One Time Pad Encrypted Messaging System

by

Nicholas Zahabiun

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Degree in Computer Engineering with Distinction

Spring 2020

One Time Pad Encrypted Messaging System

by

Nicholas Zahabiun

Approved: Dr. Cotton _____
Chase Cotton, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: Dr. Yang _____
Chengmo Yang, Ph.D.
Committee member from the Department of Electrical and Computer Engineering

Approved: Dr. Cioaba _____
Sebastian Cioaba, Ph.D.
Committee member from the Board of Senior Thesis Readers

Approved: _____
Michael Chajes, Ph.D.
Chair of the University Committee on Student and Faculty Honors

ACKNOWLEDGMENTS

Firstly, I would like to thank Dr. Cotton for mentoring me. Dr. Cotton first introduced me to cybersecurity. He was also my professor for many of my cybersecurity classes. I am now pursuing my career in Computer Networking and Security because of him. Dr. Cotton is also my first reader for this thesis. He guided me through the development process of this paper. He also put up with all the questions I asked him over the years.

Secondly, I would like to thank Dr. Yang for giving me my first research job. This job showed me I wanted to conduct research for a living. Dr. Yang also gave me my second research job. The second job was related to cybersecurity. I designed and implemented a data acquisition system for a power monitoring side-channel attack. This job solidified my interest in Cyber Security by showing me the range of expertise involved in studying it.

Thirdly, I would like to thank all my professors and teachers who taught me the important knowledge I use today. I am very fortunate to have received such high-quality tutelage throughout my life. If it was not for the teachers around me who answered all my questions, I would not know the vital knowledge I have today. It is all my teachers and mentors who are the reason I am fortunate enough to attend graduate school after I graduate.

Lastly, I would like to thank my Mother. She helped me every step of the way, throughout my life. I would not be where I am today without her.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

One Time Pad encryption (OTP) is an unbreakable cipher under three specific circumstances. Firstly, the encryption Key must have a uniform distribution and be unpredictably random. Secondly, the Key used to encrypt data can never be used more than once. Lastly, the key must remain a secret. Under these conditions, OPT encryption is completely secure. Abiding by these conditions makes OTP not practical in many scenarios. Never reusing the Key means the size of the Key must equal the size of the cleartext. This puts a constraint on storage space making OTP encryption not useful for encrypting large amounts of data. Creating large amounts of cryptographically random data is difficult in low entropy systems.

This makes the cipher not useful for encrypting large amounts of data, effectively doubling the storage needed to hold cipher text vs the cleartext. Creating large amounts of cryptographically secure random data is difficult. This makes OTP encryption not practical for large scale encryption applications. However, for a small-scale application like a messaging system, OPT encryption is shown to be very practical. Only requiring a relatively manageable sized key to send many messages. This paper tests the relative practicality and security of the OTP cipher in a simple messaging system.

This messaging system should have Confidentiality, Integrity, and Availability. Confidentiality will be derived from the One Time Pad cipher. The OTP cipher will be used to encrypt messages. Unfortunately, the OTP cipher is inherently extremely malleable. This means there must be a way to ensure the message has not been modified in transit. Integrity is achieved by using a cryptographic hash function. This hash function is used by encrypting the clear text, hashing, then encrypting the

ciphertext and hash. This is to prevent a Non-Malleability-Known-Plaintext-Attack. Availability is achieved by hardening the OTP messaging system. It should be difficult for an attacker to critically foil the messaging system. This involves designing the messaging system to include message delivery acknowledgment functionality. This is achieved by having the sender send a message until the receiver has returned a specialized acknowledgment packet.

For the OTP messaging system to work keys must be exchanged, and managed. An In-Band key exchange would involve public-key cryptography. This would defeat the purpose of using an Informationally Secure cipher like the One Time Pad. The Out-of-band key exchange is conducted by encrypting the OTP key using an authenticated cipher, AES-GCM. The AES key is verbally exchanged from the sender to the receiver. The sender would create the keys for the messages it sends. These keys must be cryptographically random according to the rules of the OTP cipher. A secure deterministic random number generator (D-RNG) cannot be used, because then the cipher would only be as secure as the D-RNG's seed value. Therefore, an external hardware-based cryptographic RNG is used. The users would need to purchase this hardware based RNG, and they would need to securely sample it for data.

The OTP messaging system would need a system to keep track of what portions of the OTP key have been used. This involves having each messaging include an internal message state. The message state for the sender goes from "uncreated", "unacknowledged", to "acknowledged". "Uncreated" means the OTP keys portion corresponding to this message has not yet been used. "Unacknowledged" means the OTP key portion has been used, and the message has been sent however, the message has not been confirmed delivered. "Acknowledged" means the message has been sent

and confirmed delivered. The message state for the receiver goes from undelivered to delivered. When the receiver receives the message it first checks if the message is authentic. If the message is not authentic the message is deleted. If the message is authentic, and the message state is undelivered, then the message is forwarded to the user. If the message is authentic, and the message state is delivered, then the message is not forwarded to the user. Regardless an acknowledgment is always sent back to the sender. This protects against replay attacks, because the user would not experience replayed messages. The way the sender and receiver know what OTP key portions correspond to each message is with the unique message number assigned to each message.

This OTP messaging system uses specific cryptographic tools and has specific statistics. The OTP cipher is used to encrypt clear text messages. The SHA3-256 hash is used to authenticate messages. Python 3.7 is used to program the messaging system. Regular socket programming with TCP is used for the internet communication protocol. The Pickle library is used to save the messaging system's state. AES-GCM is used to encrypt the message state (the message state includes the OTP keys). True RNG v3 is the hardware-based cryptographic RNG used to generate the keys. Each message has a clear text of 256 Bytes with UTF-8-bit encoding. Each message sent uses a 576 Byte portion of the OTP key. One megabyte of the OTP key is good for sending 1736 messages.

Under scrutiny, the OTP cipher pales in comparison to other commonly used modern ciphers. This messaging service would be much easier to develop with modern cryptographic tools. Other ciphers do not deal with large key sizes. Keeping track of used key portions is simply unnecessary with other ciphers. The messaging system

would also be a lot more user friendly if Informational Security is sacrificed for Computational Security. Most users are unwilling to commit to Out-of-Ban key exchanges.

Furthermore, there is no practical difference between Informational Security and Computational Security. The One Time Pad can only provide Informational Security as its upside. It is possible to create cryptosystems with the OTP cipher. Unfortunately, the OTP cipher is simply not worth its difficulty. If there were computers with infinite computational power, then the OTP would be worth its downsides. However, modern commonly used ciphers provided more than enough security with the convenience of easy to implement libraries and functionality.

Chapter 1

INTRODUCTION TO ONE TIME PAD CIPHER

One Time Pad the Unbreakable Cipher

The One Time Pad (OTP) is mathematically unbreakable [4], [1] Pg. 35. Therefore, it provides perfect Informational Security. Even with infinite computational power, one can never break the OTP cipher. The only metric revealed by the ciphertext is the cleartext length. To achieve this security certain conditions must be met. The key must be Cryptographically random, the key can never be reused, and the key must remain a secret. These conditions might seem easy to meet, however historically small mistakes have resulted in big consequences.

What makes the One Time Pad cipher unbreakable is the fact that every possible key will result in every possible cleartext. Given a ciphertext of 8-bits, there are 256 possible 8-bit keys. Each possible key decrypts the ciphertext to a unique cleartext. Since the key is completely random each cleartext has an equal chance of being the true cleartext. A 4-bit example of this is shown in Figure 1 below.

