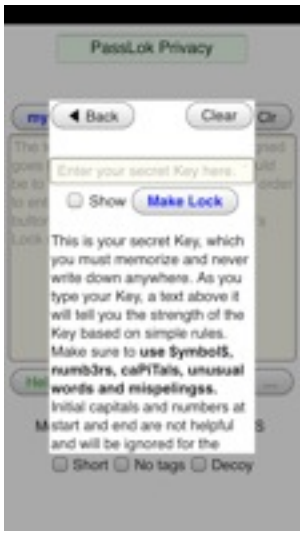
Help



This screen is displayed by clicking the **Help** button on any of the other screens. It contains context-sensitive instructions on how to do things. To know how to do a particular action, click on the corresponding title.

There is a **Learn mode** checkbox near the top of the main Help screen. When this is checked, PassLok displays a warning message, explaining what is about to happen and seeking confirmation, every time a button is pressed.

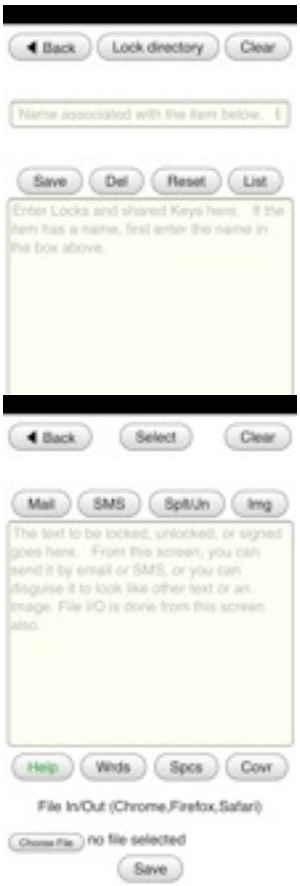
Key



This screen is displayed by clicking the **myKey** button on the main window. Here is where you input your secret Key, which should be shared with no one. The text is masked by default, but you can see it by clicking the **Show** checkbox. As you begin to type into the key box, words appear above it telling you PassLok's opinion on the strength of your Key, based on the kind and number of characters used. It cannot detect whether the text might be in a cracker's dictionary, so make sure you obfuscate or combine words as described in the previous section. If you want a more sophisticated indicator of how good your Key is, you may want to spend some time with ZXCVBN. The screen also displays some

advice on how to choose a strong Key.

Locks

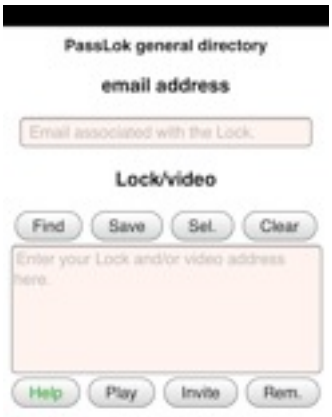


When the **Locks** button is pressed, a screen appears containing two boxes: a small one at the top, and a large one below it. This is the screen where you enter Locks and shared Keys, always into the large box, in order to lock, unlock, or verify messages. Since PassLok can store those items for future use, it has a small box where the user writes a name for the item. The buttons around the boxes are used to save, delete, reset, or do something else with the stored items, whether one at a time or all together.

Extra functions

This window appears by clicking the **(. . .)** button at the bottom of the main screen. It is essentially a copy of the main screen, displaying more buttons for functions beyond basic locking, unlocking, and signing. Whatever appears on this screen gets copied back to the main screen when the **Back** button is pressed.

Lock directory



This window is actually not part of the PassLok code, and this is why it has a different color scheme, but is closely integrated with it. The Lock directory interfaces with a database engine located at PassLok's main server, which allows people to find Locks they don't have in their local databases, by supplying the email address of the Lock's owner.

Buttons and Checkboxes:

Help

This button opens the Help section corresponding to the screen you are in.

myKey

This button opens the Key screen, where you enter your secret Key.

Locks

This one opens the Locks screen, where you enter Locks, shared Keys, cover texts, and other items you may want to store within PassLok.

Select

Selects the complete text in the box below, so it can be copied, cut, or erased. This button is not present in the Key box, for security reasons.

Clear

Clears the box below. This action is irreversible, so be careful.

Lock/Unlock

This button has context-sensitive functionality. If PassLok does not find a locked message in the main box, it will lock the contents of the main box with whatever is in the large box of the Locks screen, in this way:

- If the box contains a Lock, it will make a regular locked message, to be unlocked with the recipient's Key. The default is the anonymous locking mode described earlier, but there are two more advanced modes that are described below.
- If the box contains something else, PassLok will assume it is a shared Key and will make a Key-locked message with PL**msk tags (omitted if Tags is unchecked), to be unlocked with the same shared Key.

If a locked message occupies the main box when the **Lock/Unlock** button is pressed, PassLok will attempt to unlock the message, depending on the type of locked message it finds. In all cases, the successfully unlocked message ends up replacing the locked message originally in the main box:

- If it is an anonymous locked message (PL**msa tags), PassLok expects to find in the Key box the Key matching the Lock used to lock the message. It will return a warning if this is not found.
- If it is a Key-locked message (PL**msk tags), PassLok will take the string in the large box of the Locks screen and use it as a shared Key to unlock the message. Again, different kinds of warnings will be displayed instead if something goes wrong.
- If it is a signed message (PL**mss tags), PassLok will expect your secret Key in the Key screen and the sender's Lock in the Locks screen. If it doesn't find either one, warnings will be displayed and the unlocking will fail.
- The case of a PFS message (PL**msp tags) is similar to that of signed messages: both your secret Key and the sender's Lock are expected. Additionally, there is secret information that is stored in the database between messages; this information must remain current for the unlocking process to be succeed. Old PFS messages cannot be unlocked even if the correct Key and Lock are supplied.
- If the message has no tags and is 160 or 159 characters long, PassLok interprets it as a short message. It will then determine what type of locking was used and then try to unlock it using the shared Key in the Locks screen, your secret Key, or both your secret Key and the sender's Lock in the Locks screen, as with normal length messages. As usual, helpful comments are displayed if unlocking fails.

Finally, if there are no tags and the message is not 160 characters long, PassLok will fail to recognize it as a locked message and will attempt to lock it instead. So please make sure the tags are readable if the main box does indeed contain a locked message. It may also happen that your unlocked message is exactly 160 characters long, and PassLok insists on unlocking it. This is easily solved by adding or removing a character somewhere; a space is enough.

Sign/Verify

This button also has context-sensitive actions. First PassLok tries to find a signature at the bottom of the main box. If it does not find it, it concludes that you want to sign the text, and will do so provided you have written a valid Key in the Key box, otherwise it will complain. If PassLok finds that a signature is already there, it will attempt to verify it with the Lock present in the Locks screen, which should match the signing Key. A message will say whether or not signature verification has succeeded.

More (…)

This button transfers the contents of the main window into the extra window, where functions dealing with communications and stealth are available.
When the new screen appears, a message appears above the text-containing box, displaying the near-unique SHA256 hash of the box contents in hexadecimal code, formatted in groups of four characters. If the box contains a PassLok string having fewer than 250 characters (likely, a Lock or a signature), its tags, spaces, and extraneous characters are removed before taking the hash. This is useful to authenticate someone's Lock, for instance, since it is much easier to read the ID over the phone than the Lock itself. Bear in mind, however, that it is impossible to recover the original information from its ID.

Another likely use of this feature is for checking the integrity of PassLok itself. To do this, view the source of PassLok in a separate tab (every browser does this differently), copy it, and paste it in the main box, then click the **More(…)** button. The resulting ID should match the one published in the mirrors.html file (displayed with the Help button), for the same version of PassLok.

Short mode

When this box is checked, PassLok knows that locked messages must be at most 160 characters long so they can fit within an SMS text message. Tags are stripped and the message itself is truncated so it can fit within the limit, which is 58 ASCII characters for Key-locking, 38 characters for regular locking. A message above the text box tells the user how many characters are left before the message is truncated. It is not necessary to check this box for unlocking.

No Tags

If checked, the usual tags of the form PL**###, where ** is the version number and ### is a string identifying the item, are omitted at the beginning and the end of every Lock, message, signature, or part generated by PassLok, and emails invoked by the **Mail** button do not include text to help the recipient identify the contents. If unchecked, tags and additional text are added.

Decoy Mode

Checking this box, located below the main box, causes PassLok to expect a second, hidden message in addition to the main message in the main box, locked under a different shared Key. When you are locking or signing, a popup will appear asking for the special decoy Password and the hidden message. In unlocking or verifying, a popup asks for the decoy Password. The hidden message is displayed above the main box if it is successfully unlocked. This mode is especially useful if you suspect that you or someone else may be forced to relinquish his/her key. There is no way short of successful unlocking to tell whether or not a hidden message is present.

Make Lock

This button is in the Key screen. It makes a Lock matching whatever is entered in the Key box, and displays it in the main box. PassLok will take anything in the Key box as a secret Key to derive a Lock from, but it will alert you if the box contains a Lock. The Make Lock operation can get very slow if the Key strength indicator says it's Terrible or Weak. This will happen every time that this Key is used, which is quite frequently, so this is the moment to choose something else.

