PassLok 2.1 manual

by F. Ruiz, 1/19/2015

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 your Key
   e) Making a directory of Locks
   f) The General Lock Directory
3) Locking and Unlocking
   a) Anonymous locked messages: msa
   b) Key-locked messages
   c) Short locked messages
   d) Signed messages
4) Signatures
   a) Putting a signature on a text: sig
   b) Verifying someone's signature attached to a text
5) Real-time Chat in PassLok
6) A walkthrough of the interface
   a) Tabs
      - Main
      - Options
      - Help
   b) Dialogs
      - Key
      - Directory Edit
      - New Key, email, user name
      - Decoy in, out
      - Parts
      - Image
   c) Buttons and Checkboxes:

7) Advanced features
   a) Using fake text
   b) Hiding stuff inside images
   c) Perfect Forward Secrecy (msp) and Read-Once (mso)
   d) Decoy mode
   e) Locking with Lists
   f) Locking and signing files
   g) Random Keys
   h) Splitting and joining items
   i) Making sure PassLok is genuine
   j) Lock authentication via text or email
   k) Under the hood: Compression, Data correction, Variable Key Stretching
8) Appendix: PassLok vs. PGP

What is PassLok?

PassLok is an encryption program, which means that it turns regular readable text and files into unreadable gibberish. Only the persons holding the appropriate Keys can then retrieve the original text or files. PassLok does several types of encryption, including symmetric and asymmetric (public-key) encryption. Additionally, PassLok has the ability to disguise its encrypted output as more normal-looking text, and to hide it within image files. Beginning with version 2.1, PassLok also does real-time secure chat, which can include even video.

What is public-key encryption?

In encryption of any kind, you encrypt a text or file using a text string called a key. A key must be supplied during the decryption process to recover the original item, or the decryption will fail. Typically, the decryption key is the same as the encryption key, and in this case we speak of symmetric or 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. This 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 a 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. Secure communications between computers use some sort of public-key encryption to get it started, and then maybe switch to the other kind, which is faster, once the keys involved have been securely transmitted.

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 people 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 most of the rest

cannot help in that situation since they must be installed before they can run. PassLok can still help.

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 secret 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.
- Nothing should be installed by the operating system. 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, self-contained file, using no outside libraries. 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 items 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 4096 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 long delay to the user, this serves to encourage users to come up with stronger Keys, without forcing them directly. What's best, since all bad Keys are still in the pool, hackers are forced to run through all of them before they get to the better Keys, wasting large amounts of computer time, or risk missing them entirely.

If users decide to use much more secure random Keys, PassLok helps in several ways. The easiest is to click the Random button on the first contact screen, which is seen only when PassLok opens for the first time or a new user is created. This adds a 521-bit random token to the user's chosen Key, which is immediately encrypted with the Key and stored in the device. This provides ultimate security against those attempting to guess the Key, since they'd have to guess both the Key and the long random token, but unfortunately causes the user to be tied to the device where the token is stored (this is similar to what happens with PGP). PassLok has a button to export that token in encrypted form, however, so it can be used on a different device.

PassLok can also generate a long random Key with the touch of a button, and this is stored nowhere. 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.

As of version 2.1 PassLok has evolved into a powerful program that can do many things beyond simple asymmetric encryption: self-destruct messages, files, text- and image-based steganography (hiding), splitting into parts, rich text editing, real-time chat involving even video.

And then, PassLok sports a very clean interface with as few buttons as possible. There is a Basic mode where only the essential buttons are shown, and an Advanced mode with all the bells and whistles. Help has its own dedicated, searchable tab that is always available, and more help pops out 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. You don't need the key to lock something with an open padlock, 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.

Creating Keys and Locks

The first task for a new user is to choose a secret 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 later on, for a variety of reasons. This section describes all of these things.

The new user wizard, which appears the first time you use PassLok on a given device or click the **New User** button, leads you through the Key selection process step by step. Here's the explanation of each step:

1. **Introduction**. It is essential to have a minimum knowledge of what PassLok is supposed to do. The first screen contains an optional, fun 3-minute video that explains it. If you want to skip the whole wizard, there is also a button to do this, labeled **Exit**.
2. **User Name**. PassLok is multi-user (or multi-identity). You must provide a name if you want to store anything in the permanent directory, so that data belonging to different users don't get mixed up. Anything not public is encrypted, so only the right owners have access to their data. When you restart PassLok, all the user names recorded in the device are displayed for you to choose one.
3. **Secret Key**. A good Key is the most important element in your security, so PassLok will tell you how good it is security-wise as you type it in. If you prefer to start from an already strong Key, a button will display five random words from PassLok's internal dictionary. PassLok does not store your Key, so it is important that you remember it without having to write it down.
4. **Email or Token** (optional). If you now supply some piece of data that is unique to you, such as your email or phone number, security will be greatly enhanced since a hacker trying to guess your Key (likely from your Lock, which is public) will be forced to start the cracking process from scratch without the benefit of ready-made tables of pre-cracked Keys. This piece of data does not need to be secret, just unique to you. Unlike the Key, this piece of data will be stored in encrypted form, so you only have to supply it once. You also have the option to use a random token, by means of the **Random** button. Doing so will give you ultimate security, but it will make it difficult to use PassLok on a different machine (although there are ways to get around this, explained below).
5. **Generate and disseminate your Lock.** The last button closes the wizard and displays the Lock matching your Key and additional data on the **Main** tab. The idea is to give your Lock to everyone you wish to communicate with you, much like a (longish) phone number. To make this task super-easy, PassLok will send it by email to your friends right away, along with a short set of instructions, if only you leave a box checked before clicking the button.

How to make a strong Key

Since the cryptographic methods used in PassLok are overkill for today's computer power, the weak link in the chain is rather the 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 English 1000 Keys and their variations, and penalizes users who use those as part of their secret Key, but it doesn't stop here. It knows the strategies followed by hackers and makes them as painful as possible.

Since the Lock made from a user's Key is public, a hacker will 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 characters is typically enough.
2. Since going beyond this involves an exponentially longer computing time, the key-guessing 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 if you are the target of a personal attack.
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. (with and without spaces), 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, adding some personal data (not necessarily secret) to an already good Key will defeat the rainbow table. This is why PassLok asks you for your email address, or some other public data about yourself. To make your life a little easier, this information is stored between sessions (unlike the Key, which is never stored anywhere), so you don't have to enter it again in the same machine.

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. It can go from Terrible to Overkill, with Good being the target to hit so things don't get sluggish. In Advanced mode you also get a number for the information entropy of the Key, measured in bits. If the Key is a needle, then the

information entropy tells you how big the haystack is. The effect is exponential, so that an increase of one in the information entropy means that the haystack gets twice as large, an increase of two makes the stack four times bigger, and so forth.

In addition to the 1000 most common passwords mentioned earlier, for which PassLok gives no credit in terms of information entropy, PassLok includes a dictionary of the 10,000 most common English words. The idea is to give less credit for series of characters that can actually be found in a dictionary, as opposed to sequences that aren't there. PassLok is smart enough to know if a word has been s1mply m0d1fi3d by changing a few characters. For the time being all the words are English-based, but future versions of PassLok will use different dictionaries based on the interface language.

Generate your matching Lock

After you type your secret Key into the key input box and click OK (we are assuming your email or whatever additional data you entered the first time remains recorded), you are ready to make its matching Lock. To do this, just click the myLock button. The Lock will appear in the **Main** box.

You can tell it is a Lock because:

1. it begins with a PL**lok or PL**ezlok tag (where ** is the version number),
2. followed by an equal sign,
3. exactly 87 base64 alphanumeric characters (numbers and lower- and upper-case letters, plus + or / symbols) for regular Locks, exactly 100 broken up into groups of five for ezLoks, which constitute the actual Lock data
4. then another equal sign,
5. another set of exactly 20 alphanumeric characters, which are used to correct errors that might have slipped into the actual data,
6. a third equal sign,
7. and ends with a tag like the one at the beginning. Despite its great security, it is short enough to be shared within a text message (160 character limit) or posted on Tweeter (140 characters).

