PassLok 1.4 manual

by F. Ruiz, 9/6/2013

This version contains no images, but future versions will, 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) Creating Keys and Locks
   a) How to make a strong key
   b) Generate your matching Lock
   c) Spreading your Lock
   d) Changing Locks
   e) Making a Lock database
3) Locking and Unlocking
   a) Regular locked messages: msl
   b) Key-locked messages: msk
   c) Short locked messages
4) Signatures
   a) Putting a signature on a text: sig
   b) Verifying someone's signature attached to a text
5) A walkthrough of the interface
   a) Boxes and spaces
      - JavaScript check
      - Expandable titles
      - Key strength meter
      - Box 1: Key/Lock
      - Box 2: Text
   b) Checkboxes:
      - Learn mode
      - Show
      - Decoy mode
   c) Buttons:
      - Help
      - Make Lock
      - Split/Join
      - Clear (box 1)
      - Email
      - SMS
      - Words

What is PassLok?

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

What is public-key encryption?

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

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

What makes PassLok special?

PassLok is designed to help average people *after* they find themselves in a surveillance situation. If they had had time to communicate privately with a friend before the surveillance started, they would have been able to establish a shared secret key that would enable them to communicate using symmetric encryption. There are many good programs out there for symmetric encryption, including a bunch that are based on AES (Advanced Encryption Standard), which the US Government chose in 2001 as its encryption program, after a yearlong contest involving a dozen candidates, but these people can't use them because they don't have a shared secret with their friends.

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

What allows PassLok to function where the others cannot is its perfect portability. PassLok is not really installed on your computer; it is not running on a server, either. Since nothing secret is stored anywhere, nothing can be compromised. PassLok runs equally well on your Mac or Linux machine, your neighborhood library's public PC, or a passerby's smartphone, and it disappears after it does its job.

These are the principles guiding the design of PassLok:

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

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

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 anything as secret key, precisely so they can remember it without having to write it anywhere. 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 200,000 iterations of keystretching, 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 can crack easily. PassLok does not force users to comply to rules in choosing keys, so that a user's key can indeed be anything he/she likes. Instead, PassLok evaluates key strength and applies a variable amount to PBKDF2 key stretching according to the result. Since a large number of key stretching iterations also cause a large delay to the user, this serves to encourage users to come up with stronger keys, without forcing them directly.

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

And then, the whole thing fits in one screen: two boxes and a few buttons. Help pops out if requested or if the user chooses Learn mode, so that every button press brings out a dialog explaining what is about to happen.

Keys and Locks

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

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

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

Creating Keys and Locks

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

How to make a strong Key

Since the cryptographic methods used in PassLok are rather overkill for today's computer power, the weak link in the chain is the insecurity of the personal or shared Key chosen by the user. It has been shown that 80% of users choose bad keys, which a specialized hacking program can guess in a matter of seconds. The same program running for an hour would guess 99% of passwords made by actual users.

This is a real danger for PassLok as well. Since the Lock made from a user's Key is public, a hacker can find the 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, however, that there are only so many reasons why keys can be bad. Hackers know them, and this allow them to guess the keys very quickly. If the user avoids making those mistakes, the Key likely won't be cracked. So here's how a typical key-cracking program proceeds, and how to thwart it:

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

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

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

1. The key must have sufficient length. PassLok marks as Terrible any Key having fewer than seven 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 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 look into 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. You can add whatever you want, but PassLok is hardwired to give you a better Key strength score if you add an email address to your Key.

How can you tell if your key is any good? Just write it into box 1 of PassLok, and a text will appear giving you the score. Now, PassLok does not include a dictionary so it cannot tell whether what you type is a real word or not; that's up to you to decide. But PassLok will tell you if it's too short or the keyspace could have been made bigger by adding other kinds of characters. It will not let you get away with things like adding a number at the start or the end, and it will reward you if you add an email.

Generate your matching Lock

After you type your secret key into box 1, you are ready to make its matching Lock. To do this, just click the Make Lock button. The Lock will appear in box 2, replacing whatever was there before.

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

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

Spreading your Lock

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

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

A previous version of PassLok included a QR button to display a QR code containing a Lock. This functionality has been removed to lighten the code but the idea still holds. If you don't want to go online in any way and the person requesting your Lock is present,

you can copy the Lock into a QR code-making app (there are many good, free ones for iOS and Android) and display it on your device's screen. The other person now only needs to use a QR code reading app in his/her smartphone in order to retrieve your Lock.

Other ways of spreading your Lock are:

- Add it to your email signature. That way anyone who receives an email from you will also get your Lock. Unlike PGP public keys, which also get spread this way, PassLok Locks don't add much bulk to your signature, barely one line.
- Write it in 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.

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 pass on to you something else that suits his/her evil purposes. You'll never know that this person has become the "man in the middle." How can people know that a Lock is genuinely yours, in such case?

This is why PassLok has an ID button. When you press it with your Lock displayed in box 2, a series of numbers replaces the Lock. These numbers constitute a unique ID for your Lock, which you can read over the phone or within a video call. The person on the other end of the line can also display the ID of the Lock he/she has received, by placing it in box 2 of PassLok and clicking the ID button. If the ID shown on his/her device is the same shown on your end, then everything's good. You could have read the actual Lock to each other, but likely the ID is a lot easier and quicker to read. The ID won't work as a Lock, though, and you cannot get the Lock from the ID. To be able to send you a locked message, the other person must have your actual Lock in his/her possession.

After you have made a Lock that you are going to distribute, it is a good idea to make a video of yourself reading the ID, and post that somewhere online where people can get it. Then you can put the URL of that video in the places where you have posted your Lock. Have music playing in the background as you record the video. This way, a hacker will have a tough time chopping your original video into pieces and putting it back together so it appears that you are reading a different ID.

If it is impossible to use a rich channel such as voice or video to authenticate a Lock, there are still ways to detect by text or email if one has been given a counterfeit Lock, likely by someone who wants to listen in the conversation. It is explained in the Advanced section.

Changing Locks

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

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

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

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

Making a Lock database

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

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

If you need a Lock to lock something for somebody, a trick that I found works quite well is to keep your webmail contacts database open in one browser tab while PassLok is open in another. Navigate the address book to the individual in question, and copy the Lock into the clipboard. Then go to the tab containing PassLok and paste the Lock in box 1. You're set to go.

Even the most user-friendly PGP-based programs, such as Mailvelope, take more steps in order to do the equivalent thing, because PGP public keys are so large that they don't fit in webmail address books and therefore those programs must have their own databases, which remain locked to a particular device. 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.

A future release of PassLok in the form of browser plugins, able to integrate with popular webmail sites and their address books, is planned. But this introduces its own set of risks, which will need to be sorted out with great care.

