NCC Group Whitepaper

Secure Messaging for Normal People

September 18, 2015 – Version 1.0

Prepared by

Justin Engler — Principal Security Consultant

Cara Marie — Senior Security Consultant (Diagrams)

Abstract

"Secure" messaging programs and protocols continue to proliferate, and crypto experts can debate their minutiae, but there is very little information available to help the rest of the world differentiate between the different programs and their features. This paper discusses the types of attacks used against a variety of messaging models and discusses how secure messaging features can defend against them. The goal of this paper is to help inform those who are tech-savvy but not crypto-experts to make smart decisions on which crypto applications to use.

Table of Contents

1 Introduction

Security and privacy continue to be on the minds of the public, and many users want to know how to keep their communications private. At the same time, users are being offered more choices in how to communicate with each other than ever before, and some of these choices purport to be "secure". By the end of this paper, you will be able to understand what types of things the "bad guys" will try to do to access your communications and what features messaging systems offer to thwart those attempts.

1.1 What is a message?

For the purposes of this paper, a "message" is any information you want to send to another person or group of people. Phone calls, SMS, email, video chat, text chat, and other similar items are included, but things like communications between your computer and a server are not messages themselves as far as we're concerned in this paper.

A messaging client is whatever collection of systems you use to send messages. This could mean anything from a traditional phone up to a niche messaging application on a computer.

1.2 Cryptography is magic

There are a huge number of different types of cryptographic algorithms that are used for different situations. Algorithm choice, implementation flaws, and similar considerations can have huge impacts on the security of your data, but such details are outside of the scope of this discussion. While reading this paper, you can translate "encrypted" as "impossible for anyone else to read or modify", unless otherwise stated. You shouldn't necessarily translate it that way in the real world though, you would need to have cryptography and application security experts sign off on the actual algorithms and implementations used.

Even though we're making this assumption that encryption is unbreakable, there are plenty of other ways adversaries could get to your data. The purpose of this paper is to explain those things and what to do about them.

1.3 Cryptobabble

This paper attempts to simplify things as much as possible, but there are a few key cryptography terms that we'll be using. Sometimes their meanings can be confusing if you don't already know what they are, so let's lay a few out now.

- **Key** - A secret value that "unlocks" an encrypted message.
- **Public Key/Private Key** - These come in pairs. The idea is that you give your public key away to everyone, but you keep the private key to yourself. Anyone with your public key can send an encrypted message that can only be decrypted with the private key. You can imagine this as you distributing the blueprints to a lock that will only open with your key. Now, anyone with the blueprints (public key) can build a locked container (encrypt a message) that only the private key can open.
- **Signatures** - This is another use for public keys. If you have my public key, you can verify that a given message actually came from me. Interestingly, that message can be sent unencrypted (so anyone can read it), but you can still use the public key to validate it. For a real-world analogy, imagine that you have a sample of my handwriting signature (public key). If I send you a document and I sign it with my pen (private key), you can check to see if the signatures match.
- **Fingerprints** - Usually in cryptography a fingerprint has nothing to do with the oily smudges you leave everywhere. Instead, when we say fingerprint, we mean a short value that can represent a longer one. Keys can be very long (a number 925 digits long, for one example), so we use some math to boil that down into something manageable that a person could read and compare (a 13 digit fingerprint for the previous 925 digit key). Machines will use the full key, but if a person needs to verify the value of a key, the person could

use fingerprints instead. It's called a fingerprint because it uniquely identifies the larger value. Don't panic if someone wants to see your fingerprints!

- **Trust** - If we say that we "trust" people, we're not saying we'd let them house-sit or borrow money. It simply means that we're relying on them to vouch for someone's identity. Sometimes we'll trust an individual person, other times we'll trust a more abstract entity.

2 Your own threat model

It's impossible to determine what the "best" encrypted chat program is because everyone has different needs. To decide what kinds of security you need, first understand who is trying to attack you and what they are trying to take from you.

2.1 Adversary types

For the purposes of this talk, we're going to talk about four different types of adversary.

Opportunistic, Low-resource adversaries: These attackers aren't targeting you directly, and they aren't willing to commit much effort, skill, and/or time on you. Novice troublemakers, corporate/educational monitoring systems, analysis for use in advertising systems, etc.

Targeted, Low-resource adversaries: These attackers are specifically after your information, but they don't have massive resources. Targeted hackers, organized crime, corporate espionage, harassers, and the like make up this group.

Opportunistic, High-resource adversaries: These attackers aren't looking at you specifically, but they have massive resources to throw at the problem of bulk surveillance. This group is likely to be a government program doing large scale data collection, monitoring, and/or filtering.

Targeted, High-resource adversaries: These attackers are likely government actors who have already zeroed in on you specifically. It is highly unlikely that anything we discuss in this paper will be able to protect you in any meaningful way from this kind of attention, but hopefully we'll at least be able to explain why your choice of secure messaging program won't be much help against this class of threat.