By default, PassLok will display your ezLok, which contains no capitals (except "L") and is much easier to dictate over the phone, but you can display the regular version instead by unchecking ezLok in the Options tab before clicking myLock. Either version can be used interchangeably.

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 (accessible only in Advanced mode) 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 always read it aloud. An ezLok contains only lowercase letters, numbers, and the characters / and +, so it takes only about 45 seconds to read it as someone writes down (the dividing underscores are there only to make it easier to read, and are not needed for functionality). If you have a mobile device, you can always copy it into an app that would display a QR code, which another device can read off the screen, but the process likely won't be as fast as simply dictating it.

When you first use PassLok, it will offer to send your Lock to whomever you want by regular email. The only thing you need to do is leave a checkmark on and supply the email addresses when the ready-made message appears in your default webmail. This way you start out with a small network, without having to worry about anything beyond storing your friends' Locks when they are sent to you.

Other ways of spreading your Lock are:

- Add it to your email signature. This 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 General 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" between you and your friends, so how can people know that a Lock is genuinely yours?

This is why you can attach a web address to your Lock without affecting its function. You can authenticate your Lock by means of a video, like this:

1. Copy your Lock to a piece of paper. If it is printed bold so a camera can pick it up, so much better.
2. Then grab a smartphone and make a video of yourself reading your Lock or a part of it. It is enough to read the first 15 characters or so of an ezLok. If you have background music, so much better so nobody can hack the video to pieces and make "you" read something else.
3. Put the video somewhere online and copy its address.
4. Append that address to your Lock, preferably on the line right below it. From now on, keep the two things together every time you post your Lock.

When someone who knows you sees your Lock with a video address attached to it, he/she can always copy that address and watch you reading the Lock, which will assure him/her that it is legitimate. No need for webs of trust, certification authorities, or any of the things that were (and still are) recommended for authenticating PGP keys. Aren't you glad you live in the twenty-first century?

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. This is explained in the Advanced section.

Changing your Key

Let's say that, despite your great care to keep your Key secret, never writing it down anywhere and keeping it masked as you type it, you fear it has been compromised. Or maybe you want to use 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 a PassLok Lock.

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 other people. Those who never find out are probably best left in the dark. Same with a Lock.

Still, if you want the whole world to know that your public Lock has changed, you can head to the General Directory and change the Lock you have previously posted. This is as easy as clicking **General Directory** on the directory Edit dialog. Then you only have to supply your email address, which is required for confirmation, and click **Post**. After you click on the link emailed to you, the change in the General Directory is complete, so that users looking for your Lock will get the Lock matching your new Key.

You can check that this has happened by clicking **Find** in the General Directory screen while both your address and your Lock are displayed in their respective boxes. A green message will confirm that your Lock and the posted Lock match. It is a good practice to check from time to time even if you have not changed your Key, since email confirmation is far from perfect and there is always a chance that someone might have posted a counterfeit Lock for your email address.

Changing your Key is as easy as typing a different Key when PassLok starts, but there is a hitch that you should be aware of. As you collect into your local directory (which is explained later), they are stored encrypted by your Key, and are decrypted by the same Key when they need to be used. If you use a different Key, it won't decrypt items encrypted with the old Key, leading to all sorts of problems. This is why the preferred way to change your Key is by clicking the **Change Key** button in the Options tab. A dialog asking you to type the new Key twice appears and, if both copies agree, PassLok re-encrypts everything under the new Key.

Now, it may be that someone sends you a message locked with an old Lock of yours. You will know when you are unable to open the message with the current 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. Just restart PassLok and click **Cancel** at the Key input screen. When you try to unlock the message, you will be asked again for the Key, and at this point PassLok will accept anything you give it.

Making a directory of Locks

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 have, 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.

If this weren't easy enough, PassLok can keep its own directory within the program itself, and it doesn't get deleted when the app is closed or even if the browser's cache is flushed. When you want to store somebody's Lock or a symmetric key you share with that person, you only need to click the **Edit** button on the **Main** tab, type a unique name in the top box of the dialog that appears, then paste the Lock into the lower box, and click the **Save** button. The Lock or shared Key will be permanently saved, encrypted by your secret Key for good measure. From then on, using that Lock or shared Key is as easy as clicking on its name displayed on the **Main** tab.

Another way to use that Lock or shared Key is to click the **Edit** button and then start typing the name on the top box of the new dialog. As soon as the full name appears on the line above it, you can stop typing and PassLok will use the item. If you type Enter at this point, the item will be decrypted in the lower box, so you can see it.

Your directory does not need to stay with the particular device where you entered it. PassLok has a nifty **Move** button (visible in the Advanced interface) which takes the entire directory, encrypts it, and puts it on the **Main** tab, ready to be copied, emailed, or whatnot. PassLok will then offer to delete the directory from the device, should you want to cover your tracks, but you can cancel at this point and the packaged directory will remain in the **Main** tab.

Even the most user-friendly PGP-based programs, such as Mailvelope, take more steps in order to do this, 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.

(Chrome app only)
If you are using the Chrome Web Store version of PassLok, there is another nifty feature that you should know about, and this is that your directory entries are automatically synced through Google as you save them. This means that you can go to a computer that you have never used before (say, at the local library), open Chrome, log in with your Google account, and find the PassLok app ready for you, directory and all. The entries remain encrypted as they are stored in Google's servers, so you should not be too concerned about their having your data.

The General Lock Directory

PassLok also has an official web-based directory where people can post their Locks for others to see. Only Locks, though; shared Keys are confidential by nature, so it doesn't make sense to post them. This function is accessed from the directory edit dialog, by means of a big button at the top, labeled **General 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 General 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. When you go back, you will find that the Lock has been copied into the lower box of the directory edit dialog, so the only thing left is to give it a name and save it.

Now, you may wonder whether this is really Alice's Lock, or maybe someone else put that Lock in the general directory, in which case I may be locking messages for that someone else to read. Not good. The General Directory has two features that help prevent this scenario:

1. Posting a Lock on the General Directory involves replying to an email containing a special link, without which the posting is not complete. Someone who wants to post 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 General Girectory 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.

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 general directory so that others can find it. You open the General Directory page, write your email address in the top box, and the Lock in the bottom box; then you click the **Post** 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 too 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 a 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 general directory you can remind that person that people would like to have it listed there. There is an **Invite** button that produces a ready-made email with instructions on how to post a Lock. Unlike the other emails, this one comes from your own account, so the invited person can feel reassured.

The General Directory looks a lot like the rest of PassLok, except for a different color scheme. This is to remind you that you are looking at a different page, which connects to a server. Integration with the rest of PassLok is limited to copying the Lock found in the General Directory. This is because we want to keep the encryption code entirely separate from any server.

Locking and Unlocking

Having covered the essentials of Key and Lock management, we go now to the process of actually locking and unlocking messages. Despite its small size, PassLok is one complex piece of software, comprising five 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. If you have entered the recipients' Locks into your directory, just select them on the top box in the **Main** tab. Several Locks can be selected at once. Otherwise, click the **Edit** button and enter the Locks, one per line, in the lower box of the dialog that appears, then go back. 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.

2. Write or paste your message into the **Main** tab. Click the **Lock/Unlock** button. The locked message will replace the original message. It will normally include identifying tags at both ends, unless the **no Tags** checkbox has been checked in the **Options** tab prior to locking. Copy it and paste it into your communications program or click **Email** to open your default email.

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

1. Paste the locked message into the **Main** tab. 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 on the **Main** tab, replacing the locked message. If you haven't entered your Key or it has been so long since you used it that PassLok has forgotten it, you will be prompted to enter the Key, and then the unlocking process will continue.

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 your 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 selected, which doesn't really matter. The tags won't tell you that 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).

Here's what you do to lock a message with a shared Key:

1. If you have entered the Keys you share with the recipients into your directory, just select them on the top box in the **Main** tab. Several shared Keys (mixed with Locks, too) can be selected at once. Otherwise, click the **Edit** button and enter the shared Keys, one per line, in the lower box of the dialog that appears, then go back; there is no need to save anything.