If the Key box is empty when this button is pressed, PassLok generates a 86-character random Key and places it in the Key box, then displays its matching Lock in the main box. This is useful if you want a throwaway, very secure Key for a chat session, so that it can be forgotten when he session is over, or are planning to store securely somewhere else. PassLok does not store secret Keys.

Mail

This and the next few buttons are in the Extra screen. When this button is pressed, the contents of the box are formatted into an email message, open in the default webmail program known to the browser, with only title and recipient's address left to be filled. All of this happens in a new tab, so PassLok remains open in its tab. This will only work if PassLok finds a Lock, a locked message, or a signed text in the main box, so you don't accidentally email plain text.

SMS

This button does nothing in regular computers, but in mobile devices it opens the default SMS (texting) app. Since nothing is copied automatically from the

PassLok page, make sure to copy into the clipboard whatever you want to send before pressing this button.

Split/Join

When this button is pressed with the main box empty (you'll probably need to go back and clear it), a popup asks for the number of parts needed to retrieve the contents of the box. PassLok performs a Shamir Secret Sharing algorithm to generate those parts, which are then displayed in the main screen. Extra parts can be made by pressing this button repeatedly. These parts can now be stored in different places or sent to different people. If a sufficient number of Key parts are present in the main box when this button is pressed and the extra box is empty (again, you may need to clear it manually), the same algorithm combines them to recreate the original text, which is displayed in the extra box.

Words

When this button is pressed, any PassLok item in the box is converted into fake text, useful to avoid detection by email scanners, replacing each character with a word taken from the current cover text. To retrieve the original item from fake text, place it in the box and click this or the **Spaces** button. You must use the same cover text that was used to make the fake text.

Spaces

This button does a similar thing to the Word button, but rather than encoding the characters as words, it encodes them as spaces between words in the cover text. Seven words are needed for each character, so this makes fake text that is seven times longer than that made by the Words button, but it does not require having the same cover text loaded in order to retrieve the original.

Cover

Pressing this button with the box empty displays the current cover text. If the box contains text, it is used as new cover text when this button is pressed, provided it is sufficiently long, and then the box is erased. Even though the default cover text is in English, the new cover text can be in any language compatible with UTF-8 encoding. Cover text changes are not permanent, so that when PassLok is reloaded, it goes back to the default English text.

It is possible to store a cover text within the permanent PassLok database. To do this, go to the Locks screen, enter a name for the cover text in the small box, paste the cover text in the large box, and click **Save**. To retrieve a cover text, start typing its name in the small box of the Locks screen. PassLok will recognize it as a cover text and use it automatically.

Images

This button opens a new screen where images are displayed. When this button is pressed, the new screen displays a QR code representing the contents of the extra box. This is useful if you want to pass a Lock or similar item to someone else around you, without being online. The other person only needs to use a smartphone app for scanning QR codes, directly from your screen, in order to retrieve the item.

The images screen also contains a few buttons, which are explained below.

Choose File (images screen)

This button opens a dialog that allows the user to select an image (jpg, gif, bmp, png are all OK) which will be used to hide the contents of the extra screen. It can also load an image that contains a hidden PassLok item, so it can be revealed.

Hide

When this button is pressed, the contents of the extra screen are hidden within the least significant part of the image pixels, so it is indistinguishable to human eyes. You can then save the image by right-clicking on it (long press, on mobile devices), and selecting “save as…” from the menu that appears

Reveal

This button does the reverse of the **Hide** button. If the image displayed on screen contains a hidden PassLok item, pressing this button extracts it and writes it in the extra screen, or otherwise a message tells you that there is nothing to reveal. This function currently does not work on mobile devices.

Choose File (extra screen)

Similarly to the button of the same name on the Images screen, pressing this button opens a dialog to select a file (an image only, on unrooted mobile devices), to load into PassLok. It can be any kind of file. The file is then converted into base64 text and loaded into the extra screen, ready to be locked.

Save

This button does the reverse of the button above. When an encoded file (which looks like gibberish text, with some tags at the start) is shown in the text box, pressing this button converts it back into a file, which is then downloaded or displayed in the device. How the file is downloaded or displayed depends on the browser and the type of device.

A secondary function of this button is to show the authenticating video attached to a Lock. Just place the Lock with the video URL attached to it (or just the video URL) in the extra screen, and the video will play upon clicking this button.

Lock directory

This and the remaining buttons are found in the Locks screen, which opens when he **Locks** button is pressed. This button opens the Lock directory screen in a separate browser tab, so that users may copy and paste between the directory and Locks screens with ease.

Save

This button saves the current contents of the big box into PassLok's permanent database so it can be retrieved by typing the name in the small box. The item is stored encrypted by the secret Key, so a Key must have been entered previously by clicking the **myKey** button.

Delete

This button deletes the item whose name is displayed above the small box (not necessarily what the small box says, since name recognition is gradual), from PassLok's permanent database. This is irreversible and there is no confirming dialog, so be careful.

Reset

This button does a similar thing to the **Delete** button, but rather than deleting the complete item, it only deletes the secret information pertaining to PFS mode operation. This is useful if a PFS conversation goes out of sync and must be restarted.

If a List is displayed in the big box, clicking this button resets the current temporary List instead, but does not affect permanent storage. This is useful when one is editing a List or making a new one.

List

Clicking this button adds the contents of the large box to the current temporary List, removing any duplicates (case sensitive). If the large box is empty the current List is displayed instead. If what is displayed in the large box is an item from the permanent database, clicking **List** adds its full name to the current List, rather than the item itself.

This is useful for making a List, which will be used for locking messages for multiple recipients. To save the List in the permanent database, you must first write a name for it in the small box, then click **List** to display the List in the large box, and then click **Save**.

All

When this button is clicked, the complete stored database is displayed in the large box. The format is as follows: name of the item followed by a colon, the item itself (encrypted with the secret Key) on the following line, additional lines containing additional (encrypted) data, blank line before the next item. Lines are indented for display purposes but this has no effect on the parsing.

The main use of this button is making sure that a particular item exists in the database, since it is not necessary to know its name beforehand. It can also be used to edit the database in large chunks. The **Move** button implements a method for copying the entire database so it can be used in a different device.

Merge

When you click this button, the contents of the large box are merged into the permanent database provided it is formatted correctly, as described for the **All** button.

If the large box contains a Lock when this button is pressed, it will be combined with whatever single-line item is in the main screen, to make into an 86-character shared Key, which appears in in both boxes. The same thing happens if the main box contains a Lock instead, and the large box on the Locks screen contains a single-line item other than a Lock. This is a manual implementation of the Diffie-Hellman key exchange algorithm, which is at the core of many PassLok functions.

Move

Sometimes you want to use your stored Locks and shared Keys in a different device, or want to stop using a particular device and clear its permanent database. This button implements both. When it is clicked, the complete database is encrypted with the secret Key and the result is placed on the main screen, with identifying tags of the form PL**db. Then a popup asks the user whether to proceed with erasing the database from permanent storage. If the user cancels at this point, the database is preserved and its encrypted form remains on the main screen, so it can be copied and transferred elsewhere. If the user clicks **OK**, the database is erased, but its encrypted form remains on the main screen, from where it can be transferred or simply cleared.

To bring an encrypted database into a new device, simply paste the encrypted database into the main box and click **Lock/Unlock**. If the correct secret Key has been previously entered, the complete database appears in the Locks screen. Clicking **Merge** at this point saves it into permanent storage.

Find

This and the next few buttons are in the Lock directory screen, which is actually a page separate from the PassLok code. When the **Find** button is pressed, the email written in the top box is searched in the general directory, and the Lock corresponding to it (plus a video URL, if there is one) is displayed in the lower box. Typing Enter after the email address has the same effect.

Save

A user can put his/her Lock in the general directory by writing his/her email address in the top box, the Lock in the lower box (with an optional video URL on the line below it), and clicking this button. The process is not completed until the user clicks on a link contained in an email, which the directory sends to the given address.

Play

When a Lock in the directory has a video URL attached to it, clicking this button opens a new tab where the video is played.

Invite

By means of this button, a user can send an email preformatted with a set of instructions for downloading PassLok, making a Lock, and putting it in the directory, to the address in the top box. The email is sent from the user's own email account.

Remove

This button is similar to the **Save** button, but rather than adding a Lock or updating it, confirming the email that follows deletes all traces of the email and its Lock from the directory.

Advanced features

PassLok goes beyond simple locking/unlocking and the making and verifying of signatures. This section presents some features that, though maybe not used by everyone at first, may be important at some point.

Using fake text

PassLok can produce very secure locked messages, but it is obvious to anyone who looks at them, including scanning bots, that they are not normal text. This can alert an enemy that an encrypted communication is taking place, which might lead to unpleasantness.

