iOS Application to Facilitate One-Time Pad Key Generation and Encryption

Spencer Atkin

18 May 2018

Abstract

This research develops an iOS application that enables users to securely encrypt and decrypt messages using one-time pad (OTP) encryption. The key generation takes place by using the device's camera to collect sensor noise. In order to ensure that this noise can be used as a source of true random numbers, statistical randomness tests were performed on collected data samples. The encryption/decryption aspect of the application allow the user to enter plaintext messages or a ciphertext, and then encrypt or decrypt them using a key that is shared with a specified user.

1 Introduction

As more and more privacy abuses committed by large companies are revealed, it is becoming increasingly apparent that people need to be taking more steps to protect their personal information online. One of the most important things people do online is communicate with others. A major part of online communications are text messages sent using various services. There are currently many options for securing communications–including messages–sent over the internet using encryption, but many of the options leave something to be desired. The most secure methods are too cumbersome and technical for the layman to use. One example is using GNU Privacy Guard in the command line to manually encrypt and sign a message, then copy and paste the ciphertext and signature into a message or send them as files. The simplest options often make security compromises. For example, the cryptosystem used in Telegram, a popular messaging app, has been found to have security issues [1]. There is a great need for an application that will allow for extremely robust encryption to be performed relatively easily. This research focuses on the development of such an application. The motivation for this research comes from previous research on offensive cybersecurity and a desire to develop ways to defend against attackers.

This research consists of an iOS application to perform two main tasks: generate robust random numbers, and perform the encryption and decryption of messages. The theory behind such random number generation will be discussed, as well as the structure of the application and the methodology of its development.

2 Background

2.1 One-Time Pad

2.1.1 History

OTP encryption is a relatively old type of encryption that originally used pieces of paper shared between two people to distribute the key. All aspects of the cryptography were done on paper, including encryption and decryption of messages. The pads were truly random–generated one number at a time by dice rolls. Later, machines were made that could perform OTP encryption using reels of paper tape that contained the keys [3]. OTP encryption is mathematically unbreakable as long as several assumptions are true [4]:

- The key must be at least as long as the message being encrypted.
- The key must be truly random – not generated by a computer function, etc.

- The key and plaintext are calculated modulo 10 (digits), modulo 26 (letters), or modulo 2 (binary).
- Each key must only be used once, and the sender and recipient must both destroy their keys after use.
- There must be only two copies of the key: one held by the sender and one held by the recipient [3].

2.1.2 Pseudorandom Numbers

Pseudorandom numbers are numbers generated by pseudorandom number generators (PRNGs). A PRNG generates a sequence of numbers by feeding an initial seed value into a deterministic algorithm. The sequences are sufficiently close to truly random for most applications, but they are in fact completely determined by the initial value. If the same seed is used twice with the same PRNG, the exact same sequence of pseudorandom numbers will be generated.

There exist PRNGs that are satisfactory for use in cryptography called cryptographically secure PRNGs (CSPRNGs). They are functionally the same to regular PRNGs, but they have the requirement that an attacker that does not know the seed has a very small advantage in determining the PRNGs predictable sequence. CSPRNGs are used for things such as generating keys for AES, another type of encryption. For this and other similar uses, the difference between a CSPRNG and a true random number generator is negligible. The efficiency and ease of use benefits, however, are large.

2.1.3 Difficulties

The difficulty with performing OTP encryption on personal computing devices such as smartphones is the generation of random numbers. For the encryption to be mathematically unbreakable, a PRNG cannot be used. Even though CSPRNGs are satisfactory for other cryptography applications, they cannot be used to generate OTP keys. If there is any predictability in the numbers used for the key, the encryption is no longer unbreakable.

2.1.4 Key Generation

The solution to the problem of key generation is a true random number generator. This is a hardware device that uses some property of the physical world to generate random numbers. They will typically use microscopic processes that generate statistically random noise, such as electrical noise, or different quantum phenomena. One example of this is reversing a p-n junction in a circuit, which can cause quantum tunneling effects under certain conditions.