Locking and Unlocking

Having covered the essentials of what PassLok is and how to take care of the most critical ingredient to maintaining security, we go now to the nuts and bolts of actually locking and unlocking messages. Despite its tiny size, PassLok is one complex piece of software, comprising three 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.

Regular locked messages: msl

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 a regular locked message. It looks like a piece of gibberish, bracketed by PL**msl tags, where ** is the version number.

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

1. Fetch the recipient's Lock and paste it in box 1 of PassLok. He/she would have sent it accompanying an email, or in and SMS text message, or posted it publicly, or displayed it as a QR code, which you scanned. You need this Lock before you can do anything else. It is masked by default, so if you want to display it, check the Show checkbox. 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. PassLok will strip the tags automatically if the Tags checkbox is unchecked prior to locking.

2. Write or paste your message in box 2. Click the Lock/Unlock button. The locked message will appear in box 2, replacing the original message. Copy it and paste it into your communications program or click Email to open your default email. If you click the QR button above box 2, it will be displayed as a cellphone scannable QR code at the bottom of the page.

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

1. Write your Key into box. It is masked by default, so if you want to display it, check the Show checkbox.

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

If the unlocking fails, this is likely because the locked message has been corrupted (make sure the tags are intact) or because the Lock used to lock it does not match your secret Key. Remember one more thing, though: since the Lock is public, you won't necessarily know for sure who has sent it. Proceed accordingly. This is likely, however, 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: msk

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 personal Key, for instance, the recipient won't be able to unlock the message because he/she does not have your personal Key (remember that you should *never* give your personal Key to anyone, or even write it down). All that was said above about making good Keys applies to making shared Keys.

In this mode, the locked gibberish ends up bracketed with tags reading PL**msk, where ** is the version number. PassLok will omit the tags if the Tags checkbox is unchecked prior to locking. Here's what you do to lock a message with a shared Key:

1. Write or paste the shared Key in box 1. It is masked by default, so if you want to display it, check the Show checkbox.

2. Write or paste the message in box 2. Click the Lock/Unlock button. The locked message will appear in box 2, replacing 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 locked in this mode, which you can tell because the tags end in PL**msk, here's what you do to unlock it:

1. Write or paste the shared Key in box 1. It is masked by default, so if you want to display it, check the Show checkbox.

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

Unlike the previous mode, you would know who sent you a Key-locked message, because only people knowing the shared Key, which you also know, are able to lock the message to begin with. If the unlocking process fails, this is usually because the locked message has been corrupted (make sure the tags are intact), or because the Key used for locking is not the same you tried for unlocking. Unfortunately, this mode is hampered by the need for a previously shared secret between sender and recipient (or more people, if that should the case). One way to get around this quandary is to use the regular locking mode to exchange a new Key between the parties, and use the Key-locked mode from then on.

Short locked messages

Both regular and Key-locked messages are at least 200 characters long, which wouldn't fit in a cellular text message (SMS) limited to 160 characters. PassLok has a special mode for situations when you want the locked message to fit in an SMS. It's a symmetric mode similar to the Key-locked mode, so you need to share a secret with the recipient prior to sending the message. It is also limited in the size of the text that can be locked.

In order to fit a message as long as possible, the locked message has 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. Make sure box 1 is *empty* (otherwise the Key-locked mode will be triggered). Click the Clear button under box 1 if you have to.

2. Write or paste your message in box 2. Message length is limited to 57 ASCII characters. Non-ASCII characters use 6 spaces each, so avoid them if you can. Any text beyond the limit will be lost. Hit the Lock/Unlock button.

3. A popup will ask you for a Short Message Key, which will be necessary to unlock the message. Write it in and click OK. The locked message will appear in box 2, replacing the original message. Copy it and paste it into your communications program.

4. If you are using a smartphone, the SMS button will open your SMS app. This does not transfer the contents of box 2 to the app, though, so make sure to copy it to clipboard before you click this button. If you click the QR button above box 2, it will be displayed as a cellphone-scannable QR code at the bottom of the page.

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

1. Paste the locked message in box 2. It is okay if it is broken up by spaces, carriage returns, and special characters. Then click the Lock/Unlock button.

2. A popup will ask for the short message key, which is the same used for locking. Write it in and click OK. The unlocked message will appear in box 2, replacing the locked message.

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

Signatures

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

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

One thing is different in signing/verifying from locking/unlocking, and this is that the signed text does not turn into gibberish. Rather, the signature is a separate piece of gibberish, exactly 250 characters long plus PL**sig tags at either end (again, ** is the version number), which PassLok adds at the bottom of the signed text, which remains perfectly readable. If one wants to lock it, it can always be done after that, in either the regular or the Key-locked modes. PassLok will omit the tags if the Tags checkbox is unchecked prior to locking.

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

Putting a signature on a text

Unlike in locking/unlocking, there is only one mode in making and verifying signatures.

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

2. Write or paste the text to be signed in box 2. Click the signature/Verify button. A signature matching the text and the key will be appended at the end of the text in box 2. 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 Tags checkbox is unchecked prior to signing. It is also okay to split the sign with spaces, and punctuation other than line returns or = + or /.

Verifying a signature

The process for verifying a signature affixed at the end of a text is this:
1. Paste the lock of the person who made the signature into box 1. 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 the text with its signature appended at the end in box 2. 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 signature/Verify button. A popup message will say whether or not the signature for that text has been verified.

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

A walkthrough of the interface

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

Boxes and spaces

JavaScript check

Right under the "PassLok privacy" title is a single-line box, telling you to click on titles to get some help. The background is green. This is what you get if everything is OK and the computations are able to run.
Otherwise you get a warning message with a red background. PassLok is not operational if this is displayed. Probably JavaScript support is turned off in the browser (go to its settings to fix this), or the browser or the device itself are just not capable (use a different browser or a different device).

Expandable titles

A number of titles have an inverted triangle sign at the end. This means that an extended description will roll out if you click on the title. The description will tell you what you can do with the corresponding box or control. Click on the title again to hide the extra text.

Key strength meter

As you begin to type into Box 1, words appear somewhere above it, telling you PassLok's opinion on the contents of box 1, were it to be used as a personal or shared Key. The built-in meter evaluates keys based on the kind and number of characters used. It cannot detect whether the text might be in a cracker's dictionary, so make sure you obfuscate or combine words as described in the previous section. If you want a more sophisticated indicator of how good your Key is, you may want to spend some time with ZXCVBN.

Box 1: Key/Lock

Here is a single-line box where you write the strings that allow you to lock, unlock, sign, or verify a sign. It is masked by default because its content is secret most of the time and you wouldn't want a casual eavesdropper to see your secret stuff. You may be also showing the screen to someone. If you want to see that's being typed, there is a Show checkbox for that. There's also a Clear button (but no Select button, to discourage copying what's in the box).