2.2 Why are you keeping secrets?

When choosing a messaging system, you need to consider how much you value the information you want to keep secret. You also need to consider how much value your data has to an adversary who wants to steal it. Both of these inform how much effort you should put into your security.

Different types of data will have different value. This paper is mostly concerned with the value of the actual content of your messages, but we will also discuss some other types of data and why it is important.

Here are a few common scenarios:

Privacy-concerned average citizen: You don't have anything in particular you're trying to hide in most of your messages. You're concerned about keeping financial data, medical records, and similar information confidential, but you don't often discuss that data over messaging clients. You might also be interested in encrypted communications simply because you believe in privacy as a principle, or to help give legitimacy to other crypto users.

Business users: Businesses often have employees who need to discuss confidential information such as trade secrets or financial data.

Activists: People who do things that might draw government attention. We're going to stay away from politics in this paper and just assume that there are causes that are both morally/ethically just and are simultaneously oppressed by some government that can bring punishment to bear.

Harassed: People whose social or political beliefs are unpopular with specific individuals or groups or are targeted for harassment for other reasons (celebrities or other media personalities might fall into this category). The harassers might try to read messages to publicly air them or try to uncover a physical address or current location.

People who send nude pictures: If you want to send pictures to someone, it's likely you want that picture to only be readable by the people you actually sent it to.

3 Messages without encryption

To help explain how encrypted messaging systems can and cannot protect users, first, let's examine an unencrypted messaging system.

On the Internet, messages passed between Alice and Bob will move from their devices to the local network, then across an undefined set of Internet devices, then to the receivers' local network and onto the receiving device. If the communications aren't encrypted, eavesdroppers in any of the attacker groups above can read the message in most cases.

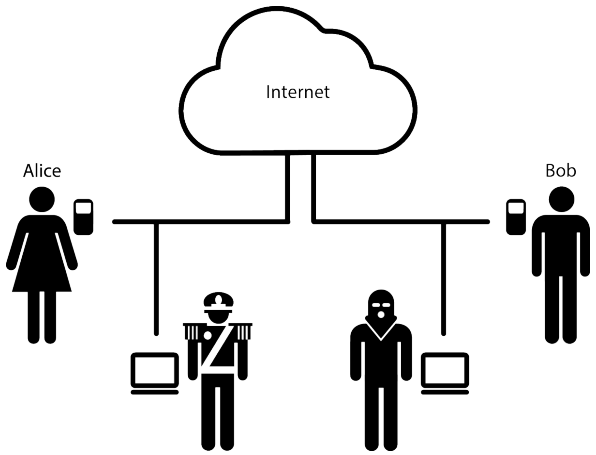

Figure 1: Alice, Bob and the Evil Eavesdroppers

The situation for SMS messages and phone calls is comparable, though the technical details vary. For mobile networks, although these technologies are not always encrypted in a meaningful way, eavesdropping on them is often more difficult than eavesdropping on IP networks. At the time of this writing, it is probably safe to assume that low-resource, non-targeted adversaries are not able to intercept SMS and mobile phone calls. Attacks against mobile phone technologies are getting easier every day, so it's likely that these types of communications will continue to get easier.

You could also envision more esoteric architectures, like bluetooth mesh networks, that also follow this same general pattern.

4 Encryption endpoints

The most common form of communications encryption on the Internet is Transport Layer Security (TLS).[1] This technology is designed to secure communications between a client and server. The diagram would look something like this.

This is much better than before, as it shuts out most eavesdroppers at the network layer. It does highlight a specific, powerful attacker.

Traffic from Alice to the server is encrypted, and then it is encrypted again from the server to Bob. The server is in the middle of the traffic and can read, store, or change any messages. Even if you trust the operators of the server, the more popular it becomes, the more likely it is to be targeted by those who want to observe message traffic. Additionally, the server operators will be operating under some legal jurisdiction, and most jurisdictions require at least some level of access to the data by law enforcement.

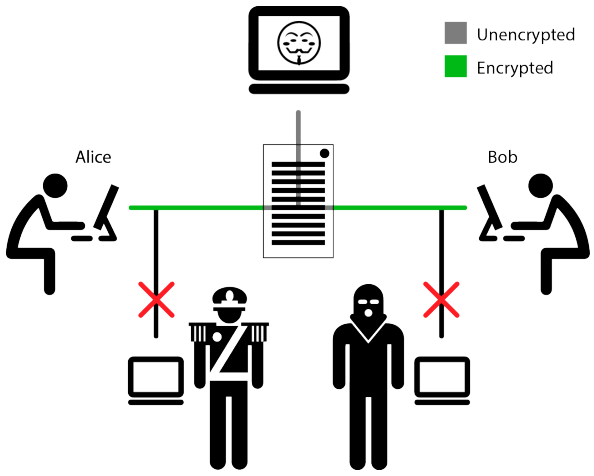

Figure 2: Malicious servers can still read TLS traffic

We also need a way to confirm that the server we're talking to is actually the correct one, and not just a server pretending to be the legitimate chat server. This is handled as an integral part of TLS. Your operating system or application has a list of trusted certificate authorities (CAs) that your system trusts to "vouch for" a given site's identity. This system can be abused in a few ways. Your system trusts a huge number of CAs by default, and it trusts any of them to vouch for any website. This means that anyone who can access or influence any CA can forge a certificate for any server, and therefore vouch for a fake server. As mentioned above, for messaging systems that use this architecture, the encryption ends at the server, so in this case it would end at the fake server.

