PassLok: more than a pretty good privacy

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, now in version 1.6.

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 that before in hardware. People have used physical stamps on a piece of paper, while they have never signed anything with a key (unless it’s been dipped in ink beforehand, but it’s kind of messy). 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, even when including the URL of an authenticating video. 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 know (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). The video for my PassLok 1.6 Lock is actually attached to the Lock itself, so that anyone who knows me and gets my Lock from a different source can check right away, without any intermediaries, that my Lock is the genuine article.

The newest version of PassLok at the time of this writing (1.6) has the ability to display locks and encrypted/signed text as a QR code, which allows users to exchange this information while meeting in person, without using any online resource. It's still not quite the same as spelling out a phone number, but it's getting dangerously close.

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.

With version 1.6, PassLok adds a PFS mode, where PFS stands for "Perfect Forward Secrecy." This means that nobody, including the participants, can decipher a recorded conversation after it ends. This is because the encryption key changes with every message, and old keys are simply discarded. PFS operation today can only be

found in certain real-time messaging programs because it is so difficult to make it work unless both ends of the conversation are connected at the same time. But PassLok has it while PGP gave up on it long ago.

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 do it through some variation of ctrl-u), 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 hit "Show ID". 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 help pages (there is a button for that, which also leads to other mirrors and old versions of PassLok), 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, you can always save it somewhere 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 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 hidden channel.

The cherry on the cake is the selection of steganography functions built into PassLok, which are completely absent in PGP. Steganography is the art of hiding things, rather than encrypting. Since encrypted output can easily be identified as such, which might cause a problem to the user, in many locations around the world, PassLok adds the ability to conceal that output as reasonably normal text, or within otherwise normal-looking images. PassLok includes a default cover text, but it can use text in any language, including Arabic, Chinese, etc. Both hiding strategies work in smartphones as well as in computers.

So, to summarize:

· PGP forces you to keep secret files that might be lost, corrupted, or compromised. PassLok forces no secret files, though fully encrypted storage is available for convenience.

· PGP forces you to install things in a particular computer, and then use that computer or one similarly equipped, which makes it dangerous if that computer is compromised. PassLok is a perfectly portable web app; you can use any PC or

smartphone in the world, so you can be sure there's no foul play. Even the stored material can be transferred to another device by means of an encrypted email.

· 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. They can carry their own video-based authentication.

· 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 output looks clearly encrypted, but PassLok has the ability to make it look like normal text or images.

As a table:

	Pretty Good Privacy (PGP) and S/MIME	PassLok
Machine-independent	No	Yes
Installation-free	No	Yes
Storage-free	No	Yes

	Pretty Good Privacy (PGP) and S/MIME	PassLok
Small keys	No	Yes
Encryption modes	1	4
Short message mode	No	Yes
Perfect Forward Secrecy	No	Yes
Hidden messages	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
Display QR code	No	Yes
Normal-looking output	No	Yes
Hide within images	No	Yes