Box 2: Text

This is a multi-line, expandable box (in a computer), where the plain text to be locked or signed, or the locked text to be unlocked, is placed. The box is also used to display the output of most functions resulting in a text string: new Locks, IDs, locked messages, signs, newly unlocked messages. With the exception of signs, which are added to the original text, the output material

always replaces the previous content of this box, so make sure you have copied it into a safe place before you do anything.

Checkboxes:

A few choices are "sticky", and are therefore toggled by checkboxes that remain checked or unchecked until they are clicked again:

Learn Mode

Checking this box, near the top of the screen, will cause a popup to appear every time a button is clicked (except for Clear and Select). The popup will detail to the user what is about to happen, and offer to cancel it. This is useful to learn the ins and outs of PassLok. The default is no Learn Mode.

Show

When this box is checked, the text in box 1 is visible. Otherwise you can only see a generic dot for every character. This is useful to avoid mistakes in typing a Key, provided you are in a snoop-safe environment. The default is not to show what's in box 1.

Tags

Unlike the other checkboxes, this one is checked by default. If checked, appropriate tags of the form PL**###, where ** is the version number and ### is a string identifying the item, are added at the beginning and the end of every Lock, message, signature, or Key part generated by PassLok, and emails invoked by the Email button include text to help the recipient identify the contents. If unchecked, tags and additional text are omitted when the corresponding items are generated.

Decoy Mode

Checking this box, located below box 2, causes PassLok to expect a second, hidden message in addition to the main message in box 2, locked under a different Key. When you are locking or signing, popups appear asking for the second key and the hidden message. In unlocking or verifying, the popups ask for the key for the hidden message, and display that message if successfully unlocked. This mode is especially useful if you suspect that someone 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.

Buttons:

PassLok functionality is invoked by pressing buttons. Three of them perform essential cryptographic functions, which are different depending on what's in the boxes. The rest invoke auxiliary functions. The following lists the buttons by location, from top to bottom and from left to right.

Help

This button opens a new tab in the browser, containing step-by-step instructions for most things a user might want to do, plus a couple introductory lessons and a link to this manual (which will open in yet another

tab). Clicking the titles on the new help tab opens or hides the respective instructions. The SHA256 ID of the source code itself, useful to ensure PassLok has not been tampered with, is on this help screen.

Make Lock

This makes a Lock matching whatever is in box 1, and displays it in box 2. PassLok will take anything in box 1 as a Key to derive a Lock from, but it will alert you if box 1 contains a Lock. The Make Lock operation can get very slow if the Key strength indicator says it's Terrible or Weak. This will happen every time that string is used as a personal or shared Key, so this is the moment to choose something else.

If box 1 is empty when this button is pressed, PassLok generates a 85-character random Key and places it in box 1, then displays its matching Lock in box 2. The string in box 1 can also be used as a shared Key, ignoring box 2. This is useful if you want a throwaway, very secure Key for a chat session, so that it can be forgotten when he session is over.

If box 2 contains a Lock when the button is pressed, the Key in box 1 and the Lock in box 2 will be combined into an 86-character Key, which appears in box 1. This is the best way to establish secure communication between two people (extensible to more people, if needed), since the other party can come up with the same shared Key without having to exchange any secret information.

Split/Join

When this button is pressed with box 2 empty, a popup asks for the number of parts needed to retrieve the Key. PassLok performs a Shamir Secret Sharing algorithm to generate those parts, which are displayed in box 2. Extra parts can be made by pressing this button repeatedly. If a sufficient number of Key parts are present in box 2 when this button is pressed, the same algorithm combines them to recreate the original Key, which is displayed in box 1.

Clear (box 1)

This button clears box 1. The action is irreversible.

Email

When this button is pressed, the contents of box 2 are formatted into an email message, open in the default webmail program known to the browser, with only title and recipient's address to be filled. All of this happens in a new tab, so PassLok remains open in its tab. This will only work if PassLok finds a Lock, a locked message, or a signed text in box 2, so you don't accidentally email plain text.

SMS

This button does nothing in regular computers, but in mobile devices, it opens the default SMS (texting) app. Since nothing is copied from the PassLok page, make sure to copy into the clipboard whatever you want to send, before pressing this button.

Words

When this button is pressed with something present in both boxes, any PassLok item in box 2 is converted into fake text, which is useful to avoid detection by email scanners. To retrieve the original item from fake text, place it in box 2 and click the same button. Pressing the button with box 1 empty will use box 2 as dictionary for making fake text, which allows the use of any language instead of the default English. Pressing the button with box 2 empty displays the current dictionary. Dictionary changes are not permanent, so that when PassLok is reloaded, it goes back to the default English dictionary.

ID

When the ID button is pressed, the contents of box 2 are replaced with their near-unique SHA256 hash in hexadecimal code, formatted in groups of four characters. If box 2 contains a PassLok string having fewer than 200 characters (likely, a Lock or a sign), its tags, spaces, and extraneous characters are removed before taking the hash. This is useful to authenticate someone's Lock, for instance, since it is much easier to read the ID over the phone than the Lock itself. Bear in mind, however, that it is impossible to recover the original information from its ID.

Another likely use of this button is for checking the integrity of PassLok itself. To do this, view the source of PassLok in a separate tab (every browser does this differently), copy it, and paste it in box 2, then click the ID button. The resulting ID should match the one published in the help file (displayed with the Help button), for the same version of PassLok.

Lock/Unlock

This button also has context-sensitive functionality. If PassLok does not find a locked message in box 2, it will lock the contents of box 2 with whatever is in box 1, in this way:

- If box 1 contains a Lock, it will make a regular locked message with PL**msl tags (omitted if Tags is unchecked), to be unlocked with the recipient's Key.
- If box 1 contains something else (except an 86-character string), PassLok will use it for a shared Key-locked message with PL**msk tags (omitted if Tags is unchecked), to be unlocked with the same shared Key.

- If box 1 contains an 86-character string, it will also use it as a shared Key, but the tags will be PL**mss (omitted if Tags is unchecked), for "signed" message, which is unlocked differently.
- Finally, if box 1 is empty, PassLok will lock the first 57 characters of box 2 into a short message exactly 160 characters long, which will fit in an SMS text message. The short message key needed for locking is obtained via a popup.

If a locked message occupies box 2 when the Lock/Unlock button is pressed, PassLok will attempt to unlock the message, depending on the type of locked message it finds. In all cases, the successfully unlocked message ends up replacing the locked message originally in box 2:

- If it is a regular locked message (PL**msl tags), PassLok expects to find in box 1 the Key matching the Lock used to lock the message. It will return a warning if this is not found.
- If it is a Key-locked message (PL**msk tags), PassLok will take the string in box 1 and use it as a shared Key to unlock the message. Again, different kinds of warnings will be displayed instead if something goes wrong.
- If it is a signed message (PL**mss tags), PassLok will expect an 86-character string in box 1, as a result of having previously combined your secret Key with someone else's Lock. If it doesn't find it, warnings will be displayed in popups and the unlocking will fail.
- If the message has no tags and is 160 characters long, PassLok interprets it as a short locked message. It will then display a popup asking for the short message key. As usual, helpful comments are displayed if unlocking fails.

