CPSC 329 Assignment 2
David Ng
30009245
T04

1.

1. In February 2016, the FBI requested that Apple create software that would allow the FBI to unlock an iPhone 5C that they had recovered belonging to one of the shooters in a terrorist attack in San Bernardino. This is the most well-known case of several wherein Apple had received orders issued by United States district courts that sought for Apple to allow the authorities to retrieve sensitive information from their phones in order to aid federal investigations. These orders had essentially demanded that Apple provide the government with software that would allow the government to bypass the security on those devices, creating an unprecedented level of access. While most of the orders were for unlocking iPhones running iOS 7 and older, a few of the requests concerned phones with additional security features that Apple had implemented since 2015. These security measures were more advanced, to the point that Apple could no longer comply with government warrants that sought to retrieve user information from the devices, lest undermining their entire efforts to protect customer security and privacy.

After the 2015 San Bernardino attack, the FBI had recovered at iPhone 5C belonging to Syed Rizwan Farook, one of the shooters involved with the attack that had killed 14 people and injured another 22. The FBI attempted to access the phone that was protected with a four-digit password, and that permitted only 10 incorrect attempts before erasing the AES encryption key that protects its stored data. After seeking aid from the National Security Agency and still failing to unlock the phone due to its advanced security features that encrypt user data, on February 9, 2016, the FBI asked Apple to create a new operating system that could be installed and run on Apple devices that disabled the security measures. Apple declined the order, stating that they had provided data to the FBI whenever they had data in their possession. Furthermore, they have complied with valid subpoenas and search warrants, and have made Apple engineers available to advise the authorities. After the FBI responded with a court order making unprecedented use of the All Writs Act of 1789 to justify an expansion of their authority, Apple and Tim Cook promptly reaffirmed their position with a publicly accessible letter to their customers on February 16, 2016, outlining the need for encryption and the threat to data security this would impose were they to unlock iPhones for law enforcement agencies.

2. One reason to support the FBI would be so that that FBI could proceed with a professional investigation for the fourteen people who were killed, and the many others who were injured. This reason makes sense, as the phone may hold clues to finding more terrorists. This reason to uphold justice and seek retribution from those responsible lies at the core of the FBI's position. The FBI want to do everything that they can under the law to investigate the San Bernardino shooting. By doing everything that they can to investigate the circumstances surrounding the shootings, they are providing the American people with an explanation on how this could have happened and what can be done to prevent similar incidents in the future. In other words, it is their way to show that they care, and will do whatever it takes to bring those responsible to justice. Aside from bringing closure to those killed or injured, access to this phone could provide the FBI with knowledge of how to prevent similar attacks in the future. This may allow them to uncover more terrorist plots and foil them before they can happen. The possibility of saving more lives is another such reason to support the FBI, as we all want to feel safe and protects in public places.

On the other hand, there are also many reasons to support Apple's position to withhold from providing the FBI with the ability to unlock the phone. One reason is that this order to create a backdoor undermines Apple's efforts to improve security features on their iPhones. By asking Apple to remove security features and provide them with a way to attack iPhone encryption, Apple is basically forced to intentionally weaken their products, possible compromising the security of all iPhone users. If Apple were to create such a backdoor, there is no way to ascertain that this will not fall into the wrong hands. To ensure that such a powerful tool is not abused, is simply to not create it. Furthermore, if Apple were to create this technique to remotely unlock their iPhones, this is essentially the creation of a master key. If this master key were to fall into the hands of hackers or cybercriminals, this could be a major cause of concern, as our privacy could be compromised. Another reason to support Apple is that the creation of a unique OS that could bypass security protections on the iPhone sets a legal precedent that expands the powers of government beyond what we have encountered thus far. We need to truly consider whether mass surveillance, conversation recording, and location tracking of innocent individuals by the government is something that should be permitted in the future should we cross this line by unlocking an iPhone at their behest.

The sources for the first question are those suggested in the assignment. These sources are reliable because they come from trusted sources (the parties involved themselves) and dated. These articles were written by Tim Cook, the CEO of Apple, and FBI Director James Comey. While they may hold a biased opinion of the issue, they nonetheless reveal the events and circumstances leading to the conflict.

2.

