efail: Outdated Crypto Standards are to blame

Hanno's blog

Tuesday, May 22. 2018

efail: Outdated Crypto Standards are to blame

I have a lot of thoughts about the recently published efail vulnerability, so I thought I'd start to writeup some of them. I'd like to skip all the public outrage about the disclosure process for now, as I mainly wanted to get into the technical issues, explain what I think went wrong and how things can become more secure in the future. I read lots of wrong statements that "it's only the mail clients" and the underlying crypto standards are fine, so I'll start by explaining why I believe the OpenPGP and S/MIME standards are broken and why we still see these kinds of bugs in 2018. I plan to do a second writeup that will be titled "efail: HTML mails are to blame".

I assume most will have heard of efail by now, but the quick version is this: By combining a weakness in cryptographic modes along with HTML emails a team of researchers was able to figure out a variety of ways in which mail clients can be tricked into exfiltrating the content of encrypted e-mails. Not all of the attack scenarios involve crypto, but those that do exploit a property of encryption modes that is called malleability. It means that under certain circumstances you can do controlled changes of the content of an encrypted message.

Malleability of encryption is not a new thing. Already back in the nineties people figured out this may be a problem and started to add authentication to encryption. Thus you're not only guaranteeing that encrypted data cannot be decrypted by an attacker, you also guarantee that an attacker cannot change the data without the key. In early protocols people implemented authentication in an ad-hoc way leading to different approaches with varying degrees of security (often refered to as MAC-then-Encrypt, Encrypt-then-MAC, Encrypt-and-MAC). The OpenPGP standard also added a means of authentication called MDC (Modification Detection Code), the S/MIME standard never received anything alike.

Authenticated Encryption

In the year 2000 the concept of authenticated encryption got introduced by Bellare and Namprempre. It can be summarized as the idea that instead of putting authentication on top of encryption let's define some construction where a combination of both is standardized in a safe way. It also changed the definition of a cipher, which will later become relevant, as this early paper already gives good guidance on how to design a proper API for authenticated encryption. While an unauthenticated cipher has a decryption function that takes an input and produces an output, an authenticated cipher's decryption function either produce an output or an error (but not both):

In such a scheme the encryption process applied by the sender takes the key and a plaintext to return a ciphertext, while the decryption process applied by the receiver takes the same key and a ciphertext to return either a plaintext or a special symbol indicating that it considers the ciphertext invalid or not authentic. (Bellare, Namprempre, Asiacrypt 2000 Proceedings)

The concept was later extended with the idea of having Authenticated Encryption with Additional Data (AEAD), meaning you could have pieces that are not encrypted, but still authenticated. This is useful in some situations, for example if you split up a message in multiple parts the ordering could be authenticated. Today we have a number of standardized AEAD modes.

Just always use Authenticated Encryption

Authenticated Encryption is a concept that makes a lot of sense. One of the most basic pieces of advice in designing crypto systems should be: "Unless you have a very good reason not to do so, just always use a standardized, off-the-shelf authenticated encryption mode."

Quicksearch

>

About me

Informationen über meine Arbeit als freier Journalist finden Sie hier.

Hanno Böck
mail: **hanno@hboeck.de**
jabber: **hanno@hboeck.de**
GPG/OpenPGP key

Hanno on Twitter

Impressum

schokokeks.org

Creative Commons

Show tagged entries

asia asia2013 berlin ccc china copyright cryptography datenschutz demonstration freesoftware gentoo hardware karlsruhe klimaschutz linux lug openstreetmap privacy russia security ssl tls travel trip2011 umwelt vorratsdatenspeicherung websecurity xss ökologie überwachung

Frontpage
Datenschutz / Privacy

There's a whole myriad of attacks that would've been prevented if people had used AEAD modes. Padding Oracle attacks in SSL/TLS like the Vaudenay attack and variations like the Lucky Thirteen attack? Use an AEAD and be safe. Partial plaintext discovery in SSH, as discovered in 2009 - and again in 2016, because the fixes didn't work? Use an AEAD and be safe. Broken XML encryption due to character encoding errors? Had you only used an AEAD and this would've been prevented. Heard of the iMessage flaw discovered in 2016? Lack of AEAD it is. Owncloud encryption module broken? If they had used an AEAD. (I'm including this one because it's my own minor contribution to the topic.)