Finally, if there are no tags and the message is not 160 characters long, PassLok will fail to recognize it as a locked message and will attempt to lock it instead. So please make sure the tags are readable if box 2 does indeed contain a locked message. It may also happen that your unlocked message is exactly 160 characters long, so that PassLok insists on unlocking it. This is easily solved by adding or removing a character somewhere; a space is enough.

Sign/Verify

This is the third button having context-sensitive actions. First PassLok tries to find a signature at the bottom of box 2. If it doesn't find it, it concludes that you want to sign the text, and will do so provided you have written a valid Key in box 1, otherwise it will complain. If PassLok finds that a signature is already there, it will attempt to verify it with the Lock matching the signing Key. A Lock must present in box 1, or there will be popups complaining. Then another popup will say whether or not signature verification has succeeded.

Select (box 2)

Selects the complete text in box 2, so it can be copied, cut, or erased.

Clear (box 2)

Clears box 2. As with the Clear button under box 1, this action is irreversible, so be careful.

Advanced features

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

Using fake text

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

To help with this problem, PassLok includes a fake text encoder, beginning with version 1.4. It is implemented into a single "Words" button above box 2. Here's how it works:

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

1) Check that the item to be converted into fake text is in box 2, and that box 1 is not empty. Since PassLok tags, which always contain the same characters, will lead to the same set of words, you may want to remove those tags before making the fake text. 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 dictionary using the process described in the next item.
3) Click the Words button above box 2. The contents of box 2 are converted into fake text using the current dictionary and displayed in box 2, replacing the previous contents.
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.

To retrieve the original PassLok item from fake text:

1) Check that the fake text is in box 2, and that box 1 is not empty.
2) If the fake text is not English, you will have to change the default dictionary using the process described in the next item.
3) Click the Words button above box 2. The fake text in box 2 is converted back into the original item and displayed in box 2, replacing the fake text.

Now, the default fake text encoding is based on a piece of English text (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 dictionary (the GNU license text) with something else. To do this, follow these steps:

1) Copy a sufficiently long text (must have at least 70 different words) and paste it into box 2.

2) Make sure box 1 is empty, and click the Words button.
3) A popup will ask you to confirm the dictionary change. If the change is unsuccessful, another popup will say why. Typically, failure to change the dictionary is due to not having a sufficient number of different words. Use a longer text and try again.
4) The recipient of your messages turned into fake text must have the same dictionary installed. One way to ensure this when using a non-English language is to display the default dictionary, copy it into a translation utility such as Google Translate, and then use the translation as the new dictionary.

Be aware that the dictionary will go back to its default if PassLok is reloaded. If you want to make the change permanent, the easiest thing to do is edit the PassLok source itself. The default dictionary is quite easy to spot, near the end of the source code.

Also be aware PassLok 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. Unfortunately, PassLok has no way to know the language of its fake text dictionary. If you dare to edit the source, one quick fix for German would be deleting the ".toLowerCase()" strings in the source, and disabling the instruction that capitalizes the output text. This instruction can be found easily since it contains two ".toUpperCase()" calls.

Locked and signed messages: mss

A regular locked message is anonymous by nature, so the recipient cannot be sure of who has locked the 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 message with your secret Key, and then lock the result with the recipient's Lock. When the recipient gets the locked message, he/she will unlock it first using his/her secret Key, and then see the signature, which then he/she can verify using your Lock. If it verifies, that means the message comes from you, sin you only have the Key that was used to make the signature.

But there is a way to do this in one step, and that is using a special shared Key that derives from both personal Keys, and which does not need to be exchanged beforehand. This is how you do it:

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

2. Paste the recipient's Lock in box 2. Click the Make Lock button. The shared Key resulting from your Key and the other person's Lock will appear in box 1, replacing your Key, and box 2 is cleared.

3. Now you can use that special Key, which will be exactly 86 characters long, to lock a message that you write or paste in box 2. The resulting locked message will have special PL**mss tags, signaling what kind of Key is required to unlock it.

When the recipient tries to unlock your message, PassLock will alert him/her that it is a special signed message and that the special shared Key must be present in box 1 so unlocking can succeed. He/she will then generate that Key by following steps 1 and 2 above, except that the process will now involve your Lock and the recipient's secret Key. Thanks to the mathematical magic of the Diffie-Hellman key exchange algorithm over elliptic curves, the recipient will find in box 1 exactly the same special 86-character shared Key that you produced, and the subsequent unlocking operation of the locked text pasted into box 2 will succeed.

Because this process involved using your Lock, which can only be produced from your Key, the recipient can be assured that the message comes from you, even though it bears no explicit sign. The process can be extended to more than two parties, as we'll see below.

Decoy mode

Now you may say, "All this business of locking and unlocking is very cute, but the world is not so cute. The moment they see you're using cryptography, someone's going to come knocking on your door and ask you for the key, first nicely, then maybe not so nicely. And if you don't talk, maybe someone else will." What's to be done in this unfortunately not-so-rare scenario, commonly known as "rubberhose attack" (in honor of the instrument commonly used to extract the key)?

This is why PassLok includes what is called in cryptography a "subliminal channel." A subliminal channel is a container for information whose 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 termed "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 there is indeed no hidden message, after supplying the personal or shared Key. Those trying to find what you're up to may be disappointed by what they read in your now unlocked messages and suspect that there is more, but they cannot know for sure. It becomes unreasonable to keep bothering you after you've supplied what they asked, and chances are they'll let you go.

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

1. Check the Decoy mode checkbox below box 2.

2. Follow the instructions for any of the three types of locking or for signing, using a shared Key, the recipient's Lock, or a popup short message Key. If Decoy mode is checked, a popup will ask for a decoy Password to lock the hidden message.

3. Write or paste into the popup box the Password for the hidden message and click OK. It is all right to enter nothing in this box, and in this case the hidden message can be retrieved without a Password. Think of it as putting the house key under the floor mat.

4. A second popup will ask for the hidden message itself. Its length is limited to 152 ASCII characters in Key-locked and signed modes, 87 characters in regular locking mode, 38 characters in short message mode, 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.

5. After clicking OK, the locked message containing both the main text and the hidden text will appear in box 2, replacing the original text. In the case of signatures, new material is added instead. Copy it and paste it into your communications program, or use the Email and SMS buttons. As with regular locked messages, it is okay to strip the tags up to the "=" sign, but not recommended. PassLok will do this automatically if the Tags checkbox is unchecked prior to locking. It is also okay to split the locked message with spaces, line returns, and punctuation other than = + or /

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

1. Check the Decoy mode checkbox below box 2.

2. Follow the instructions for any of the three locking modes, using a personal or shared Key, or nothing initially, or for verifying a signature. If Decoy mode is checked, a popup will ask for a decoy Password.