1. Since my name is David, converting this to binary code gives:

D = 00011
A = 00000
V = 10101
I = 01000
D = 00011

Therefore, the binary string M corresponding to my first name is M = 00011 00000 10101 01000 00011. To encrypt this string M to obtain cipher text C1 = M XOR K, we can first pad with some extra zeroes, or simply take only the first 25 binary digits of K. For this question, we will simply pad with extra zeroes and ignore them when we convert back to plaintext. Performing a bitwise XOR with K = 11010 10010 00100 01011 10100 10111 00101 10010, we get

00011 00000 10101 01000 00011 00000 00000 00000
XOR 11010 10010 00100 01011 10100 10111 00101 10010

11001 10010 10001 00011 10111 10111 00101 10010

Therefore, the cipher text C1 = 11001 10010 10001 00011 10111 10111 00101 10010.

2. We can recover M by using C1 XOR K. Performing this operation, we get

11001 10010 10001 00011 10111 10111 00101 10010
XOR 11010 10010 00100 01011 10100 10111 00101 10010

00011 00000 10101 01000 00011 00000 00000 00000

Thus, by performing C1 XOR K, we can successfully decrypt the cipher text and recover the original message M.

3. We recall that the cipher text C1 = 11001 10010 10001 00011 10111 10111 00101 10010. If we replace the first 5 bits of C1 with 10101, we obtain C2 = 10101 10010 10001 00011 10111 10111 00101 10010. To find the plaintext M2 corresponding to C2, we once again use C2 XOR K to decrypt C2 in order to obtain M2. Doing so gives

```
      10101 10010 10001 00011 10111 10111 00101 10010
XOR   11010 10010 00100 01011 10100 10111 00101 10010

      01111 00000 10101 01000 00011 00000 00000 00000
```

Thus, M2 = 01111 00000 10101 01000 00011 00000 00000 00000. Comparing the plaintext M2 with M, we find that only the first 5 bits have changed. Instead of being 00011, the first 5 bits are now 01111. If we were to convert this back into alphanumeric characters using MyCode, this takes D = 00011 to P = 01111. Therefore, if we changed the first 5 bits of the cipher text C1 to C2, we change the first character in the decrypted message from D to P. The rest of the message is preserved. This makes sense, since we are performing a bitwise operation between the bits representing the message and the bits representing the cipher text. There is no relation with bits having an influence on many different bits, so by only changing the first 5 bits corresponding to the original cipher text, only the first 5 bits of the original message should change.

4. No, the security of the encryption system cannot be improved by using double encryption with two randomly chosen keys. We can recall from Assignment 2 that XOR is an associative operation. That is, given three strings A, B, and C, we find that A XOR (B XOR C) = (A XOR B) OR C. In this example, if we try to compose a cipher text by using two key K1 and K2 to obtain C = (M XOR K1) XOR K2, this is the same as C = M XOR (K1 XOR K2) by associativity principle. But then, K1 XOR K2 just results in another string, which we denote K3. The cipher text can then be expressed as C = M XOR K3. We can see then, that by using double encryption with two randomly chosen keys, this does not improve the encryption system since this is equivalent to having originally chosen the key K3 instead. This assumes that the question uses these two keys and performs the XOR operation on the message and the two keys as presented in the question. The fact that they are 20 bits long does not change the fact that they offer no additional security compared to having used another key K3 that is also 20 bits long. All this does compared to a key that is 40 bits long in total is reduce the maximum possible size of the message from 8 characters to 4 characters according to MyCode. One could loop the key when they have exhausted the maximum character limit, but this would be less secure compared to a longer random key, since the key is now being repeated for the length of the message, and is not a random string that is the same length of the message.

3.

1. We are given that X1 = 56, X2 = 23, K1 = 12 and K2 = 13. We first apply F(X2, K1) to get F(X2, K1) = (23 + 12)^2 mod(64) = 35^2 mod(64) = 1225 mod(64) = 9. We then apply the XOR to F(X2, K1) = 9 with X1 = 56. 9 in binary is 001001, and 56 is 111000. 001001 XOR

111000 is 110001. Converting to decimal numbers, this is 49. Thus, 9 XOR 56 = 49. This is now Y2. Y1 is just X2. Thus, the intermediate result is (23, 49). We then repeat the procedure with our new pair (23, 49) and K2. thus, F(Y2, K2) = (49 + 13)^2 mod(64) = 62^2 mod(64) = 3844 mod(64) = 4. We now apply XOR to this result with Y1 = 23. 4 in binary is 000100, while 23 is 010111. 000100 XOR 010111 = 010011, which is 19. Thus, 4 XOR 23 = 19. This is now Z1. Z2 is just Y2, which was 49. Therefore, the cipher text (Z1, Z2) is (19, 49).