2. Write or paste the message into the **Main** tab. Click the **Lock/Unlock** button. If you entered the shared Keys directly, a prompt will ask you to confirm that these are legitimate shared Keys before the locking actually takes place. Then the locked message will replace the original text. Copy it and paste it into your communications program or click **Email** to open your default email.

If you have received a message that you suspect is locked in this mode (remember, the tags won't tell you), here's what you do to unlock it:

1. If you have entered the Key you share with the sender into your directory, just select it on the top box in the **Main** tab. Otherwise, click the **Edit** button and paste the shared Key in the bottom box of the new dialog, then go back.

2. Paste the locked message into the **Main** tab. 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 replace the locked message.

Unlike in anonymous mode, you would know who sent you a Key-locked message, because only people knowing the shared Key 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.

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 in the **Options** tab. Because of the length limitation, the size of the text that can be locked is also limited. Another constraint is that only one recipient can be selected.

In order to fit as much text 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. Here's what you do to lock a message in this mode:

1. Check the **Short mode** checkbox in the **Options** tab.

2. Write or paste your message in the **Main** tab. 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 selected the recipient's Lock or shared Key on the **Main** tab, or entered it without saving by clicking the **Edit** button and putting it in the bottom box of the new dialog, as described above. After this is done, you click the **Lock/Unlock** button. The locked message will replace the original message. Copy it and paste it into your communications program.

4. If you are using a smartphone, the locked text will be 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 near the bottom of the secreen (visible in the Advanced interface), which will open the texting app. Now you only have to type in the recipient and paste in the locked message.

If 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** tab. It is okay if it is broken up by spaces, carriage returns, and special characters. If it was locked with a shared Key, you need to select or write that Key as we saw earlier.

2. Then click the **Lock/Unlock** button. The unlocked message will replace 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.

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 (explained below), 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, since 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 the sender's and the recipient's secret Keys, and which does not need to be exchanged beforehand. Locking is done exactly as for the anonymous mode, except that you must set the locking mode on the **Options** tab as "signed" before clicking **Lock/Unlock**. Unlocking is slightly more cumbersome, in that you must select the recipient's Lock on the directory (or enter it via the **Edit** button) before clicking **Lock/Unlock**. There is no need to set the mode when unlocking. PassLok will alert you that you need to supply the sender's Lock if you forget to do so.

No information about the sender is stored within the message, but hopefully the recipient knows the source and is able to obtain the appropriate Lock (there's always the General Directory). The message is unlocked only if the correct sender's Lock is used, thus authenticating the sender without the need for a digital signature.

Signatures

Making digital signatures is rather backward from locking. Things are locked with a Lock and unlocked with the matching Key. On the other hand, things are *signed with a Key*, and then that signature is *verified with the matching Lock*.

If you think about it for a moment, you'll see that it makes sense. Signing should only be possible for one person, while it should be possible for everyone else in the world to verify that the signature was made by that one person. Hence one needs a Key, which is secret, to make a signature, and the matching Lock, which is publicly available and matches the Key uniquely, to verify it. This process is called "digital signature" in specialized jargon.

One more thing is different between signing/verifying and 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), plus 20 characters of error-correction, 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.

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.
To make a signature, write or paste the text to be signed into the **Main** tab. 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 **Email**, 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. If the Lock of the person who allegedly made the signature is in your directory, select its name on the list. Otherwise, click the **Edit** button and then paste that Lock into the bottom box of the dialog that appears, then go back.

2. Write or paste the text with its signature appended at the end into the **Main** tab. 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 message near the top 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.

Real-time Chat in PassLok

Some modern web browsers, including Chrome, Firefox, and Opera (but, sadly, not yet Safari or Internet Explorer, or anything running on iOS) support a protocol named "webRTC". RTC is short for "Real Time Chat". This offers the possibility to establish real-time secure exchanges that can include text, files, audio, or even video.

In order to start a chat session, you use PassLok to *make an invitation* involving all the people who are supposed to be able to connect to the chat. To do this, simply select all the names in the directory on the **Main** tab (you can omit yourself, since you'll be added automatically no mater what), and then click the **Chat** button at the bottom of the screen. Then a dialog appears, offering a selection for the type of chat (text and files, that plus audio, all that plus video) and a space to add an optional short message (43 characters max) which might include the time for the chat and other info. When you click **OK**, a random-looking item bracketed by PL**chat tags appears on the **Main** tab. This can be emailed or sent by any other insecure channel, since only those people you selected on the directory will be able to unlock it.

Recipients will then unlock the invitation by placing it on the **Main** tab and clicking **Lock/Unlock** or **Chat**. The sender will do the same. If there is a message, a popup will display it and ask if you want to continue (maybe it's not yet the time for the chat, so the recipient might want to cancel and try again later). Then a new screen will open containing some short instructions and including a small box and a button. The small box is to write the name you want to use during the chat so the other participants can know when you're typing something into the common log. The button will be labeled "Start" if you are the first participant arriving at the chat room, or "Join" if someone else got there ahead of you.

If you are the first, clicking **Start** will initialize the session. A message will appear telling you that you're waiting for others to join. Clicking **Join** connects you to each of those already in the session. Their chat names appear listed near the top of the screen, and messages on the log alert all participants whenever someone joins or leaves. If the chat involves audio or video, you will be asked for permission to access those features. From then on, you are connected directly to each one of the participants, using a separate encrypted channel for your interactions with each of them.

If you need to use PassLok in the middle of the chat session, there is a button that will take you back without disturbing the chat. To go back to the chat, click the **Chat** button on the **Main** tab. If something gets screwed up during the chat, there is a button to reset it at the top of the screen, at which point you will be asked again for a name. To really disconnect from the chat, restart PassLok.

You may be wondering about the security of this whole thing. A lot of it depends on the security of the webRTC protocol itself, which works more or less this way when Alice wants to chat with Bob:

1. Alice contacts an external "signaling" server so she can know what her external IP number is. This IP number is likely quite different from the one given to her computer by her local network, which she can check without calling anyone. Unfortunately, connections from outside the network need her IP number as seen from the outside.
2. After getting this address, she sends it to Bob by email or similar protocol, or perhaps using a (probably different) server that both have agreed upon previously.
3. Bob gets Alice's request to connect and so he does the same she did, and sends her his external IP number.
4. Now that both have each other's current address, their respective browsers can negotiate a direct connection between them. The process, which involves multiple back-and-forth steps, is similar to SSL/TLS, which browsers use to connect securely with servers to get "https" pages.
5. When the negotiation is complete, Alice's and Bob's browsers have a direct connection between them. Whatever Alice types/says/shows is transmitted encrypted by a random key. Bob decrypts it on his end, and what he sends back is encrypted by the same key.
6. When they disconnect, the key disappears from their machines so even if someone was recording all the traffic between them, he/she/it can no longer decrypt it.

A nice feature of the webRTC protocol is that the connection is direct after the initial exchange of addresses. Third parties are only involved in setting up the connection and then drop out of the picture.

Now, PassLok is supposed to contact no servers, so how can it be doing all this? Actually it's not. When a chat session begins, PassLok loads a separate web page, from a different server, to handle the chat. This way, any rogue code that might find its way into the chat session (not likely, but possible every time you connect to another computer) is prevented from seeing any of the truly private data that PassLok works with.

The locked invitation made by the main PassLok code contains four pieces of information:

1. An optional short message, which may be used, for instance, to announce the time that the chat session begins.
2. A code indicating the type of chat: text, audio, or video.
3. A random "chatroom name," which tells participants where to go in order to connect.

4. A random password, which the computer initiating the chat will ask of those that follow. If they don't supply the correct password, they cannot connect to the chat.