3. Write or paste into the popup box the special Password for the hidden message, if there is one, and click OK. The hidden message, if it exists, will appear in a popup even if the main unlocking fails or the signature is not verified. The main message will appear in box 2 if the main unlocking is successful.

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

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

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 impossible to remember and, if users write them down, the possibility that they'd be lost or compromised increases so much that they are not worth it anymore.

But there are exceptions, noted below. This is why PassLok helps you to generate a random Key and its corresponding Lock, should you ever need one. Just click the Make Lock button with box 1 empty. Box 1 will fill with an 85-character random password, and its Lock will appear in box 2. The 85-character length is so that the random Key (which can be used equally well as a Key) can be distinguished both from a shared Key (86 characters) and from a Lock (87 characters inside the tags). Don't worry: 85 base64 characters is still more secure than you'll ever need.

Splitting and joining Keys

There is a very clever mathematical trick that allows PassLok, starting with version 1.4, to split the secret string in box 1 (typically a Key) 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 Key, place it in box 1 while box 2 is empty and click the Split/Join button. popup will ask for the minimum number of parts that will be required to reconstruct the Key. The parts then appear on separate lines in box 2. If you need spares, click Split/Join again for every spare created, which will be added to the list in box 2. You can create from 2 up to 255 parts.

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

The process to rejoin the parts is similar: place the parts on separate lines in box 2, each part occupying a single logical line (which might wrap inside the box), and click Split/Join again. If the parts are from the same set, there are enough of them, and they are not damaged or corrupted, the Key will appear in box 1. If something is wrong, nothing happens.

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

- You want to use a high-security random Key, but since it is so hard to recall you feel forced to write it down somewhere, which is a no-no security-wise. But if you split the Key first and then write down the parts at different locations (which might be different files within the same computer), the risk of compromise is greatly reduced. When you need the Key, you retrieve the parts from their respective storage locations (which hopefully you remember), and then reconstruct the original Key. You may want to make one or two spares in case you forget the location of some parts.
- You want to send locked messages that can be read by several people, but you don't want to go through the process for generating a group Key, described below. In this case you create a Key and split it into two parts, and then make enough extra parts for all the recipients, plus one. You send each recipient one

of the parts made (locked with their personal Locks so no one else can get them), and the remaining part is sent in the open, along with the message locked with the full Key. The recipients can regenerate that Key by joining the extra part that you sent and they part they each have, and then they unlock the message.

- You want to force two or more people to cooperate, or at least talk to each other by digital means. One way to do this is to lock something important with a random Key, which gets split into parts, which you send separately to those individuals. The only way they can retrieve the important item is by pooling their Key parts together in order to reproduce the locking Key.
- Splitting can also be used as an alternative to locking with a Key, since the string split into parts can be very long. To do this, place the text to be locked in box 1 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.

Making sure PassLok is genuine

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

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

First of all, be paranoid and load 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 very difficult. At the time of this writing, the official PassLok source is https://www.autistici.org/passlok. The shorter address passlok.com redirects automatically to this source. Another mirror, which is located in US territory so it's not as secure from interference as the one above, is https://passlok.site44.com. Finally, there is a Github page containing the code at: https://github.com/fruiz500/passlok.

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

1. Load PassLok from the Web, or from storage (local or cloud) if previously saved as html as described in step 4 below. Yes, it is an excellent idea to have your own copy of PassLok stashed away somewhere safe (after you've checked its legitimacy, that is).

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). Unfortunately, this is rather difficult to do in a mobile browser today, but maybe someday mobile browsers will catch up in this respect.

3. Back in the PassLok page, make sure box 2 is clear, then paste the clipboard (CTRL-v or cmd-v) into it. Click the ID button. The self-ID of the PassLok source code will be displayed in box 2, replacing the previous contents.

4. For better security, you can do a SHA256 of the source code using an external program or online utility. In this case you may need to paste the clipboard into a text or html editor so you can save it as a file (html or txt). **DO NOT save the code using the "save" command from the browser menu**, since this command tends to modify the source page before it saves it. A copy of PassLok saved this way will still run, but its SHA256 will be different. If your operating system is Windows, do not use the built-in Notepad program, since it cannot save text with the appropriate encoding (UTF-8, no BOM). Be sure there are no extra spaces at top and bottom, since this would affect the result. Then obtain the SHA256 of this file using the external utility.

5. Go to the PassLok Help page and look at the ID for this version of PassLok. If this ID and the one obtained in step 3 or 4 are the same, the program has not been tampered with.

Now, I'm sure you realize that the process as so far described has a gaping flaw. If someone can get to the server and change the PassLok code, so can he/she also change the ID in the help page to match whatever the SHA256 of his/her special version of PassLok. Sure, but can that person change the Youtube video of me reading the genuine ID, which is linked from the same help page? It's not like I have an instantly recognizable face, but this is bound to be pretty difficult to tamper with.

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, mind you. That would be for experts to test and discover, which is facilitated by being able to see 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 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 (which should match the one you have). An interloper who is watching and perhaps modifying

your traffic won't be able to unlock the message to change it, and thus he/she must be content with preventing you from getting that locked file, in which case you'll know the Lock you have is fake.

But the easy way has the problem that the other person's answer must be exactly the answer I know, down to the smallest spelling detail, or the message won't unlock. There is another way to authenticate a Lock using a variation on the "interlock protocol," which admits answers that don't have to be exact. It is enough if the persons asking the questions can recognize the answers as valid in a more general sense. Look at this exchange between Alice and Bob, for instance:

1. Alice obtains her friend 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:
   "Dear Bob. I just got your Lock for the app called PassLok from your email signature. I fear that I'm under surveillance, so it's very important that I make sure that this Lock actually belongs to you. Here's what I want you to do:
   a. Write a question whose answer only the two of us know, and lock it with my Lock, which is included at the bottom of this email. Then split it in two parts and send me the first part. I'll be waiting for it.
   b. When I get it, I'll send you the first half of a similar question, which has been locked with your Lock. When you get it, send me the second half of your locked question.
   c. When I get that, I'll send you the second half of my question, and also the answer to yours, which then I'll be able to read. I'll send the answer locked with your Lock.
   d. If I answered your question correctly, put together the two halves of my question and unlock it. Then write the answer, lock it with my Lock, and send it back to me. Then I'll know that your Lock is authentic. Your friend, Alice."
2. When Bob gets this, he decides it's going to be fun to do all his, and writes a question whose answer only Alice knows, locks it with her Lock, which was appended to her email, splits the locked message into two parts, and sends Alice the first half.
3. Alice gets the first half, and writes her question to Bob. Then she locks it with Bob's presumed Lock, but only sends him the first half. Because nobody can unlock half a message, she must wait to get the second half of Bob's message in order to answer his question.
4. Bob gets the first half of Alice's locked question, and he sends her the second half of his locked question.
5. Alice gets the second half of Bob's question. Now she can put the two halves together and unlock them with her Key. She writes a message answering Bob's question, locks it with Bob's presumed Lock (no need to split it now), and sends it back to him along with the second half of her question to Bob.

