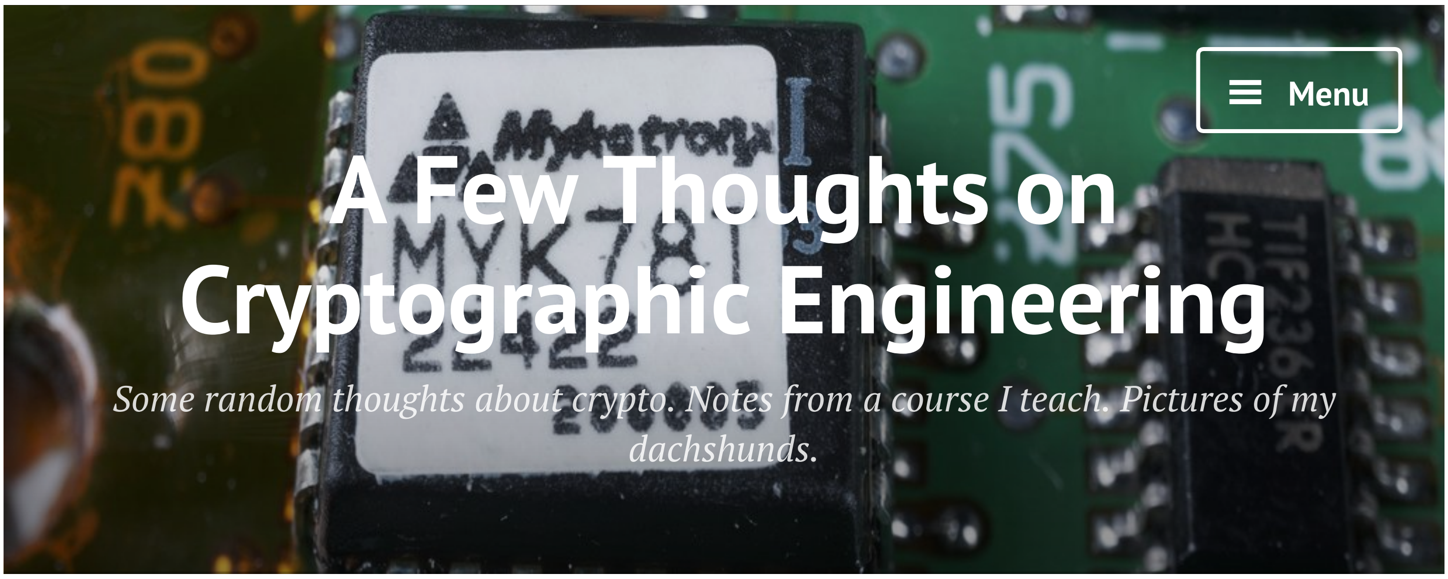


A Few Thoughts on Cryptographic Engineering

Some random thoughts about crypto. Notes from a course I teach. Pictures of my dachshunds.

Category: imessage

Attack of the Week: Apple iMessage

Today's Washington Post has a story entitled "Johns Hopkins researchers poke a hole in Apple's encryption", which describes the results of some research my students and I have been working on over the past few months.

As you might have guessed from the headline, the work concerns Apple, and specifically Apple's iMessage text messaging protocol. Over the past months my students Christina Garman, Ian Miers, Gabe Kaptchuk and Mike Rushanan and I have been looking closely at the encryption used by iMessage, in order to determine how the system fares against sophisticated attackers. The results of this analysis include some very neat new attacks that allow us to — under very specific circumstances — decrypt the contents of iMessage attachments, such as photos and videos.

The research team. From left: Gabe Kaptchuk, Mike Rushanan, Ian Miers, Christina Garman

Now before I go further, it's worth noting that the security of a text messaging protocol may *not* seem like the most important problem in computer security. And under normal circumstances I might agree with you. But today the circumstances are anything but normal: encryption systems like iMessage are at the center of a critical national debate over the role of technology companies in assisting law enforcement.

A particularly unfortunate aspect of this controversy has been the repeated call for U.S. technology companies to add "backdoors" to end-to-end encryption systems such as iMessage. I've always felt that one of the most compelling arguments against this approach — an argument I've made along with other colleagues — is that we just don't know how to construct such backdoors securely. But lately I've come to believe that this position doesn't go far enough — in the sense that it is woefully optimistic. The fact of the matter is that forget backdoors: *we barely know how to make encryption work at all*. If anything, this work makes me much gloomier about the subject.

But enough with the generalities. The TL;DR of our work is this:

Apple iMessage, as implemented in versions of iOS prior to 9.3 and Mac OS X prior to 10.11.4, contains serious flaws in the encryption mechanism that could allow an attacker — who obtains iMessage ciphertexts — to decrypt the payload of certain attachment messages via a slow but remote and silent attack, provided that one sender or recipient device is online. While capturing encrypted messages is difficult in practice on recent iOS devices, thanks to certificate pinning, it could still be conducted by a nation state attacker or a hacker with access to Apple's servers. You should probably patch now.

For those who want the gory details, I'll proceed with the rest of this post using the “fun” question and answer format I save for this sort of post.

What is Apple iMessage and why should I care?

Those of you who read this blog will know that I have a particular obsession with Apple iMessage. This isn’t because I’m weirdly obsessed with Apple — although it is a *little* bit because of that. Mostly it’s because I think iMessage is an important protocol. The text messaging service, which was introduced in 2011, has the distinction of being the first *widely-used* end-to-end encrypted text messaging system in the world.

To understand the significance of this, it’s worth giving some background. Before iMessage, the vast majority of text messages were sent via SMS or MMS, meaning that they were handled by your cellular provider. Although these messages are technically encrypted, this encryption exists only on the link between your phone and the nearest cellular tower. Once an SMS reaches the tower, it’s decrypted, then stored and delivered without further protection. This means that your most personal messages are vulnerable to theft by telecom employees or sophisticated hackers. Worse, many U.S. carriers still use laughably weak encryption and protocols that are vulnerable to active interception.