Another way to abuse these systems is to add extra CAs to the list of CAs that your system trusts. Many businesses do this automatically for their computers and networks, so that the business can monitor communications for security problems or lazy workers. Some public networks have also been known to require users to install custom CAs. Attackers can try to convince users to add attacker-controlled CAs to the trusted list for a given computer. Finally, sometimes software errors will simply skip the verification step, allowing any server to claim to be any other server.

Despite all of these disadvantages, this is still a common model, and it works well in the majority of computer communications if you're willing to allow the server to read all of your messages and you understand that the potential exists for high resource attackers to intercept your communications (and in a few selected cases, lower resourced attackers as well).

[1] Older versions were called Secure Sockets Layer (SSL). TLS and SSL are the technology behind "HTTPS" used in web pages (the lock icon in your address bar)

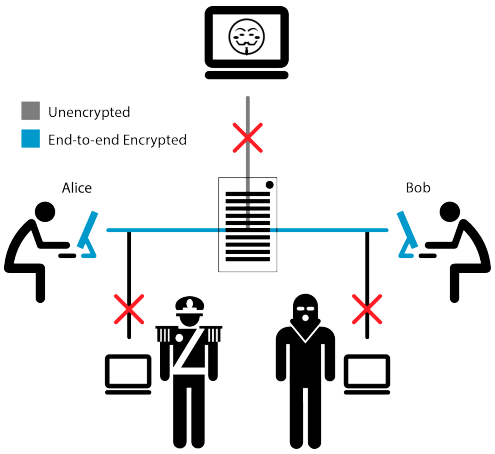

Figure 3: End-to-end encrypted traffic is only readable by Alice and Bob's computers.

A better solution would be to find a way to make the encryption endpoints be Alice's device and Bob's device, so that even if there's a central chat server, it can't actually read the data.

This is an ideal situation, but there are implementation details that make end-to-end encryption more complicated than the diagram makes it seem. We'll cover several topics that are important in encrypted messaging, and the potential ways they are addressed in both user-to-provider and end-to-end encrypted messaging.

5 Key validation

In order for end-to-end encryption to be possible, we need a way for Alice and Bob to exchange encryption keys securely, or at least to be able to verify that the keys they see for the other party actually belong to the other party. Unfortunately, it's not possible to just pass the keys over the Internet directly: think about the initial key as just another kind of message and look back at the first diagram to understand why. There's an important wrinkle here that we're mostly going to gloss over under our "cryptography is magic" clause above. It actually doesn't matter if an attacker can read our keys, there's a special type of encryption called "public key cryptography" that makes this happen. We do need to worry about an adversary changing keys that we send. This means that if we go back to the original unencrypted messages case, we're not safe because any of the eavesdroppers could replace our key with their key.

Consider another common architecture for message passing, this time using a form of end-to-end encryption, so Alice's messages to Bob don't get decrypted on the server.

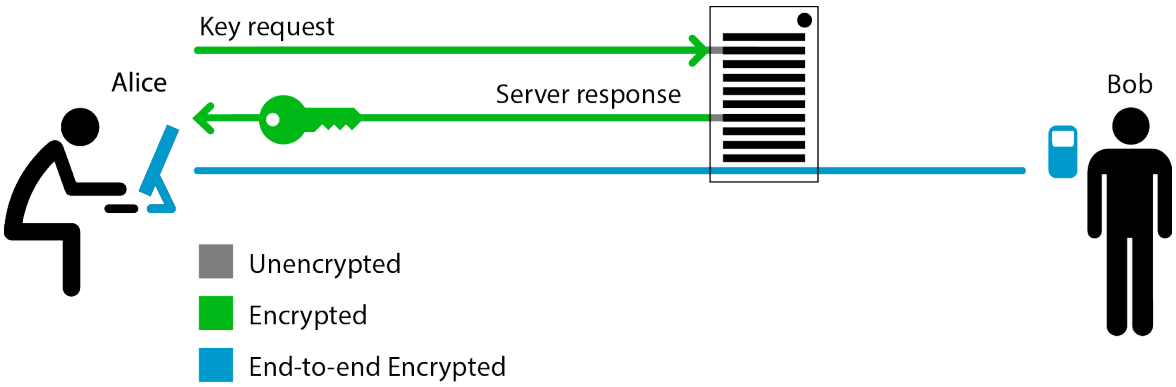

Figure 4: Alice asks Server for Bob's key, Server provides it

In the above diagram, the server has a database all of the identities. If Alice wants to talk to Bob, Alice asks the server for Bob's key, the server sends back a key, Alice uses it to encrypt a message to Bob.