2.2 Message Integrity

2.2.1 Hashes

A hash is a one-way function. This means that, given an input, an output can be determined from the function. However, given a certain output it is nearly impossible to determine what input generated that output. This property makes them very useful for cryptography.

$$f(x) \Longrightarrow y$$

$$z \Longrightarrow f(?)$$

A cryptographic hash function is a special type of hash function. They are especially useful for cryptographic applications. Ideal cryptographic hash functions have the following properties:

- The same input always generates the same output
- Computing the function's output does not take a long time
- It is not feasible to generate the hash's input from its output except by trying all possible inputs
- It is not feasible to find two inputs that generate the same output

The current standard of cryptographic hash functions are in the SHA family. For example: SHA-1 and SHA-256. There are also older hash functions that have been determined to be cryptographically broken, such as MD5 [6].

2.2.2 HMAC

HMAC (hash-based message authentication code) is a way to verify the integrity of a message. The algorithm works by iterating over the message and key with a cryptographic hash function, and returning a message digest. This digest will be the same if the message and key are identical, but will change dramatically if any part of either the message or key is different. Because of the nature of cryptographic hash functions, it is not feasible to find a message that will produce the same digest as another message. Thus, it can be used to verify that a message has not changed in transit.

2.3 Bluetooth

Bluetooth is a wireless communications protocol used to transmit data between devices at a close range (typical range 10m). Bluetooth radios are included in many smartphones, including Apple, Samsung, and Google devices, as well as many other popular Android device manufacturers. In 2010, the Bluetooth protocol was upgraded to include Bluetooth Low-Energy (BLE). As its name implies, BLE allows for bluetooth communications with a low energy requirement, and it also allows for faster data transfer speeds [7].

2.4 Cameras

2.4.1 Image Sensors

Digital cameras create images by detecting photons with components called image sensors. Image sensors consist of grid arrays of photosensors, which are electrical components that are sensitive to light. When the digital camera makes an exposure to capture an image, each photosensor measures the number of photons that hit it during the exposure time. Each photon raises the voltage that is reported by the photosensor that it hits. The matrix of voltage levels from the photosensors is then used to generate the final image.

2.4.2 Bayer Filters

A plain photosensor is sensitive to all wavelengths of visible light, and therefore all colors. In order to capture a color image, each photosensor must know what color it is seeing. To do this, a Bayer filter is overlaid on the image sensor to filter colors. The filter is a grid with each row having two colors: either a row of blue and green or a row of green and red. An example of a section of a Bayer filter is below:

Figure 1: Section of a Bayer filter

B	G	B	G	...
G	R	G	R	...
B	G	B	G	...

As a result of this pattern, the filter is 50% green, 25% blue, and 25% red. This mimics the color sensitivity of the human eye. When the camera is exposed, each photosensor will only detect either red, green, or blue. After the exposure, the values of adjacent photosensors are interpolated to create a true color image.

2.4.3 Noise

Image sensors have multiple sources of random noise when images are captured [2][5]. Shot noise occurs during the image exposure process, and results from the particle nature of light. In a short period of time, each photosensor in an image sensor has a random probability of detecting a single photon. If the image is exposed for long enough, however, the random probabilities will no longer be significant – it is likely that each sensor will detect a photon. This is similar to a coin toss experiment: tossing a coin a very large number of times will result in about equal heads and tails results. But, given very few trials, one result (either heads or tails) will tend to dominate. The principle is the same for image sensor shot noise – If an image is exposed for a short amount of time, or in an environment with very low light, the random events will dominate.

Another source of noise in an image sensor is read noise. This noise develops during processing of the image after exposure. The sources are inherent in the image sensors circuitry, coming from the level of illumination of the sensor, and its temperature. The electronic circuits connected to the sensor also insert electronic noise. The read noise of an image sensor is Gaussian, and independent at each pixel.

2.5 iOS Development

2.5.1 Overview

3rd-party developers have been able to create applications for iOS devices since March 2008, 5 months after the release of the first iPhone. Originally, only web applications were going to be permitted on the platform, but a backlash caused Apple to release the native SDK to the public [8]. In order to have access to development resources and offer apps on the iOS App Store, developers must pay a yearly fee.

2.5.2 Technology

Apple provides developers with the Cocoa Touch framework, which is used to build iOS applications and provides an abstraction layer to the operating system. Some core frameworks included in Cocoa Touch enable developers to control the UI, send push notifications, access the camera, and access the device's GPS. iOS developers use the Mac IDE called Xcode to write software and compile it for running on test devices and submitting to the App Store. For the majority of iOS development history, Objective-C was the language used to write application code. In June 2014, Apple announced Swift, a new programming language for iOS development [9]. Apple claims Swift code is simpler to read, easier to write, and faster than Objective-C code. It is also more similar to other programming languages than Objective-C, so many beginning developers will likely find Swift easier to learn. As such, many new apps are being written in Swift. All native iOS framework code is still written in Objective-C, however.