So from a security point of view, iMessage was a pretty big deal. In a single stroke, Apple deployed encrypted messaging to millions of users, ensuring (in principle) that even Apple itself couldn’t decrypt their communications. The even greater accomplishment was that *most people didn’t even notice this happened* — the encryption was handled so transparently that few users are aware of it. And Apple did this at very large scale: today, iMessage handles peak throughput of more than 200,000 encrypted messages per second, with a supported base of nearly one billion devices.

So iMessage is *important*. But is it any good?

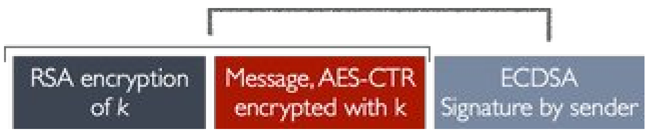
Answering this question has been kind of a hobby of mine for the past couple of years. In the past I've written about Apple's failure to publish the iMessage protocol, and on iMessage's dependence on a vulnerable centralized key server. Indeed, the use of a centralized key server is *still* one of iMessage's biggest weaknesses, since an attacker who controls the keyserver can use it to inject keys and conduct man in the middle attacks on iMessage users.

But while key servers are a risk, attacks on a key server seem fundamentally challenging to implement — since they require the ability to actively manipulate Apple infrastructure without getting caught. Moreover, such attacks are only useful for *prospective surveillance*. If you fail to substitute a user's key before they have an interesting conversation, you can't recover their communications after the fact.A more interesting question is whether iMessage's encryption is secure enough to stand up against *retrospective* decryption attacks — that is, attempts to decrypt messages *after* they have been sent. Conducting such attacks is much more interesting than the naive attacks on iMessage's key server, since any such attack would require the existence of a fundamental vulnerability in iMessage's encryption itself. And in 2016 encryption seems like one of those things that we've basically figured out how to get right.

Which means, of course, that we probably haven't.

How does iMessage encryption work?

What we know about the iMessage encryption protocol comes from a previous reverse-engineering effort by a group from Quarkslab, as well as from Apple's iOS Security Guide. Based on these sources, we arrive at the following (simplified) picture of the basic iMessage encryption scheme:

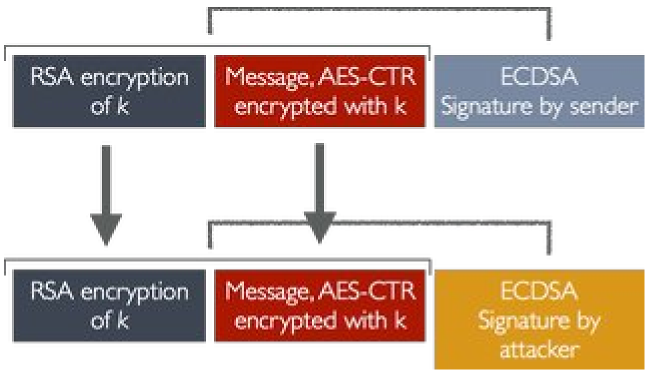


To encrypt an iMessage, your phone first obtains the RSA public key of the person you're sending to. It then generates a random AES key *k* and encrypts the message with that key using CTR mode. Then it encrypts *k* using the recipient's RSA key. Finally, it signs the whole mess using the sender's ECDSA signing key. This prevents tampering along the way.

So what's missing here?

Well, the most obviously missing element is that iMessage does not use a Message Authentication Code (MAC) or authenticated encryption scheme to prevent tampering with the message. To simulate this functionality, iMessage simply uses an ECDSA signature formulated by the sender. Naively, this would appear to be good enough. Critically, it's not.

The attack works as follows. Imagine that a clever attacker intercepts the message above *and* is able to register her own iMessage account. First, the attacker strips off the original ECDSA signature made by the legitimate sender, and replaces it with a signature of her own. Next, she sends the newly signed message to the original recipient *using her own account*:


RSA encryption of k
Message, AES-CTR encrypted with k
ECDSA Signature by sender
RSA encryption of k
Message, AES-CTR encrypted with k
ECDSA Signature by attacker

The outcome is that the user receives and decrypts a copy of the message, which has now apparently originated *from the attacker* rather than from the original sender. Ordinarily this would be a pretty mild attack — but there's a useful wrinkle. In replacing the sender's signature with one of her own, the attacker has gained a powerful capability. *Now she can tamper with the AES ciphertext (red) at will.*

Specifically, since in iMessage the AES ciphertext is not protected by a MAC, it is therefore *malleable*. As long as the attacker signs the resulting message with her key, she can flip any bits in the AES ciphertext she wants — and this will produce a corresponding set of changes when the recipient ultimately decrypts the message. This means that, for example, if the attacker guesses that the message contains the word “cat” at some position, she can flip bits in the ciphertext to change that part of the message to read “dog” — and she can make this change *even though she can’t actually read the encrypted message*.

Only one more big step to go.

Now further imagine that the recipient’s phone will decrypt the message correctly provided that the underlying plaintext that appears following decryption is correctly formatted. If the plaintext is improperly formatted — for a silly example, our tampering made it say “*7!” instead of “pig” — then on receiving the message, the recipient’s phone might return an error that the attacker can see.

It’s well known that such a configuration capability allows our attacker the ability to learn information about the original message, provided that she can send many “mauled” variants to be decrypted. By mauling the underlying message in specific ways — e.g., attempting to turn “dog” into “pig” and observing whether decryption succeeds — the attacker can gradually learn the contents of the original message. The technique is known as a format oracle, and it’s similar to the padding oracle attack discovered by Vaudenay.

So how exactly does this format oracle work?

The format oracle in iMessage is not a padding oracle. Instead it has to do with the compression that iMessage uses on every message it sends.

You see, prior to encrypting each message payload, iMessage applies a complex formatting that happens to conclude with gzip compression. Gzip is a modestly complex compression scheme that internally identifies repeated strings, applies Huffman coding, then tacks a CRC checksum computed over the *original* data at the end of the compressed message. It’s this gzip-compressed payload that’s encrypted within the AES portion of an iMessage ciphertext.

