Clayton Cohn

Prof. John Rogers

CSC 440 - 810

16 Mar 2019

Final Exam

**Short Answer**. For the short answer portion of the exam, I chose the Twofish cryptosystem. At its core, the Twofish cryptosystem is a symmetric block cipher. It has a block size of 128 bits, and takes a key of any length up to 256 bits (NIST required all competing algorithms to accept 128, 192, and 256-bit keys) (Schneier 3). It outputs 128 bits of ciphertext—the same size as the plaintext block. Twofish *is* a Feistel network. Each round includes half of the block being sent through some function, the result of which is XORed with the other half of the block (Schneier 4).

The Twofish encryption algorithm is as follows. First, we start with our 128-bit block of plaintext, and our (up to) 256-bit key. The key generates 40 subkeys via a key schedule. 8 of the subkeys will be used in the "whitening" (XORing) process—4 right after the initial plaintext input, and 4 right before the final ciphertext output. The other 32 keys will be used during the cipher's 16 rounds—2 per round. The plaintext is initially separated into 4 32-bit "words," and each word is initially whitened with one of the subkeys before the rounds begin (as mentioned above). Because Twofish is a Feistel network, there is a left side and a right side. The two left-most words make up the left side, and the two right-most words make up the right side. To start, one of the two left-side words is rotated left 8 bits. Then, each left-side word is separately input to the g function. The g function consists of four 8-bit S-boxes, followed by a "linear mixing step" based on the MDS (Maximum Distance Separable) matrix (Aparna IV). The two results of the g function are then combined via PHT (Pseudo-Hadamard Transform), and the two subkeys for that round are added (mod 2^32). On the right side, one of the words is rotated left 1 bit, and then both of the right-side words are XORed with the two results from the left side. The right-side word that did not previously rotate then rotates to the right 1 bit. At this point, the right side becomes the next round's left side, and the original left side becomes the next round's right side. There are 16 rounds in all. After the final round, all four words go through one more round of whitening with the remaining four subkeys, after which all four words are concatenated, and final ciphertext returned.

Confusion occurs in the cryptosystem during the whitening process, where the key is split into subkeys, and those subkeys are XORed with the text block before the first round and also after the last round. The S-boxes also create confusion, as S-box selection is key-dependent. The Twofish key schedule was specifically designed to "prevent related-key attacks and to provide good key mixing" (Schneier 12).

Diffusion occurs in the mixing via the MDS matrix, which was chosen specifically to "provide good diffusion" (Schneier 10). Also, the PHT, along with the key addition, "provide diffusion between the subblocks and the key" (Schneier 11). The various bit shifts also aid in diffusion.

The cipher's nonlinearity is derived from the S-boxes. Like many block ciphers, Twofish utilizes S-boxes as a "non-linear fixed substitution operation" (bdimciu 6). The S-box operation cannot be put into a linear equation, making the cipher itself nonlinear.

The Twofish cipher was designed by: Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Bruce Schneier is considered to be the primary designer (Wikipedia Twofish). He is a "cryptographer, computer security professional, privacy specialist and writer" (Wikipedia Bruce Schneier). He is from Brooklyn, New York, and is currently a fellow at the Berkman Center for Internet & Society at Harvard Law School. He received a bachelor's degree in physics from the University of Rochester, followed by a Master's in Computer Science from American University, and was later awarded an honorary PhD from the University of Westminster. He has worked for Harvard University, Counterpane Internet Security, Bell Labs, U.S. Department of Defense, and BT Group (Wikipedia Bruce Schneier).

Works Cited (short answer)

bdimciu. “Twofish Encryption Algorithm on ZYBO.” *Instructables*, Instructables, 19 Sept. 2017, www.instructables.com/id/Twofish-Encryption-Algorithm-on-ZYBO/.

K, Aparna, et al. “A Study of Twofish Algorithm.” *IJEDR*, Volume 4, Issue 2, 2016, www.ijedr.org/papers/IJEDR1602023.pdf.