To help with this problem, PassLok includes a fake text encoder, beginning with version 1.4. Those functions are available after clicking the **More (…)** button below the main box. Here's how it works:

To convert a PassLok item (Lock, locked message, etc.) into fake text:

1) Check that the item to be converted into fake text is in the main box. Since PassLok tags, which always contain the same characters, will lead to the same set of words at the start and end, you may want to remove those tags before making the fake text. This is achieved by having the No Tags checkbox checked before creating the item. Most functions in PassLok work even if the tags are missing.
2) If you wish to make text that is not English, you will have to change the default cover text using the process described in the next item.
3) Click the **More(…)** button, and then click the **Words** button or the **Spaces** button. **Words** takes every character and replaces it with a word from the cover text, resulting in gibberish text; the recipient must load the same cover text in order to recover the original item. **Spaces** encodes the text within the spaces of the cover text, using seven words per character; this results in a much longer output, but the result is readable and does not require the recipient to load the same cover text in order to retrieve the original item. If everything goes well, the contents of the main box are converted into fake text using the current cover text and displayed in the box, replacing the previous contents. If the cover text is not long enough, a popup warning will be displayed. Another thing that will cause the process to fail is if the item you want to hide is not a valid PassLok-generated item (plaintext, for instance). The hiding process provides no real security, and so PassLok will not do it unless the message has been processed first.
4) You can now email the fake text, which to an email scanner will be nearly indistinguishable from real text. You can change the punctuation and merge or split lines without changing the encoded material. Do not alter any spaces within the encoded text, however, if it was encoded with the Spaces button.

To retrieve the original PassLok item from fake text:

1) Check that the fake text is in the main box.

2) If the fake text is not English and was encoded with the **Words** button (so the text doesn't really make sense), you will have to change the cover text using the process described in the next item.
3) Click the **More(…)** button, and then either the **Words** or the **Spaces** button. PassLok will detect the method used and convert the fake text in the box back into the original item, replacing the fake text.

Now, PassLok's default cover text is a piece of English text (taken from the GNU 3.0 license). If the context of the channel where you are going to place the item is very different, it might stand out and be detected by a scanner. It will also be detected if a different language is expected. Can PassLok get around this problem?

Simple: replace the default cover text with something else. To do this, follow these steps:

1) Copy a sufficiently long text (it must have at least 70 different words, for the Words method, or 7 times the number of characters to be encoded, for the Spaces method) and paste it into the box, then click the **Cover** button.
2) If the change is successful, the box will be erased and a message will be displayed. Typically, failure to change the cover text is due to not having a sufficient number of different words. Use a longer text and try again.
3) The recipient of your messages turned into fake text must have the same cover text installed if items are encoded with the **Words** button. One way to ensure this when using a non-English language is to display the default cover text, copy it into a translation utility such as Google Translate, and then use the translation as the new cover text.

Be aware that the cover text will go back to its default if PassLok is reloaded. If you want to make the change permanent, the easiest thing to do is edit the PassLok source itself. The default cover text is a large piece of prose, quite easy to spot near the end of the source code. You can also display whatever cover text is active by clicking the **Cover** button with the text box empty.

It is possible to store often-used cover texts within PassLok's permanent database. To do this, go to the Locks screen, type a name for the cover text in the small box, then paste the cover text in the large box, and click **Save**. A stored cover text will be automatically loaded once you type its name in the small box of the Locks screen.

Also be aware that the Words encoding only capitalizes words if they follow a period. If you are writing in German or some other language with more frequent capitalization, the result won't be satisfactory with the Words method. Unfortunately, PassLok has no way to know the language of the cover text. If you dare to edit the source, one quick fix for German would be deleting the ".toLowerCase()" strings in the appropriate section of the source, and disabling the instruction that capitalizes the output text. This instruction can be found easily since it contains two ".toUpperCase()" calls.

Hiding stuff inside images

Beginning with version 1.6, PassLok includes a method to hide its output within images, so it is invisible to human eyes. It does so by replacing the least significant color information of the pixels with the material to be hidden. This function is accessed by clicking the **Image** button.

Let's say PassLok has locked a message, which you now wish to hide within an image. Click the **More(…)** button, followed by the **Image** button. The first thing you will see is a QR code containing the locked message. This is hardly any stealth at all, but it useful to pass your Lock to someone without having to go online. Ignore the QR code and click the **Choose File** button (the label uses slightly different words on different browsers). A dialog will open, asking you to select an image file to hide the stuff into. Find an image in jpg, bmp, gif, or png format and click **OK**, and the image will replace the QR code. If you now click the **Hide** button, the locked message will be hidden into the image, which will still look the same to human eyes. To save to disk, right-click on it and select **Save**. On mobile devices, press and hold on the image so you are asked if you want to save it or send it somewhere.

The process to retrieve the information hidden inside an image is similar. First you need to load the image as described above, but instead of Hide, you need to press the **Reveal** button. If the image contains a hidden PassLok item, it will be extracted and placed in the box on the extra screen. Whether or not the process succeeds, a message is displayed telling you what is happening.

Now, hiding an item inside an image, as hiding within words, provides no real security, so PassLok will refuse to do it unless what you are hiding is already locked. PassLok can certainly hide plaintext inside images, but it has been coded to accept only locked items in order to protect users who might think that hiding is enough.

Signed messages: mss

Recipients cannot be sure of who has locked an anonymous message. Key-locked messages do provide authentication about the sender, for a shared Key is needed in order to do the locking, but this means that the shared Key has been agreed to beforehand or a secure channel exists to transmit it (in which case, why not send the message that way, too?).

A way to authenticate the source, similar to how PGP and other programs do it, is to first sign the plain message with your secret Key, and then lock the result with the recipient's Lock. When the recipient gets the locked message, he/she will unlock it first using his/her secret Key, and then see the signature, which then he/she can verify using your Lock. If it verifies, that means the message comes from you, sin you only have the Key that was used to make the signature.

PassLok has a way to do this in one step, and that is using the signed locking mode. In this mode, the message is locked with a special shared Key that derives from both

personal Keys, and which does not need to be exchanged beforehand. This is how you do it:

1. Write or paste your secret Key in the key box.

2. Paste the recipient's Lock in the large box of the Locks screen. Come back to the main screen and check the **signed** mode radio button. Now you are ready to write a message in the main box and lock it with the **Lock/Unlock** button. If successful, the plain message will be replaced with a locked message bracketed by PL**mss tags.

When the recipient tries to unlock your message, PassLok will alert him/her that it is a signed message and that he/she needs to supply the sender's Lock as well as his/her secret Key in order to unlock it. No information about the sender is stored within the message, but hopefully the recipient knows it comes from you and is able to obtain your Lock (there's always the Lock directory). He/she places your Lock in the large box of the Locks screen, whether directly or by searching for it in his/her stored database. Clicking **Lock/Unlock** unlocks the message only if the correct Lock is used, thus authenticating the sender. PassLok detects automatically that it a signed message, so there is no need to click any of the radio buttons.

Because this process involves using your Lock, which can only be produced from your Key, the recipient can be assured that the message comes from you, even though it bears no explicit signature.

Perfect Forward Secrecy: msp

There is a third locking mode involving Locks, and this is PFS mode, where PFS is short for "Perfect Forward Secrecy." This kind of secrecy is achieved when nobody can read a locked message (even the sender or the receiver) after the message has been delivered. The message "self-destructs" after a while, so to speak. PassLok does this to a message when it is replied to, if the message is normal length, or even immediately after it is read, if it is a short message intended for SMS.

This is a desirable feature when one foresees that either the sender's or the recipient's secret Key might be compromised at some time in the future. If this happens, an enemy will be able to read all the messages that were locked in the other modes, which he had been storing, but the messages locked in PFS mode will remain unreadable.

How does PassLok achieve perfect forward secrecy? By changing the secret Key actually used to lock the message, every time a new message is locked for a particular recipient to read. The Lock matching that Key is sent along with the message, so the recipient can unlock it. He/she will be able to do so if he/she still remembers the Key he/she used to lock the last message from his/her side. PassLok uses 86-character random Keys for this purpose, so it follows that this mode requires those temporary Keys to be stored somewhere within PassLok's database.

So, to initiate or continue a PFS conversation (which typically involves a number of messages sent back and forth), you do the following:

1. Go to the **Locks** screen and type the name of the recipient in the small box. If a (encrypted) Lock appears in the large box, then proceed to the next step. If not, you need to enter the recipient's Lock into the permanent database, as explained earlier.

PFS mode involves storing additional data under that name, so having the recipient on the database is essential. If the secret Key had not been previously entered, PassLok will alert you so you can do it at this point. If you want to make sure that the conversation starts from scratch, without using any stored data, you can click the **Reset** button at this point in order to clear that data (the Lock itself won't be erased).

2. Now you go back to the main screen, select the **PFS** radio button, enter the message, and click **Lock/Unlock**. If all goes well, the plain message is replaced by a locked message bracketed by PL**msp tags. As you send it to the intended recipient, you may want to alert him/her if you have reset the temporary data belonging to your conversation.

Let's say you are not locking a message, but unlocking a PFS message you have received. In this case, you do the following:

1. Go to the Locks screen and bring up the sender's Lock by typing his/her name in the small box. If the PFS data need to be reset, you may do it at this point by clicking the Reset button. You also need to have your secret Key in its appropriate box.

2. PassLok knows it is a PFS message even if you don't click the radio button, so the only thing you need to do is put the message in the main box and click **Lock/Unlock**. If successful, the unlocked message will replace the locked message.