When a recipient unlocks the invitation, PassLok first displays the message and, if the user decides to go ahead, then loads the webRTC-based chat code from a different server and passes to it the rest of the data contained in the invitation. In the current version, the chat code then connects to Firebase (a high-volume game-oriented signaling server owned by Google) and gives it the chatroom name code. For the first participant, Firebase creates a database under that name, where it stores the external IP number for each computer trying to access it. It gives those addresses to computers joining later. So Firebase's role is limited to a sort of cubby hole where participants drop in their IP addresses so others can find them (and, of course, finding those addresses to begin with, which is hard for participants to know before they connect).

The code running on the first participant's computer has additional roles. An important one is allowing or rejecting incoming participants. A reason for rejection is not supplying the correct random password when connection is requested. This password was never sent to Firebase, so the only way to know it is to unlock the invitation. Thus, snooping on Firebase won't be enough to be able to connect to the chat, even as a dumb listener. Once the webRTC connection is established, the browser's built-in encryption code takes over until the session ends. Anyone intercepting the data (which doesn't travel through any central server) needs the session key for each pair of participants in order to decrypt the data transmitted between them, but this key disappears the moment one of them restarts or closes the chat.

A possible vulnerability, which the webRTC code cannot possibly defend against, is for someone to get the contents of the invitation from a participant (which includes the random password), and then connect to the chat impersonating someone. This is especially so for text-only chats, where the participants do not see or hear each other. To prevent this possibility, you may want to establish the following protocol for your chat session:

1) A new participant has just joined the chat. Someone sees that, and types in a bunch of gibberish text, like for instance "lkuyto[7987."
2) The newcomer sees this, and understands he/she is being asked for authentication. He/she copies the text, goes back to PassLok, and pastes it on the **Main** tab. Then he/she adds a signature to it by clicking the **Sign/Verify** button, copies the result, goes back to the chat, pastes it into the chat input box, and sends it.
3) Each of the other participants copies the result, goes back to PassLok, pastes it on the **Main** tab making sure the signature is on a separate line, selects the presumed participant's Lock on the directory, and clicks **Sign/Verify**. If the new participant is who he/she claims to be (meaning he/she is in possession of the Key matching the Lock), the signature will be verified.

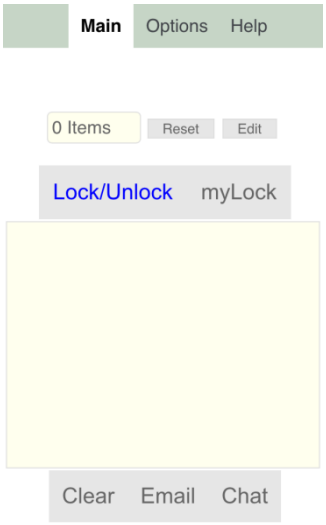
This won’t work, obviously, if the newcomer’s secret Key has been compromised. Another way it won’t work is if the stuff to be signed is predictable ahead of time, so an interloper has been able to obtain a valid signature for it.

A walkthrough of the interface

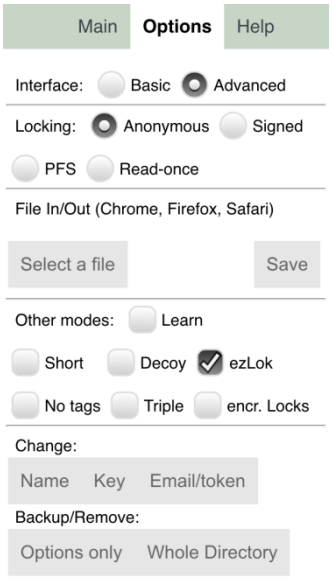
We now describe all the objects on the PassLok screens and what they do. The pictures below are from an iPhone, so the screens will look somewhat different in other platforms.

Tabs

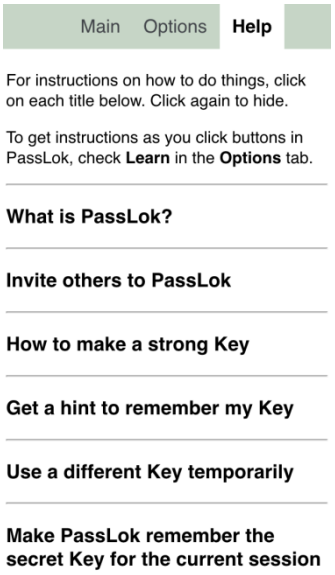
Main



Options

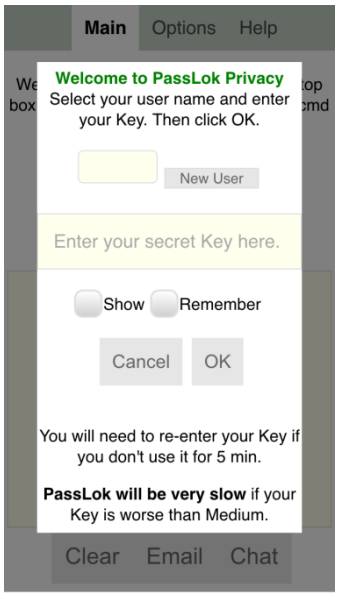


Help

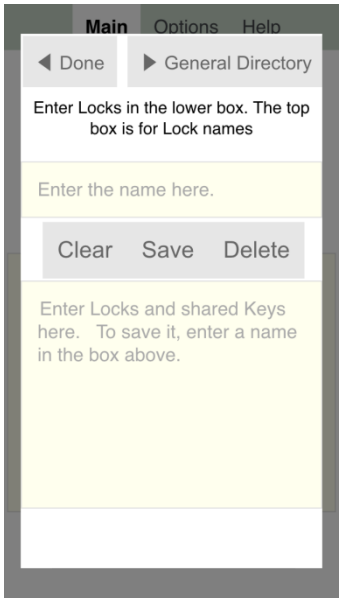


Main Dialogs

Key



Directory Edit



Each of these tabs and dialogs contain text-based buttons and checkboxes that can be clicked, text boxes that can be filled, and text. Let's see a description of them:

Buttons and Checkboxes:

Reset

This button clears the selection made in the directory box.

Edit

This one opens the directory Edit dialog, where Locks and shared Keys can be added for immediate use of for permanent storage.

Lock/Unlock

This button has context-sensitive functionality. If PassLok does not find a locked message on the **Main** tab, it will lock the contents using whatever has been selected in the directory, or entered via the Edit button. If a locked message is found, it will attempt to unlock it. Then the successfully unlocked message ends up replacing the locked message originally in the main box.

Sign/Verify (advanced)

This button also has context-sensitive actions. First PassLok tries to find a signature at the bottom of the **Main** tab. If it does not find it, it concludes that you want to sign the text in the box. If PassLok finds that a signature is already there, it will attempt to verify it using the Lock selected in the directory or entered via the Edit button. A message will say whether or not signature verification has succeeded.

myLock

When you click this, the Lock matching your Key is displayed. If you have not entered your Key yet, a dialog will ask you for it and the email/token, and then the Lock will be displayed.

Rich … Plain (non-mobile only)

This button displays or hides a ribbon above the main box, which contains buttons and selection lists for formatting text.

Clear

Clears the box.

Email

Opens your default webmail with a pre-formatted message that depends on what is contained in the main box. If the box contains nothing, the email message gives instructions on how to obtain PassLok and generate a Lock, and includes your personal Lock. This is handy to invite others to join your PassLok network.

Chat

If the **Main** tab does not contain a chat invitation, clicking this button instructs PassLok to make one for the people selected in your directory (or entered via the Edit button), including a short optional message. If there is a chat invitation, this button unlocks it and opens up the chat screen, from where you can start the chat session if you are the first to arrive, or just join it if already in course. Once a chat session has started, clicking this button returns you to the session no matter what is on the **Main** tab.
Be aware, however, that not all versions of PassLok support all of this. The iOS version, for instance, only supports making the chat invitation.

▸ (extra buttons)

This button displays a new set of buttons around the main box, dealing with communications, stealth, and secret splitting.

◂ (main buttons)

By clicking this, the buttons dealing with locking, unlocking and signing, return to the main screen.

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 used for hiding PassLok's output. 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 in PassLok's local directory. To do this, click the **Edit** button, enter a name for the cover text in the top box of the new

dialog, paste the cover text in the bottom 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.