Given this long list of attacks you would expect that one of the most basic pieces of advice everyone gets would be: "Just always use an AEAD if you can." This should be crypto 101, yet somehow it isn't.

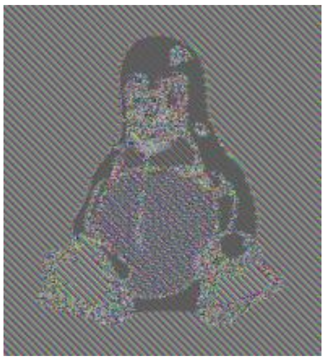
Teaching the best crypto of the 90s

Some time ago on a cryptography mailinglist I was subscribed to, someone posted a link to the material of a crypto introduction lecture from a university, saying this would be a good introduction to the topic. I took a brief look and answered that I don't think it's particularly good, citing multiple issues, one of them being the cipher modes that were covered in that lecture were ECB, CBC, OFC, CFB and CTR. None of these modes is authenticated. None of them should be used in any modern cryptosystem.

Some weeks later I was at a conference and it turned out the person across the table was a cryptography professor. We got into a discussion about teaching cryptography because I made some provocative statements (something along the lines of "Universities teach outdated crypto and then we end up with broken cryptosystems because of it"). So I asked him: "Which cipher modes do you teach in your crypto lecture?"

The answer: ECB, CBC, OFC, CFB and CTR.

After that I googled for crypto introduction lectures - and to my astonishment this was surprisingly common. This list of five cipher modes for some reason seems to be the popular choice for crypto introductions.



It doesn't seem to make a lot of sense. Let's quickly go through them: ECB is the most naive way of doing encryption with symmetric block ciphers where you encrypt every block of the input on its own with the same key. I'm inclined to say that it's not really a crypto mode, it's more an example of what not to do. If you ever saw the famous "ECB Tux" - that's the problem.

CBC (Cipher Block Chaining) is a widely used mode, particularly it's been the most popular mode in TLS for a long time, and it makes sense to teach it in order to understand attacks, but it's not something you should use. CFB mode is not widely used, I believe the only widespread use is actually in OpenPGP. OFB is even more obscure, I'm not aware of any mainsteam protocol in use that uses it. CTR (Counter Mode) is insofar relevant as one of the most popular AEAD modes is an extension of Counter Mode - it's called Galois/Counter Mode (GCM).

I think it's fair to say that teaching this list of ciphers in a crypto introduction lecture is odd. Some of them are obscure, some outright dangerous, and most important of all: None of them should be used, because none of them are authenticated. So why are these five ciphers so popular? Is there some secret list that everyone uses if they choose which ciphers to cover?

Actually... yes, there is such a list. These are exactly the five cipher modes that are covered in Bruce Schneier's book "Applied Cryptography" - published in 1996.

Now don't get me wrong: Applied Cryptography is undoubtedly an important part of cryptographic history. When it was published it was probably one of the best introductory resources into cryptography that you could get. It covers the best crypto available in 1996. But we have learned a few things since then, and one of them is that you better use an authenticated encryption mode.

There's more: At this year's Real World Crypto conference a paper was presented where the usability of cryptographic APIs was tested. The paper was originally published at the IEEE Symposium on Security and Privacy. I took a brief look into the paper and this sentence caught my attention:

"We scored the ECB as an insecure mode of operation and scored Cipher Block Chaining (CBC), Counter Mode (CTR) and Cipher Feedback (CFB) as secure."

These words were written in a peer reviewed paper in 2017. No wonder we're still fighting padding oracles and ciphertext malleability in 2018.

Choosing an authenticated mode

If we agree that authenticated encryption modes make sense the next question is which one to choose. This would easily provide material for a separate post, but I'll try to make it brief.

The most common mode is GCM, usually in combination with the AES cipher. There are a few issues with GCM. Implementing it correctly is not easy and implementation flaws happen. Messing up the nonce generation can have catastrophic consequences. You can easily collect a bunch of quotes from famous cryptographers saying bad things about GCM.

Yet despite all criticism using GCM is still not a bad choice. If you use a well-tested standard implementation and don't mess up the nonce generation you're good. Take this from someone who was involved discovering what I believe is the only practical attack ever published against GCM in TLS.