2. We now use the decryption algorithm. We first use F(Z2, K2) to get F(Z2, K2) = (49 + 13)^2 mod(64) = 62^2 mod(64) = 3844 mod(64) = 4. Now, we XOR this result with Z1 = 19. Previously, we have determined that 4 XOR 23 = 19, so 19 XOR 4 = 4 XOR 19 = 4 XOR (4 XOR 23) = 23. This is Y2. Y1 is simply Z2, which was 49. Thus, our pair is (49, 23). We now compute F(Y2, K1), which gives us F(Y2, K1) = (23 + 12)^2 mod(64) = 35^2 mod(64) = 1225 mod(64) = 9. Now, we XOR 9 with Y1 = 49. But 9 XOR 56 = 49, so by the same logic as above, we find that 9 XOR 49 = 56. This is X1, while X2 is just Y2 = 23. Thus, (X1, X2) = (56, 23). We have therefore used the decryption algorithm to decrypt the cipher text and recover the original plaintext. The decryption algorithm works to recover the plaintext from the cipher text, as shown.

3. We recall that Z1 = 19 and Z2 = 49. By flipping the least significant bit of Z1, we have changing Z1 = 19 to Z1' = 18. To find the plaintext corresponding to (Z1', Z2), we apply the decryption algorithm with (K1, K2). First, we find that F(Z2, K2) is the same as before since the values have not changed, so the result is 4. Now, we XOR 4 with Z1' = 18. This gives us 22 instead of 23. Therefore, Y2' = 22. Y1' is just Z2, so Y1' = 49. Thus, our pair is (49, 22). Now, we find F(Y2', K1) = (22 + 12)^2 mod(64) = 34^2 mod(64) = 1156 mod(64) = 4. We XOR this with Y1' = 49. 4 in binary is 000100 and 49 in binary is 110001. 000100 XOR 110001 is 110101, which is 53 in decimal form. Therefore, X1' = 53. X2 is just Y2', so X2' = 22. Thus, (X1', X2') = (53, 22).

4. The original plaintext (X1, X2) = (56, 23), while the new plaintext is (53, 22). Converting these to binary representations, this becomes X1 = 111000, X2 = 010111, X1' = 110101, and X2' = 010110. Comparing the number of bits that remain the same, we could that there are 8 bits that remain the same as the original plaintext. That is, 4 bits are different. We recall from Question 2.3 that changing the first 5 bits of the cipher text changed 5 bits in the corresponding plaintext. Thus, the ratio of the number of bits changed in the plaintext to the number of bits changed in the cipher text for the result of 2.3 was 5 / 5 = 1, while the ratio using the Simple Feistel algorithm was 4 / 1 = 4. This means that the new plaintext was changed more dramatically compared to the original plaintext using the Simple Feistel algorithm than to using the scheme of 2.3. Additionally, while only the bits of the cipher text that were changed were changed in the corresponding plaintext in 2.3, we see that although just Z1 was changed to Z1', this has produced an effect on both X1' and X2'. Since we desire pre-image resistance, the larger the ratio, the better and more secure the algorithm is for use as a cryptographic hash function. The fact that changing a small thing in the cipher text causes a large change to the resulting plaintext when deciphered is a desirable property for encryption systems. In other words, fewer bits in the calculated plaintext that remain the same is better for encryption systems, since it means that a brute force attack is the only viable way to determine the original plaintext message from the cipher text. For instance, even if the attacker had knowledge of a given (plaintext, cipher text) pair, they cannot easily deduce the plaintext of another cipher text that differs from the one they have, since even a small change in the cipher text could lead to drastic changes in the plaintext that is deciphered.