SMS (mobile only)

On mobile devices, it opens the default texting (SMS) 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.

Image

This button opens a new screen where images are displayed so that PassLok items can be hidden within them.

Split/Join

When this button is pressed, a popup asks for the total number of parts to be made, and the minimum needed to retrieve the contents of the box. PassLok performs a Shamir Secret Sharing algorithm to generate those parts, which are then displayed. These parts can now be stored in different places or sent to different people. If a sufficient number of parts are present on the **Main** tab when this button is pressed, the same algorithm combines them to recreate the original text.

Interface

This and the next few items are on the **Options** tab. The interface selector has two choices: **Basic**, which shows only the most essential buttons and settings, and **Advanced**, which displays everything.

Locking mode

There are two basic locking modes: Anonymous and Signed. The Advanced interface makes two more available: PFS (short for Perfect Forward Secrecy), and Read-once.

File In/Out

This contains two buttons for loading and saving files in and out of the **Main** tab.

Other modes

There are a number of additional modes, dealing with locking as well as other aspects, which are selected individually in this area.

Change (advanced)

This area contains buttons used when you want to change your user name, your Key, or the email/token.

Backup/Remove (advanced)

The buttons in this area make a backup (and optional delete) of either your personal settings only, or the complete directory stored on your device. In

both cases, an encrypted item is created on the **Main** tab, which restores the items after clicking **Lock/Unlock**.

"Other modes" detail

Learn

When this is checked, a popup explaining what is about to happen and requesting confirmation to continue comes up every time a button is clicked. Useful for learning how to use PassLok.

Short

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.

Decoy

Checking this 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.

ezLok

Leaving this box checked causes your Lock to be displayed in a format where no capital letters are used—except "L", which is used instead of lowercase "l", easily mistaken for number "1"—when the **myLock** button is clicked. The idea is to make your Lock easy to spell out. PassLok can use ezLoks interchangeably with regular Locks.

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.

Triple

When this is checked, every random-looking item generated by PassLok is displayed three times, with "@" signs between the copies. This is useful when the communication channel is so noisy that it is likely that data will be lost or spurious data added.

encrypt Locks

PassLok normally stores Locks in plain text, since Locks are not supposed to be secret. If you would like Locks to be stored encrypted with the secret Key like everything else, check this box.

Choose File

Pressing this button opens a dialog to select a file (an image only, on iOS devices), to load into PassLok. It can be any kind of file. The file is then converted into base64 text and loaded on the **Main** tab, 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 on the **Main** tab, 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. If the contents of the box are not an encoded file, PassLok will attempt to save it as a text file.

The Images screen (accessible in the Advanced interface) also contains a few buttons, which are explained below.

Choose File (advanced, 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 (advanced)

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 (advanced)

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 on the main screen, or otherwise a message tells you that there is nothing to reveal. This function currently does not work on mobile devices,

and this is why the whole image hiding process is available only for non-mobile devices.

General Directory

This and the remaining buttons are found in the directory Edit dialog, which opens when the **Edit** button on the **Main** tab is pressed. This button opens the General Directory screen in a frame which has no direct communication with PassLok other than copying the contents of a successful search.

Save

This button saves the item in the bottom box under the name displayed in the top box.

Delete

This button deletes the item whose entire name is displayed above the top box (not necessarily what is written in the box, since name recognition is gradual), from PassLok's local directory. This is irreversible and there is no confirming dialog, so be careful.

Reset (advanced)

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 and other settings. This is useful if a PFS conversation goes out of sync and must be restarted.

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

List (advanced)

Clicking this button adds the contents of the bottom box to the current temporary List, removing any duplicates (case sensitive). If the bottom box is empty the current List is displayed instead. If what is displayed in the bottom box is an item from the local directory, 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 to the local directory, you must first write a name for it in the top box, then click **List** to display the List in the bottom box, and then click **Save**.

All (advanced)

When this button is clicked, the complete local directory is displayed in the bottom 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 local directory, since it is not necessary to know its name beforehand. It can also be used to edit the local directory in large chunks. The **Move** button implements a method for copying the entire local directory so it can be used in a different device.

Merge (advanced)

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

If the bottom box contains a Lock when this button is pressed, it will be combined with whatever single-line item is on the **Main** tab, to make into a new Lock, which appears in in both boxes. The same thing happens if the **Main** tab contains a Lock instead, and the bottom box 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.

The following are the buttons on the General Directory interface, which is actually a page separate from the PassLok code:

Find

When this 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.

Post

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 **Post** 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 general 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 plain 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. Those functions are available after clicking the ▶ button at the bottom of the **Main** tab. Here's how it works:

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

1) Put the item to be converted into fake text into the **Main** tab. 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 ▶ 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) Put the fake text in the **Main** tab.
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 ▶ 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 **Main** tab, 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 the default if PassLok is reloaded. If you want to make the change permanent, store often-used cover texts within PassLok's local directory. To do this, click the **Edit** button, type a name for the cover text in the top box, then paste the cover text in the bottom box, and click **Save**. The stored cover text will be loaded automatically when you select its name on the list in the **Main** tab.

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

PassLok also includes a method for hiding 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 in the advanced interface.

Let's say PassLok has locked a message, which you now wish to hide within an image. Click the ▶ button, followed by the **Image** button. Then click the **Choose File** button (the label uses slightly different words on different browsers) in the new screen that appears. 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 load. 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**.

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 **Main** tab. 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 simple hiding provides enough security.

Perfect Forward Secrecy (msp) and Read-Once (mso)

There are two additional locking modes, and these are PFS mode, where PFS is short for "Perfect Forward Secrecy," and Read-Once mode. PFS secrecy is achieved when nobody can read a locked message (even the sender or the recipient) after the next message is been delivered. The message self-destructs after a while, so to speak. In Read-Once mode, self-destruction happens the moment a message is successfully unlocked.

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 and Read-Once 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. The Lock matching that temporary Key is sent along with the message, encrypted, 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 random Keys for this purpose, so it follows that these modes require those temporary Keys to be stored somewhere within PassLok's local directory.

To initiate or continue a conversation like this, which typically involves a number of messages sent back and forth, either in PFS or Read-Once mode interchangeably, you do the following:

1. Select the recipients on the **Main** tab. If there is only one, you can also go to the directory Edit dialog and start typing his/her name in the top box until the full name appears along with the corresponding Lock or shared Key. Forward secrecy modes involve storing encrypted data for each recipient, so if the secret Key had not been previously entered, PassLok will request it at this point. If you want to make sure that the conversation starts from scratch, perhaps because the conversation has somehow gone out of sync, you can click the **Reset** button when the recipient's name is displayed.

2. Now you go to the **Options** tab and select the **PFS** or **Read-Once** radio button.

3. Back in the **Main** tab, you enter the message, and click **Lock/Unlock**. If all goes well, the plain message is replaced by a locked message bracketed by PL**msp of PL**mso tags. As you send it to the intended recipients, you may want to alert them if you have reset the temporary data.

If you change your mind about what you say in the message before you actually send it, you can still write something else and lock it if you are using PFS mode. The conversation won't go out of sync. If you used Read-Once mode, however, once you reply, that's it. Locking a new message before a message from the other side is unlocked throws the conversation out of sync so that no messages could be unlocked after that. Because of this, PassLok will refuse to do it and instead will instruct you to use PFS mode (or anonymous or signed modes, for that matter) if you attempt to lock in this mode twice.

To unlock a PFS or Read-Once message you have received, you do the following:

1. Select the sender's name on the **Main** tab, or click **Edit** and start typing the name in the top box until the full name appears. If the PFS data need to be reset, you may do it at this point by clicking the **Reset** button in the directory edit dialog.

2. PassLok knows it is a PFS or Read-Once message even if you don't select the radio buttons, so the only thing you need to do now is put the message on the **Main** tab 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 your message. This situation is even more drastic in Read-Once mode. In this case the Lock involved in the message is overwritten after a successful unlock, so it is not possible to unlock a message twice.