It turns out that given the ability to maul a gzip-compressed, encrypted ciphertext, there exists a fairly complicated attack that allows us to gradually recover the contents of the message by mauling the original message thousands of times and sending the modified versions to be decrypted by the target device. The attack turns on our ability to maul the compressed data by flipping bits, then "fix up" the CRC checksum correspondingly so that it reflects the change we hope to see in the uncompressed data. Depending on whether that test succeeds, we can gradually recover the contents of a message — one byte at a time.

While I'm making this sound sort of simple, the truth is it's not. The message is encoded using Huffman coding, with a *dynamic* Huffman table we can't see — since it's encrypted. This means we need to make laser-specific changes to the ciphertext such that we can *predict* the effect of those changes on the decrypted message, and we need to do this blind. Worse, iMessage has various countermeasures that make the attack more complex.The complete details of the attack appear in the paper, and they're pretty eye-glazing, so I won't repeat them here. In a nutshell, we are able to decrypt a message under the following conditions:

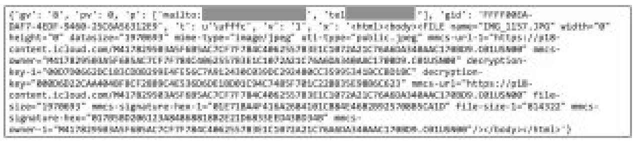
1. We can obtain a copy of the encrypted message
2. We can send approximately 2^18 (invisible) encrypted messages to the target device
3. We can determine whether or not those messages decrypted successfully or not

The first condition can be satisfied by obtaining ciphertexts from a compromise of Apple's Push Notification Service servers (which are responsible for routing encrypted iMessages) or by intercepting TLS connections using a stolen certificate — something made more difficult due to the addition of certificate pinning in iOS 9. The third element is the one that initially seems the most challenging. After all, when I send an iMessage to your device, there's no particular reason that your device should send me any sort of response when the message decrypts. And yet this information is fundamental to conducting the attack!

It turns out that there's a big exception to this rule: attachment messages.

How do attachment messages differ from normal iMessages?

When I include a photo in an iMessage, I don't actually send you the photograph through the normal iMessage channel. Instead, I first encrypt that photo using a random 256-bit AES key, then I compute a SHA1 hash and upload the encrypted photo to iCloud. What I send you via iMessage is actually just an iCloud.com URL to the encrypted photo, the SHA1 hash, and the decryption key.



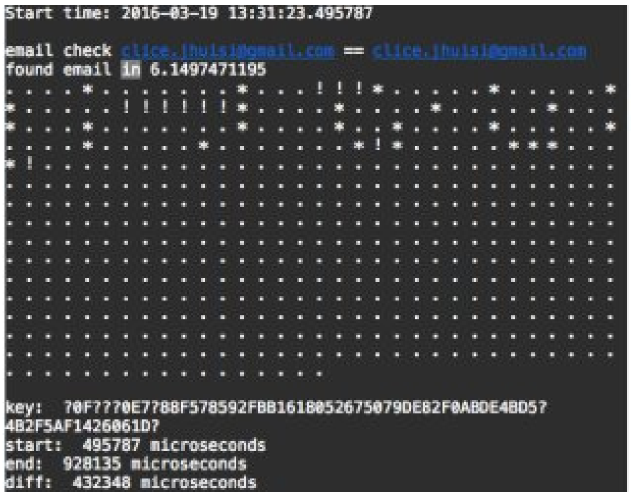
```
{'gv': '8', 'pv': 0, 'p': ['mailto:[redacted]', 'tel[redacted]'], 'gid': 'FFFF08EA-
DAF7-4E0F-9460-25C0A5631E59', 't': u'\ufffc', 'v': '1', 'x': '<html><body><FILE name="IMG_1157.JPG" width="0"
height="0" datasize="1970693" mime-type="image/jpeg" uti-type="public.jpeg" mmcs-url-1="https://p18-
content.icloud.com/[illegible]" mmcs-
owner="[illegible]" decryption-
key-1="[illegible]" decryption-
key="[illegible]" mmcs-url="https://p18-
content.icloud.com/[illegible]" file-
size="1970693" mmcs-signature-hex-1="[illegible]" file-size-1="814322" mmcs-
signature-hex="[illegible]" mmcs-
owner-1="[illegible]"/></body></html>'}
```

Contents of an "attachment" message.

When you successfully receive and decrypt an iMessage from some recipient, your Messages client will automatically reach out and attempt to download that photo. It's this download attempt, which happens *only* when the phone successfully decrypts an attachment message, that makes it possible for an attacker to know whether or not the decryption has succeeded.

One approach for the attacker to detect this download attempt is to gain access to and control your local network connections. But this seems impractical. A more sophisticated approach is to actually maul the URL within the ciphertext so that rather than pointing to iCloud.com, it points to a related URL such as i8loud.com. Then the attacker can simply register that domain, place a server there and allow the client to reach out to it. This requires no access to the victim's local network.

By capturing an attachment message, repeatedly mauling it, and monitoring the download attempts made by the victim device, we can gradually recover all of the digits of the encryption key stored within the attachment. Then we simply reach out to iCloud and download the attachment ourselves. And that's game over. The attack is currently quite slow — it takes more than 70 hours to run — but mostly because our code is slow and not optimized. We believe with more engineering it could be made to run in a fraction of a day.



```
Start time: 2016-03-19 13:31:23.495787

email check clice.jhuisi@gmail.com == clice.jhuisi@gmail.com
found email in 6.1497471195
. . . . * . . . . . . . * . . . ! ! ! * . . . . . * . . . . . *
* . . . . . ! ! ! ! ! ! * . . . . . * . . . . . * . . . . . * . . .
* . . . * . . . . . . . * . . . . * . . * . . . . * . . . . . *
. . . . * . . . . . * . . . . . . . * ! * . . . . . * * * . . .
* ! . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .

key:  ?0F???0E7?88F578592FBB1618052675079DE82F0ABDE4BD57
4B2F5AF1426061D?
start:  495787 microseconds
end:  928135 microseconds
diff:  432348 microseconds
```

Result of decrypting the AES key for an attachment. Note that the ? symbol represents a digit we could not recover for various reasons, typically due to string repetitions. We can brute-force the remaining digits.

