The PGP Problem

Jul 16, 2019

Cryptography engineers have been tearing their hair out over PGP's deficiencies for (literally) decades. When other kinds of engineers get wind of this, they're shocked. PGP is bad? Why do people keep telling me to use PGP? The answer is that they shouldn't be telling you that, because PGP is bad and needs to go away.

There are, as you're about to see, lots of problems with PGP. Fortunately, if you're not morbidly curious, there's a simple meta-problem with it: it was designed in the 1990s, before serious modern cryptography. No competent crypto engineer would design a system that looked like PGP today, nor tolerate most of its defects in any other design. Serious cryptographers have largely given up on PGP and don't spend much time publishing on it anymore (with a notable exception). Well-understood problems in PGP have gone unaddressed for over a decade because of this.

Two quick notes: first, we wrote this for engineers, not lawyers and activists. Second: "PGP" can mean a bunch of things, from the OpenPGP standard to its reference implementation in GnuPG. We use the term "PGP" to cover all of these things.

The Problems

Absurd Complexity

For reasons none of us here in the future understand, PGP has a packet-based structure. A PGP message (in a ".asc" file) is an archive of typed packets. There are at least 8 different ways of encoding the length of a packet, depending on whether you're using "new" or "old" format packets. The "new format" packets have variable-length lengths, like BER (try to write a PGP implementation and you may wish for the sweet release of ASN.1). Packets can have subpackets.

There are overlapping variants of some packets. The most recent keyserver attack happened because GnuPG accidentally went quadratic in parsing keys, which also follow this deranged format.

That's just the encoding. The actual system doesn't get simpler. There are keys and subkeys. Key IDs and key servers and key signatures. Sign-only and encrypt-only. Multiple "key rings". Revocation certificates. Three different compression formats. This is all before we get to smartcard support.

Swiss Army Knife Design

If you're stranded in the woods and, I don't know, need to repair your jean cuffs, it's handy if your utility knife has a pair of scissors. But nobody who does serious work uses their multitool scissors regularly.

A Swiss Army knife does a bunch of things, all of them poorly. PGP does a mediocre job of signing things, a relatively poor job of encrypting them with passwords, and a pretty bad job of encrypting them with public keys. PGP is not an especially good way to securely transfer a file. It's a clunky way to sign packages. It's not great at protecting backups. It's a downright dangerous way to converse in secure messages.

Back in the MC Hammer era from which PGP originates, "encryption" was its own special thing; there was one tool to send a file, or to back up a directory, and another tool to encrypt and sign a file. Modern cryptography doesn't work like this; it's purpose built. Secure messaging wants crypto that is different from secure backups or package signing.

Mired In Backwards Compatibility

PGP predates modern cryptography; there are Hanson albums that have aged better. If you're lucky, your local GnuPG defaults to 2048-bit RSA, the 64-bit-block CAST5 cipher in CFB, and the OpenPGP MDC checksum (about which more later). If you encrypt with a password rather than with a public key, the OpenPGP protocol specifies PGP's S2K password KDF. These are, to put it gently, not the primitives a cryptography engineer would select for a modern system.

We've learned a lot since Steve Urkel graced the airwaves during ABC's TGIF: that you should authenticate your ciphertexts (and avoid CFB mode) would be an obvious example, but also that 64-bit block ciphers are bad, that we can do much better than RSA, that mixing compression and encryption is dangerous, and that KDFs should be both time- and memory-hard.

Whatever the OpenPGP RFCs may say, you're probably not doing any of these things if you're using PGP, nor can you predict when you will. Take AEAD ciphers: the Rust-language Sequoia PGP defaulted to the AES-EAX AEAD mode, which is great, and nobody can read those messages because most PGP installs don't know what EAX mode is, which is not great. Every well-known bad cryptosystem eventually sprouts an RFC extension that supports curves or AEAD, so that its proponents can claim on message boards that they support modern cryptography. RFC's don't matter: only the installed base does. We've understood authenticated encryption for 2 decades, and PGP is old enough to buy me drinks; enough excuses.

You can have backwards compatibility with the 1990s or you can have sound cryptography; *you can't have both*.

Obnoxious UX

We can't say this any better than Ted Unangst:

> *There was a PGP usability study conducted a few years ago where a group of technical people were placed in a room with a computer and asked to set up PGP. Two hours later, they were never seen or heard from again.*

If you'd like empirical data of your own to back this up, here's an experiment you can run: find an immigration lawyer and talk them through the process of getting Signal working on their phone. You probably don't suddenly smell burning toast. Now try doing that with PGP.

Long-Term Secrets

PGP begs users to keep a practically-forever root key tied to their identity. It does this by making keys annoying to generate and exchange, by encouraging

"key signing parties", and by creating a "web of trust" where keys depend on other keys.

Long term keys are almost never what you want. If you keep using a key, it eventually gets exposed. You want the blast radius of a compromise to be as small as possible, and, just as importantly, you don't want users to hesitate even for a moment at the thought of rolling a new key if there's any concern at all about the safety of their current key.