5. First, we need to convert the counter 123 into binary. Doing so, we obtain 000001111011. We now split this into two 6 bit numbers to obtain X1 = 000001 and X2 = 111011 in binary. Converting these values to decimal, we have X1 = 1 and X2 = 59. We can now apply the SimpleFeistel algorithm with K1 = 12 and K2 = 13. F(X2, K1) = (59 + 12)^2 mod(64) = 71^2 mod(64) = 5041 mod(64) = 49. By appyling XOR with X1, we get 49 XOR 1 = 48. Thus, Y1 = X2 = 59, and Y2 = 48. F(Y2, K2) = (48 + 13)^2 mod(64) = 61^2 mod(64) = 3721 mod(64) = 9. Applying XOR with Y1, we get 9 XOR 59 = 50. Thus, Z1 = 50 and Z2 = Y2 = 48. Converting 50 and 48 to binary, we get 110010 and 110000 respectively. We now concatenate the two to get 110010110000. Converting this to decimal, we get that our first pseudo-random number is 3248.

We now increment the counter to 124. In binary, this is 000001111100. This gives us X1 = 000001 and X2 = 111100. Converting to decimal, X1 = 1 and X2 = 60. F(X2, K1) = (60 + 12)^2 mod(64) = 72 ^2 mod(64) = 5,184 mod(64) = 0. Applying XOR with X1 is just X1 = 1. Thus, Y1 = X2 = 60, and Y2 = 1. F(Y2, K2) = (1 + 13)^2 mod(64) = 14^2 mod(64) = 196 mod(64) = 4. Applying XOR with Y1 = 60, we get Z1 = 56. Z2 = Y2 = 1. Converting to binary, Z1 = 111000, while Z2 is 000001. Concatenating, we get 111000000001. Converting to decimal, we find that the second pseudo-random number is 3585.

To obtain our last pseudo-random number, we increment the counter to 125. In binary, this means that X1 = 000001 and X2 = 111101. Converting to decimal, X1 = 1 and X2 = 61. F(X2, K1) = (61 + 12)^2 mod(64) = 73^2 mod(64) = 5329 mod(64) = 17. Applying XOR with X1 = 1, we get Y2 = 16. Thus, Y1 = X2 = 61, and Y2 = 16. F(Y2, K2) = (16 + 13)^2 mod(64) = 29^2 mod(64) = 841 mod(64) = 9. Applying XOR with Y1 = 61, we get Z1 = 52. Z2 = Y2 = 16. Converting to binary, Z1 = 110100, while Z2 = 010000. Concatenating, we get 110100010000. Converting to decimal, we find that the third pseudo-random number is 3344.

4.

1. To find the hashed value of the message M = 1010100101, we need to split this message into blocks. Since the function f takes y1 and y2, where both are 4 bit positive integers, we shall split the message M into M = 1010 1001 01. This is incomplete, so we pad M to M = 1010 1001 0100 (we pad the last block of the message with 0's as described in the question). To harden the hash even further, the length of the message is added as an extra block. M was originally of length 10, which is 1010 in binary. Thus, adding this padding to M, we obtain M = 1010 1001 0100 1010. Now that we have he proper padding, we can find the hash value of this message using the given initialization vector and compression function. We will treat the initialization vector and the message M as being values expressed in binary, and the 11 that is added at the end of the compression function to be in decimal.

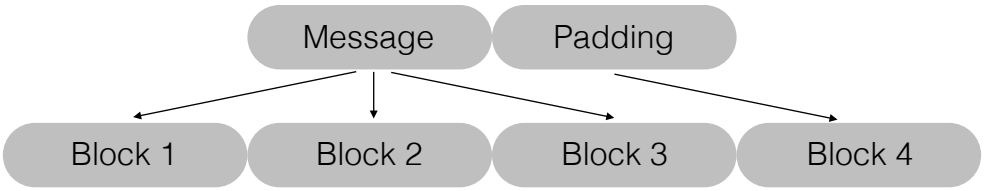
We want to first compute f(1111, 1010). These values are 15 and 10 in decimal. Thus, (15^2 + 10^2 + 11) mod(15) = 6. Block 2 = 1001 is 9 in decimal, so f(6, 9) = (6^2 + 9^2 + 11) mod(15) = 8. Block 3 = 0100, which is 4 in decimal, so f(8, 4) = (8^2 + 4^2 + 11) mod(15) = 1. Block 4 = 1010, which is 10 in binary, so f(1, 10) = (1^2 + 10^2 + 11) mod(15) = 7. Therefore, converting back to binary, the hash value of the original message M is 0111.

