TightBeam Communicator Architecture

Abstract

The application *TightBeam Communicator* uses QR codes to transmit secure connection information between two mobile devices. By using QR codes the possibility for an interception of the communication required to produce a secure connection is greatly reduced. By providing a mechanism for establishing a secure connection between two mobile devices, future communications can use this trust relationship to provide a high probability that future communications are not subject to a man-in-the-middle attack and that the identity of the message sender can be trusted.

1. Introduction

Most modern techniques for establishing a secure communication over the public Internet are subject to a number of attacks. TightBeam provides a level of security higher than those used by existing techniques. By using a unique method of sharing an initial secret, TightBeam can create a secure connection with a high level of trust. TightBeam does not use a centralized server for managing security; all keys are stored on the devices involved, to provide end-to-end encryption.

Other techniques for sharing keys such as Bluetooth or other close proximity radio communications are subject to eavesdropping. By using a QR code and the camera of a mobile device, the threat of eavesdropping is eliminated.

TightBeam does use a server for temporarily storing encrypted messages. Specifically, TightBeam uses Amazon's Simple Queue Service (SQS) to store these messages. Note however, that the messages are stored encrypted, and cannot be read without the key stored on the receiver's device.

Currently TightBeam is an iOS application. As discussed below, TightBeam uses a number of security features provided by the iOS platform to help secure your messages in the event of a stolen phone. On that note, remember to set a lock code on your iOS device, and make sure you are set up to remotely wipe your data.

2. Existing Techniques

There are of course many existing techniques for establishing a secure connection over a public or untrusted network. Each technique offers its own advantages and disadvantages. The advantage shared with all common techniques is the simplicity of sharing the secret required to encrypt a message. Most commonly the use of RSA (or another asymmetric encryption algorithm) is used in conjunction with a certificate authority.

2.1 TSL and SSL

TSL and SSL provide a technique for a client and server to establish a secure connection. A server publishing a public key to a certificate authority that is then used by the client to encrypt messages that only the server can decrypt. This allows the client to securely send a key the server, which in turn can be used to send encrypted messages back and forth. If however the certificate authority is compromised and a man-in-the-middle proxy is in place, messages sent between the client and the server will be compromised.

3. Managing Identity

For the majority of encrypted communications, TSL is a perfectly acceptable technique, especially if you and your organization have control of the certificate authority. However, being dependent on a third party for managing identity opens a significant attack surface. To avoid this risk, it is up the actors of any security protocol to manage the identity of the trusted actors. This presents a significant technical problem, which is why protocols like TSL exist for managing identity in a large system. For smaller organizations, the opportunity to use a non-distributed identity management system becomes plausible. By using a non-distributed identity system the risk of compromise is greatly reduced.

TightBeam offers a system for users to establish a secure connection with another TightBeam user, which provides a very high level of confidence that messages sent are secure.