Alice has just blindly trusted the server to tell the truth and provide Bob's key, but at this point, there's no way for her to verify that the server provided Bob's key.

5.1 Trust on first use

There are solutions to this problem, but they can become cumbersome quickly. First, simplest, and least secure is Trust on First Use (TOFU). The first time Alice asks for Bob's key and gets a response, Alice saves the key (or some representation of the key). The next time Alice tries to send a message to Bob, Alice either uses the saved key or checks to see if the key received from the server matches the one received earlier.

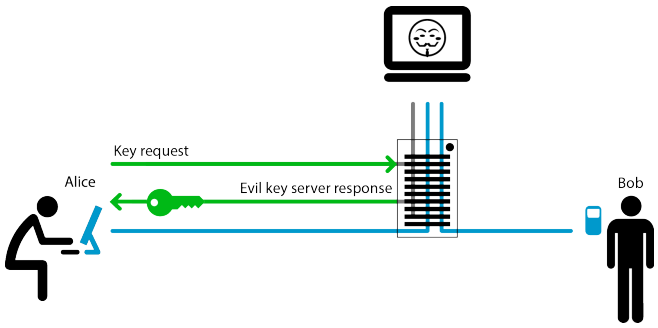

Figure 5: Server provides bad key (MITM)

One weakness to TOFU lies in the first message. If the server is lying for this first response, then there is no way to securely communicate with Bob. The server could read or modify the traffic at any time. This makes TOFU an acceptable choice only when dealing with opportunistic attackers who aren't already actively targeting you or your communications partner.

There is a further problem here. If Bob needs to change his key suddenly, there is no way for Alice to tell the difference between Bob and an attacker claiming to be Bob. This is a huge and essentially unsolvable problem for systems that have TOFU as their only key validation method.

5.2 Out-of-band validation

A better way to solve this problem is to find a way to validate the key so that Alice knows that the key received is the one actually used by Bob. The easiest way to accomplish this is to validate the keys out of band. This is usually done with either a hexadecimal string commonly called the "fingerprint", or a QR code which is a graphical representation of the same data. The fingerprint is not a secret value you want to protect (unlike your own, real print on your fingers)- instead, it is a public value you would want to print on your business cards, advertise on your website, or tattoo in a handy location[2]

However, this key exchange is only as secure as the communications method used. If the fingerprint was exchanged over the very chat you were attempting to authenticate, if you *were* being attacked, the attacker would simply replace the fingerprint with whatever data you expect to see. Fingerprints must be exchanged over a separate channel that you know (or at least hope!) has not been compromised. (Hence "out-of-band".) The best assurance is provided by physically visiting your communications partner and verifying keys/fingerprints - but other options potentially include a phone call, SMS, a video chat - anything where you can be reasonably sure the attacker is unable to modify the data of.

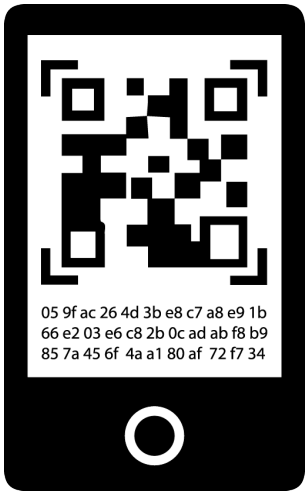

Figure 6: QR code and "fingerprint"

Consider also that some types of verification are harder to forge. A live videochat of Bob reading off his fingerprint to Alice is difficult to intercept and modify in real time with a faked video of Bob reading off the wrong fingerprint. An SMS from Bob with his fingerprint might be easier for some attackers to intercept and modify in transit, but if the validation was ad hoc and not part of some automatic validation scheme, it would probably require an adversary who was already prepared and waiting to jump into the communications between two specific individuals. This adversary would also need access to the phone network or otherwise be able to modify or forge SMS messages in transit.

5.3 Transitive trust

Another method to determine if the keys you're receiving are legitimate is to ask someone else. Consider a case where Alice wants to talk to Bob, but she can't validate Bob's key. However, Alice trusts Carol, and Carol trusts Dave. Dave has Bob's key and has validated it. Now Alice can have some assurance that the key she is using for Bob is a legitimate key.

This strategy has a few names that imply a few variations.

5.3.1 Web of Trust

The most popular term since the 1990s has been "web of trust". The web of trust implies a public database where everyone can see everyone else's "connections" - who trusts whom. These connections may not be symmetric - just because Alice trusts Carol, doesn't mean carol trusts Alice.

[2] Okay, maybe not tattoo it, that makes changing your key even harder!