6. Bob gets Alice's email and can now unlock both messages from her. He sees the correct answer to his question in the first one, so he unlocks the one containing Alice's question. He writes the answer, locks it with Alice's Lock, and sends it to her. Had he been unable to unlock Alice's question, he would have told her so. If her answer to his question was wrong, he would have told her, too.
7. Alice gets Bob's message, unlocks it and, seeing the correct answer to her question, is satisfied that Bob's Lock is authentic. Otherwise she gets a message from Bob telling her that things didn't work out, or something other than a message answering her question, or nothing at all, and she decides that "Bob's Lock" was bad.

An alternative to splitting locked messages is to make the ID of the locked or unlocked message and send it ahead of the message itself, or apply a signature to the message and send the signature ahead of the message. The recipient will then check the ID or signature after the message is received, and will know that something's wrong if it is not the same. Another option is not to lock the answers to the questions, in steps 5 and 6, since authentication works just as well if those messages are not locked; locking just preserves those answers, which might be sensitive, from a less-than-powerful eavesdropper who might see the exchange.

If Alice does not know Bob well enough to be able to ask him a question whose answer is known only to the two of them, or maybe Bob doesn't know Alice well enough to ask a similar question, there is still something they can do, so long as Alice can recognize Bob in some way, and Bob can recognize Alice. Instead of a personal question, the asker can direct the other person to simply say or do something contained in the "question" message, but to do so in a video or audio recording, which is then put somewhere in the cloud, and the URL is sent back as an answer. The asker will then see or hear the other person, whom he/she recognizes, saying or doing something that she/he would not be saying or doing unless she/he has read the question message.

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

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

Mallory will then get the whole question from Bob, so he will be able to unlock it, re-lock it, and pass it along to Alice in step 4, and then get from her a reply that will satisfy Bob in step 6. But the damage has been done. Mallory is committed to sending Bob the second half of a question from "Alice" that is most likely not the question the real Alice

asked, or otherwise Bob won't be able to unlock the message, and Mallory's cover will be blown. Bob might not discover the ruse at this point, but it is highly unlikely that his answer, or whatever else Mallory can come up with to replace it, will satisfy the real Alice's question. Then she'll know someone's in-between and Bob's Lock is not authentic.

If now they repeat steps 2 to 6 all over with one new question 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 question in step 6, and he will alert her of that fact. It is possible that a 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 answer to her question, or she will decide that "Bob's Lock" is bad.

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

Group Keys

Sometimes a Key that is shared by more than two people is needed. For instance, if a long document is to be sent by email to a bunch of people, and one foresees there will be a series of follow-up emails from people in the group, which everyone else ought to see. In that case, locking messages for each person can become cumbersome. It is best to set up a Key that is shared by everyone in the group, and use that for locking and unlocking everything.

There are several ways to do this. I'm giving you three below. Choose the one you like or make up your own. In all that follows, I will be assuming that everybody's personal Locks are known to everyone else in the group.

The easy way:

1) Make a list of the users in the group and have a way to pre-assign the task that follows to one of them. This could be random, by some sort of order (alphabetical, ages, whatever), or assigned by the boss.
2) This designated person then comes up with a strong Key, which he/she figures everyone can remember, or a random one, and sends it to everyone else (plus him/herself, if it is random) encrypted with their respective Locks.
3) Now they all unlock their messages with their respective Keys, and retrieve the shared Key, which now they all have. If they forget it, they can always retrieve it from the original locked message.

This is quite a simple method but it has the problem that multiple messages have to be sent, unless all the locked Keys are put in one message sent to everyone, likely with tags saying whose Lock was used in each case.

The harder way:

1) Make a list of participants, ordered according to a pre-established criterion, and split it two halves. If the number is odd, get as close as possible. Then split every half into two exact or approximate halves, and so forth until the largest piece has only two names. This sets up a sort of "tournament brackets" system, as in the "March Madness" NCAA basketball tournament.
2) Now everybody knows they have to make the Key they share with the other person in the pair. Alternatively, the first person comes up with a random Key, which he/she sends to the other.
3) Then the first person of each pair sends the Lock matching that shared or random Key to the members of the pair that they'd be matched with in the next bracket (possibly locked with their shared Key, for authentication). The other person receives it and makes the shared Key resulting from that Lock and his/her previous shared Key.
4) And so on and so forth. Eventually, it will come to two individuals exchanging Locks, and a Key shared by everybody will have been established, since they have all been getting copied in all the exchanges.

This second method takes quite a bit more communication but has the advantage that a shared secret never travels, even as a locked message, only the Lock derived from it. Compromising one of the Keys still reveals the final secret, for the attacker having access to all the emails would be able to pose as the participant who lost that Key.

The third one, which involves only one message sent to everyone. There is some work needed to prepare it, and then some work needed after it is received. Here's the first part:

1) Make a list of participants according to some criterion, not necessarily pre-established, and assign a person to initiate the computation. This person will head the list. Let's call her Alice. Alice, like everyone else in the group, has access to everyone's permanent Locks.
2) Alice writes her Key in box 1 of PassLok, and the Lock belonging to the next person in her list (let's call him Bob) in box 2. She clicks Make Lock so the shared Key for the two of them is displayed in box 1. Then she clicks Make Lock again so its matching Lock appears in box 2. The takes that Lock and writes it into a text file (or starts an email), perhaps preceded by a label like "Alice+Bob".
3) Then she takes the Lock belonging to the next person on her list (Carol), writes it into box 2, and clicks Make Lock twice, as before. She appends that string to the text file or email, perhaps preceded by something like "+Carol".
4) Alice keeps doing that with every Lock on her list, perhaps stopping at the next-to-last one (which belongs to Yvonne) rather than the last. She writes it on her list, and finally sends the whole thing to everyone (perhaps excepting Bob). Now everyone can make a Key that will be shared by everyone else, while an outsider cannot.

Here's how those receiving this message, which does not need to be locked in any way, retrieve the group Key:

1) Let's say last on the list is Zenon. He takes the Lock listed below the "+Yvonne" label (the last one, unless Alice added a "+Zenon" entry so everyone can check that they

got the correct group Key), puts it in box 2 of PassLok, and writes his secret personal Key in box 1, then clicks Make Lock. Box 1 will display the shared group Key.