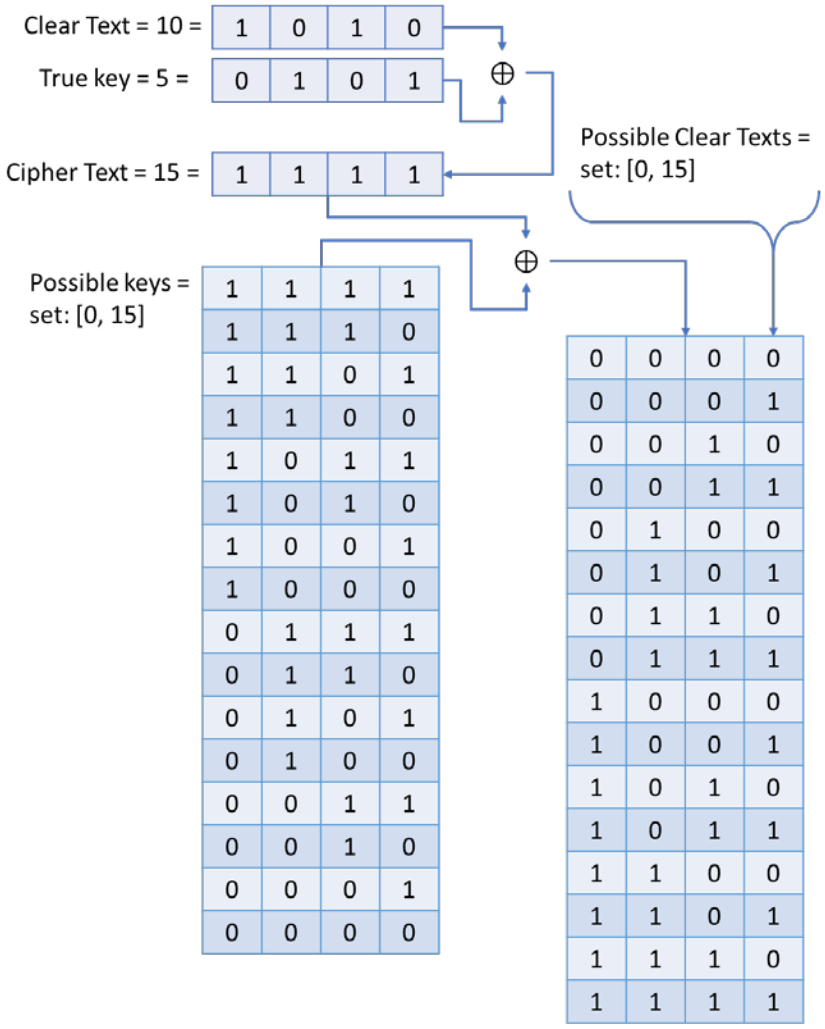


Figure 1 This figure shows how the One Time Pad cipher is Informationally Secure at the size of a nibble (4 bits). Given a 4-bit ciphertext, there are 16 possible 4-bit keys. Each of these 16 possible 4-bit keys creates a unique 4-bit clear text. This means each possible key creates all possible clear texts. Each key is equally likely to be the true key. Therefore, it is impossible to tell which clear text is the true clear text. Note the "⊕" symbol represents the "exclusive or" instruction.

The One Time Pad is a simple cipher to compute. The only logical operation needed is the exclusive or (xor). Cipher Text equals Clear Text xor Key. This simple

computation coupled with Informational Security makes the OTP seem very appealing when choosing ciphers to use insecure communication channels. Consequently, in the mid-1900s militaries and governments often used the OTP [5]. This history teaches us the mistakes of the past, so we do not make them in the future.

When the Key used in the OTP is not cryptographically random the ciphertexts become vulnerable to frequency analysis and other forms of cryptanalysis. If the key contains a statistical bias then the ciphertext will also contain a statistical bias. If the key is created with a deterministic random number generator when a portion of the key is revealed the rest can be calculated.

For a key to be cryptographically random it must be statistically unpredictable, evenly distributed, and have no discernable patterns. The key must have backtracking and prediction resistance. If an attacker receives a piece of the key, they must not be able to predict future or previous portions of the key. The key must have a uniform distribution, such that there is no higher chance one bit will occur over another. The security of the key is dependent on its method of generation.

When implementing the One Time Pad, a common mistake is reusing the key. This is called the Two-Time Pad. When the same key is used to encrypt more than one portion of the cleartext, the xor of the ciphertexts equals the xor of the cleartexts. This means an attacker can use a technique called Crib-Dragging [6]. This technique involves guessing common phrases that would be in cleartext, then xor the phrase with the ciphertext to see if the result is readable. If the result is readable, it is most likely part of the true cleartext. When the Two Time Pad is used, the cipher becomes amateurishly easy to break.

A big shortcoming of the One Time Pad cipher is its malleability. It is very easy for an attacker to modify ciphertext in a way that meaningfully affects the cleartext. Any bit an attacker flips in the ciphertext will result in the corresponding bit flipping in the cleartext. If an attacker knows the cleartext they can modify the ciphertext such that it deciphers to any modified cleartext they chose. The One Time Pad cipher has no built-in authentication protocol, therefore if an attacker modifies the ciphertext one would never know. An example of OTP malleability is shown below in Figure 2.

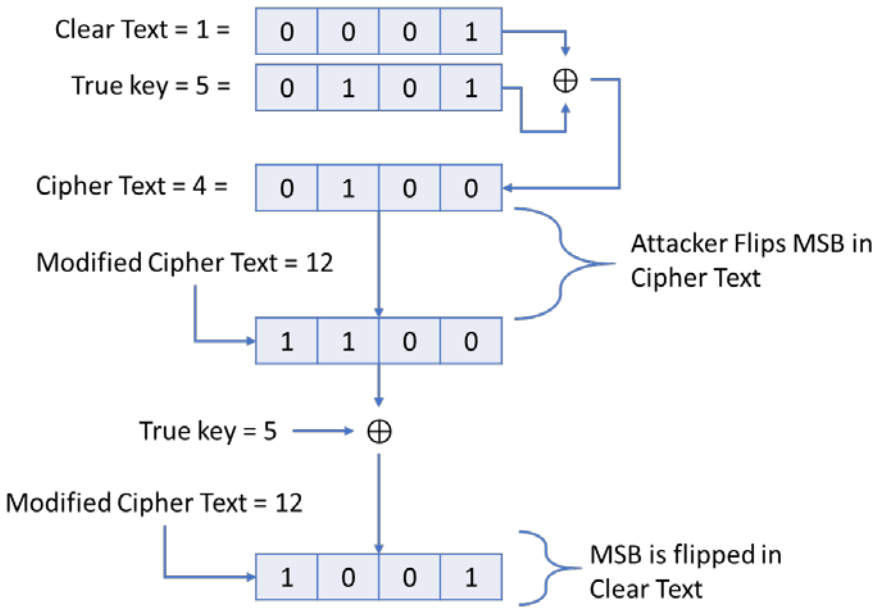


Figure 2 This figure shows how an attacker can modify an OTP ciphertext to meaningfully change the cleartext. Any bit flipped in ciphertext will result in the corresponding bit being flipped in the cleartext. This example shows the most significant bit (MSB) being flipped in the ciphertext causing the MSB to be flipped in the cleartext.

Malleability is a big problem in the cybersecurity of the OTP cipher. A ciphertext could detail the amount of money to be sent in a financial transaction. An attacker could flip a bit in the thousands place. This could cause a user to send thousands of dollars rather than hundreds.

One Time Pad in Cyber Security Practice

Cyber Security is all about the CIA (Confidentiality, Integrity, and Availability). Confidentiality is about making sure no unauthorized individuals can view confidential data. Integrity refers to making sure sensitive data has not been modified by any attackers. Availability is about making sure systems are always available to users to access.

Cryptography is said to be the strongest link in the Cyber Security chain. However, it is also the easiest link to get wrong. Cryptography is simply the process of making information illegible then legible again. The cipher can be considered the fundamental tool of cryptography. A cipher is simply an algorithm that can take a key to convert cleartext into ciphertext then back into cleartext. The One Time Pad is a symmetric key cipher, this means it uses the same key to encrypt and decrypt data.

For a cipher like One Time Pad to be considered secure in cryptography, it should abide by many standards. One such standard is Indistinguishability. This means the ciphertext should be indistinguishable from random data. The OTP Cipher does meet this standard when used properly. Any data XORed with random data should appear as random data [1] Pg. 37.

Non- malleability is a standard that means an attacker cannot modify a ciphertext in a way that meaningfully modifies the cleartext. The OTP cipher does not meet this standard. Any bit flipped in the ciphertext will result in the corresponding bit

being flipped in the cleartext. This means if an attacker knows the cleartext corresponding to a ciphertext they can modify the ciphertext such that it will decrypt to anything they chose. An example of OTP cipher malleability is shown in Figure 2 above. To give the OTP Non- malleability one may use secondary cryptographic tools like hash functions.