The PGP cheering section will immediately reply "that's why you keep keys on a Yubikey". To a decent first approximation, nobody in the whole world uses the expensive Yubikeys that do this, and you can't imagine a future in which that changes (we can barely get U2F rolled out, and those keys are disposable). We can't accept bad cryptosystems just to make Unix nerds feel better about their toys.

Broken Authentication

More on PGP's archaic primitives: way back in 2000, the OpenPGP working group realized they needed to authenticate ciphertext, and that PGP's signatures weren't accomplishing that. So OpenPGP invented the MDC system: PGP messages with MDCs attach a SHA-1 of the plaintext to the plaintext, which is then encrypted (as normal) in CFB mode.

If you're wondering how PGP gets away with this when modern systems use relatively complex AEAD modes (why can't everyone just tack a SHA-1 to their plaintext), you're not alone. Where to start with this Rube Goldberg contraption? The PGP MDC can be stripped off messages –– it was encoded in such a way that you can simply chop off the last 22 bytes of the ciphertext to do that. To retain backwards compatibility with insecure older messages, PGP introduced a new packet type to signal that the MDC needs to be validated; if you use the wrong type, the MDC doesn't get checked. Even if you do, the new SEIP packet format is close enough to the insecure SE format that you can potentially trick readers into downgrading; Trevor Perrin worked the SEIP out to 16 whole bits of security.

And, finally, even if everything goes right, the reference PGP implementation will (wait for it) release unauthenticated plaintext to callers, *even if the MDC doesn't match*.

Incoherent Identity

PGP is an application. It's a set of integrations with other applications. It's a file format. It's also a social network, and a subculture.

PGP pushes notion of a cryptographic identity. You generate a key, save it in your keyring, print its fingerprint on your business card, and publish it to a keyserver. You sign other people's keys. They in turn may or may not rely on your signatures to verify other keys. Some people go out of their way to meet other PGP users in person to exchange keys and more securely attach themselves to this "web of trust". Other people organize "key signing parties". The image you're conjuring in your head of that accurately explains how hard it is to PGP's devotees to switch to newer stuff.

None of this identity goop works. Not the key signing web of trust, not the keyservers, not the parties. Ordinary people will trust anything that looks like a PGP key no matter where it came from – how could they not, when even an expert would have a hard time articulating how to evaluate a key? Experts don't trust keys they haven't exchanged personally. Everyone else relies on centralized authorities to distribute keys. PGP's key distribution mechanisms are theater.

Leaks Metadata

Forget the email debacle for a second (we'll get to that later). PGP by itself leaks metadata. Messages are (in normal usage) linked directly to key identifiers, which are, throughout PGP's cobweb of trust, linked to user identity. Further, a rather large fraction of PGP users make use of keyservers, which can themselves leak to the network the identities of which PGP users are communicating with each other.

No Forward Secrecy

A good example of that last problem: secure messaging crypto demands forward secrecy. Forward secrecy means that if you lose your key to an attacker today, they still can't go back and read yesterday's messages; they had to be there with the key yesterday to read them. In modern cryptography engineering, we assume our adversary is recording everything, into infinite storage. PGP's claimed adversaries include world governments, many of whom are certainly doing exactly that. Against serious adversaries and without forward secrecy, breaches are a question of "when", not "if".

To get forward secrecy in practice, you typically keep two secret keys: a short term session key and a longer-term trusted key. The session key is ephemeral (usually the product of a DH exchange) and the trusted key signs it, so that a man-in-the-middle can't swap their own key in. It's theoretically possible to achieve a facsimile of forward secrecy using the tools PGP provides. Of course, pretty much nobody does this.

Clumsy Keys

An OpenBSD signify(1) public key is a Base64 string short enough to fit in the middle of a sentence in an email; the private key, which isn't an interchange format, is just a line or so longer. A PGP public key is a whole giant Base64 document; if you've used them often, you're probably already in the habit of attaching them rather than pasting them into messages so they don't get corrupted. Signify's key is a state-of-the-art Ed25519 key; PGP's is a weaker RSA key.

You might think this stuff doesn't matter, but it matters a lot; orders of magnitude more people use SSH and manage SSH keys than use PGP. SSH keys are trivial to handle; PGP's are not.

Negotiation

PGP supports ElGamal. PGP supports RSA. PGP supports the NIST P-Curves. PGP supports Brainpool. PGP supports Curve25519. PGP supports SHA-1. PGP supports SHA-2. PGP supports RIPEMD160. PGP supports IDEA. PGP supports 3DES. PGP supports CAST5. PGP supports AES. There is no way this is a complete list of what PGP supports.

If we've learned 3 important things about cryptography design in the last 20 years, at least 2 of them are that negotiation and compatibility are evil. The flaws in cryptosystems tend to appear in the joinery, not the lumber, and expansive crypto compatibility increases the amount of joinery. Modern protocols like TLS 1.3 are jettisoning backwards compatibility with things like RSA, not adding it. New systems support *just a single suite of primitives*, and a simple version number. If one of those primitives fails, you bump the version and chuck the old protocol all at once.

If we're unlucky, and people are still using PGP 20 years from now, PGP will be the only reason any code anywhere includes CAST5. We can't say this more clearly or often enough: you can have backwards compatibility with the 1990s or you can have sound cryptography; you can't have both.