The moment you lock a new message in PFS mode, a new temporary Key is made and the previous one is overwritten. This means that you won't be able to read your previously locked message, or the recipient's reply to it. This situation is even more drastic if you use Short mode together with PFS mode. In this case, you can only unlock a message once, and you can only make one reply that will be successfully unlocked.

PFS mode is available if you are locking a message for several recipients, as is described below. The PFS data pertaining to each recipient is handled separately, so you can continue your PFS conversations with each separate recipient after a joint message has been made. PFS conversations work best if each message sent is followed by a reply from the other side (this is strictly necessary for short messages), although it is also possible to maintain an uneven exchange if no short messages are used (or the parties wait for short messages to be replied to).

PassLok is probably the only program out there that achieves perfect forward secrecy for asynchronous conversations (email and such). Typically, PFS-enabled programs such as CryptoCat (a JavaScript implementation of the "Off The Record" protocol) require that the participants in a PFS conversation be online at the same time, so the server can change and distribute temporary keys to them as needed.

Decoy mode

Now you may say, "All this business of locking and unlocking is very cute, but the world is not so cute. The moment they see you're using cryptography, someone's going to come knocking on your door to ask you for your secret Key, first nicely, then maybe not so nicely. And if you don't talk, maybe someone else will." What's to be done in this

unfortunately not-so-rare scenario, commonly known as “rubberhose attack” (in honor of the instrument commonly used to extract the Key)?

This is why PassLok includes what is called in cryptography a “subliminal channel.” A subliminal channel is a container for information whose very presence cannot be detected. PassLok makes use of the random data that is part of every locked message, and optionally hides a second message in there. Since the presumably random data is actually produced by encoding a random string using the same AES function that encodes the hidden message, there is no way to tell that a hidden message is indeed present. Trying to extract the presumed hidden message without the proper key will fail in exactly the same way as if no hidden message exists.

The result is called “plausible deniability.” Since there is no way to tell whether or not there is a hidden message, you or your friend on the other side can claim that indeed there is no hidden message, after supplying the secret or shared Key that they demanded. Those trying to find what you’re up to may be disappointed by what they read in your now unlocked messages and suspect that there is more, but they cannot know for sure. At some point, it becomes unreasonable to keep bothering you after you’ve supplied what they asked, and chances are they’ll let you go.

So here’s how to use PassLok’s subliminal channel, which is called Decoy mode:

1. Check the **Decoy** mode checkbox below the main box.

2. Follow the instructions for any type of locking or for signing. If Decoy mode is checked, a popup will ask for a decoy Password to lock the hidden message, and the hidden message itself, which is of limited length: 152 ASCII characters in Key-locked and signed modes, 87 characters in anonymous or PFS modes, 37 characters for short messages (Key-locked and signed only), 40 characters for signatures. Non-ASCII characters use 6 spaces each, so avoid them if you can. Any text beyond the limit will be lost.

3. Write or paste into the popup box the Password and the hidden message and click **OK**.

4. After clicking OK, the locked message containing both the main text and the hidden text will appear in the main box, replacing the original text. In the case of signatures, new material is added instead. Copy it and paste it into your communications program, or use the **Mail** and **SMS** buttons. As with regular locked messages, it is okay to strip the tags up to the "=" sign, but not recommended. PassLok will do this automatically if the no Tags checkbox is checked prior to locking. It is also okay to split the locked message with spaces, line returns, and punctuation other than = + or /

When the recipient gets the message, he/she can unlock or verify it in the conventional way, in which case it behaves no differently from a message that was not locked or signed in Decoy mode or, suspecting there is more than meets the eye, checks Decoy mode first, in order to reveal the hidden message. Here’s the process:

1. Check the Decoy mode checkbox below the main box.

2. Follow the instructions for any locking mode, or for verifying a signature. If Decoy mode is checked, a popup will ask for a decoy Password.

3. Write or paste into the popup box the special Password for the hidden message, and click **OK**. The hidden message, if it exists, will appear above the main box even if the main unlocking fails or the signature is not verified. The main message will appear in the main box if the main unlocking is successful.

All of this, of course, requires that the decoy Password, which is secret, be shared before the exchange takes place. Since Decoy mode is more advanced than what most users are going to require, the assumption is that the parties are sophisticated enough to establish this secret before they see a need to start using this mode. Everything that is said above about the strength of a secret Key still applies (perhaps even more) to a decoy Password.

Space for the hidden message is quite limited, but there are ways to expand it to include a large item. For instance, the hidden message could contain the URL and password of a compressed and encrypted file, which has been uploaded to an anonymous cloud service.

Locking for multiple recipients

From time to time, you may want to send the same message to several people. If this message is very confidential and you need to send it by some insecure channel such as email, you've got a problem. Encryption programs such as PassLok can help, but then another problem arises: if the program can only encrypt the message for one recipient at a time, then the process can get tedious, to say nothing of the risk or missing someone or using the wrong keys. This is why professional encryption programs such as PGP and PassLok since version 1.6 are able to encrypt messages so that multiple recipients can decrypt them.

The underlying process is simple: come up with a shared Key just for the message, use it to lock the message, and then give this Key to each of the recipients, possibly along with the locked message. This solves nothing if the new Key must itself be locked individually for each recipient so it can be sent securely. But since the Key normally will be much shorter than the message, there is little harm sending to all the recipients a package containing all the locked versions of this Key, along with the locked message.

When a recipient gets it, he/she first looks for the Key that unlocks the message. This Key will have been locked so that only he/she can retrieve it, and placed somewhere within the data that accompanies the locked message. The appropriate locked Key bears an identifying tag, so the recipient can find it easily. Then he/she unlocks the Key using his secret Key or perhaps a permanent shared Key in common with the sender, and then unlocks the message with the message Key.

Sounds complicated? That's because it is meant to be complicated so it is secure, but PassLok makes it easy for the user, thanks to an item called List. A PassLok List contains several lines of text, each of which is a name on the local database, a Lock, or a shared Key. If a line is not a 87-character Lock or a name already on the database, PassLok will show the user those lines, telling him/her that they will be understood to be shared Keys. If anything on the list show is actually a name for which you thought you had a

Lock, but you didn’t, you should cancel and remove that name from the List (or enter the corresponding Lock into your local database), since using those names as shared Keys would make the message easy to unlock by anyone.

This is what you do to lock a message meant for multiple recipients:

1. First you need to have a List written in the large box of the Locks screen. A List contains several lines of text. Valid items for each line are: names on the local database, Locks, or single-line pieces of text that are thus taken as shared Keys (which you will be asked to accept via a popup). Each of those lines will lock the message for one recipient. If the List contains names from the database, you also need to have entered your secret Key in its special box. If the List is already saved in the database, you only need to type its name in the small box and you're ready to go.

2. Then you go back to the main screen, type your message, and click **Lock/Unlock**. Short mode is not available for multiple recipients (the locked message becomes too long to fit within the 160-character limit), but all the other modes remain active. You can mix Key-locked and regular locked messages, but the same locking mode (anonymous, signed, or PFS) must be used for all recipients that use a Lock. It may be difficult to keep several PFS conversations in sync, so be careful about using PFS mode for multiple recipients at one time. Decoy mode is available, too, but there is only one decoy Password for all recipients.

3. If all goes well, the multiple-locked message replaces the plain message. Tags are the same as for single-locked messages. You may notice the presence of "%" characters within the locked message, which are used to separate the different versions of the message Key encrypted for each recipient. It is not possible to tell who are the other recipients, besides yourself (if you are a recipient, that is), because the identifying tags are also encrypted.

Unlocking a multiple-locked message is no different from unlocking a single-locked message. You supply the appropriate unlocking credentials (secret Key, sender’s Lock, shared Key, depending on the mode), place the locked message in the main box, and click **Lock/Unlock**. If something goes wrong, there will be a message telling you about it.

Other encryption programs, such as PGP, offer the possibility to encrypt for multiple recipients, but PassLok has a few advantages. First, you can have recipients with whom you share a Key, and recipients for whom you have a Lock, and include both types of locking within the same locked message. Then, Lists can be saved into the permanent database, so making a multiple-locked message takes the same effort as locking for a single recipient, once you've made a List.

PassLok has a special button to help you make Lists easily. It is labeled, not surprisingly, **List**. Clicking this button adds the item or items currently being displayed on the Locks screen to a temporary List, which is recalled when the **List** button is clicked with nothing on the large box. Clicking the **Reset** button erases this temporary List, in case a mistake has been made. If the new item is itself a List, containing several lines, the **List** button merges them into the temporary List, eliminating any duplicates and placing them in alphabetical order. To save the temporary List into the permanent database, first give it a name in the small box, then click **List** so the temporary List is displayed in the large

box, and then click **Save**. If you do not save the temporary List, it goes away when you close PassLok.

Locking and signing files

Locking text is nice, but what if what I need to exchange with someone is not text, but a picture or a recording? Can PassLok help with that? Sure it can. Here are two completely different methods to do that.