Semantic Security is a security notion that ciphertexts cannot leak any information about their cleartexts if the key remains secret. This means the same cleartext encrypted twice must result in different ciphertexts[1]. The OTP cipher does have semantic security.

In Cyber Security practice, the way secure communication is represented is with the Bob, Alice, and Eve model. Bob is trying to send a message to Alice. Eve acks as a Man-in-the-Middle. Therefore, Eve can monitor and control the network traffic between Bob and Alice. This model is used to see if Eve can do something like catch a message from Bob to Alice then modify it to say something different. If Bob simply used the OTP cipher to encrypt a short text message to Alice he would be in trouble. Eve could easily catch and modify the message and Alice would be none the wiser. However, if Bob also used other cryptographic tools like hash functions he could make it significantly harder for Eve to modify his messages. This is shown in Figure 3 below.

Figure 3 Bob and Alice communicate over a monitored network controlled by Eve. Bob and Alice send data back and forth, while all data must go through Eve. Usually, Bob is trying to send some message to Alice. Alice is usually trying to tell Bob she received the message.

OTP Messaging System Proposal

The One Time Pad is not commonly used in modern encryption systems because of its drawbacks. Its combination of malleability, large key sizes, and Two Time Pad risk makes the cipher far less appealing in comparison to a modern cipher like AES (Advanced Encryption Standard). However, there are certain use cases where the OTP cipher can still seem practical compared to other ciphers. One of these cases would be a simple text-based messaging system. Since the size of the data being encrypted is small the key size will not be an issue. Other cryptographic tools used in conjunction with the cipher give it Non-malleability. The simple nature of a text messaging system makes accidentally reusing keys unlikely. This is because it would be easy to keep track of relatively small key sizes.

Creating a One Time Pad Encrypted Messaging System would show the practicality of the OTP cipher in a modern context. This will determine the difficulties of generating and exchanging cryptographically random keys that are the size of the messages being encrypted. It is important to know the challenges that arise when using other cryptographic tools to circumvent the inherent malleability of the OTP cipher.

Chapter 2

ONE TIME PAD MESSAGING SYSTEM CONCEPTUAL METHODOLOGY

Requirements of Security

A lot of foresight is needed in implementing a secure messaging system. One must make sure they consider any plan an attacker might attempt. The messaging system must remain secure even if the attacker knows exactly how the system works. The system should still work even if the attacker can monitor and control all the traffic between the users.

To achieve this level of security one must consider CIA. A cipher is used to encrypt the message to obtain Confidentiality. When the attacker intercepts the message, they cannot decrypt the ciphertext. A cryptographic hash function is used to conserve Integrity. If an attacker modifies a message then the hash would no longer match the modified message rendering it invalid to the receiver. Availability should make the system robust, and allow users to know when the messaging service is working as intended. This requires the system the messaging service is running on to be hardened.

It is often said that key management is half the battle in secure communication. The messaging service needs a way to create and transfer keys. The keys should be created from a cryptographic non-deterministic random number generator (RNG). The keys should be retrieved from the source in a secure manner such that other applications on the system cannot also read the data generated from the RNG. The keys should be exchanged in a manner that cannot be intercepted by malicious parties. The keys should be stored in an encrypted manner.

Once keys are exchanged the OTP cipher can easily be used to make message data Confidential. However, a secure cryptographic hash function must be used to verify the data's integrity. There needs to be a message authentication algorithm in place. It is important to verify messages have not been modified by any attackers. This algorithm should rely on both the OTP cipher and the hash function for its security because the OTP cipher is the subject of this research.

Early Message Delivery Process

The message delivery design process begins by sending a single message from a sender to a receiver. The Bob, Alice, and Eve model will be used. Bob will be the sender of the message. Alice will be the receiver of the message. Eve will be a malicious third party with control over Bob's and Alice's network. For now, it is assumed the receiver (Bob) and the sender (Alice) both have a copy of the shared One Time Pad secret key.

Firstly, using the OTP cipher Bob should encrypt his cleartext being sent to Alice to ensure the message's Confidentiality. Since it would leak data to reveal the length of each cleartext all ciphertexts should be a fixed length. To fix the length of messages Bob will simply use a Mode of Operation to standardize messages of any size. If the cleartext length is too short it is padded with zeros to make the messages a standard size. If the length of the cleartext is too long the cleartext is simply split into multiple messages. Having fixed-length messages also allows Bob not to communicate the length of each message to Alice, because each message is the same size.

Next, Bob would need to compute a cryptographic hash of the cleartext to send to Alice to authenticate the message's Integrity. Alice can use this hash to determine if the message has been modified. If the message does not match its hash the message is

considered inauthentic and Alice would not accept the message. This is called authenticated encryption. There are multiple ways to do authenticated encryption with a cryptographic hash function, Encrypt-and-MAC, MAC-then-Encrypt, and Encrypt-then-MAC. MAC stands for Message Authentication Code, also known as a keyed cryptographic hash function. Since at this point Bob and Alice only have the OTP key they will be using a non-keyed hash function for now. If Encrypt-and-MAC is used, the hash function could reveal data about the cleartext. Eve would only need to guess hashes of common cleartexts to reveal the content of the message. If Encrypt-then-MAC is used, Eve can still easily modify the ciphertext. Eve would just need to precompute the hash of the modified ciphertext then replace the old hash with the new one. MAC-then-Encrypt is the most secure choice [1]. Unfortunately, MAC-then-Encrypt will increase the key size needed to send a single message by the length of the cryptographic hash.

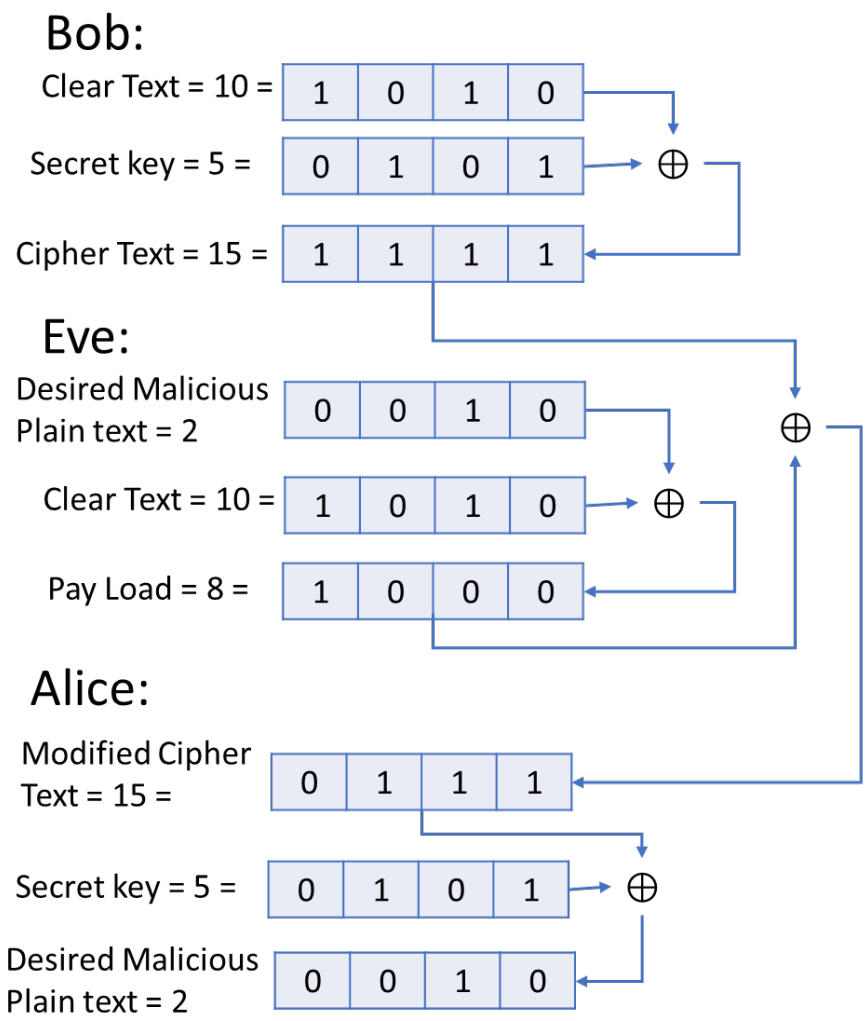
Lastly, Bob would need some way to confirm the message was successfully delivered to the receiver to ensure Availability. Without checking if a message was successfully delivered Eve can remove messages from Bob and Alice's conversation without getting caught. Message delivery confirmation would involve some sort of acknowledgment message being sent back from Alice to Bob. This acknowledgment message would need to be something Alice could only produce if the message was properly received. Alice can return the recomputed hash of the decrypted message. Of course, this acknowledgment hash should be encrypted using another portion on the OTP key. Once Bob sends his message he will start a timer and patiently wait for the acknowledgment to be received. If a time out occurs Bob will resend the message and restart the timer. If the acknowledgment is received the Bob will decrypt the hash and

check if it authenticates the original message. If Bob determines the acknowledgment is legitimate he will stop the timer and consider the message successfully sent. If Bob resends the message too many times he will give up and leave the message as undelivered.