Other popular modes are Poly1305 (usually combined with the Chacha20 cipher, but it also works with AES) and OCB. OCB has some nice properties, but it's patented. While the patent holders allow some uses, this still has caused enough uncertainty to prevent widespread deployment.

If you can sacrifice performance and are worried about nonce generation issues you may have a look at AES in SIV mode. Also there's currently a competition running to choose future AEADs.

Having said all that: Choosing any standardized AEAD mode is better than not using an AEAD at all.

Both e-mail encryption standards - OpenPGP and S/MIME - are really old. They originate in the 90s and have only received minor updates over time.

S/MIME is broken and probably can't be rescued

S/MIME by default uses the CBC encryption mode without any authentication. CBC is malleable in a way that an attacker can manipulate encrypted content with bit flips, but this destroys the subsequent block. If an attacker knows the content of a single block then he can basically construct arbitrary ciphertexts with every second block being garbage.

Coupled with the fact that it's easy to predict parts of the S/MIME ciphertext this basically means game over for S/MIME. An attacker can construct an arbitrary mail (filled with some garbage blocks, but at least in HTML they can easily be hidden) and put the original mail content at any place he likes. This is the core idea of the efail attack and for S/MIME it works straight away.

There's an RFC to specify authenticated encryption modes in Cryptographic Message Syntax, the format underlying S/MIME, however it's not referenced in the latest S/MIME standard, so it's unclear how to use it.

HTML mails are only the most obvious problem for S/MIME. It would also be possible to construct malicious PDFs or other document formats with exfiltration channels. Even without that you don't want ciphertext malleability in any case. The fact that S/MIME completely lacks authentication means it's unsafe by design.

Given that one of the worst things about e-mail encryption was always that there were two competing, incompatible standards this may actually be an opportunity. Ironically if you've been using S/MIME and you want something alike your best bet may actually be to switch to OpenPGP.

OpenPGP - CFB mode and MDC

With OpenPGP the situation regarding authenticated encryption is a bit more complicated. OpenPGP introduced a form of authentication called Message Detection Code (MDC). The MDC works by calculating the SHA-1 hash of the plaintext message and then encrypting that hash and appending it to the encrypted message.

The first question is whether this is a sound cryptographic construction. As I said above it's usually recommended to use a standardized AEAD mode. It is clear that CFB/MDC is no such thing, but that doesn't automatically make it insecure. While I wouldn't recommend to use MDC in any new protocol and I think it would be good to replace it with a proper AEAD mode, it doesn't seem to have any obvious weaknesses. Some people may point out the use of SHA-1, which is considered a broken hash function due to the possibility of constructing collisions. However it doesn't look like this could be exploited in the case of MDC in any way.

So cryptographically while MDC doesn't look like a nice construction it doesn't seem to be an immediate concern security wise. However there are two major problems how MDC is specified in the OpenPGP standards and I think it's fair to say OpenPGP is thus also broken.

The first issue is how implementations should handle the case when the MDC tag is invalid or missing. This is what the specification has to say:

Any failure of the MDC indicates that the message has been modified and MUST be treated as a security problem. Failures include a difference in the hash values, but also the absence of an MDC packet, or an MDC packet in any position other than the end of the plaintext. Any failure SHOULD be reported to the user.

This is anything but clear. It must be treated as a security problem, but it's not clear what that means. A failure should be reported to the user. Reading this it is very reasonable to think that a mail client that would display a mail with a missing or bad MDC tag to a user with a warning attached would be totally in line with the specification. However that's exactly the scenario that's vulnerable to efail.

To prevent malleability attacks a client must prevent decrypted content from being revealed if the authentication is broken. This also goes back to the definition of authenticated encryption I quoted above. The decryption function should either output a correct plaintext or an error.

Yet this is not what the standard says and it's also not what GnuPG does. If you decrypt a message with a broken MDC you'll still get the plaintext and an error only afterwards.

There's a second problem: For backwards compatibility reasons the MDC is optional. The OpenPGP standard has two packet types for encrypted data, Symmetrically Encrypted (SE) packets without and Symmetrically Encrypted Integrity Protected (SEIP) packets with an MDC. Appart from the MDC they're mostly identical, which means it's possible to convert a packet with protection into one without protection, an attack that was discovered in 2015.

