(1)

For Alice and Bob to ensure that they both are using the same key for their message encryption and decryption, the properties of integrity and confidentiality must be met. Certainly, our new (insecure) channel that we would like to certify our shared key on must maintain the property of confidentiality (assets are only viewed by certified parties), such that there can be no third parties with the analytical and cybersecurity tools to crack the transmission. The purpose of having and confirming the shared key becomes pointless once an attacker has breached the confidentiality of the key asset.

In addition, integrity (data is only modified by those who are certified for it) is another property that must be controlled in this situation. In Alice and Bob's insecure communications about whether they share the same key, it is integral that they are truly seeing each other's messages and not one modified by a hacker or other entity. Similar to confidentiality, the security of our channel fully depends on the fulfillment of the property of integrity such that messages are exchanged accurately and without the possibility of modification from a third party.

(2)

The scheme mentioned of Alice sending $x = k \wedge r$ where r is some random number (n-bit), and then Bob sending $y = k \wedge x$ has a major security vulnerability on data confidentiality that would make our system vulnerable and insecure from attacks as soon as the transmission exchange is made. Notably, we have two pieces of data, x, and y, in an insecure channel for a third party to attempt to gain some useful insights from our communication. We could use the following method to attain the key, which should be secret. Just like Alice and Bob, we are most interested in the case that k_a and k_b are the same such that we will be able to use their key to decrypt message exchanges, so we will say (and assume) that $k = k_a = k_b$. Either way, let's see what we can derive from the provided.

r ~ random n-bit
k ~ n-bit secret key

Insecure channel
{ $X = k_a \wedge r$ }
{ $Y = k_b \wedge X$ }

A hacker will be able to attain the secret key k because the protocol mentioned above allows a leak of data composing our key via the XOR operation and the property of

reversal (associativity). If a hacker were to take the XOR of X and Y, then our hacker would obtain k and then we would lose the property of integrity on our data, as another party would gain access to view and use the 'secret' key.

X XOR Y

$(k_a \wedge r) \wedge (k_b \wedge X)$

$(k_a \wedge r) \wedge (k_b \wedge (k_a \wedge r))$ where $(k_a \wedge r)$ is a like term

And based on associativity, the XOR expressions above simplify down to just k_b. This means that even in the case that Alice and Bob's keys are not the same, our hacker could easily obtain Bob's key. More importantly, we should note if these keys are the same then of course our attacker could attain Bob's key (which is the same as Alices), and any future communication would become insecure. In either case, we can note that the confidentiality of this particular exchange is not met as the attacker could use this protocol to at least get Bob's key (find new information), which may also be the shared key, proving the faulty property of confidentiality in this example protocol.

1.

(1)

An attacker will only be able to deduce which password a user has out of the two that are pre-suspected in the case that t=1 or t=3. This kind of attack will not be susceptible to frequency analysis as this type of analysis should not apply to such small text, and both (or generally any) passwords should be just as likely as another. There is little relevance in employing frequency analysis, though that sometimes may work with larger (non-password) data.

For t=1, the attacker can use the fact that p1 has all "increasing" letters (in terms of Ascii, Unicode, etc), whereas p2 has so-called jumbled letters. The attacker can use this small fact when taking the encrypted ciphertext of the password to deduce which was the original password. The attacker can leverage information about the order to figure out which cipher is more similar (matching in pattern) to the password, based on the fact that the message was linearly encoded (using Vigenere ~ Caesar cipher), and therefore apply the same analytics of the letter (value) ordering. Notably, if the ciphertext follows some scheme "[a+i][b+i][c+i][d+i]", then the attacker can be sure that the password is p1="abcd". If the attacker observes "[b+i][e+i][d+i][g+i]", then the attacker can be sure that the password is p2 = "bedg". Let's say if i is 1 (base case), then we will get c1 = 'bcde' and c2 = 'cfeh'. C1 is completely increasing whereas c2 is not in lexicographical order. Keep in mind that even if we see c1='yzab', this is considered increasing still as our cipher uses 1:1 mapping where the set of letters is circular (rotational). This case is (hopefully) pretty simple.