So far, the basics behind the CIA have been covered for one message. By the design laid out so far, each message sent would consume the message-size plus twice the cryptographic hash-size of the One Time Pad key. This amount of key being used is okay because the size of the messages and hashes are relatively small. Now, the messaging system needs to work for multiple messages. This can be achieved using a simple message number such that each number corresponds to each message. The One Time Pad key will be divided up into portions corresponding to what message each portion is used for. When Bob sends a message, he will attach the message number at the end of the message unencrypted. When Alice receives the message, she will use the message number to determine what portion of the One Time Pad key she will use to decrypt the message and encrypt her acknowledgment reply.

Unfortunately, this simplistic approach to a One Time Pad messaging system has a couple of big problems beyond not being fully developed and having no key management. This approach is vulnerable to replay attacks, and it is not Non-Malleability Known-plaintext attack (NM-KPA) secure [1] Pg. 46. A replay attack would simply be if an attacker intercepted a message before it reached the receiver then sent it again later. If Bob had an important command to tell Alice, Eve can block then repeat that command at any time. The NM-KPA attack can be done if an attacker knows what the sender is saying to the receiver. If the attacker knows the cleartext

they can modify the packet to send a forged message. An example of the NM-KPA is shown below in Figure 4.



Mathematical Proof of Non-NM-KPA security:
PT = Plain Text
PT' = Desired Malicious Plain Text
CT = Cipher Text
CT' = Malicious Modified Cipher Text
$CT \oplus PT \oplus PT' = PT \oplus Key \oplus PT \oplus PT' = Key \oplus PT' = CT'$

Figure 4 This figure shows an example of an NM-KPA attack on the OTP cipher. If an attacker knows the clear text, then the attacker can modify the ciphertext to be any clear text they chose. The mathematical proof of the NM-KPA attack is shown below the example.

Eve could simply guess Bob says “Hello” in the first message. Eve would then calculate the hash of “Hello”. With each piece of the puzzle, Eve now knows the clear text. Once Eve knows the clear text she may conduct the NM-KPA attack as shown above.

Final Message Delivery Process

To further the One Time Pad Messaging system’s development the security issues must be fixed first then a key management system must be implemented. There are many ways to prevent replay attacks, and achieve NM-KPA security.

Preventing relay attacks is as simple as making sure massages cannot be delivered to the user multiple times. Alice must always send back acknowledgment packets when she receives a valid packet. This is in case Bob did not receive a previous acknowledgment packet. This means Alice must remember the first time each message is validated and sent to the user. This can be done with a simple message state that progresses from unreceived to received. After Alice validates a message she checks the message state corresponding to the message number. If the message state is unreceived the message is forwarded to the user, and the message state is progressed to received. If the message flag is received the message is not forwarded to the user. Either way, once a message is confirmed authentic by the receiver an acknowledgment packet is returned to the sender. This allows the receiver to always acknowledge messages without the user being shown multiple messages affected.

There are two main ways to achieve NM-KPA security. The first way is using a cipher that is not easily malleable without knowing the key. The second way is using a MAC that cannot be created for a forged message without an attacker knowing the

key. The purpose of this One Time Pad messaging system is to create a messaging system that relies heavily on the One Time Pad for security. This constraint is imposed to thoroughly test the practicality of the OTP cipher. Luckily, a cipher can be used in conjunction with a cryptographic hash function to effectively create a MAC. A set of steps must be taken to achieve this level of security. First, encrypt the message. Second, find the hash of the encrypted message. Third, encrypt the message concatenated with the hash. This is effectively the same as before however, now the attacker would not know how to modify the ciphertext such that the new hash can be properly inserted. The disadvantage of this solution is that the key size is practically doubled. The advantage of this solution is it relies as on the One Time Pad cipher and hash to gain NM-KPA security. This solution means Alice would no longer return the encrypted hash of the cleartext as the acknowledgment. Alice would return the hash of the encrypted message.

This updated messaging process is now more secure. An attacker in control of communication alone should not be able to forge messages between the sender and receiver unless they have a copy of the OTP key. Users will not receive messages multiple times even if resent by an attacker. The full OTP messaging system process is laid out in Figure 5 below. The notations used in Figure 5 are just below it in Figure 6.

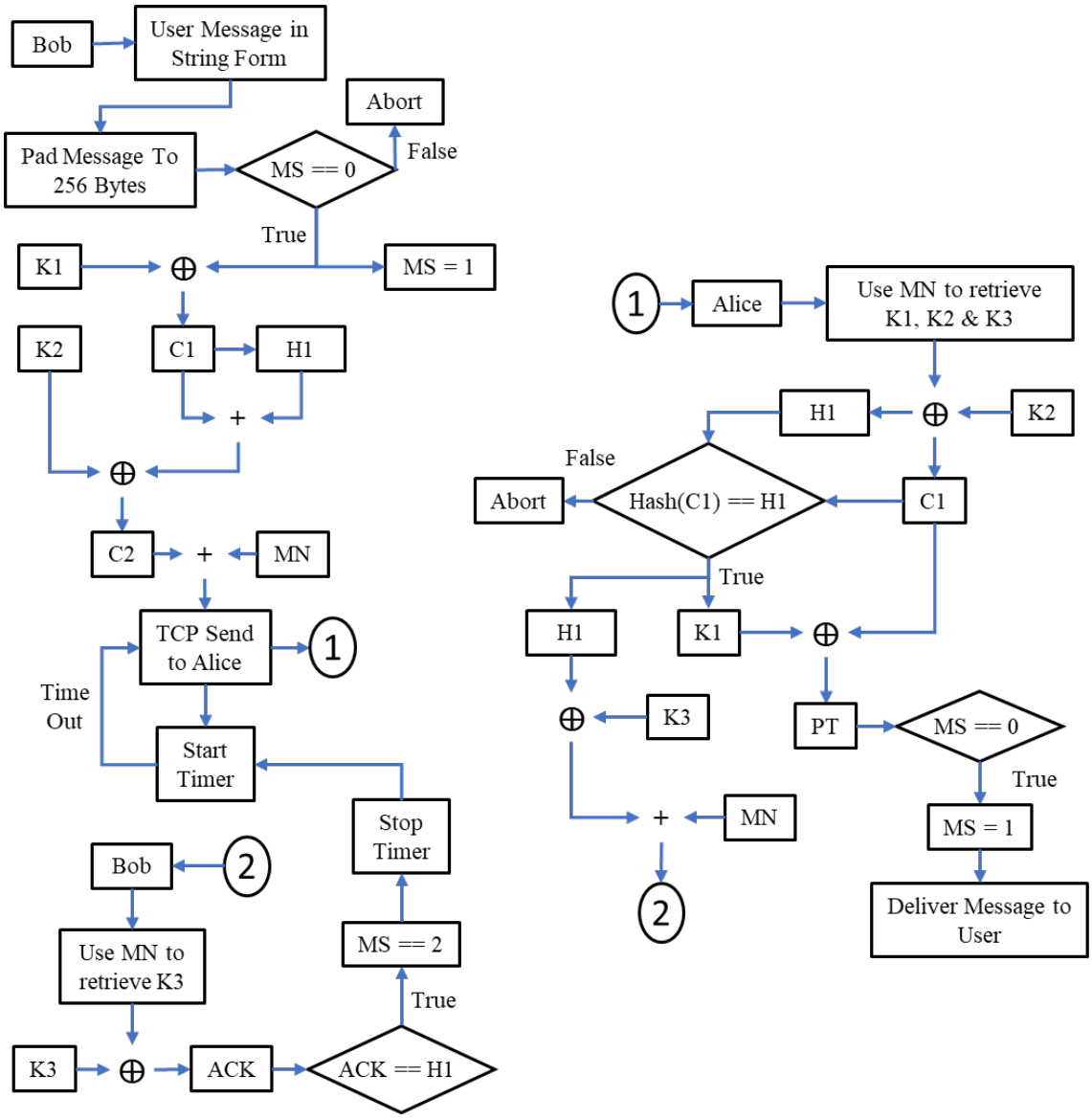


Figure 5 This figure is a flow chart of the entire OTP messaging process.