Janky Code

The de facto standard implementation of PGP is GnuPG. GnuPG is not carefully built. It's a sprawling C-language codebase with duplicative functionality (write-ups of the most recent SKS key parsing denial of service noted that it has multiple key parsers, for instance) with a long track record of CVEs ranging from memory corruption to cryptographic side channels. It has at times been possible to strip authenticators off messages without GnuPG noticing. It's been possible to feed it keys that don't fingerprint properly without it noticing. The 2018 Efail vulnerability was a result of it releasing unauthenticated plaintext to callers. GnuPG is not good.

GnuPG is also effectively the reference implementation for PGP, and also the basis for most other tools that integrate PGP cryptography. It isn't going anywhere. To rely on PGP is to rely on GPG.

The Answers

One of the rhetorical challenges of persuading people to stop using PGP is that there's no one thing you can replace it with, *nor should there be*. What you should use instead depends on what you're doing.

Talking To People

Use Signal. Or Wire, or WhatsApp, or some other Signal-protocol-based secure messenger.

Modern secure messengers are purpose-built around messaging. They use privacy-preserving authentication handshakes, repudiable messages, *cryptographic ratchets* that rekey on every message exchange, and, of course, modern encryption primitives. Messengers are trivially easy to use and there's no fussing over keys and subkeys. If you use Signal, you get even more than that: you get a system so paranoid about keeping private metadata off servers that it tunnels Giphy searches to avoid traffic analysis attacks, and until relatively recently didn't even support user profiles.

Encrypting Email

Don't.

Email is insecure. Even with PGP, it's default-plaintext, which means that even if you do everything right, some totally reasonable person you mail, doing totally reasonable things, will invariably CC the quoted plaintext of your encrypted message to someone else (we don't know a PGP email user who hasn't seen this happen). PGP email is forward-insecure. Email metadata, including the subject (which is literally message content), are always plaintext.

If you needed another reason, read the Efail paper. The GnuPG community, which mishandled the Efail disclosure, talks this research down a lot, but it was accepted at Usenix Security (one of the top academic software security venues) and at Black Hat USA (*the* top industry software security venue), was one of the best cryptographic attacks of the last 5 years, and is a pretty devastating indictment of the PGP ecosystem. As you'll see from the paper, S/MIME isn't better.

This isn't going to get fixed. To make actually-secure email, you'd have to tunnel another protocol over email (you'd still be conceding traffic analysis attacks). At that point, why bother pretending?

Encrypting email is asking for a calamity. Recommending email encryption to at-risk users is malpractice. Anyone who tells you it's secure to communicate

over PGP-encrypted email is putting their weird preferences ahead of your safety.

Sending Files

Use Magic Wormhole. Wormhole clients use a one-time password-authenticated key exchange (PAKE) to encrypt files to recipients. It's easy (for nerds, at least), secure, and fun: we haven't introduced wormhole to anyone who didn't start gleefully wormholing things immediately just like we did.

Someone stick a Windows installer on a Go or Rust implementation of Magic Wormhole right away; it's too great for everyone not to have.

If you're working with lawyers and not with technologists, Signal does a perfectly cromulent job of securing file transfers. Put a Signal number on your security page to receive bug bounty reports, not a PGP key.

Encrypting Backups

Use Tarsnap. Colin can tell you all about how Tarsnap is optimized to protect backups. Or really, use any other encrypted backup tool that lots of other people use; they won't be as good as Tarsnap but they'll all do a better job than PGP will.

Need offline backups? Use encrypted disk images; they're built into modern Windows, Linux, and macOS. Full disk encryption isn't great, but it works fine for this use case, and it's easier and safer than PGP.

Signing Packages

Use Signify/Minisign. Ted Unangst will tell you all about it. It's what OpenBSD uses to sign packages. It's extremely simple and uses modern signing. Minisign, from Frank Denis, the libsodium guy, brings the same design to Windows and macOS; it has bindings for Go, Rust, Python, Javascript, and .NET; it's even compatible with Signify.

Encrypting Application Data

Use libsodium It builds everywhere, has interface that's designed to be hard to misuse, and you won't have to shell out to a binary to use it.

Encrypting Files

This really is a problem. If you're/not/making a backup, and you're /not/archiving something offline for long-term storage, and you're /not/encrypting in order to securely send the file to someone else, and you're /not/encrypting virtual drives that you mount/unmount as needed to get work done, then there's no one good tool that does this now. Filippo Valsorda is working on "age" for these use cases, and I'm super optimistic about it, but it's not there yet.

Update, February 2020

Filippo's age has been released. It's a solid design with simple, easily auditable implementations in Go and Rust. You can build binaries for it for every mainstream platform. Age is, of course, much younger than PGP. But I would bet all the money in my pocket against all the money in yours that a new vulnerability will be found in the clangorous contraption of PGP before one is found in age. Look into age!

Hopefully it's clear that this is a pretty narrow use case. We work in software security and handle sensitive data, including bug bounty reports (another super common "we need PGP!" use case), and we almost never have to touch PGP.