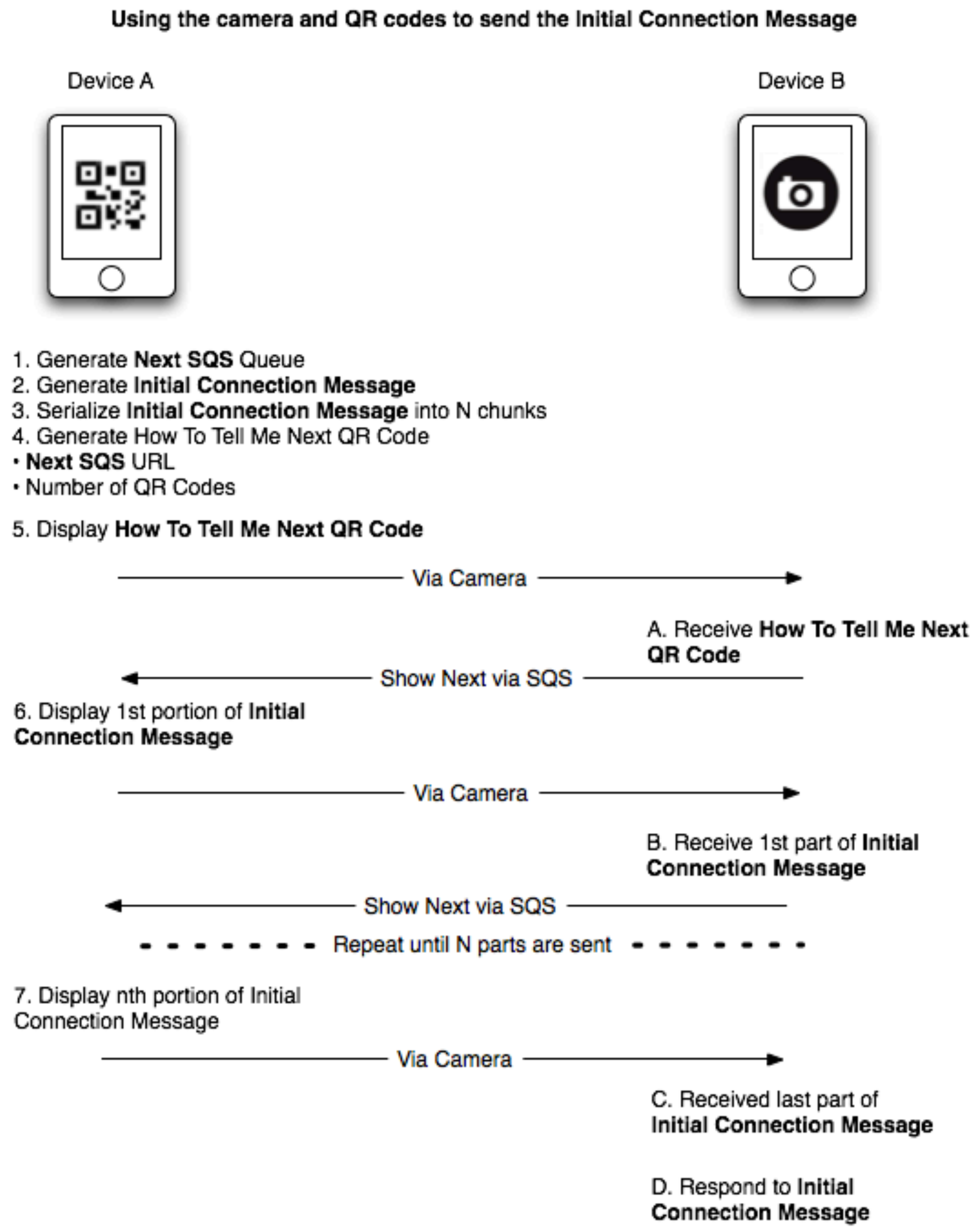
4. TightBeam Protocol

TightBeam provides a protocol that is used to establish a trusted, secure connection suitable for digital communication. There are several steps in this process as outline below.

The term public/private key is used below. However, please note that the public key is never distributed publicly. This document uses the familiar term public/private key to indicate the keys used in an asymmetric encryption algorithm. The “public” key mentioned below is considered as important of a secret, as the “private” key.

4.1 Passing the Initial Connection Message Via QR Code

TightBeam establishes a connection between two devices by first passing an Initial Connection Message. This message is passed from one device to another by using a series of QR codes that contain the Initial Connection Message. Figure 1 shows how this Initial Connection Message is passed from one device to another.


Using the camera and QR codes to send the Initial Connection Message
Device A
Device B
1. Generate Next SQS Queue
2. Generate Initial Connection Message
3. Serialize Initial Connection Message into N chunks
4. Generate How To Tell Me Next QR Code
• Next SQS URL
• Number of QR Codes
5. Display How To Tell Me Next QR Code
Via Camera
A. Receive How To Tell Me Next QR Code
Show Next via SQS
6. Display 1st portion of Initial Connection Message
Via Camera
B. Receive 1st part of Initial Connection Message
Show Next via SQS
Repeat until N parts are sent
7. Display nth portion of Initial Connection Message
Via Camera
C. Received last part of Initial Connection Message
D. Respond to Initial Connection Message
Figure 1 – Initial Connection Message

In Figure 1, we see two device, Device A and Device B. Each device adopts one of two roles. The first role is that of the QR code generator, and the other role is the QR code reader. In this example, Device A will produce the QR codes that encapsulate the Initial Connection Message, while Device B will scan the QR Codes and reassemble the Initial Connection Message.