Notation:

K1: key portion 1, 256 Byes	MS: Message state, integer
K2: key portion 2, 256+32 Bytes	MS = 0 Bob: Default, Message Unmade
K3: key portion 3, 32 Byes	MS = 1 Bob: Message unacknowledged
PT: Clear Text, 256 Bytes, UTF-8	MS = 2 Bob: Message acknowledged
C1: Cipher Text 1, 256 Bytes	MS = 0 Alice: Message undelivered
H1: SHA3-256 Hash of C1, 32 Bytes	MS= 1 Alice: Message delivered
C2: K2 ⊕ (C1+H1), 256+32 Bytes	MN: Message Number, integer, 6 Bytes
=: Instantiation	⊕: XOR
Abort: End Message Prosses	+: Concatenation
①, & ②: Jump Points	==: Boolean Comparison

Figure 6 This is the notation for Figure 5.

Bob has a message to send from the user. The message is first padded to become the clear text PT. The message is a string from the user. The PT is a fixed length encoded byte array. Bob then makes sure his message state MS is zero for "unused". If MS is zero, MS is set to 1. Bob now encrypts PT with K1 into C1. Bob calculates the hash of C1, H1. H1 and C1 are encrypted with K2, then concatenated with the message number MN to create a packet. Bob sends this packet to Alice. Bob now starts a timer. If a timeout occurs, the packet is resent. When Bob receives an acknowledgment packet he uses the MN to retrieve K3. K3 is then used to decrypt the ACK. If the ACK matches H1 them Bob stops the timer and sets MS to 2.

When Alice receives a packet, she uses the MN to retrieve K1, K2, and K3. Alice decrypts C2 with K2 to get C1 and H1. Alice checks if the hash of C1 matches H1, if not Alice aborts the process. Alice then decrypts C1 with K1 to get PT. If MS equals zero Alice delivers PT to the user, then sets MS to one. Alice now encrypts H1with K3 to create an ACK. Then Alice sends the ACK concatenated with MN to Bob.

Developing Key Management Processes

Now a system for key management must be made. Bob and Alice must both have a copy of a large file with random data to use as a One Time Pad key. Before the keys can be exchanged, the keys must be created. The sender (Bob) should be tasked with key creation. This is because the sender has control over what the messages say and who they get sent to.

It is not easy to create a large amount of cryptographically random data. A deterministic pseudo-random number generator (RNG) cannot be used, because an attacker might determine the initial value or an internal state then recover future or previous portions of the key. This is called backtracking resistance and prediction resistance. Operating systems like Linux and have built-in system calls for retrieving cryptographically random data. These RNGs have sources of entropy based on difficult to predict environment values. This is fine for most cryptographic systems that only need a few bytes of random data. However, most systems do not have nearly enough entropy to generate the amount of random data needed for the One Time Pad messaging system in a timely manner. Therefore, a dedicated hardware-based RNG will be used. A hardware-based RNG can generate random data at a relatively fast and constant rate. Unfortunately, this solution will incur a financial cost to the user.

One must be careful when retrieving random data for cryptographic purposes. This can be done by using library functions and following the sample code provided by the developers of the hardware-based RNG. It is Cyber Security best practices to use up-to-date cryptographic libraries. Those who are not experts should not try to implement cryptographic systems on their own for non-educational purposes. When retrieving random data do it in a manner such that another program cannot easily retrieve the same random data.

Once the program is securely retrieving the data it should be encrypted at rest. Save the data in an encrypted state. A block cipher like AES should be used to encrypt the random data. The password used to encrypt and decrypt the random data will be taken from the user. The password-file should be privileged read-only, and its directory should be privileged accesses only. Only a program with administrator privileges on the system should be able to read the password file.

Unfortunately, transferring this key-file without the data being intercepted is tricky. This OTP messaging system should rely on the OTP cipher as much as possible. Unfortunately, the OTP cipher cannot be used to encrypt an OTP key exchange for obvious reasons. Two methods for key exchanges will be explored. In-Band key exchange and Out-of-Band key exchange.

An In-Band key exchange is done over the internet. An Out-of-Band key exchange is completed over a medium separate from the internet. In-Band key exchanges are usually less secure than Out-of-Band key exchanges. It is easier for a Man-in-the-Middle attacker to intercept data over the internet. It would usually be much more difficult for an attacker to monitor or control a non-internet communication channel.

To conduct an In-Band key exchange a Diffie-Hellman key exchange would most likely be used. The Diffie-Hellman key exchange is special because it allows two parties (Bob, and Alice) to exchange a shared secret key over a public channel. The Diffie-Hellman key exchange is safe even when all communications are visible to attackers. The Diffie-Hellman key exchange can be conducted anonymously. This means Bob and Alice could have no prior communication or knowledge of each other before creating their shared secret. Unfortunately, the Anonymous-Diffie-Hellman key

exchange is vulnerable to Man-in-the-Middle attacks. This means Eve can take over the key exchange and impersonate Alice. To get around this attack one should use the Authenticated-Diffie-Hellman key exchange. The Authenticated-Diffie-Hellman is considered cryptographically secure, prevents single party key control, and provides perfect Forward Secrecy [1] Pg. 314. Unfortunately, the Authenticated-Diffie-Hellman is vulnerable to replay attacks by Eve. However, this can be avoided with a key confirmation. Always make sure to use a Key Derivation Function on the shared secret of the Diffie-Hellman key exchange. This is because the shared secret alone is not cryptographically random. Always make sure both parties in a Diffie-Hellman key exchange are using safe primes. The Authenticated-Diffie-Hellman relies on RSA signatures for authentication. Once an Authenticated-Diffie-Hellman shared secret is created and a key is derived the OTP key file can be securely encrypted with AES then sent from Bob to Alice. Afterward, an authentication packet can be sent back and forth. This can all be done over any secure TLS connection.

The problem with In-Band key exchange is the loss of Informational Security over the network. AES can only be as secure as its key. A properly done Diffie-Hellman key exchange is only as secure as the Discrete Logarithm Problem is hard [1] Pg. 303. If Bob and Alice conducted an Out-of-Band key exchange they would not have to worry about a malicious entity monitoring their communications. An Out-of-Band key exchange can be done as simply as loading the encrypted OTP key files onto a USB stick then transferring it over by hand. Bob and Alice could verbally exchange the key used to decrypt the OTP key file. Other less secure Out-of-Band key exchanges can be done over Blue Tooth and NFC. However, these wireless protocols are not inherently secure. Out-of-Band key exchanges should be done in person or by

a trusted currier, and in a secluded secure location. The big disadvantage of Out-of-Band key exchanges is that Bob and Alice must meet in person. This restraint provides inherent security and trust. However, it reduces the usability of the OTP system. Ultimately, how Bob and Alice exchange their keys is up to them.

Chapter 3

ONE TIME PAD MESSAGING SYSTEM IMPLEMENTATION

Programing Language

The conceptual design of the OTP messaging system has been created. Now the messaging system must be created. This messaging system should be created in a way that runs on multiple platforms. The messaging system should be easy to run. The messaging system should be intuitive. The messaging system should be designed to be updated over time. The messaging system construction process begins with sending a single message. The sender and the receiver can already be considered to have a copy of the OTP key.