The easy one, which works even on a smartphone. Deal with the item as if it were a large attachment that your email program cannot handle. This is typically what you'd do:

1. Take the picture, recording, or whatever, and upload it to a cloud service. For best results, archive it locally before the upload, using an encryption password. If the cloud service is anonymous, so people don't know who uploaded it, so much better.
2. The cloud service will give you a short URL to download the file you just uploaded. Put that URL in PassLok and lock it in whichever mode you prefer, along with the archive password if there is one. Then send the locked result to the recipient.
3. The recipient will unlock the message, retrieve the file from its cloud storage location, and optionally decrypt it using the password.

The harder one, which involves loading the file itself into PassLok:

1. Go to the extra functions screen by clicking **More(…)**. At the bottom of this screen is a button labeled something like **Choose File** (the actual wording depends on the browser used). When you click it, a dialog opens so you can navigate to the file (images only, on mobile devices). When you click **OK**, the file is converted into base64 text and loaded into the box.
2. Now you can lock or sign the file as if it were any piece of text. After locking, it will look just as random as any locked text and there will be no indication that this is actually a file.
3. The recipient will unlock the locked file in the normal way, and then will click the **More(…)** button in order to access the file I/O functions. He/she then will click the **Show** button, which will instruct PassLok to convert the encoded file back to its original form. What actually happens at this point depends on the browser. Mobile browsers usually display the file, provided it belongs to one of the recognized types. On a PC or Mac, Chrome and Firefox offer to save the file to the default downloads directory and its original name, Safari downloads it with a generic name, Internet Explorer does nothing.

If you don't want to use PassLok's built-in file encoder, there are other encoding programs that you can download for just about any operating system (the function is actually built into OSX and Linux). If you choose a web program, be aware that your file is going to be sent to a server, likely unencrypted, to it can be turned into base64 text. If you use one of these programs, just copy and paste the text between it and PassLok, and you're set.

This second method works so long as the encoded file can be fit in the browser memory, which is usually a few megabytes. It is relatively slow but does not involve storing the file anywhere. On the other hand, when a file travels in an email, it is also being stored at least in the mail server, so this is not as different as one might think.

If what you want to do is to sign the file rather than to lock it, you can't really use the first method, because signing an Internet location wouldn't prove much about the file itself. That means you either have to load into PassLok the whole file, encoded to base64 as described above, or better yet, load a unique ID of it, which is what you'd sign.

This is the easiest process to sign a large file:

1. Take a SHA256 (or any other hash, really) of the file. You can do that directly via Terminal commands in OSX and Linux, and there are several free programs for Windows that do the same. PassLok itself displays the SHA256 of text in the extra functions screen, near the top of the screen.
2. Put that hash string into the main box of PassLok and apply your signature, using your secret Key. After the signature is made, you can cut it out and give it to people along with the file, or keep the hash string and the signature together.
3. Someone wishing to verify the sign will first have to take the SHA256 of the file (which should be the same you signed), then load it into the main box of PassLok and append the signature if it wasn't there already. Then verify the signature on the hash using the sender's Lock.

If you insist on using only PassLok, you can load the file with the **Choose File** button, as described earlier. At this point the SHA256 of whatever is contained in the extra functions screen (in this case the file encoded as base64 text) is displayed at the top of the screen. Copy that and then go to the main screen and paste it there. You can sign this as if it were the actual file. To verify the signature accompanying a file, repeat the process in order to obtain the SHA256 of the encoded file, then put it on the main screen and place the signature on the line below and click **Sign/Verify**. If you choose to sign a file this way, you should warn anyone wishing to verify it, since the hash of the original file will be different from the hash of its base64 encoding.

Random Keys

Security experts recommend using long random Keys rather than stuff that you come up with. This makes them impossible to crack by brute force or dictionary attacks. The problem is that they are also impossible to remember and, if users write them down, the possibility that they'll be lost or compromised increases so much that they are not worth it anymore.

But there are exceptions. This is why PassLok helps you to generate a random Key, should you ever need one. Just go to the Locks screen and click **Save** with nothing in the large box. The box will fill with an 86-character random string made of base63

characters, which is more secure than you'll ever need. As a matter of fact, you could split the random Key into two 43-character Keys, and each of them would still be as random as the encryption built into PassLok could ever take (43 base64 characters have as many variations as 256 binary digits, which is the limit for AES, on which PassLok is built).

You will likely want to remember the random string. PassLok will store it for you in the local database if you supply a name for it in the small box before you click **Save** (the large box must still be empty). As usual, you will need to have written your secret Key in its proper box before you do this.

Splitting and joining items

There is a very clever mathematical trick that allows PassLok, starting with version 1.4, to split a piece of text into a collection of parts, each of which by themselves are completely useless. The algorithm is called the Shamir Secret Sharing Scheme, and it is based on the properties of polynomials. To split a text, go to the extra functions screen click the **Split/Join** button while the text is displayed in the box. A popup will ask for the minimum number of parts that will be required to reconstruct the text. The parts will then appear on separate lines in the main box. If you need spares, click **Split/Join** again for every spare created, which will be added to the list in the main box. You can create from 2 up to 255 parts.

Parts can be readily identified by the "PL**p^^^" tags (** is the version number and ^^^ is the number of parts needed to reconstruct the original) surrounding the actual data. It is all right to strip the tags up to, and including, the "=" sign. As with the other kinds of items made by PassLok, the tags are meant to identify the item and provide some protection against accidental corruption. If the original text is shorter than 5 characters, you will need to strip the tags in order to reconstruct the text.

The process to rejoin the parts is similar: place the parts on separate lines in the main box, each part occupying a single logical line (which might wrap inside the box), navigate to the extra functions screen, and click **Split/Join**. If the parts are from the same set, there are enough of them, and they are not damaged or corrupted, the original text will appear in the box. If something is wrong, PassLok will tell you with a message.

There are a number of situations where you may want to split a piece of text into several parts. For instance:

- You want to use a high-security random Key, but since it is so hard to recall you feel forced to write it down somewhere, which is a no-no security-wise. But if you split the Key first and then store the parts at different locations (which might be different files within the same computer), the risk of compromise is greatly reduced. When you need the Key, you retrieve the parts from their respective storage locations (which hopefully you remember), and then reconstruct the original Key. You may want to make one or two spares in case you forget the location of some parts.

- You want to send locked messages that can be read by several people, but for some reason you don't want to make a multiple-locked message. In this case you create a random Key and split it into two parts, and then make enough extra parts for all the recipients, plus one. You send each recipient one of the parts made (locked with their personal Locks so no one else can get them), and the remaining part is sent in the open, along with the message locked with the full random Key. The recipients can regenerate that Key by joining the extra part that you sent and the part they each have, and then they unlock the message.
- You want to force two or more people to cooperate, or at least talk to each other by digital means. One way to do this is to lock something important with a random Key, which gets split into parts, which you send separately to those individuals. The only way they can retrieve the important item is by pooling their Key parts together in order to reproduce the locking Key.
- Splitting can also be used as an alternative to locking with a Key, since the string split into parts can be very long. To do this, place the text to be locked in the box and generate two parts. Since both parts must be joined to retrieve the text, one of them can be kept secret and used as a Key while the other is sent by insecure channels.
- If you are the Coca-Cola Company and want to embrace the XXI century, make a Word file containing the secret formula for Coke, load it into PassLok using the **Choose File** button, and then split it with the **Split/Join** button. Give different parts to different people for safekeeping.

Making sure PassLok is genuine

My biggest concern regarding the actual security of PassLok is the integrity of its code. If an attacker manages to intercept the html page before it comes to your device, he could replace it with one that outwardly looks and behaves the same, but which uses weaker encryption that he/she is able to break without difficulty. There are countless ways in which this can be achieved. This is a real problem, which is shared by all security programs running on a browser, or even directly on the computer.

If you got PassLok from an app store (hopefully starting with PassLok 2.0), there is a process by which the developer (that's me) digitally signs the code and the publisher (Apple or Google) vouches for its authenticity. In this case, an attacker wanting to tamper with the code would have to make the publisher issue a fake certificate so a counterfeit app would be accepted by your device. Probably quite difficult for a regular hacker, though not so much for a government agency, given recent history.

I don't claim to have found the ultimate solution to this problem, but at least there is something that can be done, and it is the very transparency of an html page, which makes it so vulnerable to tampering, that makes it possible.

First of all, be paranoid and load the web version of PassLok only by SSL or TLS. That's the "https" at the start of the page address, in the browser. This means that the PassLok page only leaves the server after an encrypted communication has been established between it and the browser running on your device. An attacker wishing to switch a

tampered version with the genuine one would have to spoof the server's digital certificate, which is quite difficult. At the time of this writing, the official PassLok source is passlok.com, which also hosts the general Lock directory and a copy of the informational page at http://passlok.weebly.com. Then, there are the mirrors. The most secure one, because it is located outside of US territory, is https://www.autistici.org/passlok. Another mirror, which is located in US territory so it's not as secure from interference as the one above, is https://passlok.site44.com. Finally, there is a Github page containing the code at: https://github.com/fruiz500/passlok, which also displays it in functional form as https://fruiz500.github.io/passlok.

Since I am not the sole administrator of any of those sites, there is still a chance that someone might still gain access to the PassLok source and change it. How can you tell if that happens? Because I'm publishing the ID of the genuine source in an entirely different location and by a different, harder to fake, method. Here's what you do to check it:

1. Load PassLok from one of the authorized sources listed above, or from storage (local or cloud) if previously saved as html as described in step 3b.