When a message is locked for several recipients, the data pertaining to each recipient is handled separately, so you can continue your conversations with each separate recipient after a joint message has been made. Forward secrecy conversations work best if each message sent is followed by a reply from the other side (this is strictly

necessary for Read-Once mode), although it is also possible to maintain an uneven exchange if no Read-Once messages are involved.

PassLok is probably the only program out there that achieves perfect forward secrecy for asynchronous conversations (email and such). Typically, Forward secrecy-enabled programs such as CryptoCat (a JavaScript implementation of the "Off The Record" protocol) require that the participants in a conversation be online at the same time, so the server can distribute temporary keys to them as needed. PassLok also has a real-time chat feature, but security is handled in a completely different way. More on this later.

Decoy mode

Now you may say, "All this business of locking and unlocking is very cute, but the world isn't 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 one of your friends will." What's to be done in this unfortunately not-so-rare scenario, commonly known as "rubberhose attack" (in honor of the instrument traditionally 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 see in your now unlocked messages and suspect there is more, but they cannot know for sure. At some point, it becomes unreasonable to keep bothering you after you've given them 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 on the **Options** tab.

2. Follow the instructions for any type of locking or for signing. If Decoy mode is checked, a popup will ask for a Decoy Lock or shared Key, and for the hidden message itself, which is of limited length: 152 ASCII characters in signed mode, 87 characters in anonymous, PFS and Read-Once 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. After clicking **Lock/Unlock**, the locked message containing both the main text and the hidden text will appear on the **Main** tab, replacing the original text or, in the case of signatures, appended to the signed text. As with regular locked messages, it is okay to strip the tags up to the "=" sign. 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 usual 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 the Decoy box first, in order to reveal the hidden message. Here's the process:

1. Check the Decoy mode checkbox on the **Options** tab.

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 key.

3. Write or paste into the popup box the special Key for the hidden message. If this is not a shared Key, uncheck the Shared checkbox. Then click **OK**. The hidden message, if it exists, will appear in the message area of the **Main** tab even if the main unlocking fails or the signature is not verified. The main message will be displayed in the normal way if the main unlocking is successful.

All of this, of course, requires that the sender know the special Key, if it's a shared Key. 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 use this mode. Everything that is said above about the strength of a secret Key still applies (perhaps even more) to a Decoy Password. There is also the possibility of using special Locks for hidden messages, in which case they don't need to be kept secret. But if your special Lock is public, you run the risk that an enemy may apply the rubber hose to you back side until you cough up the Key for that Lock, too.

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, if needed) of a compressed and encrypted file, which has been uploaded to an anonymous cloud service.

Locking with Lists

PassLok can lock an item for multiple recipients, who presumably use different secret Keys, as easily as for a single recipient. This is the right place to ask how can this be possible.

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. You select the recipients directly on the **Main** tab or, if you want to keep complete stealth and not even use the directory, you only need to enter the recipients' Locks or shared Keys, one per line, on the bottom box of the directory Edit dialog. Then, PassLok has Lists including several recipients (more on this later) that can be saved into the local directory, so locking for multiple recipients takes the same effort as locking for a single recipient, once you've made a List.

The underlying process is this: PassLok comes up with a shared Key just for the message, uses it to lock the message, and then locks this Key for each of the recipients and sends the locked Keys along with the locked message. When a recipient gets the package, he/she first needs to find the message Key that has been locked for him/her. Now, there may be several of those, each locked for a different recipient, so how does PassLok identify the right one? Because the locking process adds an identifying tag to each locked Key. This should be done in a way that only the recipient can succeed at identifying the tag, or the locked message would be leaking information (the intended recipients) to whoever happens to intercept it.

So the identifying tag for each locked Key is based on information that only the sender and each recipient know. If they are using a shared Key, for instance, the tag is based on that shared Key, after it is encrypted using itself as Key, in a way that is different for each message. If a Lock is used, then the tag is based on that Lock, encrypted , for signed, PDF, or Read-Once messages with the permanent shared Key made by combining the sender's Key and each recipient's Lock; for anonymous messages, the Lock is encrypted with the result of combining each recipient's Lock and the dummy Key used for the message, which is the same for all recipients and is sent in plaintext along with the rest of the message.

After each recipient computes this id tag information, he/she then looks for it in the locked message. Once it is found, the data immediately following it is the encrypted message Key, which then can be decrypted using whatever mode was selected by the sender, which is transmitted via a special character at the start of the locked message. Note that only the recipient can find this tag, since it is encrypted in a way that only he/she (and possibly the sender) can decrypt. Anyone intercepting the message cannot find out who the intended recipients are, even if he/she has all the Locks belonging to those recipients. An intended recipient can only find out if he/she is on the list; the

identities of the other recipients remain unknown as far as any hint contained in the locked message.

Sounds complicated? That's because it is meant to be complicated so it is secure, but PassLok makes it easy for the user, in two ways. The first is the directory on the **Main** tab. The sender only needs to select the recipients (several can be selected), and click **Lock/Unlock** exactly as for just one recipient. On the recipient's side, there is no need to select anything once the correct secret Key has been entered, at least for anonymous messages. The sender must be identified for the other types, but this simply involves one click on the directory.

The second labor-saving device is Lists. PassLok can handle multiple recipients as a unit, by making a list of their names (or their individual Locks or shared Keys), one per line, and saving it under a new name. The List name appears on the **Main** tab directory like any other name, except that it is highlighted so the user knows it is actually a List of several recipients. When you click on a List name, the message area tells you the names contained on the List. PassLok scans for duplicates automatically, so that you can click on Lists (or individual recipients) that may have lines in common, but only one instance of each recipient will be considered.

There is a special button to help you make Lists of multiple recipients. It is labeled, not surprisingly, **List**, and it is visible on the directory Edit dialog when the Advanced interface is selected. Clicking this button adds the item or items currently being displayed on the bottom box to a temporary List, which is recalled when the **List** button is clicked with nothing on the 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 local directory, give it a name in the top box, then click **List** so the temporary List is displayed in the bottom 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 **Options** tab and find 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 **Main** tab.
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 go to the **Options** tab and click the **Save** 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 types they recognize. 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 use one of these programs, just copy and paste the text between it and PassLok, and you're set.

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 text. This 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 either have to load the whole file into PassLok, encoded to base64 as described above, or better yet, load a unique ID of it, which is what you'll 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.
2. Put that string into the main box of PassLok and apply your signature by clicking the **Sign/Verify** button (visible in the Advanced interface). The signature is then appended to the hash.
3. Someone wishing to verify the signature will first have to take the SHA256 of the file (which should be the same you signed), append the signature to it, put the result on the **Main** tab of PassLok, select the sender's Lock on the directory, and click **Sign/Verify**. If everything works okay, a green message at the top will say so.

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 likelihood 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 open the directory Edit dialog and click **Save** with nothing in the bottom box. The box will fill with an 86-character random string made of base64 characters, which is way more secure than you'll ever need. As a matter of fact, you could split the result into two 43-character strings, and each of them would still max out the encryption algorithm on which PassLok is built (43 base64 characters are equivalent to 256 binary digits, which is the upper limit for AES).

You will likely want to remember the random string. PassLok will store it for you in the local directory if you supply a name for it in the top box before you click **Save** (the bottom box must still be empty). As usual, you will need to have written your secret Key in its proper box before you can save anything.

Another place where random 86-character strings appear is as tokens to be used in conjunction with your secret Key. This is done by clicking the **Random** button in the New User wizard when you get to this optional step, or a similar button in the Change Email/token dialog called from the **Options** tab. The result is equivalent to adding that random string to your secret Key, which makes it essentially impossible to guess from looking at the matching Lock no matter how bad your Key is. The random token is stored in your device, encrypted under your Key, so you don't need to enter it again. The catch is that you'll be tied to that device from then on, since you wouldn't have the token stored in other devices. To help you cope with this, PassLok can export it as a special locked item (along with the rest of your settings) if you click any of the Backup/Remove buttons at the bottom of the **Options** tab. To use your random token

on a new device, set up a new user with the same secret Key (but obviously not the same token), place the backup item on the **Main** tab, and click **Lock/Unlock**.