The need for an online response is why our attack currently works against attachment messages only: those are simply the messages that make the phone do visible things. However, this does not mean the flaw in iMessage encryption is somehow limited to attachments — it could very likely be used against other iMessages, given an appropriate side-channel.

How is Apple fixing this?

Apple's fixes are twofold. First, starting in iOS 9.0 (and before our work), Apple began deploying aggressive certificate pinning across iOS applications. This doesn't fix the attack on iMessage crypto, but it does make it much harder for attackers to recover iMessage ciphertexts to decrypt in the first place.

Unfortunately even if this works perfectly, *Apple* still has access to iMessage ciphertexts. Worse, Apple's servers will retain these messages for up to 30 days if they are not delivered to one of your devices. A vulnerability in Apple Push Network authentication, or a compromise of these servers could read them all out. This means that pinning is only a mitigation, not a true fix.

As of iOS 9.3, Apple has implemented a short-term mitigation that my student Ian Miers proposed. This relies on the fact that while the AES ciphertext is malleable, the RSA-OAEP portion of the ciphertext is not. The fix maintains a “cache” of recently received RSA ciphertexts and rejects any repeated ciphertexts. In practice, this shuts down our attack — provided the cache is large enough. We believe it probably is.

In the long term, Apple should drop iMessage like a hot rock and move to Signal/Axolotl.

So what does it all mean?

As much as I wish I had more to say, fundamentally, security is just plain hard. Over time we get better at this, but for the foreseeable future we’ll never be ahead. The only outcome I can hope for is that people realize how hard this process is — and stop asking technologists to add unacceptable complexity to systems that already have too much of it.

By Matthew Green in Apple, attacks, imessage, messaging | March 21, 2016 | 2,693 Words | 13 Comments

Let’s talk about iMessage (again)

Yesterday’s New York Times carried a story entitled “Apple and other tech companies tangle with U.S. over data access“. It’s a vague headline that manages to obscure the real thrust of the story, which is that according to reporters at the Times, Apple has *not* been forced to backdoor their popular encrypted iMessage system. This flies in the face of some rumors to the contrary.

While there's not much new information in here, people on Twitter seem to have some renewed interest in how iMessage works; whether Apple could backdoor it if they wanted to; and whether the courts could force them to. The answers to those questions are respectively: "*very well*", "*absolutely*", and "*do I look like a national security lawyer*?"

So rather than tackle the last one, which nobody seems to know the answer to, I figure it would be informative to talk about the technical issues with iMessage (again). So here we go.

How does iMessage work?

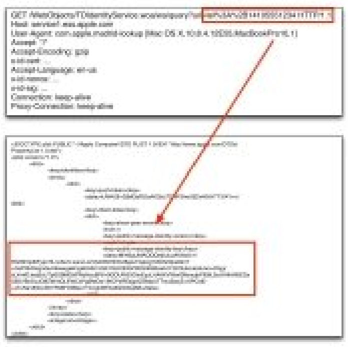
Fundamentally the mantra of iMessage is "keep it simple, stupid". It's not really designed to be an encryption system as much as it is a text message system that happens to include encryption. As such, it's designed to take away most of the painful bits you expect from modern encryption software, and in the process it makes the crypto essentially invisible to the user. Unfortunately, this simplicity comes at some cost to security.

Let's start with the good: Apple's marketing material makes it clear that iMessage encryption is "end-to-end" and that decryption keys never leave the device. This claim is bolstered by their public security documentation as well as outside efforts to reverse-engineer the system. In iMessage, messages are encrypted with a combination of 1280-bit RSA public key encryption and 128-bit AES, and signed with ECDSA under a 256-bit NIST curve. It's honestly kind of ridiculous, but whatever. Let's call it good enough.

iMessage encryption in a nutshell boils down to this: I get your public key, you get my public key, I can send you messages encrypted to you, and you can be sure that they're authentic and really came from me. Everyone's happy.

But here's the wrinkle: where do those public keys come from?

Where *do* you get the keys?



```
GET /WebObjects/TDIdentityService.woa/wa/query?uri=[illegible]
Host: service1.ess.apple.com
User-Agent: com.apple.madrid-lookup [Mac OS X,10.8.4,12E55,MacBookPro10,1]
Accept: */*
Accept-Encoding: gzip
x-id-cert: ...
Accept-Language: en-us
x-id-nonce: ...
x-id-sig: ...
Connection: keep-alive
Proxy-Connection: keep-alive
```

```
[illegible]
```

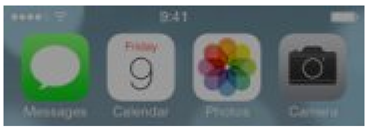
Key request to Apple's server.

It's this detail that exposes the real weakness of iMessage. To make key distribution 'simple', Apple takes responsibility for handing out your friends' public keys. It does this using a proprietary key server that Apple owns and operates. Your iPhone requests keys from Apple using a connection that's TLS-encrypted, and employs some fancy cryptographic tokens. But fundamentally, it relies on the assumption that Apple is good, and is *really* going to give you you the right keys for the person you want to talk to.

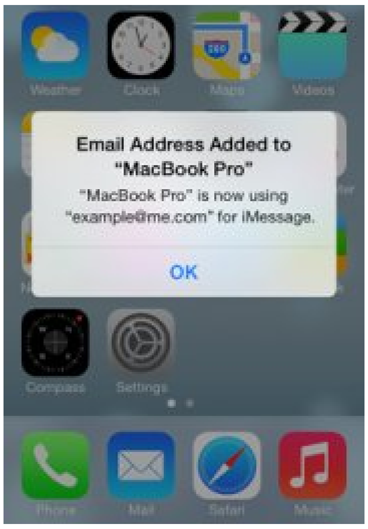
But this honesty is just an assumption. Since the key lookup is completely invisible to the user, there's nothing that forces Apple to be honest. They could, if inspired, give you a public key of their choosing, one that they hold the decryption key for. They could give you the FBI's key. They could give you Dwayne "The Rock" Johnson's key, though The Rock would presumably be very non-plussed by this.