The process starts with Device A creating an SQS Queue that will be used by Device B to send a Show Next QR Code message. These messages are unencrypted, though over SSL, and only contain the message, “show me QR code X”, where X is the index of the series of QR Codes that represent the Initial Connection Message. The contents of the Initial Connection Message will be covered in the next section.

Once Device A has created the Initial Connection Message, it is serialized and broken into a number of chunks. Each chunk will be one QR code in the series. Since multiple QR codes are required to communicate an Initial Connection Message with a large key, Device A needs to tell Device B how to ask for another QR code once a QR code is read. This is accomplished by Device A by creating an initial QR code that contains the URL of an SQS Queue created especially for this communication and the total number of QR codes that will make up the entire message, including the first QR code.

Once the initial QR code is created it is presented to Device B, which reads the code, finds the SQS queue used for communication and sends a Show Next message, containing the index of 1.

Device A receives this message and shows the first QR code that represents the first chunk of the Initial Connection Message. Device B, then reads the first chunk and proceeds to ask for the second QR code in the series. This process is continued until Device B reads all of the chunks of the Initial Connection Message. At which point, the Initial Connection Message is used to establish a secure connection between Device B and Device A, as explained in the following section.

4.2 Using the Initial Connection Message to Establish a Secure Connection

Once Device B receives the Initial Connection Message, it can be used to establish a secure connection between the two devices, as show in Figure 2.