Splitting and joining items

There is a very clever mathematical trick that allows PassLok to split a piece of text (or a file encoded as 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, get the extra set of buttons on the **Main** tab and then click **Split/Join** while the text is displayed in the box. A popup will ask for the total number of parts to be made, and 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. 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) bracketing the actual data. As usual, it is all right to strip the tags up to, and including, the "=" sign, since 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), display the extra set of buttons, and click **Split/Join**. If the parts are from the same set and there are enough of them, 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 security no-no. 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 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 as many parts as recipients plus one, with two being required to reconstruct the original. You send each recipient one of the parts (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 shared Key that 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 webpage 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 done. 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 as a packaged app from the Chrome Web store, the code has been signed as it was uploaded to the store, so that the web browser (Chrome) will check that the signature matches the actual code before it accepts it. In this case, an attacker wanting to tamper with the code would have to make Google issue a fake certificate so that Chrome would accept a counterfeit app. Probably quite difficult for a regular hacker, not so much for a government agency. So you can feel secure about Chrome app version of PassLok, so long as you trust Google. The same can be said about the versions of PassLok for Android, which is distributed from the Google Play store, and for iOS, which is vouched for by the Apple app store.

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, what 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 for an individual (though apparently not for the NSA and similar entities). At the time of this writing, the official PassLok source is

https://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, 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 a chance that someone might still gain access to the PassLok source and change it without my knowledge. How can you tell if that happens? Because I'm publishing a hash 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 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). You can also save the page to hard disk at this point. Alternatively, you can get the source from PassLok's GitHub repository at: https://github.com/fruiz500/passlok. **DO NOT save the running code using the "save" command from the browser menu**, since this command modifies the source code before saving it, only the source displayed by the method above. If your operating system is Windows, do not paste the clipboard into the built-in Notepad program, since it cannot save text with the appropriate encoding. If you are pasting the code somewhere, 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 hash of the code on the clipboard using a program different from PassLok. A hash is a mathematical operation that converts a (usually long) piece of data into a fixed-length random-looking string, which is a sort of fingerprint of the original. An important property is that the process is one-way: you cannot get the original from the hash. The probability that two given pieces of data will produce the same hash string is negligible. Popular hash algorithms are MD5, SHA1 and, more recently, SHA256. 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 does not add extra space at the top of the file, which would affect the result. Online-convert (http://hash.online-convert.com/sha256-generator), fileformat.info (http://www.fileformat.info/tool/hash.htm) and freeformatter.com

(http://www.freeformatter.com/sha256-generator.html) have worked 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 (not Notepad) 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 on the **Main** tab to a text file by clicking the **Save** button in the **Options** tab, but be aware of the danger that this function might have been tampered with.

4. Currently, the SHA256 for the most recent version of PassLok is published in five places. These are:

1. The "Get PassLok" section of PassLok's informational website at http://passlok.weebly.com.
2. Within the description of the Chrome packaged app, at this location: https://chrome.google.com/webstore/detail/passlok-privacy/epcchpdljafmfegifkigklfcmkphfmbh .
3. Within the description of the Android app: https://play.google.com/store/apps/details?id=com.fruiz500.passlok
4. Within the description of the iOS app: https://itunes.apple.com/us/app/passlok-privacy/id879861603?mt=8&uo=4
5. At my blog, prgomez.com: http://prgomez.com/nonfiction/passlok/65-passlokcurrent.html

These are all different from the actual servers of the PassLok webpage for one important reason: whoever manages to tamper with the code on the server can also tamper with the published hash if it is served from the same location. Hence, the servers are different.

Now, I'm sure you realize that the process as so far described has a gaping flaw. The pages containing the SHA256, although different from those containing the code, can still be hacked in order to place there a hash that would match a counterfeit code. It would mean hacking into several servers but it would still be possible. Sure, but can that individual or corporate entity change the YouTube video of me reading the genuine ID, which is always linked right below the hash? 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. This would be for experts to test and discover, which is greatly facilitated by being able to read the code.

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 presumed 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 whatever 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 think of 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 participants 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 obtained your PassLok Lock from (cite source), but I still wonder if it is authentic since I am unable to view the authenticating video. Therefore, I ask you to help me authenticate it through the interlock protocol. Here's what I want you to do:

1. *Write 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 appended to this message, and copy it to a safe place. Then split it in two and send me the first half only.*
2. *When I receive your first half, I will also write a message asking you to do something in a picture or short video. I'll lock that message with your Lock, and*

I'll send you half of the locked message first. When you get it, go ahead and send me the other half of your locked message.

3. *When I get your second half, I'll put together the two halves together and unlock your message. Then, following your instructions I'll send you the picture or video right away, unlocked, along with the second half of my locked request.*
4. *When you get it, please verify that what I sent conforms to your instructions. If so, put together the two halves of my locked message to you, unlock it, and send me what I ask in the message as soon as possible, unlocked as well. 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 at the beginning of step 2. 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 1 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 at the end of step 2, so he will be able to unlock it, re-lock it, and pass it along to Alice, and then get from her a reply that will satisfy Bob in step 4. But the damage has been done. Mallory is committed to send Bob a second half from "Alice" that will match the first half but which most likely does not contain 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 that someone is in-between and Bob's Lock is not authentic.

If now they repeat the process 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 complete 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 a powerful enemy.

Under the hood: Compression, Data Correction, Variable Key Stretching.

There is some really cool stuff happening as you work with PassLok that don't involve any buttons. Let's talk here about some of them.

Typically, a locked message will be longer than an unlocked one. This is especially true if it involves non-Latin characters, which the computer internally encodes as a longish string of binary data (as much as six times longer than a Latin character). To compensate for this, PassLok includes smart Lempel-Ziv (LZ) compression of plaintext. This algorithm replaces frequently used strings or characters with shorter codes, resulting in as much as a 50% reduction in output size. When the plaintext is long, this also translates into faster processing. An example: the Gutenberg.org version of Shakespeare's Hamlet (nearly 180,000 Latin characters long) takes about four seconds to lock or unlock in my 2007-model Dell machine if compression is not used, but barely one second with compression applied right before the encryption step. The locked item is also less than half the size.

The "smart" bit about LZ compression is that it isn't used where it won't be helpful. One case is for encoded files, which tend to look fairly random without any compression. Since compression is based on finding repetitions, random-looking data don't compress well. PassLok detects encoded files and skips the compression/decompression steps.

Data correction means that an item may have suffered some sort of corruption and still remain functional. Without data correction, when text is locked any change in the locked result will make it impossible to unlock. A similar think happens with a Lock that is dictated over the phone and copied down with errors: the defective Lock will no longer match the original Key, so that material locked with it won't be unlockable. So how does PassLok implement this wonder?

Two ways: Reed-Solomon algorithm and triple encoding. Triple encoding is the easiest one to understand, so we'll cover this first. If instead of sending one copy of a locked item you send three identical copies, whenever two of them agree we can safely conclude that the result is correct. This also works for deletions or spurious additions at any point, so long as two of the copies still agree. Triple encoding is very easy to implement but, duh, increases the size of the output by a factor of three, so PassLok makes it optional. It will always detect triple encoding if it is used, but to make triple-encoded output the user needs to check the Triple checkbox on the **Options** tab. You can always obtain the same result by copying the single output, adding a "@@" string (without the quotes), then a copy of the output, another "@@" string, and then another copy of the output, leaving no spaces in-between.

Reed-Solomon is the algorithm used in CDs, DVDs and other media in order to deal with scratches etc. QR codes also use this algorithm. The idea is to add some more data that identifies any corruption of the original, and even supplies the correct values. PassLok implements this by adding some more random-looking data before all final tags. The error-correction data can be identified easily because it is preceded by a "=" character, just like the tags. It can detect and correct up to 10 wrong or missing characters for every 255 characters of the original, including the error-correction code itself. Error correction usually takes place when an item is used, but in the case of Locks it happens as soon as a Lock is entered, meaning that it will appear to change (it loses its tags, for instance) without the user having clicked any button. This serves as confirmation that the Lock has been identified as such.