Indeed it gets worse. Because iMessage is designed to support several devices attached to the same account, each query to the directory server can bring back *many keys* — one for each of your devices. An attacker can simply add a device (or a fake 'ghost device') to Apple's key server, and senders will encrypt messages to *that* key along with the legitimate ones. This enables wiretapping, provided you can get Apple to help you out.

But why do you need Apple to help you out?



As described, this attack doesn't really require direct collaboration from Apple. In principle, the FBI could


Email Address Added to "MacBook Pro"

"MacBook Pro" is now using "example@me.com" for iMessage.

just guess the target's email password, or reset the password and add a new device all on their own. Even with a simple subpoena, Apple might be forced to hand over security questions and/or password hashes.

The real difficulty is caused by a final security feature in iMessage: when you add a new device, or modify the devices attached to your account, Apple's key server sends a notification to *each of the existing devices already to the account*. It's not obvious how this feature is implemented, but one thing is clear — it seems likely that, at least in theory, Apple *could* shut it off if they needed to.* After all, this all comes down to code in the key server.

Fixing this problem seems hard. You could lock the key server in a giant cage, then throw away the key. But as long as Apple retains the ability to update their key server software, solving this problem seems fundamentally challenging. (Though not impossible — I'll come back to this in a moment.)

Can governments force Apple to modify their key server?

It's not clear. While it seems pretty obvious that Apple *could* in theory substitute keys and thus enable eavesdropping, in practice it may require substantial changes to Apple's code. And while there are a few well-known cases in which the government has forced companies to turn over keys, changing the operation of a working system is a whole different ball of wax.

And iMessage is not just any working system. According to Apple, it handles several billion messages every day, and is fundamental to the operation of millions of iPhones. When you have a deployed system at that scale, the last thing you want to do is mess with it — particularly if it involves crypto code that may not even be well understood by its creators. There's no amount of money you could pay *me* to be 'the guy who broke iMessage', even for an hour.

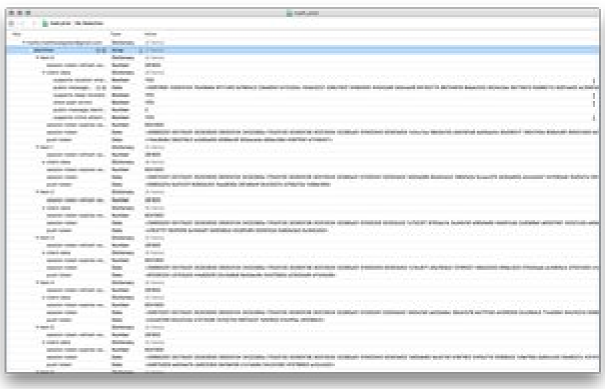
Any way you slice it, it's a risky operation. But for a real answer, you'll have to talk to a lawyer.

Why isn't key substitution a good solution to the 'escrow' debate?

Another perspective on iMessage — one I've heard from some attorney friends — is that key server tampering sounds like a pretty good compromise solution to the problem of creating a 'secure golden key' (AKA giving governments access to plaintext).

This view holds that key substitution allows only *proactive* eavesdropping: the government has to show up with a warrant before they can eavesdrop on a customer. They can't spy on everyone, and they can't go back and read your emails from last month. At the same time, most customers still get true 'end to end' encryption.

I see two problems with this view. First, tampering with the key server fundamentally betrays user trust, and undermines most of the guarantees offered by iMessage. Apple claims that they offer true end-to-end encryption that they can't read — and that's reasonable in the threat model they've defined for themselves. The minute they start selectively substituting keys, that theory goes out the window. If you can substitute a few keys, why not all of them? In this world, Apple should expect requests from every Tom, Dick and Harry who wants access to plaintext, ranging from divorce lawyers to foreign governments.

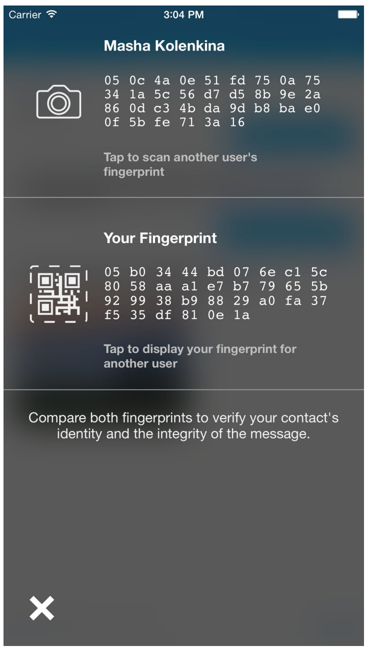


A snapshot of my seven (!) currently enrolled iMessage devices, courtesy Frederic Jacobs.

The second, more technical problem is that key substitution is relatively easy to detect. While Apple's protocols are an obfuscated mess, it is at least *in theory* possible for users to reverse-engineer them to view the raw public keys being transmitted — and thus determine whether the key server is being honest. While most criminals are not this sophisticated, a few are. And if they aren't sophisticated, then tools can be built to make this relatively easy. (Indeed, people have already built such tools — see my key registration profile at right.)

Thus key substitution represents at most a temporary solution to the 'government access' problem, and one that's fraught with peril for law enforcement, and probably disastrous for the corporations involved. It might seem tempting to head down this rabbit hole, but it's rabbits all the way down.

What can providers do to prevent key substitution attacks?



Masha Kolenkina

```
05 0c 4a 0e 51 fd 75 0a 75
34 1a 5c 56 d7 d5 8b 9e 2a
86 0d c3 4b da 9d b8 ba e0
0f 5b fe 71 3a 16
```

Tap to scan another user's fingerprint

Your Fingerprint

```
05 b0 34 44 bd 07 6e c1 5c
80 58 aa a1 e7 b7 79 65 5b
92 99 38 b9 88 29 a0 fa 37
f5 35 df 81 0e 1a
```

Tap to display your fingerprint for another user

Compare both fingerprints to verify your contact's identity and the integrity of the message.

Signal's "key fingerprint" screen.

From a technical point of view, there are a number of things that providers can do to harden their key servers. One is to expose ‘key fingerprints’ to users who care, which would allow them to manually compare the keys they receive with the keys actually registered by other users. This approach is used by OpenWhisperSystems’ Signal, as well as PGP. But even I acknowledge that this kind of stinks.