2. Flipping the fourth bit in message M, it becomes 1011100101. Once again, its length is 10, and it also needs to be padded with two extra 0's. Thus, M becomes M = 1011 1001 0100 1010. We once again compute the hash value. The initialization vector 1111 is 15 in decimal, while 1011 is 11 in decimal. f(15, 11) = (15^2 + 11^2 + 11) mod(15) = 12. Block 2 = 1001, which is 9 in decimal, so f(12, 9) = (12^2 + 9^2 + 11) mod(15) = 11. Block 3 = 0100, which is 4, so f(11, 4) = (11^2 + 4^2 + 11) mod(15) = 13. Block 4 = 1010, which is 10, so f(13, 10) = (13^2 + 10^2 + 11) mod(15) = 10. Therefore, in binary, the hashed value is 1010. We note that 3 bits have changed from the original M = 0111.

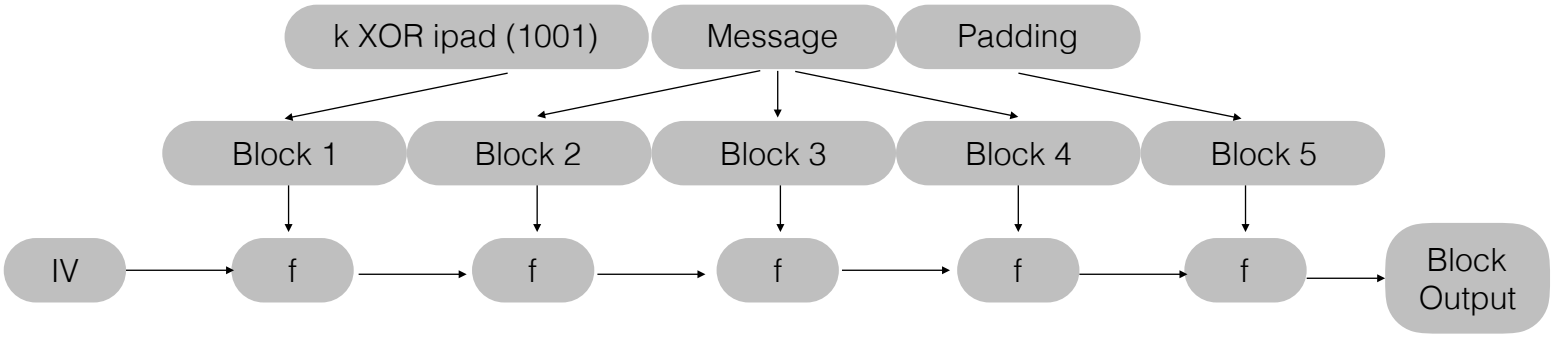
3. The question states that the diagram needs to include input and output of each block, and any additional parameters that are needed for the computation. We will interpret this to mean that we need to clearly identify where the output and input of each block occurs in the diagram, instead of substituting real values. That will be reserved for Part 4 of this question. We will construct tinyHMAC. We know that for a secret key k which is the same length as the input block size, and a message m, we have

$$\text{tinyHMAC}(k, m) = h((k \text{ XOR opad}) \mid h((k \text{ XOR ipad}) \mid m))$$

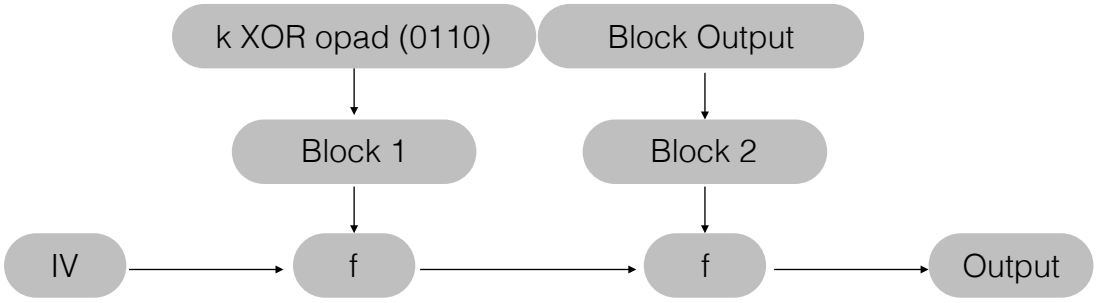
First, the message is padded accordingly. From the previous parts of the question, we know that 10 bit long messages take up 4 blocks including the padding that is added. This looks like the following:



We now have to concatenate to the beginning of this message k XOR with ipad, which form a block before the message. Thus, we concatenate them before applying hash function:



The result from the above is h((k XOR ipad) | m). The output of this is a single block. We will call this Block Output. This is concatenated with k XOR opad before the hash function is applied again.