The messaging system is a computer program. Therefore, the first thing that must be decided is what programing language will the messaging system be coded in. The programing language must be easy to develop with. The programing language must run the program in a timely manner. The programming language must not consume too many system resources when run. The programing language used should be able to run on Windows, Linux, and Mac with minimal modification between versions. There are many programming languages to choose from C, C++, Python, Java, Golang, and many more. Python 3.7 will be used to develop this messaging system. This is because Python is very easy to develop software with. Python runs on any commuter system with Python installed. Python code does not need to be compiled into operating system specific executables. Although some modification is needed to run python code on different operating systems, it is relatively small compared to other programming languages. Compiled languages like C, C++, and Golang will run faster than Python. However, this messaging system requires

relatively little computational power. Therefore, the difference in program speed will not affect user experience. Most importantly Python has many cryptographic and networking libraries at its disposal. These libraries are kept up to date and secure by experts. This means it is easy to rely on these libraries for security. It is the job of the messaging system developer to always keep its libraries up to date and secure.

Clear Text Size

Now that the Python development environment is chosen. The process of the messaging system will be programmed. The first thing that needs to be chosen is clear text length or the size of the PT as defined in Figure 5. The user's text message must be encoded into a byte array to properly encrypt. Each clear text is the same length. This means there is a limit to how many characters a user can fit into a message. Also, regardless of how much the user writes into each message the same amount of OTP key will be consumed. SMS messages are usually a maximum of 160 characters [3]. SMS messages are usually encoded with GSM-7. This is a 7-bit encoding algorithm. This means SMS messages are 140 bytes in size.

The One Time Pad messaging system will use 256-Bytes plain texts. The OTP messaging system will also use UTF-8 encoding. This means each Byte in the plain text will correspond to one character in the message for the most part. Most messages can fit into 256 characters. This clear text size means over 512 Bytes of OTP key will be consumed when a single message is created.

Cryptographic Hash Function

Once the clear text is encrypted the hash must be calculated. There are many cryptographic and non-cryptographic hash functions. The size of the hash will result in

twice that amount of OTP key being used per message. Cryptographic hash functions have certain standards they must meet to be considered cryptographically secure. Unpredictability means the output of a Hash function should be cryptographically random. Preimage Resistance means given the hash value of random data it is practically impossible to find the input that results in that hash. Collision Resistance means it is practically impossible to find two inputs that result in the same output hash. However, due to the Pigeonhole Principle, no hash can have true Collision Resistance [1] Pg. 175. This is because there are infinite possible inputs and only a finite amount of outputs. Therefore, some inputs must compute to the same hash. The hash should be relatively easy and fast to compute. The hash should be computable on most computer systems. Luckily, the National Institute of Standards and Technology (NIST) holds world-renowned cryptography competitions to create American cryptographic hash algorithm standards. In these NIST competitions, cryptographers from all over the world compete to create advanced cryptographic hash algorithms. When a cryptographic standard is adopted by NIST it is considered state of the art cryptography [1] Pg. 175. NIST cryptographic hash functions are the most extensively scrutinized by cryptanalysts. This means NIST standard hash functions are most ubiquitous in cryptography. Being so common, most cryptographic libraries support NIST cryptography standards. Also, processor manufacturers design special hardware to compute NIST standard cryptography faster. NIST standard hash functions are called the SHA (Secure Hash Algorithm) family of hash algorithms. SHA3-256 will be used to authenticate the OTP messaging system. SHA3-256 produces a hash message H1 with a length of 32 Bytes. Therefore, 64 Bytes of the OTP key will be consumed each message to encrypt this hash.

Communication Protocol

Now that the initial message has been created there must be a method to send the message. There are many internet communication protocols. Some of these protocols have built-in cryptographic security. Some of these protocols do not even have reliable data transfer. This messaging system should be secure regardless of its internet communication protocol. There have been many times in the past when supposedly secure internet communication protocols have had debilitating failures. These failures can be from old week protocols being left out on the internet. Like old HTTPS connections that use a prime P which is too short to be cryptographically secure [1] Pg. 97. Or, TLS connections that used RC4 as the stream cipher, despite RC4 is no longer being considered secure [1] Pg. 359. Sometimes, shoddy software developers simply do not implement their code correctly. Some servers use AES-GCM as their stream cipher, then proceed to use a repeating nonce. Some developers even use all zeros as their nonce [1] Pg. 237. There have even been times where malicious entities have manipulated Certificate Authorities [1] Pg. 363.

However, secure internet communication protocols that are up to date and implemented correctly can be considered secure for the time being. This means using a secure internet communication protocol for the One Time Pad messaging system's data transfer cannot decrease the messaging system's level of security. However, this OTP messaging system is for educational purposes. Therefore, the OTP messaging system's method of internet communication will TCP using socket programming. If this messaging system was a consumer product it would have multiple layers of security. TCP was chosen because it has no security features, but it is the most basic version of reliable data transfer.

Key Management

Next, the key retrieval process must be implemented. Both Bob and Alice have a copy of the OTP key file. The OTP key file is simply an encrypted key file with a large amount of random data in it. As stated earlier Bob and Alice manually exchange a user key. This user key can be used to encrypt the OTP key file. The OTP key file can be encrypted with AES-GCM. When each user starts the OTP messaging system program the OTP key file is decrypted in memory. Then, each user reads from the file in the same manner. Each user cuts out chunks of data as OTP key portions, then assigns message numbers to each set of OTP key portions. Since the same code is used to read the same files the users know each message number corresponds to the same keys. This allows each user to easily instantiate each message data structure with the appropriate keys and message numbers.

The messaging system itself is a simple data structure. The messaging system is essentially a list of messages. The user iterates through this message list when receiving and sending messages. The sender, and the receiver both use the same data structure as their messaging system. The sender and receiver simply use different methods for sending and receiving data. When the sender and receiver start their programs for the first time they decrypt then read the OTP key file. Then as data is being read from the key file the list of messages is being appended to. Each message simply has its message state, its set of keys, and an empty clear text. Once the message list is filled the sender is prompted to create a message to send. The receiver simply waits for messages to be received while returning ACKs (acknowledgment packets) in the background.

For the keys to be exchanged they must first be generated. It has already been decided only OTP keys will be saved. These OTP keys are simply files filled with

cryptographically random data. This data should come from a hardware-based cryptographic random number generator (RNG). There are many hardware-based RNGs to choose from. The rate of random number generation should be high enough to create multiple megabytes of random data in just a few seconds. The random data generated should be tested for randomness. The RNG should have a simple, and secure way to connect to the system. The RNG should have libraries and examples on how to connect to it. The RNG should be inexpensive.

Ultimately, the TrueRNG v3 was chosen. The TrueRNG v3 was chosen because for multiple reasons. Most computers running python have USB ports for the TrueRNG v3. The TrueRNG v3 is designed to be used for cryptographic purposes. It has a random data generation rate of 400 kilobits per second. The TrueRNG v3 does come with documentation demonstrating how to use it. Lastly, it was a low price.

Alice and Bob will both create keys for the messages they send. This is so each user has control over their own message's confidentiality. The sender (Bob) will simply read random data from the TrueRNG v3, encrypt it using AES-GCM, then write it to an encrypted file. Bob will then use a secure method to transfer this file to Alice. If Bob & Alice share a data storage device like a memory card they must make sure to write over the residual memory on the data storage device after transfer. Once Bob and Alice both have a copy of the OTP key file, they may start reading data. Both Bob and Alice read the data from the OTP key file using the same code. This ensures all the message's OTP key portions and message numbers match.

Save Messaging State

Now that the messaging system is complete. Bob and Alice need a way to start and stop their message systems whenever they like. When a python program is

stopped all the local storage data on the stack/heap is lost. If the messaging system is simply restarted, then the message states are reset to unsent and undeceived. This is very dangerous due to the two-time pad vulnerability.