A more user-friendly approach is to deploy a variant of Certificate Transparency, which requires providers to publish a *publicly verifiable proof* that every public key they hand out is being transmitted to the whole world. This allows each client to check that the server is handing out the actual keys they registered — and by implication, that every other user is seeing the same thing.

The most complete published variant of this is called CONIKS, and it was proposed by a group at Princeton, Stanford and the EFF (one of the more notable authors is Ed Felten, now Deputy U.S. Chief Technology Officer). CONIKS combined key transparency with a ‘verification protocol’ that allows clients to ensure that they aren’t being sidelined and fed false information.

CONIKS isn’t necessarily the only game in town when it comes to preventing key substitution attacks, but it represents a powerful existence proof that real defenses can be mounted. Even though Apple hasn’t chosen to implement CONIKS, the fact that it’s out there should be a strong disincentive for law enforcement to rely heavily on this approach.

So what next?

That’s the real question. If we believe the New York Times, all is well — for the moment. But not for the future. In the long term, law enforcement continues to ask for an approach that allows them to access the plaintext of encrypted messages. And Silicon Valley continues to find new ways to protect the confidentiality of their user’s data, against a range of threats beginning in Washington and proceeding well beyond.

How this will pan out is anyone's guess. All we can say is that it will be messy.

Notes:

* How they would do this is really a question for Apple. The feature may involve the key server sending an explicit push message to each of the devices, in which case it would be easy to turn this off. Alternatively, the devices may periodically retrieve their own keys to see what Apple's server is sending out to the world, and alert the user when they see a new one. In the latter case, Apple could selectively transmit a doctored version of the key list to the device owner.

By Matthew Green in Apple, imessage, messaging | September 9, 2015 | 1,794 Words | 10 Comments

---

Can Apple read your iMessages?

About a year ago I wrote a short post urging Apple to publish the technical details of iMessage encryption. I'd love tell you that Apple saw my influential crypto blogging and fell all over themselves to produce a spec, but, no. iMessage is the same black box it's always been.

What's changed is that suddenly *people seem to care*. Some of this interest is due to Apple's (alleged) friendly relationship with the NSA. Some comes from their not-so-friendly relationship with the DEA. Whatever the reason, people want to know which of our data Apple has and who they're sharing it with.

And that brings us back to iMessage encryption. Apple runs one of the most popular encrypted communications services on Earth, moving over two billion iMessage every day. Each one is loaded with personal information the NSA/DEA would just love to get their hands on. And yet Apple claims they can't. In fact, *even Apple can't read them*:

> *There are certain categories of information which we do not provide to law enforcement or any other group because we choose not to retain it.*

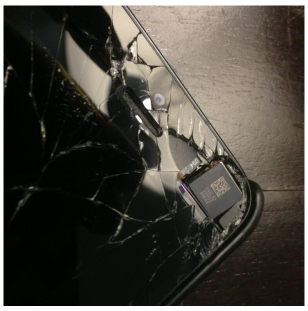
> ***For example, conversations which take place over iMessage and FaceTime are protected by end-to-end encryption so no one but the sender and receiver can see or read them. Apple cannot decrypt that data.***

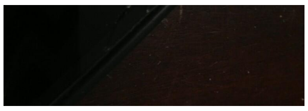
This seems almost too good to be true, which in my experience means it probably is. My view is inspired by something I like to call "Green's law of applied cryptography", which holds that *applied cryptography mostly sucks*. Crypto never offers the unconditional guarantees you want it to, and when it does your users suffer terribly.

And that's the problem with iMessage: users don't suffer enough. The service is almost magically easy to use, which means Apple has made tradeoffs — or more accurately, they've chosen a particular balance between usability and security. And while there's nothing wrong with tradeoffs, the *particulars of their choices* make a big difference when it comes to your privacy. By witholding these details, Apple is preventing its users from taking steps to protect themselves.

The details of this tradeoff are what I'm going to talk about in this post. A post which I swear will be the *last post I ever write on iMessage*. From here on out it'll be ciphers and zero knowledge proofs all the way.

Apple backs up iMessages to iCloud





That’s the super-secret NSA spying chip.

The biggest problem with Apple’s position is that it just plain isn’t true. If you use the iCloud backup service to back up your iDevice, there’s a very good chance that Apple can access the last few days of your iMessage history.
For those who aren’t in the Apple ecosystem: iCloud is an optional backup service that Apple provides for free. Backups are great, but if *iMessages* are backed up we need to ask how they’re protected. Taking Apple at their word — that they really can’t get your iMessages — leaves us with two possibilities:

1. iMessage backups are encrypted under a key that ‘never leaves the device‘.
2. iMessage backups are encrypted using your password as a key.

Unfortunately neither of these choices really works — and it’s easy to prove it. All you need to do is run the following simple experiment: First, lose your iPhone. Now change your password using Apple’s iForgot service (this requires you to answer some simple security questions or provide a recovery email). Now go to an Apple store and shell out a fortune buying a new phone.

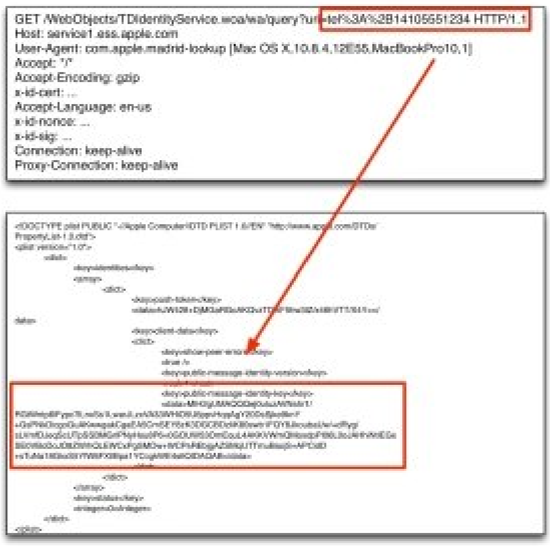
If you can recover your recent iMessages onto a new iPhone — as I was able to do in an Apple store this afternoon — then Apple *isn’t* protecting your iMessages with your password or with a device key. Too bad. (**Update 6/27:** Ashkan Soltani also has some much nicer screenshots from a similar test.)