2. Direct your browser to "view source". Each browser does this differently, but most non-mobile browsers have this capability. Typically, you load the source on a separate tab by typing CTRL-u (Windows) or option-cmd-u (OSX). On the page displaying the source, select all (CTRL-a or cmd-a), then copy to clipboard (CTRL-c or cmd-c). Alternatively, you can get the source from PassLok's GitHub repository at: https://github.com/fruiz500/passlok. **DO NOT save the code using the "save" command from the browser menu**, since this command modifies the source code before saving it. A copy of PassLok saved this way will still run, but its SHA256 will be different. If your operating system is Windows, do not use the built-in Notepad program, since it cannot save text with the appropriate encoding. Be sure there are no extra spaces at top and bottom, since this would affect the result.

3. Now you have to take the SHA256 of the code on the clipboard using a program different from PassLok. You have several options:

    a. Use an online utility, where you paste the text on a box and click a button. You must make sure that pasting did not add extra space at the top of the file, which would affect the result. The one at Xorbin (http://www.xorbin.com/tools/sha256-hash-calculator) has worked quite well in our tests.

    b. Save the clipboard to a text file, and then use a local program or utility to take the SHA256 of the file (built into the OS in Linux and OSX, not so in Windows). In order to save to file, you will need to start a new file in a good text editor and paste the clipboard there. Then obtain the SHA256 of this file using the local program.

c. PassLok does have the ability to save anything contained in the box of the extra functions screen to a text file by clicking the **Save** button, as well as to take the SHA256 of the box contents (displayed on the line above), but be aware of the danger that these functions might have been tampered with.

4. Go to the PassLok Help page accessed from the extra functions screen and look for an item near the end, entitled "Verify PassLok's Integrity," which describes this process. This item contains a button labeled **Show PassLok's ID** which, when clicked, opens a separate tab where the correct ID of the source code (or SHA256) is listed for the current version and a few old ones in the archive, plus links to authenticating videos if available. If this ID and the one obtained in step 3 are the same, the program has not been tampered with.

Now, I'm sure you realize that the process as so far described has a gaping flaw. If someone can get to the server and change the PassLok code, so can he/she/it also change the ID in the help page to match whatever the SHA256 of his/her/its special version of PassLok is. Sure, but can that individual or corporate entity change the Youtube video of me reading the genuine ID, which is linked right below the ID? It's not like my face is as instantly recognizable as Michael Jackson's (thank God), but this is bound to be pretty difficult to tamper with no matter what.

The process can be improved, to be sure, but hopefully this will allow most users to rest at ease and use PassLok with a minimum assurance that it is safe from tampering. Not that the code itself has no security flaws. That would be for experts to test and discover, which is greatly facilitated by being able to see the code.

Once you are satisfied that the web version of PassLok is genuine, you can check a native app of the same version against it. They both should produce the same Locks starting from the same Key, and make locked messages and signatures that can be unlocked or verified by the other. Since an attacker would not know the contents of your messages beforehand (that's why you should test different messages, of different lengths), it would be hard for him/her/it to do something different under special circumstances, so if messages lock, unlock, sign, or verify interchangeably between the web version and a native version, you can be confident that they are the same where it matters.

Lock authentication via text or email

If you cannot make contact with the other person through a rich channel such as voice or video, you're going to have to start using somebody's Lock without knowing for sure if the Lock is genuine. Trust will build up gradually, as the messages sent back and forth serve to confirm the identity of the participants. But there are ways to authenticate a Lock with only a few messages traveling back and forth.

The easiest thing to do is to send a message to the Lock owner, including a question whose answer only the two of you know, and asking him/her to send you his/her Lock, itself locked with a Key that is the answer to that question. If the answer is correct, you'll be able to open the locked message, and thus retrieve the genuine Lock. An interloper who is watching and perhaps modifying your traffic won't be able to unlock a

message locked this way, and thus he/she must be content with preventing you from getting that locked file, in which case you'll know that the Lock you have is fake.

But this easy way has the problem that the other person's answer to my question must be exactly the answer I remember, down to the smallest spelling detail, or the message won't unlock. It may also be that, although I know this person quite well (a coworker comes to mind), I can't come up with any secret that only the two of us would know. There is another way to authenticate a Lock using a variation on the "interlock protocol," which admits some flexibility. It is enough if the persons asking the questions can recognize the answers as valid in a more general sense, or just recognize the other person's face or voice.

Look, for instance, at this scenario. Alice has obtained Bob's Lock, but she fears that it might be counterfeit and someone else might be unlocking the messages she sends to Bob, reading them, perhaps changing them, and then re-locking them with Bob's actual Lock for him to read. So Alice sends Bob this email (which is also included in PassLok's help system, should you want to use it):

> *Dear Bob. I just got your Lock for the app called PassLok from your email signature. I fear that I'm under surveillance, so it's very important that I make sure that this Lock actually belongs to you. Here's what I want you to do:*
>
> *(1) Write me a message asking me to take a picture or video of myself doing something of your choice. Lock the message with my Lock, which is at the bottom of this message, but don't send it back to me just yet. Instead, save it and display the locked message's ID, and send me that.*
>
> *(2) When I receive your ID, I will also write a message asking you to do something in a picture or short video, which I'll lock with your Lock. But I'll send you the ID first. When you get it, go ahead and send me the locked message containing your instructions.*
>
> *(3) When I get that, I'll check that the ID matches and then I'll follow your instructions. I'll send you the picture or video right away, unlocked, along with my locked request. Expect to receive it within half an hour of your message.*
>
> *(4) When you get it, please make sure it matches the ID I sent you earlier, and that what I sent conforms to your instructions. If everything is okay, go ahead and follow my instructions and send me the result as soon as possible, unlocked as well, within half an hour if you can. Then I'll know that your Lock is authentic.*
>
> *Your friend, Alice.*

Let's see how the protocol contained in this email foils Mallory, who is able to intercept and modify their communications without them knowing anything. He poses as Alice before Bob, and as Bob before Alice. In this case, Alice does not have Bob's genuine Lock, but one that Mallory made in order to impersonate Bob. Likewise, Bob does not get the Lock that Alice sent in step 1, but one for which Mallory has the Key.

Things begin to go wrong for Mallory in step 3. Since Mallory cannot yet read Alice's request but nevertheless has to send something to Bob to keep the exchange going, he must send him a request from "Alice" that likely has nothing to do with the real Alice's request. That, or pretend in step 2 that Bob is refusing to go along with the game, which is not going to do much to reassure Alice.

Mallory will then get the whole request from Bob, so he will be able to unlock it, re-lock it, and pass it along to Alice in step 4, and then get from her a reply that will satisfy Bob in step 6. But the damage has been done. Mallory is committed to send Bob a request from "Alice" that will match the ID previously sent but which most likely is not the request made by the real Alice, or otherwise Bob won't keep up with the protocol and Mallory's cover will be blown. Bob might not discover the ruse at this point, but it is highly unlikely that his reply, or whatever else Mallory can come up with to replace it, will satisfy the real Alice's request. Then she'll know someone's in-between and Bob's Lock is not authentic.

If now they repeat steps 2 to 4 all over with one new request from either side, but this time with Alice asking the first question, Bob will also notice that something is wrong. But what if there is no Mallory, and "Bob's Lock" was not being used to listen in but was simply corrupted or mistaken for another Bob's Lock? Then Bob will simply be unable to unlock Alice's request in step 4, and he will alert her of that fact. It is possible that Mallory could still be watching without attempting to modify the messages passing through him unless he really has to, but it is unlikely that he could replace Bob's announcement that the protocol failed with something that would satisfy Alice, because at that stage Alice won't accept anything but a correct reply to her request, or she will decide that "Bob's Lock" is bad.

It took some homework and three emails from each side, after which they still don't know each other's authentic Lock (which would be impossible with Mallory changing everything, anyway), but Alice has avoided being duped by an enemy.

Appendix: A Comparison between PassLok and PGP

This is an article I published on my blog some months ago. It is on the long side and obviously biased, but I couldn't resist adding it here as an appendix because it lays out why PassLok might yet succeed where others failed. I have edited a few things here and there as PassLok has evolved.

Email and chat encryption, certainly very desirable for privacy, has been available for a long time but very few people use it. In this article, I show why this has happened and why this is about to change with PassLok, the new privacy app derived from URSA.

Email and chat encryption until today has normally been done using PGP (short for "pretty good privacy") or GPG, its open-source cousin. PGP, created by Phil Zimmermann, was first released in 1991 with the intention of bringing strong encryption to common folks. Back then, computers were using UNIX, DOS, or an early version of MacOS as operating system. DES was yet to be cracked. The Advanced Encryption Standard (AES) winner had not yet been picked. Zimmermann based his program on IDEA, a contender in the AES contest, adding the RSA method for public key cryptography. The first version of PGP ran from a DOS command line. It was awesome and it put Zimmermann in a heap of trouble with the Feds.

There are many good articles on the workings of PGP out there, so I will only describe the essentials, from which all its subsequent history has derived. The basic steps have not changed much in the later versions, whether controlled from a command line or a graphical user interface. What follows is a bit technical but not excessively so. Stay with me.