2) Yvonne takes the Lock prior to that (with the label "+Xavier") and does the same with it and her personal Key. Then he writes Zenon's Lock in box 2 and clicks Make Lock. Now she's got the group key, too.
3) Everyone repeats a similar process, writing their secret Key in box 1 and the Lock prior to their name in box 2, and then successively adding the Lock of every person below them on the list and clicking Make Lock every time, ending with Zenon's Lock. The final merged Key obtained will be the group Key.
4) Bob has the most work to do, for he has to start with Alice's Lock (likely not listed because he has it already), and then all the other Locks except his own, all the way to Zenon's. Like Alice, he doesn't need any email to retrieve the group Key, only the order in which the Locks were added. Alice is done before she sends that email, as soon as she adds Zenon's Lock to the mix.

The list of names does not need to be pre-made, so long and the originator identifies which Locks are added at which steps, and it can be random. Anyone can be Alice, the originator, perhaps it will be the person who has the first document to lock for the whole group to read. In fact, Alice can start with a random Key/Lock, so long as that random Lock heads the email she's sending to the group. Like the other two methods, however, compromising one Key can reveal the group Key to an attacker.

PassLok and forward secrecy

When two or more people are involved in a chat session, they remain next to their computers until the session is over. This allows the use of throwaway random Keys, since those can easily be kept in a PassLok window. Then the session is over, that page is closed and the random key is lost, since it was never saved anywhere. This achieves "perfect forward secrecy," which means that nobody, including those involved in the chat, can retrieve the plain text of the conversation after it's over. If a Key is compromised later on, this does not necessarily reveal the conversation to an attacker.

I'll first show how the chat session could be set up if forward secrecy is not required, and then how it would be done to provide forward secrecy.

Without forward secrecy:

1. Alice enters the chatroom. She doesn't find anyone listed whose Lock she might know.

2. Bob enters the chatroom and notices Alice's name, whose Lock he has. He makes the shared Key combining his personal Key and Alice's Lock, and posts a message locked with that shared Key, prefaced by @Alice.

3. Alice notices the message and sees that it is "Bob's". She makes the shared Key by combining her personal Key with Bob's Key, and unlocks the message. Then she

posts the Lock corresponding to their shared Key, so that others might join later. She and Bob start chatting using their shared Key.

4. Carol enters the chatroom and wants to join. She makes a new shared Key merging her personal Key with the shared Lock, and posts a message locked with that new Key, prefaced with @Bob, because she knows Bob.

5. Bob sees the message and makes the shared Key by combining the previous shared Key with Carol's Lock. Then he posts Carol's Lock, plus the Lock matching that new shared Key so that others might join later. Alice sees that and generates the new shared Key. Now it's the three of them sharing a Key.

6. More people come in, and they each merge the last common Lock with their personal Key to make the new shared Key, then post a message addressed to some participant they know, so they can vouch for them and post the message with their Lock and the Lock of the new shared Key, so everyone else knows the shared Key has been expanded.

This is quite a simple protocol, modeled after the third process for making a group Key. Unfortunately, if somebody's personal Key is compromised after the chat is over, an attacker would be able to produce all the successive chat Keys from the chatroom log, and the messages from the moment that person joined the chat would be plain to read.

With forward secrecy:

1. Alice enters the crowded chatroom but there isn't anyone whose Lock she has in her address book, so she can't be sure of their identities.

2. Bob enters the chatroom and finds Alice listed as a chatroom user. He wants to chat with her in complete privacy without leaving the main chatroom. He opens two PassLok instances, generates a random Key/Lock set in one instance. Then he copies Alice's permanent Lock, which he retrieves from his address book, into box 2 of the other instance, and writes his personal Key into box 1 and clicks Make Lock to merge them into box 1. He copies the dummy Lock from the other PassLok tab and pastes it into box 2, then locks it with the shared key made from his permanent Key and Alice's permanent Lock. He posts the locked result into the chatroom window, prefaced by an "@Alice" tag.

3. Alice sees Bob's posting and wonders if the Bob she knows is the one who wants to chat in private. She does the same as Bob did in step 2 above, except that she uses the shared Key resulting from her permanent Key and Bob's permanent Lock, to unlock the message posted by Bob. When this unlocks, she knows this is the Bob she knew. Alice then makes a random pair of Key and Lock in the other PassLok tab and posts that Lock in the chatroom, followed by the Lock she retrieved from Bob's

message. She then takes Bob's dummy Lock and merges it with the random key she has just generated. The result is a dummy Key she shares with Bob, so she can start locking and posting messages for him to read.

4. When Bob sees this dummy Lock posted in the chat window, he knows this is the Alice he wants to chat with, because she was able to unlock his first message. He takes her dummy Lock from the posting and merges it with his dummy Key. The result is the shared Key, which allows him to post and read messages from Alice.

5. The process by which more people join their private chat is very similar. Every newcomer into the chatroom who wants to join the encrypted chat will need to find someone who can vouch for him/her. Then he/she makes a random Key/Lock pair and posts that Lock, encrypted with the Key he/she shares with that participant (made by merging the newcomer's personal Key and the current participant's permanent Lock), prefaced with an @NameOfParticipant tag, to call his/her attention.

6. The current participant notices the posting, retrieves the newcomer's permanent Lock from his/her address book, and makes the shared Key, with which he/she unlocks the message. Satisfied that it is OK to admit the newcomer into the chat, he/she posts the Lock of the current chat Key (which is about to change), followed by the newcomer's random Lock extracted from the request posting.

7. The newcomer sees his/her dummy Lock posted, and realizes the request has been successfully unlocked and accepted. He/she then uses the other Lock, and his/her random Key (which wasn't posted) to produce the new chat Key. The other participants see the same posting and use the newcomer's Lock and their previous chat Key to make the new chat Key, which they start using from then on. One more participant has been added to the common chat, and the chat Key has changed.

8. There is no need to change Keys as people leave the chatroom. If someone leaves and then comes back after the Key has changed, he/she will need to be admitted to the chat all over again, starting with step 5. When everyone leaves and the last PassLok instance is closed, all the Keys are lost and nobody can retrieve any part of the private conversation from a log of the chat window. At most, if somebody's personal Key is compromised later on, an attacker can see that a dummy Lock was posted, but since no Keys were ever posted he/she still won't be able to read any of the messages.

If there are only two people chatting back and forth, there's also the possibility of changing the Key for every message. This way, if any of their personal Keys is

compromised, the only thing that an attacker can get from a log of their chat is the setup but none of the actual content. Here's how Alice and Bob would communicate:

1. Alice want to talk to Bob so she types her secret Key in box 1, fetches Bob's Lock and puts it in box 2, and then clicks Make Lock to combine them into their shared Key. Then she opens another PassLok window and clicks Make Lock to generate a random Key and its Lock. She cuts that Lock and pastes it in box 1 of the first PassLok window. Then she clicks Lock/Unlock to lock it and sends the result to Bob.