The output of this is the result of tinyHMAC(k, m).

4. We first calculate that k XOR opad = 1010 XOR 0110 = 1100 and k XOR ipad = 1010 XOR 1001 = 0011. Converting from binary to decimal, we find that the initialization vector IV =

15, k XOR opad = 12, and k XOR ipad = 3. As before, the padded message is M = 1010 1001 0100 1010. According to the above diagram, we first calculate Block Output. IV and k XOR ipad from Block 1 are fed into f, giving f(15, 3) = (15^2 + 3^2 + 11) mod(15) = 5. Block 2 = 1010, which is 10 in decimal, so f(5, 10) = (5^2 + 10^2 + 11) mod(15) = 1. Block 3 = 1001, which is 9 in decimal, so f(1, 9) = (1^2 + 9^2 + 11) mod(15) = 3. Block 4 = 0100, which is 4, so f(3, 4) = (3^2 + 4^2 + 11) mod(15) = 6. Block 5 = 1010, which is 10, so f(6, 10) = (6^2 + 10^2 + 11) mod(15) = 12. This is Block Output. We now evaluate f(IV, k XOR opad) = (15^2 + 12^2 + 11) mod(15) = 5. Block output was 12, so f(5, 12) = (5^2 + 12^2 + 11) mod(15) = 0. Therefore, by using the above tinyHMAC to calculate, we have found that the tag of the above message M when k = 1010 is 0000.

5. The first design (a) does not offer much improved security or efficiency. Originally, tinyHMAC(k, m) produces as output a 4 bit value. Since it applies the hash function h() last, this forces the output to be in the range of 0000-1110. This is because the hash function is simply a repeated application of the one-way compression function, that can only generate numbers from 0 to 15. It cannot be 1111, since 15 mod(15) = 0. Since we now apply tinyHMAC(k1, m) XOR tinyHMAC(k2, m), we can now obtain 1111 (Consider for instance that tinyHMAC(k1, m) produced 1110 and tinyHMAC(k2, m) produced 0001). Therefore, as opposed to the original 1/15 chance that one could randomly correctly guess the output, the first design (a) that uses XOR afterwards makes it a 1/16 chance. Security has improved very slightly. Efficiency remains the same, as we still use 4 bits for each message. Thus, design (a) retains the same efficiency while slightly increasing the security.

The second design (b) is a concatenation of two tinyHMAC calculations. It is clear that this reduces efficiency, as it takes two 4 bit values and concatenates them, now requiring 8 bits. Compared to the original 4 bits, we now require an additional 4 bits for each message. In terms of security, we can make the simplifying assumption that tinyHMAC(k1, m) and tinyHMAC(k2, m) both have 16 possibilities. Thus, 16*16 = 256 different possibilities. This means that an attacker now has a success chance of 1/256 compared to the original 1/15. This is a drastic improvement. We can also calculate the different possibilities more carefully without making the simplifying assumption. We split the cases into whether or not the first bits of the two sets of four are 1 or 0. If they are both 0, the remaining 6 bits can be 2^6 different possibilities. If one of them is 1, then the bits of the tinyHMAC where 0 is leading can be 2^3 = 64 different possibilities, while the bits of the tinyHMAC where 1 is leading can only be 7 possibilities (last bit cannot be 1). Thus, 8*7 = 56. We multiply by 2 since either the first set or second set can start with a leading 1. This is 11 possibilities. The last case is when both sets are leading with 1. We have 7 * 7 = 49. Adding these possibilities, we have 225 possibilities, which means that the attacker has a success chance of 1/225, which is still much better security than the original. This can also be calculated as 1/(15 * 15). Thus, the second design (b) has much better security, but is less efficient since it requires more bits.

6. We want to make the hash pre-image resistant to 40 bit security, so the success chance needs to be 1/2^(40) = 1/1099511627776. The hash output therefore has to be at least a length of 40. At a length of 40, if we allow each bit to be 1 or 0, we have 2^40 different possibilities for the hash output. Thus, the success chance is 1/2^(40). Therefore, the smallest size of hash output that would allow for 40 bit security is when the hash output is at a length of 40. Here, we assume that the hash output can be any of the 2^40 possibilities when each bit can be either 0 or 1.