The first thing that PGP asks you to do is type some garbage so it can seed its random number generator, then it makes a pair of private and public keys using the RSA (Rivest-Shamir-Adleman) algorithm. You cannot design your private or public keys because only certain sequences of characters are valid keys in RSA. The security of RSA keys is based on the difficulty of separating two large prime numbers that have been multiplied with one another. The private key consists essentially of those two numbers, together but clearly identified, while the public key is essentially the result of multiplying them. You can always get the public key from the private key, but the reverse operation is much harder to do. The larger the numbers, the harder it gets. Currently (2013), NIST recommends using 2048-bit factors for decent security. That's 342 base64 characters (base64 is used for displaying keys and encrypted text in PGP), or 617 decimal digits. Those are big numbers, so you can imagine the difficulty of multiplying them, let alone trying to factor them. Try doing that by hand.

The private key is to be kept jealously guarded, whereas the public key is to be publicized so people can retrieve it and send you encrypted messages. The private key is needed to read messages encrypted with the matching public key, so this process ensures that only you can read them. The actual encryption algorithm is IDEA, which is much faster than RSA and does not impose limits on the size of the message. The random encrypt/decrypt key used for the IDEA encryption is what RSA actually encrypts with a public key and later decrypts with the private key. You can also use the private key to make a signature (actually, a random-looking string, plus some identifying tags) of a given text. People can load the appropriate text, the signature, and your public key into PGP, and then verify that this particular signature of this particular text could only have been made with the private key matching the public key provided.

Very clever indeed. This opens up the possibility of exchanging confidential information without having a pre-established secret password, making binding contracts, and a bunch of other neat things. Today, PGP and similar methods are at the root of secure communications between computers and the digital certificates that tell them that another computer or a given piece of software are legit. But Phil Zimmermann's original

vision for secure communications between regular people has largely failed. We still email or text each other in a way that anyone in-between can read what we write. Why?

In a 1999 paper entitled "Why Johnny Can't Encrypt," researchers Whitten and Tygar addressed the slow pace of adoption of PGP (and indeed of most other security-oriented software) and placed the blame on confusing icons and menu structures, and on the assumption that users had a minimal knowledge of public key cryptography. This was in 1999, when PGP was in its first GUI version (5.0). Ten years later, a follow-up study involving PGP 9.0 found that the software led people to make key pairs with a greater chance of success, but encryption security had actually gotten worse since most users ended up sending unencrypted messages unknowingly. There was also a "Johnny 2" study that concluded that the problem wasn't going to go away until key certification was handled in a completely different way.

I think the root reason for these problems, which have so far have prevented PGP and its cousin S/MIME from making it in the general user world is that PGP, as well as S/MIME, was originally based on RSA. The newer, commercial versions of PGP are rather based on a different public key algorithm named DSA, which does not have the same restrictions on the keys as RSA, but the processes and conventions imposed by RSA are still there. Because an RSA private key can only be made in certain ways (not true with DSA, but the RSA key-generation process was retained for compatibility), PGP has to make the private key for you. The result, as we discussed above, is a very long, impossible to remember string of random-looking characters. Therefore, PGP stores the private key in a computer file since to do anything beyond encrypting messages for someone else to read, you must have your private key.

And here is the problem. Either the place where you store your private key is perfectly secure from tampering and eavesdropping, or you must add further security to protect it. PGP "solves" this problem by encrypting the private key with IDEA, using a separate key, before it writes it down to a file. Thankfully, IDEA admits arbitrary keys, so PGP asks you to come up with a "passphrase" (code for long, hard to guess password), which is used to encrypt the private key. When you need to use your private key, PGP retrieves the file containing it in encrypted form, asks you for the passphrase, and decrypts it using IDEA. The private key is never displayed in decrypted form (at least for you to see it).

This may solve the problem for a corporation, but it doesn't solve it for the general population. You still need to have that file containing your encrypted private key, in addition to remembering the passphrase that unlocks it. If you lose the file, you're done. If you are using someone else's computer and don't have a way to retrieve that file remotely or don't carry it along in a flash drive, you're done. The public key also has problems arising from their authenticity, but what makes PGP rather painful to use by the general public is having to manage that "secret key ring", as they call it.

Okay, you *can* write out the private key file so it is displayed in conventional ASCII characters, and then you can paste it into your Google files or any other place that you can access online. It will still be encrypted so that no one can use it without your passphrase. But then you'll be trusting someone else to guard your precious private key. If they fail, someone could delete it, corrupt it, or switch it with a counterfeit key. On

top of that, most online services have been known to open their users' accounts to more or less accredited "investigators," often without a warrant. (Note: this was written months before Edward Snowden revealed the NSA's pervasive data-collecting programs)

PGP public keys are even longer than the private keys, and just as random-looking. Therefore, they must be stored somewhere. Because they are "public" it is okay to display them in the open and upload them to public places. This is what PGP keyservers are: computers where people have uploaded their public keys so other users can find them and thus can send them messages encrypted with those public keys, for their eyes only to read. The problem is that there is no intrinsic assurance that a certain key belongs to a certain individual. PGP again "solves" this problem with something called "web of trust." Essentially, people add electronic signatures (made with their private keys) to other people's public keys, to certify that they believe the keys belong to the people the keyserver says they belong to. The signatures are either made by other individuals, whom you may or may not know, or by an intrinsically trusted, professional Certification Authority, for a fee. Think of a digital notary public.

So when I want to write a PGP-encrypted email to a certain individual, I am asked to fetch his/her public key from a key server, where keys are usually catalogued by name or email address, then load it into PGP (often permanently by means of its "public key ring"), and then enter the message and tell PGP to encrypt it. The more recent versions of PGP can do this from a GUI, including the key-fetching process, but are still hard to use by a majority of people. Even if I succeed in obtaining the appropriate keys, they usually have no assurance, other than the digital signature of people they don't know, that the keys actually belong to that person and not to a third malevolent party. I know this because I've made keys that I've successfully uploaded to a key server, using all kinds of pretend names except my own.

To be sure, PGP encourages certain standard practices that provide a modicum of reassurance. One of them is that people should sign their public keys (thereby making them twice their original length) before they post them. The idea is that people then can verify that, if you were able to sign your public key with your private key, you must have both keys so at least it's not someone who just ran into your public key somewhere who is posting that key. Another best practice is to add a signature to a text right before it is encrypted. This way the recipient of the text has assurance, by checking the signature, of the sender's identity. Unfortunately, signing a message before encrypting it makes it longer and harder to process later on as well.

So much for specific aspects of usability, but how about the interface itself? Whitten and Tygar piled up most of their criticism against PGP in this area. Leaving aside whether user interaction is from a command line, and in the original PGP, or there are graphics involved as in the versions after 5.0, PGP forces the user to adopt a metaphor that does not match anything else in the user's experience, namely, to lock things with one key and unlock them with another. People have never done this before and are therefore confused from the very start as to how one would go about it. Consistent use of technical words such as "encrypt", which people tend to associate with tombs and corpses rather than computers the first time they hear it, doesn't help much, either.

How does PassLok help with all this? To begin with, in PassLok your private key is whatever you want. If you want it to be "I feel depressed without fried Twinkies," that's fine with PassLok. It will complain that it's kind of weak but will still make a public key to match it, and that's that. Assuming that you can remember it, you don't need to write it anywhere, and certainly you don't have to use a flash drive, or store it in a file anywhere, plain or encrypted.

And by the way, PassLok doesn't have "public keys" and "private keys." It has "Locks" that people can put on a text so nobody can read it, and "Keys" that unlock a locked text for their possessors. Everybody has used those before in hardware. All of this may sound like a little exercise in word substitution, but it can make all the difference in a user's comprehension of what's going on.

PassLok Locks are much shorter than PGP public keys: just 103 characters, tags included, versus several hundred. They are a lot more secure, too, since 521-bit elliptic curve keys are believed to be equivalent to RSA keys longer than 15,000 bits. Because PassLok keys are short, they can be sent by text messaging or made into QR codes so that people can read your key out of your calling card, which helps a lot with authentication. They also fit within the "extra info" fields of just about all webmail contact lists, where you can upload keys as you get them instead of storing them in a special key ring file, or fetching them every time from a key server. Giving someone a PassLok Lock is only slightly more involved than giving a phone number, which is precisely the metaphor that PassLok uses for distributing and authenticating Locks.

PassLok uses no web of trust or certification authority to authenticate its public keys. How are then people supposed to get someone's Lock with a minimum of confidence that it does indeed belong to that person? Here the problem may be the question itself. Ask yourself: how do get someone's cell phone number with a minimum of confidence that it does indeed belong to that person? I don't know what you do, but I look at that person's physical or electronic card first, or ask someone else who might have it. The phonebooks in my house have been gathering dust for years, as have PGP key servers all over the world (many of which aren't current or active anymore, once you check). When I get a phone number, I use it right away, and let the actual use serve as verification that it is correct.