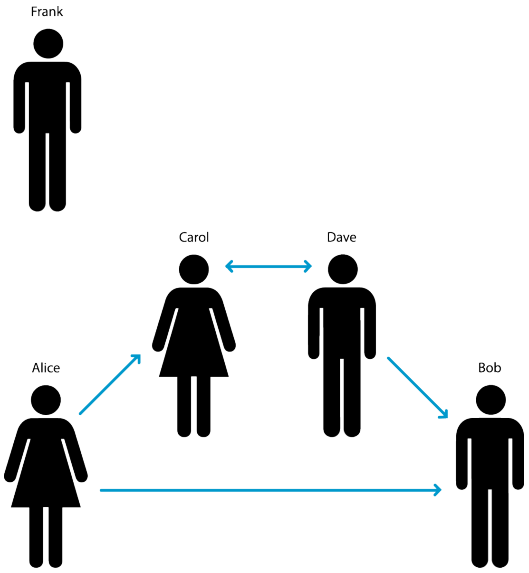

Figure 7: The Web of Trust

There are many subtle problems with the web of trust. The biggest one for the purposes of this paper is that a new user is an island (as shown above with Frank). Poor Frank trusts no one, and no one trusts Frank. If Frank were real-life friends with Dave, an attacker would easily be able to impersonate Dave - because Frank has no way to trust the Dave that is in the graph. Frank could look at Dave and see he is connected to mutual friends - but Frank can't validate *those people either!* Someone could have replaced Frank's entire social circle – and Frank would have no idea. This sounds far-fetched, but it's generally easy to create spam accounts, and this attack has been demonstrated before.

Another variant of this attack is a mutual friend of the group, Gary, who has never signed up for the service because he's staunchly opposed to the idea of being near anything that transmits like a radio. (He also lives in a cabin in the forest, removed his car stereo, and uses his own waste as manure, calling it "humanure", which is why his friends don't visit him much.) If one day "Gary" signed up for the service - the social circle wouldn't find out it wasn't the real "Gary" until they talked to him in person!

So, even for the Web of Trust, the first verification done must be another type: either TOFU or Out-Of-Band. And this is required whenever social circles do not touch. If you communicate with a lot of different groups of people: this could be very often.

Other attacks focus on the transitive nature of the web of trust. Everyone in the web must have the same standards for how they "trust" someone. If Carol did a poor job of verifying Dave's key, Alice's message to Bob could be compromised. The proliferation of social networks, and the easiness in which we may accept friends on them, is a great example of how "Trust" may become "An attractive person messaged me". An adversary on the web might do this intentionally, causing anyone who chains trust in this way to receive keys belonging to that adversary instead of the real ones.

Additionally, the public nature of the web of trust means that it is possible for anyone to see with whom someone communicates. That is a large leak of the social graph and metadata.

5.3.2 Trusted Introductions

An alternative to the public web of trust are private, trusted introductions. These take the form of someone you already trust (in the diagram above, Carol could introduce Dave to Alice.) This seems similar to the web of trust - but it's different because Alice has no way to learn that Dave knows Carol (unless one of them tells her). Once Alice learns this, she can ask Carol to perform an introduction. The introduction is a private message no one else sees.

Once introduced, Alice and Dave can communicate independently of Carol and introduce each other to more people.

5.4 Validation is forever until it isn't

It's worthwhile to note that in a secure chat solution, key validation is only required *once*. The exception to that is if your communications partners lose their keys, (For example, getting a new device or factory resetting an existing one.) their fingerprint will change - and validating their new key is required just as before. Once the new key is validated, the old key should not longer be accepted by the application - you wouldn't want someone having their phone stolen, and the thief being able to impersonate them in an encrypted, authenticated chat.

6 Group messaging

The above scenarios have assumed communications that are intended only for two people. Some other issues arise when attempting to secure multiparty chats. The rest of the examples in this section assume some sort of end-to-end encryption between all of the participants in the conversation, where each user is obligated to send the same message to all users in the chat.

6.1 Ice cream attack

When a group of people are in a room, it's obvious to see who is talking, and you can avoid interrupting people (or do it on purpose.) But when several people are on a phone call, there is that slight delay between when you start talking, and when you hear someone else start talking. This delay is magnified when users are typing on keyboard, especially thumb-typing on small keyboards. It may seem like a small problem, but ordering messages is anything but.

The ice cream attack becomes even more powerful if the attacker is part of the chat, and purposely delays some other users' messages to change the context of what they say!

Figure 8: Poor Dave, he loves ice cream so much, but he types so slow!

6.2 Membership changes

Another problem with group chats – especially when you take into account the problem of message ordering – is that it's difficult to remove members. Suppose the four friends have had enough of Bob's fanaticism and want to remove him from their ice cream chat.