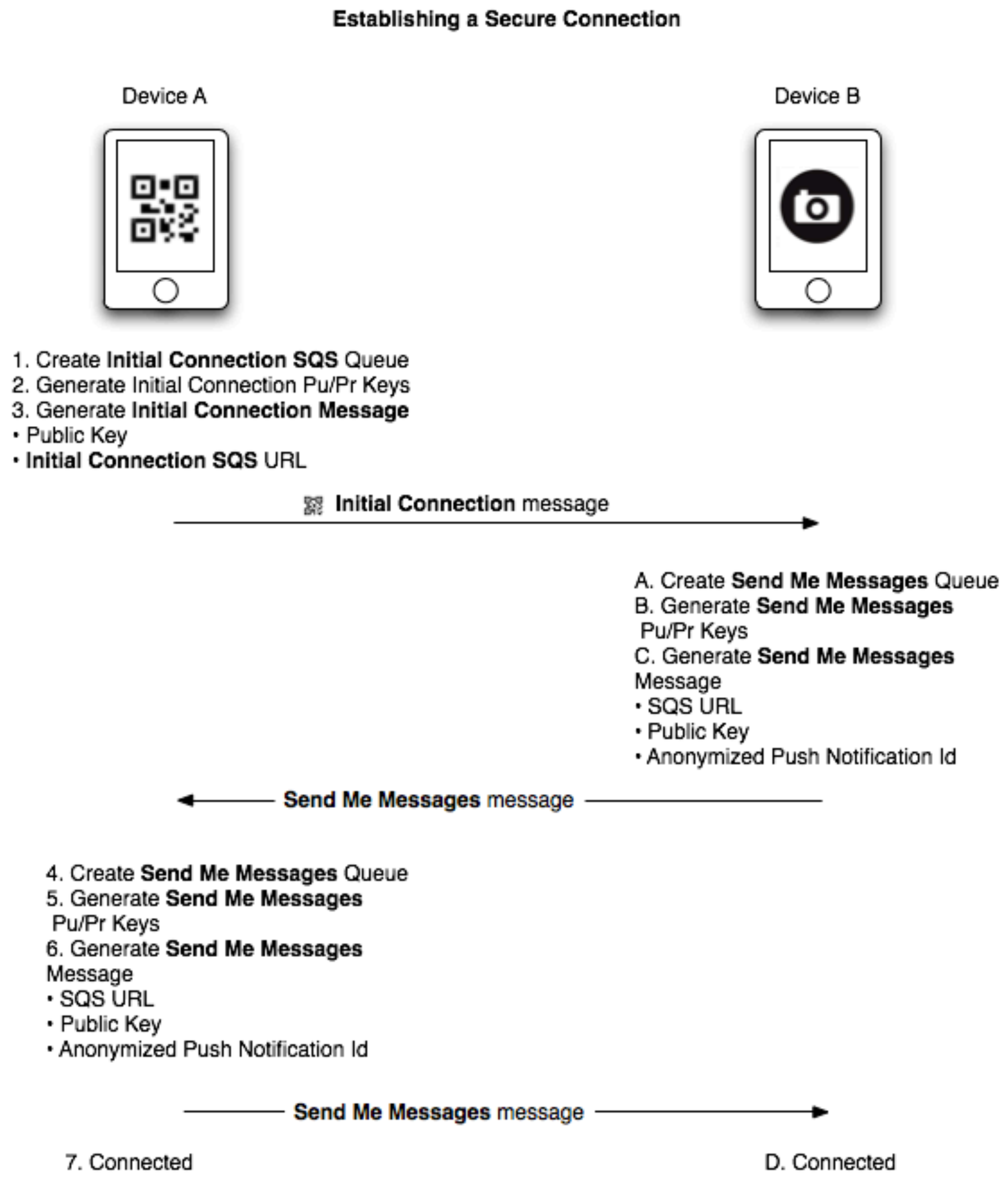

Establishing a Secure Connection
Device A
Device B
1. Create Initial Connection SQS Queue
2. Generate Initial Connection Pu/Pr Keys
3. Generate Initial Connection Message
• Public Key
• Initial Connection SQS URL
Initial Connection message
A. Create Send Me Messages Queue
B. Generate Send Me Messages Pu/Pr Keys
C. Generate Send Me Messages Message
• SQS URL
• Public Key
• Anonymized Push Notification Id
Send Me Messages message
4. Create Send Me Messages Queue
5. Generate Send Me Messages Pu/Pr Keys
6. Generate Send Me Messages Message
• SQS URL
• Public Key
• Anonymized Push Notification Id
Send Me Messages message
7. Connected
D. Connected

Figure 2. Establishing a Secure Connection

In Figure 2, we see the process of establishing a secure connection between two devices, A and B. The process is kicked of by Device A by sending the Initial Connection Message to device B via QR codes. To generate the content of the Initial Connection Message a new SQS queue is created for one time use. Also, a Public/Private key pair is generated for use by Device B to respond to Device A. The URL for the SQS queue and the public key make up the content of the Initial Connection Message. This public/private key pair is used only once and is destroyed after it is used.

Once Device B receives the Initial Connection Message, it generates a Send Me Messages message. The Send Me Messages message is comprised of a public key from a new public/private pair, and the URL of a new SQS queue. In addition, a random Id is generated which will be used to enable push notifications. The Send Me Messages message created by Device B is then sent to the SQS queue from the Initial Connection message, encrypted with the public key from the Initial Connection message. This message is then received by Device A and the contents are recorded. Device A then generates its own Send Me Messages message and sends it to the SQS queue from Device B's Send Me Messages message. This second Send Me Messages message is encrypted with the public key from the Device B's Send Me Messages message. Once this second Send Me Messages message is received by Device B, both device have all of the information they need to communicate securely.

The following section looks at the details of how future messages are sent securely between two devices.

4.3 Sending Secure Messages

After the initial connection process, the two devices involved now have all of the required information to send secure messages, as shown in Figure 3.