Error correction is based on math not unlike that of the SSSS algorithm (used in the Split/Join function), and it seems like magic, but it can fail sometimes. If there is more than 10 errors for a given group of 255 characters, a message will alert the user that the RS algorithm has failed. Since it is possible that all the errors be in the error-correction code itself, retrying the function after having deleted the tags (including the error-correction code, which won't be used in this case) might do the trick. The RS algorithm cannot detect whether characters are missing or have been added later, so if this has happened the process will fail as well. Unfortunately, there is no way to correct for this if the item was not triple-encoded. Triple-encoding, on the other hand, should handle this event without any trouble.

Key stretching means that, every time a Key is used in PassLok, what is actually used is not the Key itself, which might be a simple word and therefore easy to guess by hackers, but the result of applying a complex mathematical operation to that Key. The result is a random-looking, fairly long string that would be pretty much impossible to guess by using dictionaries. Even better, it is harder to do the operation backwards than to try all possible inputs, so that the operation is effectively one-way only. Key stretching has been used since the beginning of computer-based cryptography, so there are a number of well-researched standard algorithms. PassLok uses the SCRYPT algorithm, which has the nice feature that it is hard to run using parallel processors, resulting in no significant gains over a single processor. This, which might sound at first like a bug, is actually what we want: we want a long, tedious computation consuming lots of computer time, in order to make Key-guessing as hard as possible. The actual user will only be running this computation once, which is usually okay, while a hacker trying to guess the Key will be running it millions of times.

PassLok goes one step further, and it uses the WiseHash algorithm, combining SCRYPT with a smart Key strength evaluation function based on a dictionary and true entropy measurements. This means that weaker Keys, which score lower in the strength-measuring algorithm, are hit with more spurious computations than stronger Keys. This is interesting for two reasons. The first is that the user experience is worse for weaker

Keys, thus providing an incentive for the user to come up with a stronger Key. Although there is little perceived difference going from a Good to an Excellent to an Overkill Key (to paraphrase the labels PassLok displays in Basic mode), things get really slow as you drop down to Medium, to say nothing of Weak and Terrible Keys.

The second reason is that bad Keys remain valid Keys, even as they are penalized. A hacker trying to guess your Key by making Locks for all strings in the dictionary will be forced to consume a large amount of computer time mucking through the bad Keys, or risk missing the easy ones. Add the email/token feature, which means that there is no such thing as a pre-computed list of Keys and their matching Locks valid for all users (also called “rainbow table”), and you’ve got an extremely secure way to use Keys that you can actually remember.

Appendix: A Comparison between PassLok and PGP

This is an article I published on my blog some time 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 (2015), 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. A "Confused Johnny" study made in 2013 said that users actually preferred to see some of the innards of encryption, so they could be sure that encryption was taking place, but that they preferred not being involved with key handling.

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 hosed. 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 hosed. 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 to the world 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, usually 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, I usually have no assurance, other than the digital signature of people I don’t know, that the keys actually belong to the right 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 are 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. 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. Assuming you can remember it, you don't need to write it down anywhere, and certainly you don't have to use a flash drive, or store it in a file anywhere, plain or encrypted. PassLok will complain that it's kind of weak but will still make a public key to match it. Even better, it will compensate automatically for the weakness of your key. The only thing you might notice is that processing is kind of slow, which will hopefully encourage you to use a better key.

And by the way, PassLok doesn't have "public keys" and "private keys." It has "Locks" that people can put on a text or file so nobody can read it, and "Keys" that unlock a locked item. 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. If you feel lazy, you can just paste the Lock into PassLok, and then PassLok will remember it from then on, especially if you use the

Chrome app version, which syncs everything between computers as soon as you log into your Google account. 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 I 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 feel better about it 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-I,n 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 spelling out my Lock for the whole world to see, and it becomes one with my Lock. I've actually done that every time the evolution of PassLok has made my old Lock obsolete.

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 of verifying the source, so it accepts everything, including public keys that might have been made 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. There are at least four different PGP keys bearing my name, and not even I can tell which is the good one.

PassLok had no key server for the longest time because it is not strictly necessary. Starting with version 1.7, however, a "General 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. 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. Signed 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 (plus a more stringent "Read-Once" mode, starting with version 2.1), in addition to the anonymous and signed modes. PFS is short for "Perfect Forward Secrecy," which means that nobody, including those originally involved in the message exchange, can read an old message after the exchange is over. In the case of Read-Once messages, they sort of "self-destruct" and become unreadable right away. In other programs, this typically requires participants in a conversation to be connected at the same time to some server (not PGP, though, since it does not have a real-time component or anything resembling forward secrecy), 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 never leave the device where they were made and are destroyed after they are used, the messages become unreadable to everyone.

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, was 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, under any operating system. 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://, https://, ftp:// and so forth, which a webpage needs to execute in order to communicate with the outside.

For the more paranoid amongst us, hashes of the source code (such as the SHA256 hash) are published with each new version of PassLok, so you can check its integrity. Just tell the browser to show the source code, copy it, and paste it into a hash-making program or online utility. If the code is genuine, there should be a match with the published hash. If you are still concerned that the sources displaying the hash codes might get hacked by someone who also changed the working code, watch the optional video of me reading the hash, which is a lot harder to fake. Once you have a copy of PassLok that you trust, you can save it somewhere within your device, or in the cloud so you can use it when you need 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. PassLok hashes, on the other hand, are published on *different* servers, and they include videos of the author reading them. Unlike with compiled programs, PassLok users don't have to trust it in any way before they verify the code as genuine. The code is perfectly dead until they put anything on it and start 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 signature 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 detect whether or not there is one at all. This functionality is accessed by checking a single "Decoy mode" box, so named 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, the new data replaces the noise within the image, so human eyes are unable to detect anything being there.

Let me just mention one last killer PassLok feature that PGP can only dream about. It turns out that PassLok is not restricted to asynchronous communications like email. Beginning with version 2.1, PassLok has a real-time chat component that includes the exchange of text, files, audio, and even video. For several participants at the same time. End-to-end encrypted. Using a single button on the interface. PassLok chat, which is based on the new webRTC protocol currently supported by Chrome, Firefox, Opera, and a couple more browsers, begins with a user making a locked invitation for other users to join in a chat. This invitation can be sent out safely by email or similar means. When the time for the chat arrives, participants only need to click a button to open the invitation, give themselves names for the chat, and join in. The connection is direct between each pair of participants. Everything is encrypted by the browser itself and there is no record anywhere.

And I'll stop here. I could go on talking about data compression, which makes Passlok-locked items less than half the size of items encrypted with other programs, or about Reed-Solomon error correction, which preserves the functionality of most PassLok items even if they are corrupted, or about variable key stretching, which flicks the finger at hackers by forcing them to spend a lot of computational effort on the worst possible Key choices. All under the hood, so the user doesn't even know it's there but is benefitting from it nonetheless. The list of features unique to PassLok goes on and on.

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. The Chrome app version syncs automatically between computers.
- 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 authenticated 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 does only asynchronous communication such as email. PassLok includes real-time, peer-to-peer secure chat, which can involve even video.
- PGP requires a noise-free channel. Not so PassLok, which includes two kinds of automatic error correction.
- 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 (storage available)
Small public keys	No	Yes
Spread public keys w/o Internet	No	Yes
Encryption modes	1	5
Short message mode	No	Yes
Hidden messages	No	Yes

Fake text steganography	No	Yes, 2 modes
Image steganography	No	Yes
Secret Sharing	No	Yes
Transparent authentication	No	Yes
Error correction	No	Yes, 2 kinds
Real-time chat	No	Text, files, audio, video
Encryption bitlength	128 or 256	256 always
RSA-equivalent public key strength	1024- or 2048-bit	+15000-bit
Variable key stretching	No	Yes
Main interface pages	many	1
Learn mode	No	Yes
Built-in tutorial videos	No	Yes
Jargon-free	No	Well . . . almost