How do they remove him? There aren't many good options:

1. Any member can arbitrarily remove any other member from a group chat. This works well for small circles of friends, but completely breaks for random groups of people on the Internet
2. It requires a majority vote. This works horribly in a text messaging scenario where people take a long time to reply or vote. Alice decides to call a vote to kick Bob... but any discussion about doing so happens in front of Bob - because he's not kicked yet! And it requires a majority to vote
3. A majority vote with a time limit? It's not clear this is a particularly usable solution either.
4. There is the concept of a "moderator" who has the authority - First off, there's the social aspect of "Which of your friends would you both trust and respect enough to moderate your circle of friends?" But there's also the very real problem that the moderator has to be online and responsive or you have the same delay problems.

6.3 Group chat protocols

Overall the encryption protocols for group chat systems are not as well understood as those for only two parties, so there could be other issues lurking beneath the surface here that we don't know about yet. This means that if your security needs are particularly acute, you probably should avoid group chat.

7 Open source

Many commentators on the topic of secure messaging clients will tell you that only open source programs are safe. It is important to understand what exactly open source software contributes to the security posture of a secure messaging program

7.1 Security audits

Errors in application design, coding, or cryptography can undermine the security of a secure messaging system. In the past, a commonly held belief in some circles was that open source software would be more secure as a result of its openness to be examined by anyone. The past couple years have shown that the availability of source code to the public is not in itself enough to produce a secure result. The discovery of complex bugs requires a concerted, methodical approach by talented individuals that simply cannot be applied en masse to all open source software. But, open source software significantly lowers the barrier to both having an audit conducted, and verifying that developers take security concerns seriously.

Generally speaking, you should use messaging systems that have been audited by experts and where the problems uncovered during those audits were fixed in a timely manner. If a crypto vendor claims that an audit was done, but they don't show you the results (hopefully both the findings and the fixes), you're pretty much taking them at their word that the system is safe for use.

7.2 Does it do what it says?

A better argument for open source secure messaging systems is that an examination of the source is the best way for the public to determine if a program is actually providing the protections it claims. With a closed source system, the public can only take it on faith, based on how much they trust the creators of the software.

That being said, most members of the public are not able to do this analysis themselves. Instead, some well-regarded members of the community might do some analysis of a particular program and then provide some commentary about the security guarantees it provides.

7.3 Did we get the program we analyzed? (reproducible builds)

Unless you are going to manually inspect and build every program you use, you will have to at some point take someone else's word that the programs that you use do what they say they do – and this applies to open source software as much as closed source. If someone you trust analyzes the source of a program, how do you know that you will receive the same thing that was analyzed? For closed source software, even if there was an audit conducted by someone you trust, there is no guarantee that what you actually download is what was audited.

For some threat models, one should assume that software delivery channels (Download sites (even over HTTPS), App Stores, etc.) are potentially hostile. If your threat model includes these kinds of attackers, you cannot trust any software provided over any of these channels, even if the application is open source and/or audited by a trusted party.

On systems where it is easy to build your own software, you could hope that the people you trust who did the audit provided some sort of signature over the code that was audited. Then, when you download the code yourself, you could verify that the signature matches, and you could be reasonably certain that the software you build is the same as the one that was analyzed.

But for people who have things to do, places to see, and sweet, sweet cinnamon buns to eat at every opportunity - building every piece of software I want to run on my system is out of reach. So how can I, the cinnamon bun-loving public, know that a program I download was produced from the source code that was audited? Unfortunately, this is a very difficult problem currently. There are a few projects that have achieved

the holy grail of "reproducible builds" - but this is only possible on certain platforms (like desktop Operating Systems and Android).

A trusted auditor could review the code, provide commentary, and then build the code and provide the fingerprints for that build. A typical user could then use a verification program to ensure that a downloaded binary actually corresponds to the code that was audited.

It's not likely that this will ever be usable for regular users for platforms like iOS, where only programs from the App Store are allowed and binaries are modified by Apple before being delivered to a user[3]

7.4 Operating systems and open source

To continue down the rabbit hole, an application can only be as secure as the operating system it runs on. Some threat models would include an attacker that could control operating system vendors. Most desktop and laptop users are running closed-source operating systems. Mainstream mobile operating systems contain some open-source components, but enough closed-source functionality exists in enough critical areas to render the distinction meaningless for the purposes of this discussion.

We're still awaiting the year of Linux on the desktop. Even if we had it, most users aren't able to build an OS from scratch anyway, so a subverted distribution channel would still be an issue.

7.5 How many operating systems are on your device? And what do they run on?

And if we're going to look into the rabbit hole, we may as well look in all the parts of the burrow and see just how much is down here. (I've always suspected rabbits connect their burrows into a thriving rabbit-tropolis under my yard.)

Modern computing equipment actually includes many CPUs or microcontrollers, in some cases running multiple operating systems. Desktops and Laptops have dedicated microcontrollers in UEFI/BIOS, disk drives and many peripherals. Many phones have a separate secured mode (TrustZone) running concurrently with the main operating system. A System-on-a-Chip (SOC) used in a phone might have multiple discrete CPUs. A phone might also have one or more baseband processors used to handle network communications. Often the interconnections between these various operating systems and components are arcane, if not completely opaque to the user, and none of them are likely to be open source in the near future.