There are multiple ways to solve this problem. One way is to create an algorithm that can write all messages' data to a file in a consistent manner. Alice and Bob will both run the algorithm to write their current message data to an AES-GCM encrypted file. Creating and debugging an algorithm like this on one's own can be difficult. Luckily, Python has a library called Pickle. This library allows a user to save an arbitrary object to a file. When they are done using the OTP messaging system Bob and Alice simply need to Pickle, encrypt, then save their message objects to a file. When Bob and Alice are ready to communicate again they simply read, decrypt, and un-Pickle their message objects back. To be clear, the Pickle file is all that is needed to run the program. The OTP key is included in the Pickle file. The Pickle file can be considered the OTP key file. One should keep in mind an unencrypted Pickle file is very insecure [2]. There are many techniques involving modifying a Pickle file to take over a Python program.

Messaging System Information

Ultimately, the messaging system is written in Python 3.7. Any computer running python 3.7 should be able to run the messaging system with minimal modification. The messaging system has been tested to work in both the Windows 10 and Ubuntu 18.04.3 LTS. Linux and Windows each use different code to read from USB ports. Each message clear text is 256-Bytes long with 256 characters. Each SHA3-256 hash is 32 Bytes. Each message sent, and confirmed delivered consumes

576 Bytes of OTP key data. It takes the RNG 20 seconds to make enough random data to support the sending of 1763 messages.

Messaging System Overhead

The original cleartext message is 256 Byes. The message concatenated with the hash is 288 Bytes. The message number is an integer converted into 6 Bytes. The total TCP data sent from Bob to Alice per message is 294 Bytes. The acknowledgment packet consists of a 32 Byte hash and a 6 Byte message number. The total TCP data sent from Alice to Bob per message is 38 Bytes. It takes at least 332 Byes of TCP data to be sent over the network to send one 256 Byte message.

It takes 256 Bytes of the OTP key to encrypt the message. It takes 288 Bytes of OTP key to encrypt the message concatenated with the hash. It takes 32 Bytes of the OTP key to encrypt the hash again. It consumes a total of 576 Bytes of the OTP key to send one 256 Byte Message.

To send a single message at least 576 Bytes are XORed, 2 comparison functions are ran, 2 hashes are computed, 2 TCP sends are executed, and 2 TCP receives are executed. Computationally speaking the xor function is parallelable. Comparison functions are also parallelable. Hash function run times depend on the input size. All these functions need to run in a manner as to not allow timing attacks.

Chapter 4

ONE TIME PAD MESSAGING SYSTEM TESTING

Message Modification, and Fabrication

Once again, the Bob, Alice, and Eve model will be followed. Eve should not be able to Modify or Fabricate any messages without a copy of the OTP key. Eve attempts to modify a message by flipping a bit in the ciphertext. This will cause the corresponding bit to be flipped in the cleartext. When this bit is flipped the hash will no longer match the clear text, then Alice will drop the message. There is a possibility Eve can also flip bits in the hash, to match the modified clear text. The NM-KPA vulnerability attack is shown in Figure 2. This problem has been fixed. This is because the hash is no longer of a plain text predictable by Eve. The message is now encrypted before being hashed. This means Eve will not know how to modify the hash to match the modified message's ciphertext. Of course, both the encrypted message and the hash are encrypted again as to not leak data. The probability Eve can modify a message and hash is approximately (1 / 2^ (256)). This is due to the security of the SHA3-256 hash function. This is probability is infinitesimally small. The reason Eve cannot modify a message without knowing the OTP key is the same reason Eve cannot fabricate a message without knowing the OTP keys.

Timing Attack

A timing attack is a simple side-channel attack. An attacker measures the time it takes to execute certain operations. These time measurements are then used to reveal sensitive data about the cryptosystem. The OTP cipher has a constant-time encryption with xor which further suppresses side channel vulnerability. The hash may still be time sensitive.

A common form of timing attack is measuring the time it takes for two values to match. The attacker provides one value then an oracle returns if the value provided matches a secret value. If the oracle uses an insecure matching algorithm then an attacker can reverse engineer the secret value. An example of an insecure matching algorithm is checking each byte sequentially then returning false if two bytes do not match. An attacker can manipulate this process by checking all 256 possible values of the first byte. When the oracle takes longer to return false, the first byte is correct. The attacker would then check all possible values of the second byte. When the oracle takes longer to return false, the first two bytes are correct. This goes on for the length of the secret value. This process goes on until the entire secret value is revealed.

In the OTP messaging system Alice does execute an insecure comparison function. When Alice decrypts a message with K2, she checks to see if the hash H1 matches ciphertext C1. Only if there is a match will Alice return an acknowledgment packet ACK. Bob must wait a certain amount of time before resending a message. If Alice immediately returned a non-acknowledgment packet (NCK) to possibly expedite Bob's message resending process, then the OTP messaging system would be vulnerable to timing attacks. Eve could simply create an arbitrary ciphertext. Then Eve could use a timing attack to reverse engineer the H1 hash of the arbitrary C1 ciphertext. This could result in Eve forging a message to Alice. Luckily, Alice's comparison function was replaced with a secure one, and Alice does not respond to inauthentic messages. Bob also has a secure comparison function for authenticating received ACKs.

Troubles with TCP socket programing

TCP send and receive functions need specified amounts of data to send and receive. If a TCP receive function is executed to receive 10 Bytes of data then only receives 5 Bytes, the function will wait till it receives another 5 Bytes. This is because the TCP receive function is blocking.

This is a problem. If an entire message has not been sent for any reason, then the receiver will wait forever for a response. This can affect Availability. If Eve removes a portion of a message from Bob, then Alice can be left waiting for the rest of the message. When Bob resends the message, Alice will fill the end of the message with the beginning of Bob's recent message. This will cause Alice to see the message as inauthentic. When Alice receives the next message the first portion will be filled with the tail end of Bob's recent message. Bob and Alice may never get back on track.

The solution is simple. TCP send and receive functions can run in a non-blocking manner. This can be done by setting a timeout in Python's API. This timeout allows Bob and Alice to only attempt to send and receive data for certain amounts of time. If a timeout occurs then an exception is thrown. This means Eve would have to actively block messages to stop the OTP messaging system. Eve cannot simply ruin Bob's and Alice's message alignment.

Message System Testing Methodology

The messaging system is tested as it is built. However, when the messaging system is complete, one would want to test it as strenuously as possible. This allows the messaging system to retain function even under the most extreme circumstances.

This testing is done by modifying every receive and send function call in the program. Every time a data is sent or received there is a 20% chance something will

go wrong. This is an arbitrary percentage which allows the messaging system to experience moderate difficulty in testing. Any number of bytes can be changed, added, or removed. These modifications represent errors caused by attackers or extraneous circumstances. Every time an error is made the nature of the error is printed to the log file. Every time a TCP send or receive timeout occurs is printed to the log file. Every time Bob sends or resends a message is printed to the log file. Every time Alice receives an authentic message or inauthentic message is printed to the log file. Every time Bob receives an ACK or inauthentic ACK is printed to the log file. This process creates comprehensive log files for the sender and receiver. These log files show how the messaging system program handles itself when things are going wrong.

Testing Randomness

This OTP messaging system heavily relies on the randomness of the keys. Therefore, one would want to test the randomness of the key generated. There are many statistical steps for testing randomness. Unfortunately, one can easily create nonrandom data that passes any statistical test [1] Pg. 66. For example, the decimal digits of pi can pass any statistical randomness test, however, they are not random. There is no sure-fire way to measure the randomness of numbers. The randomness of the OTP key was tested using NIST standards. This was done using tools made by Zdeněk Říha, and Marek Sýs [7]. No significant statistical bias was found.

Chapter 5

CONCLUSION

Future Research

There is still work to be done on this messaging system. The biggest weakness of this system is both users must be active at the same time to exchange messages. This is also the case for other popular messaging systems like SMS. To send an SMS message both the sender's and recipient's phones must also have a cellular connection. This is fine for SMS messages because users have cellular connection nearly 24/7. This messaging system runs on Python. It would be difficult to keep a computer running Python nearly 24/7.

Preferably there would be a trusted secure server working between the users. This server can save users' messages before delivery. This allows Alice to receive a message long after Bob has sent them. Of course, this server would save messages in an encrypted state, utilizing end-to-end encryption. This means the users would need to conduct key exchanges with this server to preserve the CIA. This server would also need to be hardened. Extensive cybersecurity analysis and testing were required to secure Bob and Alice's connection. This cybersecurity analysis and testing would need to be repeated to secure a connection with a server.