If t=2, then we can no longer use the fact that one password has an increasing order and that one does not. This is because both p1 and p2 have letters separated by one

character that also hold a difference of 2. This means that our key, let's say k = "ij" has the opportunity to change these 'separating' values into any (random) order, and these separating values hold the conclusion we needed for t=1 to identify increasing values or schemes. Due to the structure of p2, we can not use our observations from p1 to try to understand from the ciphertext and size of the key as to which password the user might be using. They are both equally likely.

For t=3, we have lost the repetition on separated characters with a difference of two that was held for t=2 meaning that we can again use differences in p1 and p2 to make assumptions (conclusions, declarations) as to which of the two passwords the user is using. Upon encryption of p1="abcd" and p2="bedg" with a key with length 3, we can use another heuristic to determine which password the ciphertext started from. Now that the key is = "ijk", our attacker will say the password is p1 if the ciphertext follows "[a+i][b+j][c+k][d+i]". The attacker identifies this ciphertext as related to p1 as in the 1st and 4th positions we have begun repeating the key. Thus, if the attacker sees that |cipher[0]-cipher[3]|= 3, then he will know that the first and last characters have a difference of three which only happens in p1 = "abcd" as a and d are 3 apart and have repeated the key. Alternatively, if the cipher follows "[b+i][e+j][d+k][g+i]", then we can see that |cipher[0]-cipher[3]| = 5 and the password is p2="bedg", as b and g are 5 apart. This is less intuitive maybe but also makes logical sense.

For t=4, no new information can be attained, as if the key size is the same length as the text being encoded, then each character on both sides (passwords) is maintained secret and hidden from the attacker, as there is no chance to understand from the cipher as we are uniquely encrypting character-by-character in the case t = 4.
The attacker knows the exact value that is given as c, but since the attacker does not know anything about k the attacker will not be able to backtrack with just c and the pre-supposed passwords. This follows the same security for the OTP which says that as long as the key is the length of the message being encrypted, then we maintain perfect security.

(2)

The mono-alphabetic substitution cipher is easy to break with a chosen plaintext attack. The definition of perfect security follows that the ciphertext leaks no new information about the plaintext, as the message m and ciphertext c are unconditionally independent. To get the secret key, the attacker needs as many distinct chosen plaintext characters as the scheme had distinct characters in the alphabet. This will allow the attacker to determine and track letter substitution as the plaintext is replaced deterministically from the cipher. For an English message, we will need a message composed of at least 25 distinct letters to find the mappings for each letter. The final letter mapping will be inferrable, and not revealed.

However, if we want this mapping so-called ‘revealed’ (fully), we will want to use all 26 as at this point we will not need to make any inferences. The shortest chosen single-message plaintext that I could find is “the quick brown fox jumps over the lazy dog”, as it uses unique letters minimally. Mono-alphabetic substitution schemas are perfectly secure against ciphertext-only attacks when they use a randomly generated (and confidential) replacement key that is probabilistic and conditional, as opposed to deterministic (1:1 mappings), as well as when the size of the key (decryption scheme) is at least the size of the message being encrypted, amounting to the fact that in using a one-time pad (variation) or something as close as possible to it in our system helps in achieving perfect security amidst threats of frequency analysis.

2. [crib.py]~decryption algorithm of faulty (reused) OTP with crib dragging
I will use the crib-dragging resource found at taosquared.net to reveal the 11 messages currently in ciphertext (secrecy). We will use the first two of the messages to start. We will also hope that this is a two (or multi) time pad as opposed to a one-time pad.

C0 = M1 xor k
000d1a07263a376b111c3a07390b154606021a4c003c1043071704254c03091000
C1 = M2 xor k
0d0d19530674332a0b593b163101521f0a16541c1c210306160609384c0004151f

If this is a multi-time pad, we can attempt decryption by crib-dragging (guessing words positionally) on both new messages derived from XOR with the new ‘key’ of sorts generated as XOR_M1_M0. Since our key is XORed in C0 (M0 xor k) and C1 (M1 xor k), we can take the XOR between these two (C1 and C0) to get a product of just M1 and M0, with no more key involved. This takes the part of the system that we don’t and can’t know and focuses on other attack principles and vulnerabilities (most brute force attempts).