Because of all of these interconnected pieces, all of which are closed-source, and at least some of which are connected/updatable from a network interface (often without user knowledge/interaction), even a fully open and audited "main" operating system doesn't provide a significant security hurdle in the face of some types of dedicated attackers.

Even if we handwave away all of the above and assume that all of the operating systems and firmware on a device are open and secure, various types of hardware design backdoors or flaws are available to highly motivated attackers. Some progress is being made in providing fully open hardware designs, which is a great step forward. However, revelations in the past couple years show that even if we assume hardware and software that were designed and built securely, some attackers can add or modify hardware in a system to provide access.

After taking all of the above into account, it should become clear why open source is not a security panacea.

[3] Some subset of this would be possible for jailbroken users.

8 Metadata

Metadata is information about communications that isn't the actual content of the messages themselves. This could include the times and dates, lengths, and parties involved in a communication. Often metadata can reveal as much information, if not more, than the message content. As one pretty important man once said: "We kill people based on metadata."[4]

8.1 Direct collection of metadata

In almost all cases, assume that the vendor of the chat application can collect metadata directly. End-to-end encryption can protect message content but cannot prevent a provider from collecting metadata. A highly motivated attacker might attempt to gain access to a messaging provider's networks to collect metadata (either about specific individuals, or in bulk). A provider serious about protecting privacy will keep as few communications logs as possible and might consider obfuscating the information. A provider with commercial interests such as analytics or targeted advertising is likely to retain metadata information, which might make it more accessible to other attackers.

Even if the server runs open-source software, there is still no real way to verify that the server is not collecting metadata.

8.2 Inferring metadata as a global passive adversary

Attackers with access to a significant portion of the network infrastructure between two users and/or the messaging server might be able to infer metadata by watching for patterns in timing or message size.

This becomes more difficult if the messaging server and/or client attempt to obfuscate communications by delaying messages or sending dummy messages, but these techniques might be defeatable by statistical techniques.

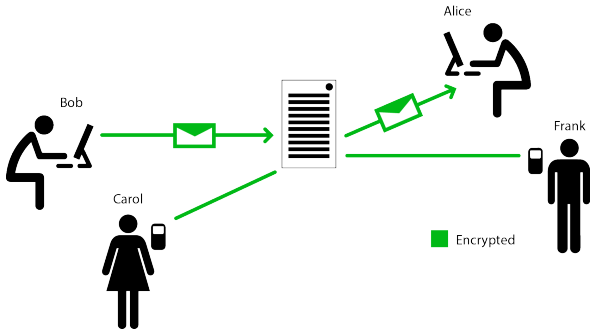

Figure 9: Even when encrypted, timing and size can match senders and receivers

8.3 Identifiers as metadata

The chat server will be able to identify your public IP address under most circumstances. This means that even if your provider doesn't have your real name, they might be able to link your messages to a particular address in the real world. If an attacker has access to IP logs from the chat server and your Internet provider, the attacker might be able to match these to lead to your identity.

Some chat systems require you to register and communicate using a valid phone number. These would require you to take extra steps to mask your identity as compared to a system that allows arbitrary usernames. A collection of phone numbers is likely to be more useful to attackers than a list of arbitrary usernames.

8.4 Address book harvesting

Many chat systems will attempt to upload your address book from your phone to the server. This allows the collection of metadata that doesn't include call information, but instead provides the ability to create a graph of "who knows who".

Some chat programs will attempt to obfuscate this in various ways before sending to the server (hashes and

[4]Retired General Michael Hayden - https://youtu.be/UdQiz0Vavmc?t=27s

bloom filters come to mind), but these techniques are not effective at scale. At the end of the day, you must assume that if the chat service provides any sort of matchmaking by contact list (phone numbers, email address, etc.), the chat server can read your social graph.

8.5 Is there any cure for metadata?

All of these explanations may make it seem hopeless, and that one cannot protect oneself from giving away metadata to anyone. That's not the case – tools like Tor (and to a lesser extent VPNs) do effectively hide a lot of metadata. Integrating something like this into an application can go a long way to disguise who's talking to who. But like group messaging any claims should be viewed very suspiciously.

In particular, Tor does not claim to address a "Global Passive Adversary". This is generally understood to mean an eavesdropper who can effectively watch the whole internet at once. Adversaries who can watch both ends of a Tor connection might also be able to infer metadata without a "global" reach.

9 Physical device access/seizure

For some threat models, it makes sense to consider a case where your adversary is able to access your device physically. The security of your device (especially full disk/hardware backed encryption and a strong passcode) plays a big role here, but is mostly out of scope for this discussion. For the rest of this section, we assume that your adversary was able to bypass any existing OS or hardware encryption so we can focus on the features of the applications. [5]

9.1 Logs & transcripts

It should go without saying, but a system that saves logs or transcripts of old messages risks exposure of either metadata or message content if an adversary is able to access that data. The chat transcripts are easy to see by just scrolling up (or the equivalent) - the logs would be hidden from view.