Availability in this messaging system is protected by an acknowledgment packet. This packet is sent from Alice to Bob to confirm a message has been delivered. This packet is unforgeable and repeatable. There is always a chance Eve will choose to stop all communication between Bob and Alice. In this case only Bob will know messages are not being delivered. There should be a way for Bob and Alice to know if Eve is stopping their communication. This could be accomplished with a heartbeat

function in the messaging system. This function is simply periodic messages automatically sent between Bob and Alice. These messages will continuously let Bob and Alice know they have a working connection. Their heartbeat message should be unforgeable and indistinguishable from other messages. This would better protect availability.

A common practice in Cyber Security is penetration testing. Professionals, and experts design and develop secure systems handling sensitive data. It is always Cyber Security best practices to have an external set of hackers to test a system for security. It would be preferable to release the OTP messaging system to a team of professionals to run independent tests of its security. Any security vulnerabilities found would then be patched.

Currently, this program can only be running in sender and receiver mode. If each user wants to send, and receive messages they must run the program two times. Each user would need to keep track of two OTP key files. The messaging system should be redesigned in a parallel programming manner. There should be a separate thread for sending messages and receiving messages. Both sent and received messages should be displayed to the user in chronological order. This would best be done with a Graphic User Interface (GUI).

This messaging system currently works in a command-line environment. This is fine for testing purposes. However, the common user would require a GUI to use a messaging system. This means there would be a menu interface with options like, generate keys, set messaging save state, send messages, and view messages. The messaging system should be installed on a user's computer. The messaging system

should run on the host system's startup to notify the user when new messages have been received.

The messaging system uses TCP for educational purposes. If the public used this system, then there should be multiple layers of secure communication. Updating the messaging system to use a secure internet communication protocol like TLS 1.3 would be preferable. This would mean an attacker would need to bypass TLS 1.3 to attack the OTP messaging service.

What to do Differently

This messaging system relies extremely heavily on the OTP for security. The messaging system was designed this way to best test the practicality of the OTP cipher. However, in retrospect, this messaging system would have been much easier to make without so much reliance on the OTP. In the end, AES-GCM had to be used to securely store and transmit keys.

A MAC should have been used to make authenticated encryption. The messaging system nearly doubles its key size to use encrypt, hash, then encrypt. This could have been avoided by using a keyed cryptographic hash. A smaller portion of the OTP key could have been used as the hash key. Of course, this would mean the key would act both as an OTP key and a hash key.

Stream Ciphers

Stream Ciphers are very similar to the One Time Pad cipher. The encryption methods are the same. Both ciphers xor cryptographically random data with their plain texts. However, their key derivation methods are different. Stream ciphers create a stream of cryptographically random data from a fixed initial value, & key. This means

a perfect stream cipher is as secure as its key. The OTP derives its key from a truly random source. If implemented correctly, the OTP has perfect informational security. However, there may not be a truly random source of data in existence. The best RNG in the world may only be as practically secure as a securely seeded stream cipher. The difficulties in implementing the OTP cipher may not be worth "perfect" security. There may currently be no practical difference between 256 bits of security, and 512 bits of security. However, in the future 1024 bits of security might be the minimum secure standard.

Informational vs Computational Security

The One Time Pad cipher is not practical in modern cryptographic systems. Theoretically, this messaging system has Informational Security in transit. For argument's sake, the messaging system will be considered Informationally Secure. Computational Security is the level of cryptographic security such that a cipher cannot be broken with a practically feasible amount of computational power [1] Pg. 80. Most cryptographic communication systems have Computational Security using only 128 to 256 bits of security. 256 bits of security will very likely be secure for our lifetime. 512 bits of security is considered all that is needed to be practically secure forever. Breaking 256 bits of security is a practically impossible task. Breaking 512 bits of security is like multiple practically impossible tasks. Regardless of how many bits of security over 256 the task of breaking the cryptography is practically the same. Therefore, there is no practical difference between having Informational Security and Computational security.

The greatest advantage the OTP cipher is its Informational Security. However, using a cipher with this level of security has too many drawbacks. The OTP cipher

needs keys equal to the size of its clear text. The OTP key can only be cryptographically random data. The OTP cipher can never reuse the same key to encrypt multiple clear tests. The OTP cipher is easily malleable. These drawbacks already render the OTP cipher unusable is most cryptographic systems. However, this research considers the viability of using the OTP cipher in an optimal scenario.

This optimal scenario is a simple text messaging system. The OTP cipher should work well for an application involving small amounts of sensitive data like text messages. Unfortunately, the task is not as simple as saving keys the size of the text messages sent. Keys need to be used for authentication two. It takes significant overhead to work around the malleability of the OTP cipher. These keys need to be stored in a Python-compatible format. This greatly increases the size of the key's storage space.

The most difficult aspect of key management is key exchange. Exchanging keys In-Band with public-key cryptography would defeat the purpose of the OTP cipher. Public key cryptography like RSA and EC-DH are not Informationally Secure. Therefore, the users are left exchanging keys Out-of-Band. Exchanging keys Out-of-Band is not user friendly. Ultimately, if users do not want to go through the procedures of using a cryptographic system its security is irrelevant. Technically, since the keys are stored with AES encryption the messaging system reduces to computational security.

The method for key generation is also undesirable. OTP keys must be derived from a truly random source. This source cannot be a secure pseudorandom function, because then the cipher is only as secure as the RNG's seed. Using an external

hardware-based RNG has its complications. Users would need to purchase an RNG, and the RNG needs to be sampled securely.

The most common mistake made when using the One Time Pad is the Two Time Pad. It is imperative OTP keys are not reused. Implementing an OTP system involves extra procedural design to ensure keys are never reused. This is in stark contrast to a cipher like AES, which can reuse keys, and remain Computationally Secure.

Other commonly used secure ciphers like the block cipher AES or the stream cipher Salsa20 are far more practical than the One Time Pad. These ciphers can reuse their keys. There is extensive documentation showing how to use these ciphers. These ciphers have well-established methods for authentication. There are secure communication protocols like TLS1.3, which already use these ciphers behind the scenes [8]. If this messaging system was designed with modern cryptographic tools, then it would be more user friendly and have a much shorter development process.

Conclusion

It is possible to securely use the One Time Pad cipher in a modern cryptographic system. However, the benefit of Informational Security is outweighed by the rules needed to implement the OTP cipher securely. The difference between Informational Security and Computational Security is insignificant. There is a significant developmental difficulty in implementing the OTP cipher securely. If there were computers with infinite computational power then the OTP cipher would be very useful. Today's commonly used ciphers are more than enough to provide adequate security and usability in any cryptographic system.

Appendix

Bibliography

1. Aumasson, Jean-Philippe. *Serious Cryptography: a Practical Introduction to Modern Encryption*. 1st ed., vol. 1 1, ser. 1, No Starch Press, 2108.

2. "Pickle - Python Object Serialization." *Pickle - Python Object Serialization - Python 3.8.3 Documentation*. N.p., n.d. Web. 19 May 2020.

3. Kamp, Paul. "What Is GSM-7 Character Encoding?" *Twilio*. N.p., 2020. Web. 19 May 2020.

4. Shannon, Claude. *Communication Theory of Secrecy Systems*. 1949, *Communication Theory of Secrecy Systems*, netlab.cs.ucla.edu/wiki/files/shannon1949.pdf.

5. "History of the One Time Pad." *Introspective Networks*. N.p., 02 Apr. 2018. Web. 27 May 2020.

6. McGee, Zachary. "About Crib Dragging." *Crib Drag - Online Interactive Crib Dragging Tool*. N.p., n.d. Web. 27 May 2020.

7. Říha, Zdeněk, and Marek Sýs. "Faster Randomness Testing." *STS*. N.p., 2017. Web. 27 May 2020.

8. Jackson, Brian. "An Overview of TLS 1.3 - Faster and More Secure." *Kinsta Managed WordPress Hosting*. N.p., 13 Apr. 2020. Web. 27 May 2020.