C1 XOR C0 as
XOR_C1_C0 = (M1 xor k) xor (M0 xor k) = M1 xor M0 =
And it may be most accurate to rename, XOR_M1_M0 =
0d000354204e04411a450111080a47590c144e501c1d134511110d1d00030d051f
R0 = C0 xor XOR_M1_M0, R1 = C1 xor XOR_M1_M0,
R0 = C0 xor (M1 xor M0), R1 = C1 xor (M1 xor M0)

R1, R0 are messages that are unintelligible without further cryptanalysis/ crib dragging. In this case with crib dragging, we will want to use the XOR property of reversal at (from) each possible position in either M1 or M0 (it doesn’t matter because

our key is a part of both), to find more intelligible English character sequences. We will let the resource provided above do this manual (arguably tedious) labor, and my program will do the final decoding once we decipher R0 and R1 as P0 and P1 (English messages) respectively. In this way, I used the online software to find suspected texts for P0 and P1 to ensure the product of my program was accurate.

Upon many crib-dragging guesses such as “Brennan”, “cyber”, “aREaLWAYS”, and “ and “, I realized that I had to narrow and extend my searches so I would find English readable text as a result (even if not initially appearing as such, it may just be). Based on the fact that the decodings (crib dragging) for M1 and M0 no longer are encrypted with a secret key, but instead with C0, we have developed a model that can search for a variety of terms, phrases, etc that will lead us to our plaintext P0, P1 and ultimate decryption of all 11 messages. Once we have attained the ‘key’ ~ P0, P1, as well as its size (repeatability), we will be able to use this ‘key’ through all messages as confirmation that the system is insecure and chose to reuse a one-time pad.

After some crib dragging random words into both messages, I searched “ can ” in M1 which allowed me to find “ng te” which I initially ignored as illegible text. I then figured that “can” starts a question and the most obvious word that would follow was “you”, which brought further confirmation that I was on the right track as it found ‘ridg’ in M1 which was just glaringly specific to the English language. Note from the final solution that ‘ridg’ is not involved in any of the texts, and it may be part of our key or just a complete coincidence. Given “ng” and “te”, I realized “testing” revealed both “yep I can” as well as “n read “. Putting these segments together with the guess “testing testing”, I found “yep I can read”.

Now that I had found two big parts “testing testing “ as well as “can you “, I decided to use some of my context clues. This finished process revealed the decrypted messages from crib-dragging guesses at different positions in “testing testing can you read this” which found “yep I can read you perfectly well”. Upon putting these strings into a character counter, we have found the length as 33 which matches the message space provided, which checks out well with the fact that our hex representation says each (single) ASCII character comes from 2 hex digits.

P0 = testing testing can you read this, P1 = yep I can read you perfectly well

Once we find out which message the revealed texts P1 and P0 come from (c0 or c1) via XORs with each of the suspected ciphers (c0 / c1) with any other cipher that is not the first or second message, we can take note of which English message came from which cipher. Next, we will want our ciphers to lose exclusive or on the key. To do this, we will

simply run XOR on all other 10 cipher messages (c1-c10) with c0 and xor this list of (c0 xor [c1, c2, …, c10]) with the obtained text (either P1 or P0 [in hex]), I used P0 (since I wanted continuous range [1,10]) to obtain the original messages.

Ciphs = c0 xor [c1, c2, …, c10] to obtain hex codes for c1-c10
P0_hex = hex(P0)

M2 = awesome one time pad is working
M3 = yay we talk about the answers now
M4 = i hope no student can read this
M5 = that would be quite embarrassing
M6 = luckily OTP is perfectly secret
M7 = wasnt there a catch or something
M8 = maybe yet I didnt pay attention
M9 = we should really learn about it
M10 = nah dont think we need to

3.

a. The Dual Elliptic Curve Deterministic Random Bit Generator which was introduced by the National Security Agency for inclusion in ANSI X9.82 where the standardization process began in the early 2000's. NSA paid RSA to incorporate their generator into their system for $10 million. However, the generator had a 'secret' backdoor only available to the NSA as identified by Microsoft employees, which was realized as the algorithm was extremely slow (thousands of times slower than its competitors) and contained a lot of bugs. While I certainly don't enjoy making assumptions, there is a strong correlation between the lawless state of computing and what this means for the digital enforcement of good practice and it appears in the chaotic state of the world post 9/11, members affiliated with the US government used this lens of 'national security' to take new domain on the information they are provided.