The sad thing is there’s really no crypto to understand here. The simple and obvious point is this: if *I* could do this experiment, then someone at Apple could have done it too. Possibly at the request of law enforcement. All they need are your iForgot security questions, something that Apple almost certainly does keep.*

Apple distributes iMessage encryption keys

But maybe you don't use backups. In this case the above won't apply to you, and Apple clearly says that their messages are end-to-end encrypted. The question you should be asking now is: *encrypted to whom*?

The problem here is that encryption only works if I have your encryption key. And that means before I can talk to you I need to get hold of it. Apple has a simple solution to this: they operate a *directory lookup* service that iMessage can use to look up the public key associated with any email address or phone number. This is great, but represents yet another tradeoff: you're now fundamentally dependent on Apple *giving you the right key*.



```
GET /WebObjects/TDIdentityService.woa/wa/query?uri=tel%3A%2B14105551234 HTTP/1.1
Host: service1.ess.apple.com
User-Agent: com.apple.madrid-lookup [Mac OS X,10.8.4,12E55,MacBookPro10,1]
Accept: */*
Accept-Encoding: gzip
x-id-cert: ...
Accept-Language: en-us
x-id-nonce: ...
x-id-sig: ...
Connection: keep-alive
Proxy-Connection: keep-alive
```

```
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
        <dict>
                <key>identities</key>
                <array>
                        <dict>
                                <key>push-token</key>
                                <data>[illegible]</
data>
                                <key>client-data</key>
                                <dict>
                                        <key>show-peer-errors</key>
                                        <true />
                                        <key>public-message-identity-version</key>
                                        [illegible]
                                        <key>public-message-identity-key</key>
                                        <data>[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]</data>
                                </dict>
                        </dict>
                </array>
                <key>status</key>
                <integer>0</integer>
        </dict>
</plist>
```

HTTPS request/response containing a "message identity key" associated with an iPhone phone number (modified). These keys are sent over SSL.

The concern here is that Apple – or a hacker who compromises Apple's directory server – might instead deliver *their own key*. Since you won't know the difference, you'll be encrypting to that person rather than to your friend.**

Moreover, iMessage lets you associate multiple public keys with the same account — for example, you can you add a device (such as a Mac) to receive copies of messages sent to your phone. From what I can tell, the iMessage app gives the sender no indication of how many keys have been associated with a given iMessage recipient, nor does it warn them if the recipient suddenly develops new keys.

The practical upshot is that the integrity of iMessage depends on Apple honestly handing out keys. If they cease to be honest (or if somebody compromises the iMessage servers) it may be possible to run a man-in-the-middle attack and silently intercept iMessage data.

Now to some people *this is obvious,* and to other's *it's no big deal*. All of which is fine. But people should at least understand the strengths and weaknesses of the particular design that Apple has chosen. Armed with that knowledge they can make up their minds how much they want to trust Apple.

Apple can retain metadata

While Apple may encrypt the contents of your communication, their statement doesn't exactly rule out the possibility they store *who you're talking to*. This is the famous meta-data the NSA already sweeps up and (as I've said before) it's almost impossible not to at least collect this information, especially since Apple actually delivers your messages through their servers.

This metadata can be as valuable as the data itself. And while Apple doesn't retain the *content* of your messages, their statement says nothing about all that metadata.

Apple doesn't use Certificate Pinning

As a last – and fairly minor point – iMessage client applications (for iPhone and Mac) communicate with Apple's directory service using the HTTPS protocol. (Note that this applies to directory lookup messages: the actual iMessages are encrypted separately and travel over ~~XMPP~~ Apple's push network protocol.)

Using HTTPS is a good thing, and in general it provides strong protections against interception. But it doesn't protect against all attacks. There's still a very real possibility that a capable attacker could obtain a forged certificate (possibly by compromising a Certificate Authority) and thus intercept or modify communications with Apple.

This kind of thing isn't as crazy as it sounds. It happened to hundreds of thousands of Iranian Gmail users, and it's likely to happen again in the future. The standard solution to this problem is called ‘certificate pinning‘ — this essentially tells the application not to trust unknown certificates. Many apps such as Twitter do this. However based on the testing I did while writing this post, Apple doesn't.

Conclusion

I don't write any of this stuff because I dislike Apple. In fact I love their products and would *bathe* with them if it didn't (unfortunately) violate the warranty.

But the flipside of my admiration is simple: I rely on these devices and want to know how secure they are. I see absolutely no downside to Apple presenting at least a high-level explanation to experts, even if they keep the low-level details to themselves. This would include the type and nature of the encryption algorithms used, the details of the directory service and the key agreement protocol.

Apple may Think Different, but security rules apply to them too. Sooner or later someone will compromise or just plain reverse-engineer the iMessage system. And then it'll all come out anyway.

Notes:

* Of course it's possible that Apple is using your security questions to derive an encryption key. However this seems unlikely. First because it's likely that Apple has your question/answers on file. But even if they don't, it's unlikely that many security answers contain enough entropy to use for encryption. There are only so many makes/models of cars and so many birthdays. Apple's 2-step authentication may improve things if you use it — but if so

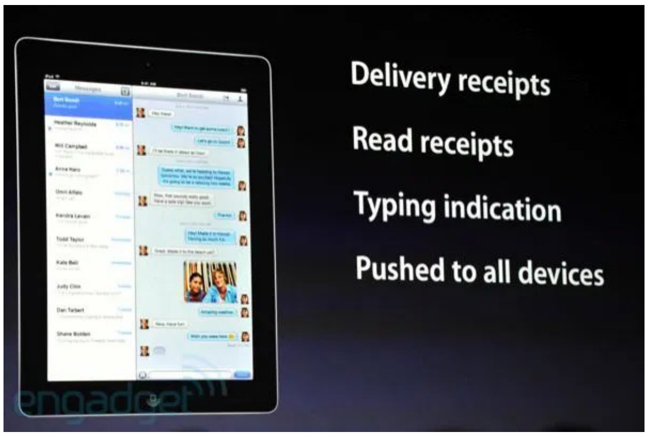
Apple isn't saying.