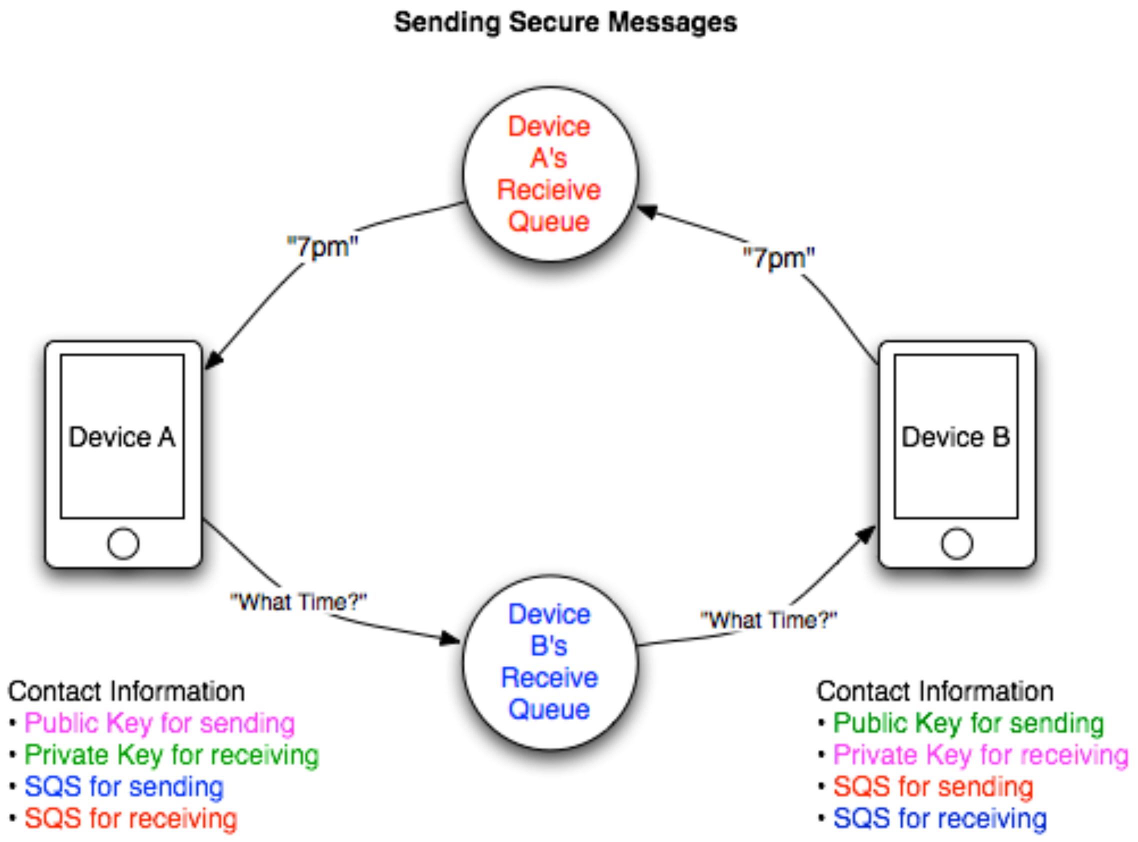


Figure 3. Sending Secure Messages

In Figure 3, we see the artifacts involved in sending secure messages. The items listed under "Contact Information" are the pieces of data exchanged by the two devices during the process of establishing a secure connection. The color codes indicate related data.

Device A’s purple public key is the public key for Device B’s purple private key. Similarly, Device B’s green public key is the private key that goes with Device A’s green private key. Also note that Device A’s receive queue (red, top) is known to Device B as the SQS for sending. Likewise, Device B’s receive queue is known to Device A as the SQS for sending.

When Device A sends the message “What Time?” to Device B, it first encrypts the message with its purple private key. The encrypted message is then sent to the blue queue, known to Device A as the SQS for sending. Device B checks the same blue queue (SQS for receiving) and decrypts the message with its purple private key. Device B sends the response “7pm” to Device A by encrypting the message with the green public key and posting it to the red queue known to Device B as SQS for sending. Lastly, Device A checks the red SQS for receiving, pulls the message down and decrypts it with its green private key.

In this way each device knows enough information to encrypt and send a message to the other device.

4.4 Notifications

On mobile devices it is typical to receive OS level notifications when messages from any application are received. TightBeam is no different, the utility of a messaging app needs to include this feature. However, the way iOS implements notifications leaves a little to be desired in terms of security. In the naïve approach to using push notifications, it would be required to send the message to a server, decrypt it and then post the message to Apple. Also, since each device generates a Device Token used to identify which device should receive a notification. This Device Token must not be sent to another iOS device, because doing so would allow Apple to track who is sending messages to who, though Apple would not be able to read the messages.