3 Development

3.1 Overview

The product created through this research is a mobile application that runs on iOS devices (iPhone and iPad). Development was done on a Mac computer using the Xcode IDE. The Swift and Objective-C programming languages were used. Swift code makes up the bulk of the code base, and certain low-level data operations are written in Objective-C. A UI framework that makes it easier to write UI code was used.

3.2 Requirements

This research will be deemed a success if an application is developed that can perform all the required functions. Namely, the application must be able to:

- Generate random numbers that pass a statistical randomness test
- Connect two peers and generate a shared key between them
- Encrypt plaintext messages and allow the user to export the ciphertext
- Allow the user to import a ciphertext, and decrypt it to a readable plaintext message

The application should perform all these functions while not being exceedingly complex in its design or operation, and should require minimal instruction before a user is able to make use of all its features.

3.3 Research Implementation

The main theory behind this research is OTP encryption. The purpose of the application is to perform OTP key generation, and OTP encryption and decryption. To perform the key generation, the design of image sensors was harnessed. The principles of image sensor noise were used to generate the random numbers needed for the OTP key and because they were from a truly random source the encryption performed by the application can be considered to be mathematically unbreakable.

3.3.1 Randomness

The iPhone camera was used as a source of randomness to generate the OTP key. To do this, the RAW capture API functionality was used. This means that when a picture is taken, the raw image data can be handled before any processing is performed on it. Rather than being returned in the JPEG or PNG format, the image is simply a matrix of values representing the voltage level of each photosensor in the image sensor.

Each voltage value in the image data is represented as a 16-bit integer. To ensure that good data is being used, only the two least significant bits of each value are used. These bits are all concatenated to form one large random bit stream. At first, this stream was used for the key without any changes. However, when statistical tests were run on the stream, it was determined that the data tended to be biased towards 1 with approximately 49% 0s and 51% 1s. To fix this, the stream is processed before being used for the key according to a method devised by John von Neumann. The stream is analyzed in bit pairs, and the following rules are used: if the two bits are the same, they are discarded. Otherwise, the first digit of the pair is used. Fully:

$$0,0 \Longrightarrow \emptyset$$

$$1,1 \Longrightarrow \emptyset$$

$$0,1 \Longrightarrow 0$$

$$1,0 \Longrightarrow 1$$

The result of this method is a stream half as long as the stream before processing, with any bias towards a certain bit removed. The implementation of this bias removal algorithm is included below.

```
size_t removeBias(uint8_t *stream, size_t streamSize) {
    uint8_t *unbiased = malloc(streamSize);
    size_t count = 0;
    int bit = 0; // 0-7: bit positions in the current uint8

    uint8_t nextInt = 0;
    for (int x = 0; x < streamSize; x++) {
        uint8_t thisByte = stream[x];
        for (int shift = 0; shift < 4; shift++) {
            uint8_t masked = thisByte & 0x03;
            thisByte >>= 2;

            if (masked == 0x01) { // Nibble is 01
                bit++;
            } else if (masked == 0x02) { // Nibble is 10
                nextInt |= (0x01 << bit++);
            }

            if (bit == 8) {
                unbiased[count++] = nextInt;
```

```
                    nextInt = 0;
                    bit = 0;
                }
            }
        }
        memcpy(stream, unbiased, count);
        free(unbiased);
        return count;
}
```

For the camera to function properly as a source of randomness, the user will cover up the device's camera before capturing an image. This will ensure that a low amount of light is entering the sensor, and it will be impossible to determine sensor values based on what the user took a picture of. The camera's exposure time was lowered so that shot noise would be a source of randomness alongside the Gaussian read noise from the sensor.

3.3.2 Encryption/Decryption

The process of encryption and decryption are relatively simple, and they are fundamentally the same. In order to encrypt a plaintext or decrypt a ciphertext, a message of N bytes is aligned with the first N bytes of the key that the message is being encrypted/decrypted for. An XOR operation is then performed on the pair of byte sequences to achieve the result. The N key bytes used for this operation are then securely deleted from the device so that they cannot be recovered or used again. This step is performed for both encryption and decryption.