To be secure, messaging systems should offer the ability to erase message content, metadata, and/or internal contact lists. [6]

Some systems will provide options to not log at all, or to automatically remove records after a certain period of time (but see "Auto-deleting messages" below for some limitations of this strategy). This is helpful in limiting the amount of data available if your device is seized before it can be wiped.

9.2 Forward secrecy

Imagine an adversary who doesn't have the ability to decrypt your traffic, but records it all anyway in the hope that later decryption will be possible. Some time later, your device (or the other encryption endpoint) gets taken by this attacker.

For some types of encryption, the attacker now has control of the keys used to encrypt all of the recorded traffic, so the attacker can now read all of it. However, a combination of a few cryptographic techniques can make it such that the recorded network data will still be unreadable even if the attacker steals your device. This property is called Forward Secrecy (FS)[7]

[5] Much of the discussion below will also apply if an attacker uses some sort of exploit to gain logical access to your device as a privileged user.

[6] This can be challenging on flash storage systems, as their controllers often write data in unpredictable or opaque ways - but the application can at least attempt to delete the data.

[7] And may also go by the name 'Perfect Forward Secrecy'.

10 Not as Advertised

Not everything in the secure messaging space works like it says on the tin. Some of these are negligently misleading at best, whereas others simply have some subtle details that aren't immediately obvious.

10.1 Auto-deleting messages

Many chat programs have a setting that deletes a message for both parties after some automatic or user-selected countdown value. This can be useful to help keep message history to a minimum, but it relies on the cooperation of your communication partners. If the person receiving the message chooses to save the message beyond the purported lifetime of the message, there are a variety of techniques available, and there is nothing that the sender can do to prevent this (or even to reliably detect it). For an obvious example, just imagine that the receiver has a second camera and takes a photo of the screen whenever one of these messages comes in.

10.2 One time pads

One time pad cryptography (also called the Vernam Cipher) is provably secure from cryptanalysis when used correctly.

Unfortunately, it's nearly impossible to be used correctly in a meaningful way. These types of ciphers require large amounts of truly random input and a secure way to synchronize this random data between the communication partners.

If you see a modern secure messaging application that claims security by virtue of its one time pad implementation, it's likely that the system is either not really secure (for example, it uses pseudorandom data generated from a shared key, or it transmits the pad material over some other less-secure medium), or not practical for normal use.

10.3 Special crypto hardware

A few devices are available that treat the phone (or other device) as an untrusted network device and provide their own encryption via some attached hardware. Leaving aside any security vulnerabilities that might allow a hostile device to attack the crypto hardware directly, a hostile phone might simply elect to turn on its own microphone to listen to the conversation before it is encrypted.

10.4 Geofencing

Some messaging applications claim to only send messages to devices within a certain geographic area. In almost all cases (see mesh networks below for a counterexample), this all happens on the server based on a receiver's reported location. It is relatively easy to purposely misreport a location to allow one to read or send data that one would not normally be allowed to.

10.5 Mesh networks

Mesh networks have a variety of technical advantages and constraints, but in terms of security they are no different than any other messaging system. There is no special magic that makes them more secure other than the fact that you must actually have a receiver able to receive/transmit to a member of the mesh.

These deserve special mention because they have been used during protests where traditional methods (SMS, Voice, and mobile data) were unavailable (or perhaps intentionally deactivated by the authorities). There is nothing intrinsic to mesh networks that would make them safe. The authorities (or other adversaries) could connect to them and log message data (if unencrypted) or metadata (usernames or device identifiers).

10.6 Military-grade

This essentially means that a particular set of encryption algorithms were used, but doesn't address any of the topics in this paper. Claiming "military grade encryption" in marketing materials is something like claiming your car is safe because it has a bulletproof windshield.

10.7 Bespoke cryptography

The details are beyond the scope of this paper, but it's generally safe to assume that something using custom cryptographic algorithms or protocols is less secure than something using well-known ones. A special subset of this is that cryptography algorithms that are kept secret are usually very poorly tested and not likely to be secure.

10.8 Multiple synchronized devices

There are a variety of technical reasons why having multiple devices all synchronized to the same account can be difficult if the messages use end-to-end cryptography. Either one key is shared amongst multiple devices (which means that one stolen device can masquerade as all of the rest) or more complex cross-signing schemes are needed. Cross-signed schemes end up with more complex problems, like "If there are two devices, and one is stolen, which one can invalidate the other?". Alternatively, a server could manage the multiple identities, but this endangers the end-to-end and key verification properties of a system.

11 Conclusion

Hopefully we've illustrated what the different secure messaging clients can offer to help preserve privacy and also shown their limitations. No software application alone will be likely to provide effective protection against a government that is specifically targeting you, but the correct choice of application, used correctly, can have a major impact in keeping your data safe from prying eyes in all other cases.