This could've been avoided, for example by using different key derivation functions for different packet types. But that hasn't happened. This means that any implementation that still supports the old SE packet type is vulnerable to ciphertext malleability.

The good news for OpenPGP is that with a few modifications it can be made safe. If an implementation discards packets with a broken or missing MDC and chooses not to support the unauthenticated SE packets then there are no immediate cryptographic vulnerabilities. (There are still issues with HTML mails and multipart messages, but this is independent of the cryptographic standard.)

Streaming and Chunking

As mentioned above when decrypting a file with GnuPG that has a missing or broken MDC then it will first output the ciphertext and then an error. This is in violation of the definition of authenticated encryption and it is also the likely reason why so many mail clients were vulnerable to efail. It's an API that invites misuse. However there's a reason why GnuPG behaves like this: Streaming of large pieces of data.

If you would want to design GnuPG in a way that it never outputs unauthenticated plaintext you'd have to buffer all decrypted text until you can check the MDC. This gets infeasible if you encrypt large pieces of data, for example backup tarballs. Replacing the CFB/MDC combination with an AEAD mode would also not automatically solve this problem. With a mode like GCM you could still decrypt data as you go and only check the authentication at the end.

In order to support both streaming and proper authenticated encryption one possibility would be to cut the data into chunks with a maximum size. This is more or less what TLS does.

A construction could look like this: Input data is processed in chunks of - let's say - 8 kilobytes size. The exact size is a tradeoff between overhead and streaming speed, but something in the range of a few kilobytes would definitely work. Each chunk would contain a number that is part of the authenticated additional data in order to prevent reordering attacks. The final chunk would furthermore contain a special indicator in the additional data, so truncation can be detected. A decryption tool would then decrypt each chunk and only output authenticated content. (I didn't come up with this on my own, as said it's close to what TLS does and Adam Langley explains it well in a talk you can find here. He even mentions the particular problems with GnuPG that led to efail.)

It's worth noting that this could still be implemented in a wrong way. An implementation could process parts of a chunk and output them before the authentication. Shortly after I first heard about efail I wondered if something like this could happen in TLS. For example a browser could already start rendering content when it receives half a TLS record.

An upcoming new OpenPGP standard

There's already a draft for a future version of the OpenPGP standard. It introduces two authenticated encryption modes - OCB and EAX - which is a compromise between some people wanting to have OCB and others worried about the patent issue. I fail to see how having two modes helps here, because ultimately you can only practically use a mode if it's widely supported.

The draft also supports chunking of messages. However right now it doesn't define an upper limit for the chunk size and you could have gigabytes of data in a single chunk. Supporting that would likely again lead to an unsafe streaming API. But it's a minor change to introduce a chunk limit and require that an API may never expose unauthenticated plaintext.

Unfortunately the work on the draft has mostly stalled. While the latest draft is from January the OpenPGP working group was shut down last year due to lack of interest.

Conclusion

Properly using authenticated encryption modes can prevent a lot of problems. It's been a known issue in OpenPGP, but until now it wasn't pressing enough to fix it. The good news is that with minor modifications OpenPGP can still be used safely. And having a future OpenPGP standard with proper authenticated encryption is definitely possible. For S/MIME the situation is much more dire and it's probably best to just give up on it. It was never a good idea in the first place to have competing standards for e-mail encryption.

For other crypto protocols there's a lesson to be learned as well: Stop using unauthenticated encryption modes. If anything efail should make that abundantly clear.

Posted by Hanno Böck at 23:08 | Comment (1) | Trackbacks (0)

Trackbacks

Trackback specific URI for this entry

No Trackbacks

Comments

Display comments as (Linear | Threaded)

Partial plaintext discovery in SSH, as discovered in 2009 - and again in 2016
____^^^^^^^^^^^^

Hi, I've discovered a broken link in your article. "again in 2016 " was linked to "https://blog.hboeck.de/archives/by partly the same people" instead of the actual research paper. I guess you accidentally mixed up your parentheses with links in Markdown. Please make sure "by partly the same people" is plaintext, and link to the actual research paper is linked at "again in 2016".

Thanks for your great, in-depth article!

#1 Anonymous on 2018-05-24 12:48 (Reply)

Add Comment

Name	
Email	
Homepage	
In reply to	[Top level]
Comment	

☐ **Remember Information?**
☐ **Subscribe to this entry**

Submit Comment Preview