Due to the nature of the encryption, if one bit is flipped in the ciphertext, it will also be flipped in the plaintext. In this way, it is possible for an attacker to tamper with the message while it is in transit without having to read what the message says. Thus, a technique must be applied to ensure that the ciphertext has not been changed in any way after it was encrypted. To do this, an HMAC digest is computed on the message and sent along with the ciphertext. This digest is then computed again by the recipient and compared to the received digest to ensure that they match.

If encryption is being performed, the HMAC digest is simply computed and sent along with the ciphertext. If decryption is being performed, the HMAC digest is computed and then compared against the received digest. If they differ, the message was tampered with.

3.3.3 Testing

All testing of the application was performed continuously as the application was being developed. The researcher used two iPhone devices to run the application and test all features, including inter-device communications. No formal UI testing was performed, but the interface was designed to be simple and easy to comprehend. If tests were conducted, it is expected that the testers would have little problem discovering how the application's interface functions.

3.3.4 Limitations

There were no real limitations on the process of this research. All required devices were possessed by the researcher, including a Mac computer and multiple iPhone devices that were able to run the application for testing. The research was also completed within the required time constraints.

4 Discussion

This research has resulted in the development of a mobile application that is very easy to use, but also gives users access to very secure encryption. It is now possible for non technical users to protect their online communications in a way that was not previously possible. In addition, a method was developed for generating truly random numbers on an iOS device using the camera. This means that iOS applications now have the ability to to perform cryptography using random numbers that do not come from a PRNG.

During this research, much was learned about random number generators and their applications in cryptography. In addition, a large amount of research was done on image sensors and the way they function so that a method of random number generation could be developed. A deep understanding was developed of the sources of the noise that arises when an image is captured, as well as how raw image data is stored before it is processed.

To extend this research, the application's features could be extended to make for a more seamless user experience. For example, a chat feature could be built in to the application, which would allow users to send and receive encrypted messages inside the application instead of needing to copy and paste them into another application. In addition, although the application is simple to use, onboarding could be shown on the first app launch to ensure that no user will have any confusion about using the application.

References

[1] Jakobsen, Jakob & Orlandi, Claudio. (2015). On the CCA (in)security of MTProto. *Cryptology ePrint Archive, Report 2015/1177.* Retrieved 14 May 2018 from `https://eprint.iacr.org/2015/1177.pdf`

[2] Hong, S., & Liu, C. (2015). Sensor-Based Random Number Generator Seeding. *IEEE Access, 3,562-568.* `http://dx.doi.org/10.1109/access.2015.2432140`

[3] Rijmenants, D. (n.d.). One-time Pad. Retrieved 4 October 2017, from `http://users.telenet.be/d.rijmenants/en/onetimepad.htm`

[4] Wagner, N. (2017). *The Laws of Cryptography: The One-Time Pad.Cs.utsa.edu.* Retrieved 4 October 2017, from `http://www.cs.utsa.edu/~wagner/laws/pad.html`

[5] Wallace, K., Moran, K., Novak, E., Zhou, G., & Sun, K. (2016). Toward Sensor-Based Random Number Generation for Mobile and IoT Devices. *IEEE Internet Of Things Journal,* 3(6), 1189-1201. `http://dx.doi.org/10.1109/jiot.2016.2572638`

[6] Chad R, Dougherty (31 Dec 2008). Vulnerability Note VU#836068 MD5 vulnerable to collision attacks. *Vulnerability notes database.* CERT Carnegie Mellon University Software Engineering Institute. Retrieved 10 May 2018.

[7] (2016, June 10). Bluetooth 5 Promises Four times the Range, Twice the Speed of Bluetooth 4.0 LE Transmissions. Retrieved 8 May 2018, from `https://www.cnx-software.com/2016/06/10/bluetooth-5-promises-four-times-the-speed-twice-the-range-of-bluetooth-4-0-le-transmissions/`

[8] (2011, October 21). Jobs' original vision for the iPhone: No third-party native apps. Retrieved 8 May 2018, from `https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/`

[9] (2010, September 9). *Swift Has Reached 1.0.* Retrieved 8 May 2018, from `https://developer.apple.com/swift/blog/?id=14`