To address the first issue, TightBeam does not send the message to a device as a notification. TightBeam simply sends a simple generic message “You have a new secure message.” when a message is sent. Second, TightBeam keeps a map of Device Tokens to unique Notification Ids. This mapping allows each Device to trigger a notification on a target device without knowing the target device’s Device Token.

5 iOS Implementation Details

iOS provides a number of unique security features that TightBeam takes advantage of. The following is a description of each of these features, and how TightBeam is configured in regard to these features.

5.1 App Sandbox

All iOS applications are installed in their own sandbox. This can typically be thought of as a unique UNIX user for each app. This means that one app cannot access the files of another, each apps files and memory is separate from every other. This prevents other apps installed in iOS from being able to read messages or keys used by TightBeam.

5.2 Keychain

iOS provides a Keychain for securely storing small amounts of data, including encryption keys. TightBeam uses the Keychain to store the public and private keys used to encrypt and decrypt messages. However, these keys are stored with the flag `kSecAttrAccessibleAlwaysThisDeviceOnly`, which indicates that the keys should not be backed up. The upside to this is that keys will not be backed up, so they will not be exposed to potential hackers. The downside is that your connection to your secure contacts will be lost if your iPhone is broken, or needs to be restored from a backup.

5.3 Core Data

TightBeam uses Core Data to store connection information and messages. Core Data is configured with `NSFileProtectionComplete` to protect your messages. The implications of this are that TightBeam can only function while in the foreground. However, this adds an additional level of security as the Core Data database is encrypted while not in use.

6 Potential Attacks

No security system is perfect, and TightBeam is no different. The following is a description of possible attacks against TightBeam and what steps have been taken to prevent them.

6.1 Advances In Math

TightBeam uses the standard iOS encryption library to encrypt messages sent from one device to another. The assumption by Apple and all people who use this library is that these encryption algorithms are very hard to break. However, this assumption is based in unknown mathematical principles. It is entirely possible that these algorithms might be found to be simple to crack. Future version of TightBeam will use the most advance encryption technique known, however, if a math genius breaks these algorithms, there is not a lot we can do. But fear not, the social chaos that would ensue as every bank and trading company comes to its knees, might make your messages not all that interesting.

6.2 Interception of the Initial Connection Message

TightBeam assumes that the Initial Connection Message is securely sent between devices. To help ensure this security, the use of QR codes and the camera is used to transmit this message. To intercept this message, a video camera would need to be setup capable of recording the Initial Connection Message QR codes. If this message was intercepted, a man-in-the-middle attack is possible. To help avoid this, the information in the Initial Connection Message is used only once. A potential man-in-the-middle attack would have to intercept the first Send Me Messages message and install its own before the intended device reads the message.

Again, using the camera and not any sort of radio to communicate the Initial Connection Message mitigates this attack. Eavesdroppers easily record any sort of radio communication, such as Bluetooth. If you are very concerned about this attack vector, exchange the Initial Communication Message in a bathroom or under your coat.

6.3 Server Credentials Compromised

TightBeam relies on SQS for sending messages. This implies an Amazon Services account that controls these resources. If this account was compromised, an attacker could delete queues, or delete messages in queues, which would disrupt communications. However, the content of the messages, and who is sending or receiving them would not be compromised.

6.4 Lost Device

At some point a user of TightBeam will lose their device and an attacker will find it. If the user fails to set a strong passcode and the user fails to remotely wipe their device, it is possible for the attacker to send messages to the user secure contacts. This is an unfortunate event, however the damage is limited to the lost device and those contacts. Other contacts of the exposed contacts are not affected. The damage is limited to the cell of contacts available on the lost device. If you suspect one of you contacts lost their device, simply delete that contact in TightBeam. Further, setting your messages to auto-delete will help establish deniability.

7 Conclusion

TightBeam communicator uses a unique method for establishing trust between iOS devices. This method allows a user to trust that nobody can read the messages sent to another user, while at the same time, allowing a user to believe that a given message was sent by who they think sent it. Additionally, TightBeam goes through some hoops to help insure that the connection between devices also hard to track, by obfuscating the relationship between devices using push notifications and using randomly generated ids for queue names, notification ids, and key tags within Keychain.