2. Bob sees a lock and signed message from Alice and knows he must get her Lock, put it in box 2, and then combine it with his personal Key before he can unlock the message. After unlocking, he sees a Lock different from Alice's permanent Lock, so he presumes it is a dummy Lock to be used in his next message to her. He opens a second instance of PassLok and generates a random Key and its Lock. He takes the Lock that Alice sent him and puts it in box 1 of the first PassLok window, and the dummy Lock he just produced in box 2. Then he maybe adds some text and clicks Lock/Unlock to lock that, and sends the result to Alice.

3. Alice uses the dummy Key in the second PassLok window to unlock the message that Bod just sent her. The she puts the Lock contained in that message in box 1. On the first PassLok page, she generates another dummy Key and its Lock, copies the Lock, and pastes it in box 2 of the second PassLok page, along with a message. The she locks it with the Lock that she just got from Bob, and sends him the result.

4. They keep going back and forth, every time generating a new random Key and its Lock, which the other person will use to lock the next message. When they are done communicating, they close both PassLok instances and all the dummy Keys are forgotten, along with the possibility of unlocking their messages.

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:

- 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.

- 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.
- 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:

- Find a base64 encoding program. There are programs that you can download for just about any operating system (the function is actually built into OSX and Linux). If you choose a web program, be aware that your file is going to be sent to a server, likely unencrypted, to it can be turned into base64 text.
- Load the file into the program and do a base64 encoding of it, which will result in a (long) piece of text. Copy it into the clipboard.
- Now paste it into box 2 of PassLok and lock it in regular or Key-locked mode. The resulting (even longer) locked text can be sent safely by the usual channels.
- When the recipient gets it, he/she will first unlock it, then copy the plain base64 code into a base64 decoder, and turn it again into a binary file. Since the encoding process eliminates any information concerning the name and kind of the original file, the recipient will need to supply a name that makes sense when the decoding program announces that it is ready to save the file. The actual name could have been sent in a separate message, or even the same, making sure it will not be mixed with the code representing the file contents.

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

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

This is the easiest process to sign a file:

- 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.
- Put that hash string into box 2 of PassLok and apply your signature, using your secret Key. After the signature is made, you can cut it out and give it to people along with the file, or keep the hash string and the signature together.
- Someone wishing to verify the sign will first have to take the SHA256 of the file (which should be the same you signed), then load it into box 2 of PassLok and append the signature if it wasn't there already. Then verify the signature on the hash using the sender's Lock.

Appendix: A Comparison between PassLok and PGP

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

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 (mid-2013), NIST recommends using 2048-bit factors for decent security. That's 342 base64 characters (base64 is used for displaying keys and encrypted text in PGP), or 617 decimal digits. Those are big numbers, so you can imagine the difficulty of multiplying them, let alone trying to factor them. Try doing that by hand.

The private key is to be kept jealously guarded, whereas the public key is to be publicized so people can retrieve it and send you encrypted messages. The private key is needed to read messages encrypted with the matching public key, so this process

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

Very clever indeed. This opens up the possibility of exchanging confidential information without having a pre-established secret password, making binding contracts, and a bunch of other neat things. Today, PGP and similar methods are at the root of secure communications between computers and the digital certificates that tell them that another computer or a given piece of software are legit. But Phil Zimmermann's original vision for secure communications between regular people has largely failed. We still email or text each other in a way that anyone in-between can read what we write. Why?

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

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

And here is the problem. Either the place where you store your private key is perfectly secure from tampering and eavesdropping, or you must add further security to protect it. PGP "solves" this problem by encrypting the private key with IDEA, using a separate key, before it writes it down to a file. Thankfully, IDEA admits arbitrary keys, so PGP asks you to come up with a "passphrase" (code for long, hard to guess password), which is used to encrypt the private key. When you need to use your private key, PGP retrieves the file containing it in encrypted form, asks you for the passphrase, and decrypts it

using IDEA. The private key is never displayed in decrypted form (at least for you to see it).

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

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

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

To be sure, PGP encourages certain standard practices that provide a modicum of reassurance. One of them is that people should sign their public keys (thereby making them twice their original length) before they post them. The idea is that people then can verify that, if you were able to sign your public key with your private key, you must have both keys so at least it's not someone who just ran into your public key somewhere who is posting that key. Another best practice is to add a signature to a text right before it is

encrypted. This way the recipient of the text has assurance, by checking the signature, of the sender's identity. Unfortunately, signing a message before encrypting it makes it longer and harder to process later on as well.

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

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

And by the way, PassLok doesn't have "public keys" and "private keys." It has "Locks" that people can put on a text so nobody can read it, and "Keys" that unlock a locked text for their possessors. Everybody has had that 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 keyring file, or fetching them every time from a keyserver. 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. It uses no keyservers. How are then people supposed to get someone's Lock with a minimum of confidence that it does indeed belong to that person? Here the problem may be the question itself. Ask yourself: how do get someone's cell phone number with a minimum of confidence that it does indeed belong to that person? I don't know what you do, but I look at that person's physical or electronic card first, or ask someone else who might have it. The phonebooks in my house have been gathering dust for years, as have PGP keyservers all over the world (many of which aren't current or active anymore, once you check). When I get a phone number, I use it right away, and let the actual use serve as verification that it is correct.

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

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

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

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

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

only need to ensure that your enemies don't know about it. PGP can't offer any such assurance for the paranoid.

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

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

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

Beginning with version 1.4, PassLok also has the ability to disguise its output as common text. 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.

So, 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.
- 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.
- 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 keyservers. They can be summarized into ID numbers for authentication through a separate channel or through any of the many audiovisual methods freely available today.

- PGP is built on the RSA public key algorithm, which would need keys longer than 15,000 bits for a security level comparable to that of PassLok, which is built on 521-bit elliptic curve math.
- PGP started using only the 128-bit IDEA algorithm for its main encryption method (it allows more secure methods now). PassLok uses the strongest, 256-bit version of AES (a.k.a. Rijndael), the winner of the 2001 NIST contest for best encryption algorithm. S/MIME uses triple-DES, which lost to AES in that contest. IDEA didn't even compete.
- PGP only admits one encrypted message at a time. So does S/MIME. In PassLok, a second message whose very existence is impossible to verify can be conveyed at the same time as the main message.
- PGP outputs easily identifiable encrypted strings. PassLok can produce output that looks like English, or any other language.

As a table:

	Pretty Good Privacy (PGP) and S/MIME	**PassLok**
Portable	No	Yes
OS-independent	No	Yes
Installation-free	No	Yes
Storage-free	No	Yes
Small keys	No	Yes
Encryption modes	1	4
Short message mode	No	Yes
Hidden messages	No	Yes
Fake text output	No	Yes
Secret Sharing built-in	No	Yes
Transparent authentication	No	Yes
Keyserver-free	No	Yes
Encryption bitlength	128 or 256	256 always
RSA-equivalent public key strength	1024-bit or 2048-bit	+15000-bit