This move by the NSA x RSA showed the declaration the US government made to take over as much data from as many streams as possible. Based on the presentation by Around 2007 two Microsoft workers, who were tasked with implementing the Dual_EC_DRBG, found that NSA affiliates had a perfectly misleading combination of information and implementation within their generator such that NSA would have access to others' use of the RBG (making the system completely insecure with a loss of confidentiality to the US government on the new internet encryption standard). After this (around 2014), NIST removed the

backdoor for Dual EC DRBG from the standard and urged users to switch to another of the available (approved) encryption schemes.

b. A major ethical concern regarding the system is the base of trust between the government and companies. Affiliates with the government said that the agency "fully support[s] and [would] not undermine efforts to create encryption standards" (Wiki). Notably, this is a very hypocritical thing to say after they had infiltrated and gained access to all systems using the encryption standard that they implemented in ANSI X9.82. It is difficult to believe that the agency did not undermine international internet user security when they actively had prime access to all data transmission that was following their protocol.

In this way, there is a major concern about totalitarianism and surveillance technologies that would allow governments and corporations alike to gain access to data they say that they don't, and in doing one thing but saying another. This brings an interesting, extremely crucial, and hypo-understood discussion on digital user privacy that continues to unfold in the surveillance world, both in discussion for government entities and most recently and relevantly in the capitalist system for corporations and conglomerates that similarly seek profits and insights at the negligence of user decision rights and the unfounded, unprecedented and unrecognizable territory of computing security.

While this was many years ago, and surveillance technologies both on mobile and other ubiquitous devices have certainly advanced by multiple folds, we should be appalled at the lack of ethical standards that the NSA perpetrated onto the world community, the dangers and possibilities it could have fostered, and the reality of the domain it opened for other entities to launch attacks and make declarations on private user data ownership as a (very profitable) business model for neoliberal capitalists.

c. NIST guidelines aim to place many stakeholders into the decision-making of cryptographic technologies for our country. These guidelines aim to increase transparency and collaboration in the ideation, development, and implementation of our security standards. NIST Cryptographic Standards and Guidelines Development Process from March 2016 provides some key pillars that the field must follow. Some of these include openness, balance, integrity, continuous improvement (questioning), and global acceptability.

Given the amount of time it took for the realization of the backdoor until the system was taken off of the internet standard (about 7 years), the guidelines document provides the general question for when an issue is large enough to be taken up on, "Does it appear to be a matter that the communities of interest

consider to be both important and practical to address?" which may be able to make flaws, or in this case these designed insecurity features by NSA, easier and faster to fix in the future. When guidelines like the one provided by the NIST are not applied to the field, we can see that any 'Eve' or malicious attacker will be able to intercept data transmissions. For this reason, it must be assumed by the field that all attackers have unlimited computational resources (bounded by nothing) and are generally much 'smarter' than you (in terms of cryptanalysis) with the worst possible intentions. This is the inherent motivation for the security field to always be ahead of the attackers. This model does not work without proper recognition of entities becoming too powerful as with the RSA implementation of the backdoor encryption algorithm.

It doesn't matter if the entity is the US government which has its own nationally and domestically embedded intentions (arguably good or bad, depending on who you ask), or a banking hacker (which is generally always viewed as malicious), as all insecurities give rise to others to find such backdoors. In this case, it is fortunate that those who realized the backdoor had good intentions and handled the situation very properly, with a rebuttal of information and education. While many people would find themselves stressed with so much information, certain entities would use their gained knowledge to create the most user harm. It should be noted that it took a very long time for the computing society to realize the insecurity their technology rests on, which is a worrisome fact that insecurities are founded after their implementation and therefore require some (unprecedented) territory to handle. If this territory has not been defined, entities can and will make their own decisions irrespective of societal interest.

The diversification of stakeholders means that no one organization will be (as easily) able to implement and deceive the rest of the community as was done in this case. In this sense, it is integral for the maintenance of cryptography to follow the guidelines discussed in this document, other guidelines motivated by other periods and projects (security flaws), and the general (and universal) laws that people follow inherently based on fostering a good community. Generally, and arguably more important than making a single guideline, it is most important that meetings and conventions on these guidelines and policies happen every so often to ensure the update and progressive attitude of the cryptographic community against the growing intelligence of attackers. This shows the human understanding that a guideline or policy may not be able to stop these unethical behaviors and system implementations, but constantly having these conversations, asking these questions, and making updates to codes of conduct, policies, and guidelines.