It's the same way with a particular PassLok Lock. The first time I use it, I am not so confident that it will encrypt messages for the intended individual and not someone else, but hopefully I'll be able to tell after a couple exchanges. If I am paranoid about someone intercepting our communications and placing himself in the middle, PassLok has a simple authentication mechanism built in where I can get the other person on the line and ask him/her to spell out the unique ID of his/her Lock (or mine). I can post a video of myself showing my government-issued IDs and spelling out my Lock's ID for the whole world to see. I've actually done that for my URSA 3.2 key, which is identical to my PassLok 1.0 lock, and you can see it here if you're curious, and I keep doing it every time the evolution of PassLok has made my old Lock obsolete, last time with version 1.3.

PGP has had key servers since the beginning. The idea is that people load their public keys to the key server so that others can find them and use them; after all, public keys are made to be publicized. Neat idea, but unfortunately most PGP key servers are far

from useful because of some serious fundamental flaws. The first is that anyone can upload a public key. The server has no means to verify the source, so it accepts everything, including public keys that might have been generated by someone trying to pose as someone else. The second is that keys are never deleted (though they can expire or be revoked by means of an additional certificate), leading to multiple keys for the same user that need to be sorted out by those who want to write to this user.

PassLok had no key server for the longest time because it is not strictly necessary. Starting with version 1.7, however, a “general Lock directory” has been added as a convenience, but it is significantly different from the old PGP key servers. For one thing, users need to reply to a confirmation email every time they add, delete, or modify an entry (yes, entries can be fully deleted), so that a poser would need to be actively interfering with the victim’s email in order to succeed. Then, each entry has space for an authenticating video, and the interface has a button to simplify the action of viewing it. This video, similar in concept to the code-authenticating video mentioned below, shows the Lock’s owner reciting a part of his/her Lock or of its SHA256 hash. This way, people who have met this person can be assured that the Lock is authentic, without a need for certificates and webs of trust.

Improving on PGP, which only has public-key encryption to which a signature might be added to authenticate the sender, PassLok adds a simpler “signed” encryption mode, while retaining the ability to do anonymous public-key encryption. The signed message ends up being shorter, too, and is easier to process on the receiving end than an anonymous locked message. Authenticated encryption is available as an option, but it doesn't involve signing the message. Again, this is because PassLok has under its hood the Diffie-Hellman key exchange algorithm, which involves both parties, rather than the RSA algorithm, which is one-sided.

Starting with version 1.6, PassLok includes a third “PFS” locking mode, in addition to the anonymous and signed modes. PFS stands for Perfect Forward Secrecy, which means that nobody, not even those originally involved in the message exchange, can read an old message after the exchange is over. This typically requires that the participants in a conversation be connected at the same time to some server (certainly, PGP does not have anything resembling this), but PassLok achieves this very desirable attribute through email or any other asynchronous channel. It does so by generating a new random Key for each new message, which is stored only until the message is replied to, and then is overwritten by another Key. Since the encryption Keys are destroyed after they are used and they never leave the device where they were made, nobody can read the messages after that.

Starting with v1.4, PassLok also includes the Shamir Secret Sharing Scheme, seamlessly built into its interface via a single button. Among other things, this means that a user can in fact store random keys without worrying too much about their being lost or compromised. The only thing he/she needs to do is split the key into several parts, which then are stored at separate locations that only the user knows. Spare keys can be produced, too, just in case.

PGP was born as a command-line process to which a graphical user interface was later added, and it shows. PassLok, on the contrary, has been born as a web app. There are

no multiple versions of PassLok compiled to run on different operating systems, just the one page, which runs without modification both in PCs and mobile devices. You can actually read the code if you feel inclined it, and I do recommend that you do precisely that sometime so you can be sure the page is not sending or receiving any information to or from the outside. It can be as easy as doing a "show source" in the browser (every browser does this differently, but they all use ctrl-u or opt-cmd-u as a shortcut), followed by searches for things like http://, ftp:// and so forth, which a webpage needs to have in order to communicate with the outside.

For the more paranoid amongst us, PassLok offers the ability to perform a check of its integrity before you do anything with it. Just tell the browser to show the source code, copy it, paste it into the Public Key box, then click the ID button. PassLok will make a SHA256 checksum of itself and display it in the Plain Text box. You can always get the checksum for the latest version from the instructions page (first link near the top of the page), and make sure they match. If you are concerned that the official website might get hacked by someone who wants to switch that code so it matches an altered version of PassLok, watch the optional video of me reading the checksum, which is a lot harder to fake. Once you have a copy that you trust, you can save it somewhere in your device or in the cloud so you can use it when you need it. It's not secret information, so you only need to ensure that your enemies don't know about it. PGP can't offer any such assurance for the paranoid.

Much has been made of the presumed vulnerability of client-side encryption methods like PassLok. The argument usually goes like this: an enemy could modify the code before it gets delivered to the client application (the browser, in PassLok's case), and the user wouldn't have a clue that it has happened; case closed. That is, if the user never bothers to check its integrity. To go even further, I would argue that exactly the same thing could happen to a program, such as PGP, that is downloaded from a server and then installed on the device.

Developers of compiled software fight this by publishing an MD5, SHA1 or, more recently, SHA256 hash of the installation file so users can check it for tampering. The hash is usually on the same page as the download link and without a video of the developer reading it out loud. A PassLok user does not have to trust it in any way before he/she verifies that it is genuine. The code is perfectly dead until he/she puts anything on it and starts pushing buttons. This is usually not the case with compiled software, including PGP, to say nothing of the impossibility of looking at the code itself and try to figure out what it is doing. So which is more secure against tampering?

And for the super-paranoid, those who lay awake at night worrying that they, or someone they communicate with, might get their secrets beaten out of them by non-digital methods, PassLok has a subliminal channel built-in, which PGP and S/MIME have never had and probably will never have. This means that every PassLok message and sign is capable of containing an additional message, encrypted under a separate key. Those who do not have that key cannot obtain the second message, and neither can they test whether or not there is one. This functionality is accessed by checking a single "Decoy mode" box, named so because it is perfectly possible to generate completely

misleading exchanges while the actual information is being conveyed through the subliminal channel.

Beginning with version 1.4, PassLok also has the ability to disguise its output as common text or within images, what is known as steganography. The process is a simple one-button click, which is reversed with the same one-button click. The default text is English, but you can change the language quite easily. This way, anyone scanning emails will have a much harder time detecting that encryption is taking place. If you hide PassLok's output inside an image (four formats are supported), the new data replaces the noise within the image, so human eyes are unable to detect anything being there.

To summarize:

- PGP forces you to keep secret files that might be lost, corrupted, or compromised. PassLok uses no secret files of any kind. If a user insists on storing secret material, PassLok can split it into parts that are useless unless the whole set, or a substantial part of it, is put together. Anything that PassLok stores for users' convenience is encrypted.
- PGP forces you to install things in a particular computer, and then use that computer or one similarly equipped, which makes it dangerous if that computer is compromised. PassLok is a perfectly portable web app; you can use any PC or smartphone in the world, so you can be sure there's no foul play. Even its Lock directory is portable, and can be moved easily from one device to another.
- PGP public keys are very long and unwieldy, including certifying signatures in addition to the keys themselves; signatures can only be read by the PGP program. There are at least two kinds of keys, RSA and DSA of different bit lengths, which are incompatible with each other. People are unable to read the certificates, and must rely on the software to check things out. PassLok Locks (there's only one kind) are short. They are transmitted and verified in the same way as a phone number without official key servers. They can be summarized into ID numbers for authentication through a separate channel or through any of the many audiovisual methods freely available today.
- PGP is built on the RSA public key algorithm, which would need keys longer than 15,000 bits for a security level comparable to that of PassLok, which is built on 521-bit elliptic curve math.
- PGP started using only the 128-bit IDEA algorithm for its main encryption method (it allows more secure methods now). PassLok uses the strongest, 256-bit version of AES (a.k.a. Rijndael), the winner of the 2001 NIST contest for best encryption algorithm. S/MIME uses triple-DES, which lost to AES in that contest. IDEA didn't even compete.
- PGP only admits one encrypted message at a time. So does S/MIME. In PassLok, a second message whose very existence is impossible to verify can be conveyed at the same time as the main message.

- PGP outputs easily detectable encrypted strings. PassLok can produce output that looks like English, or any other language. It can also hide its output within images, in a fashion undetectable to the eye.
- PGP uses difficult to understand terminology derived from cryptologists' jargon. PassLok only talks about Keys and Locks. In PassLok, context-sensitive help and tutorial videos are always just one click away.

As a table:

	Pretty Good Privacy (PGP) and S/MIME	**PassLok**
Portable	No	Yes
OS-independent	No	Yes
Installation-free	No	Yes
Storage-free	No	Yes
Small public keys	No	Yes
Spread public keys w/o Internet	No	Yes
Encryption modes	1	4
Short message mode	No	Yes
Hidden messages	No	Yes
Fake text steganography	No	Yes, 2 modes
Image steganography	No	Yes
Secret Sharing Scheme	No	Yes
Transparent authentication	No	Yes
Encryption bitlength	128 or 256	256 always
RSA-equivalent public key strength	1024-bit or 2048-bit	+15000-bit
Interface pages	many	1
Learn mode	No	Yes
Built-in tutorial videos	No	Yes
Jargon-free	No	Yes (well . . . almost)