** In practice it's not clear if Apple devices encrypt to this key directly or if they engage in an OTR-like key exchange protocol. What is clear is that iMessage does not include a 'key fingerprint' or any means for users to verify key authenticity, which means fundamentally you have to trust Apple to guarantee the authenticity of your keys. Moreover iMessage allows you to send messages to offline users. It's not clear how this would work with OTR.

By Matthew Green in Apple, imessage, messaging | June 26, 2013 | 1,596 Words | 21 Comments

---

Dear Apple: Please set iMessage free



Normally I avoid complaining about Apple because *(a)* there are plenty of other people carrying that flag, and *(b)* I honestly like Apple and own numerous lovely iProducts. I'm even using one to write this post.

Moroever, from a security point of view, there isn't that much to complain about. Sure, Apple has a few irritating habits — shipping old, broken versions of libraries in its software, for example. But on the continuum of security crimes this stuff is at best a misdemeanor, maybe a half-step above 'improper baby naming'. Everyone's software sucks, news at 11.

There is, however, *one* thing that drives me absolutely nuts about Apple's security posture. You see, starting about a year ago Apple began operating one of the most widely deployed encrypted text message services in the history of mankind. So far so good. The problem is that *they still won't properly* *explain how it works.*

And nobody seems to care.

I am, of course, referring to iMessage, which was deployed last year in iOS Version 5. It allows — nay, *encourages* — users to avoid normal carrier SMS text messages and to route their texts through Apple instead.

Now, this is not a particularly new idea. But iMessage is special for two reasons. First it's built into the normal iPhone texting application and turned *on* by default. When my Mom texts another Apple user, iMessage will automatically route her message over the Internet. She doesn't have to approve this, and honestly, probably won't even know the difference.

Secondly, iMessage claims to bring 'secure end-to-end encryption' (and authentication) to text messaging. In principle this is huge! True end-to-end encryption should protect you from eavesdropping even by Apple, who carries your message. Authentication should protect you from spoofing attacks. This stands in contrast to normal SMS which is often not encrypted at all.

So why am I looking a gift horse in the mouth? iMessage will clearly save you a ton in texting charges and it will secure your messages for free. Some encryption is better than none, right?

Well maybe.

To me, the disconcerting thing about iMessage is how rapidly it's gone from *no deployment* to securing billions of text messages for millions of users. And this despite the fact that the full protocol *has never been published by Apple* or (to my knowledge) vetted by security experts. (Note: if I'm wrong about this, let me know and I'll eat my words.)

What's worse is that Apple has been hyping iMessage as a secure protocol; they even propose it as a *solution* to some serious SMS spoofing bugs. For example:

> *Apple takes security very seriously. **When using iMessage instead of SMS, addresses are verified which protects against these kinds of spoofing attacks**. One of the limitations of SMS is that it allows messages to be sent with spoofed addresses to any phone, so we urge customers to be extremely careful if they're directed to an unknown website or address over SMS.*

And this makes me nervous. While iMessage may very well be as secure as Apple makes it out to be, there are plenty of reasons to give the protocol a second look.

For one thing, it's surprisingly complicated.

iMessage is not just two phones talking to each other with TLS. If this partial reverse-engineering of the protocol (based on the MacOS Mountain Lion Messages client) is for real, then there are *lots* of moving parts. TLS. Client certificates. Certificate signing requests. New certificates delivered via XML. Oh my.

As a general rule, lots of moving parts means lots of places for things to go wrong. Things that could seriously reduce the security of the protocol. And as far as I know, nobody's given this much of a look. It's surprising.

Moreover, there are some *very real* questions about what powers Apple has when it comes to iMessage. In principle 'end-to-end' encryption should mean that *only* the end devices can read the connection. In practice this is almost certainly not the case with iMessage. A quick glance at the protocol linked above is enough to tell me that Apple operates as a Certificate Authority for iMessage devices. And as a Certificate Authority, it may be able to substantially undercut the security of the protocol. When would Apple do this? How would it do this? Are we allowed to know?

Finally, there have been several reports of iMessages going astray and even being delivered to the wrong (or stolen) devices. This stuff may all have a reasonable explanation, but it's yet another set of reasons why we it would be nice to *understand* iMessage better than we do now if we're going to go around relying on it.

So what's my point with all of this?

This is obviously not a technical post. I'm not here to present answers, which is disappointing. If I knew the protocol maybe I'd have some. Maybe I'd even be saying good things about it.

Rather, consider this post as a plea for help. iMessage is important. People use it. We *ought* to know how secure it is and what risks those people are taking by using it. The best solution would be for Apple to simply release a detailed specification for the protocol — even if they need to hold back a few key details. But if that's not possible, maybe we in the community should be doing more to find out.

Remember, it's not just our security at stake. People we know are using these products. It would be awfully nice to know what that means.

By Matthew Green in Apple, imessage, messaging, Uncategorized | August 19, 2012 | 925 Words | 57 Comments

Matthew Green

I'm a cryptographer and professor at Johns Hopkins University. I've designed and analyzed cryptographic systems used in wireless networks, payment systems and digital content protection platforms. In my research I look at the various ways cryptography can be used to promote user priva

My academic website

My twitter feed
Top Posts
Useful crypto resources
Bitcoin tipjar
Cryptopals challenges
Applied Cryptography Research: A Board

Journal of Cryptographic Engineering (not related to this blog)

Search …

Top Posts & Pages

The strange story of "Extended Random"
Why I'm done with Chrome
How safe is Apple's Safe Browsing?
Zero Knowledge Proofs: An illustrated primer
Attack of the week: RC4 is kind of broken in TLS
Let's talk about PAKE
Why is Signal asking users to set a PIN, or “A few thoughts on Secure Value Recovery”
What is Differential Privacy?
Ok Google: please publish your DKIM secret keys
What is the Random Oracle Model and why should you care? (Part 1)

Banner image by Matt Blaze

Archives

Select Month ▾