Schneier, Bruce. “The Twofish Encryption Algorithm.” *Schneier on Security*, 1998, www.schneier.com/academic/archives/1998/12/the\_twofish\_encrypti.html.

Wikipedia Contributors. “Bruce Schneier.” *Wikipedia*, Wikimedia Foundation, 13 Mar. 2020, en.wikipedia.org/wiki/Bruce\_Schneier.

Wikipedia Contributors. “Twofish.” *Wikipedia*, Wikimedia Foundation, 6 Oct. 2019, en.wikipedia.org/wiki/Twofish.

**Monoalphabetic vs. polyalphabetic**. The first thing I would do would be to see which characters composed the ciphertext. If, for example, the entire ciphertext was made up of the letters "ADFGX," I think it'd be pretty clear which cipher was used (or at least that it was polyalphabetic); furthermore, if the number of types of letters was significantly less (or more) than the number of letters in the alphabet, we could be relatively sure that the cipher used was polyalphabetic. It is, after all, the one-to-one relationship that makes a cipher monoalphabetic. Otherwise, in the absence of this glaring obviousness, I would run the ciphertext through frequency analysis. If it's monoalphabetic, it will have a distribution that falls in line with the distribution of the letters' natural occurrence in language. If it's polyalphabetic, the distribution will be much more uniform. This obviously assumes that there is enough ciphertext to create a reliable distribution to begin with. It also assumes that a polyalphabetic cipher's ciphertext doesn't give us a monoalphabetic frequency distribution (and vice versa) just by chance.

**Breaking Vigenére with a crib**. Using the (attached) Python code I utilized in the first and second assignments, the decrypted plaintext is:

I AM MYSELF INCLINED TO THINK THAT DECIPHERING IS AN AFFAIR OF TIME, INGENUITY, AND PATIENCE; AND THAT VERY FEW CIPHERS ARE WORTH THE TROUBLE OF UNRAVELLING THEM. ONE OF THE MOST SINGULAR CHARACTERISTICS OF THE ART OF DECIPHERING IS THE STRONG CONVICTION POSSESSED BY EVERY PERSON, EVEN MODERATELY ACQUAINTED WITH IT, THAT HE IS ABLE TO CONSTRUCT A CIPHER WHICH NOBODY ELSE CAN DECIPHER. I HAVE ALSO OBSERVED THAT THE CLEVERER THE PERSON, THE MORE INTIMATE IS HIS CONVICTION. IN MY EARLIEST STUDY OF THE SUBJECT I SHARED IN THIS BELIEF AND MAINTAINED IT FOR MANY YEARS.

The key is, "PHILOSOPHER," and the passage is from Charles Babbage's *Life of a Philosopher*.

**RSA encryption and signature**. This problem was done in the attached python file. The results are below:

=RSA=

Cohn, Clayton

n 8187300668217788611846832210295454261

e 3

cm 250321993341100643804362776828247673629

cs 141993753745530359302700541300342466938

**Finding p and q**.

Use φ(n) = (p-1)(q-1) = pq-p-q+1 to determine p+q.

φ(n) = (p - 1)(q - 1) = pq - p - q + 1 = pq - (p + q) + 1

We know that n = pq, which makes φ(n) = n - (p + q) + 1.

φ(n) = n - (p + q) + 1

Thus,

p + q = n + 1 - φ(n)

Use (p-q)2 = p2-2pq+q2 to find p-q. (Hint: Add 4pq-4pq to the right side).

(p - q)^2 = p^2 - 2pq + q^2

= p^2 - 2pq + q^2 + 4pq - 4pq

= p^2 + 2pq + q^2 - 4pq

= (p + q)^2 - 4pq

= (p + q)^2 - 4n

Thus,

p - q = sqrt((p + q)^2 - 4n)

pq\_plus = p + q

pq\_minus = p -q

pq\_plus + pq\_minus = (p + q) + (p - q) = 2p

Thus,

p = (pq\_plus + pq\_mius) / 2

pq\_plus - pq\_minus = (p + q) - (p - q) = 2q

Thus,

q = (pq\_plus - pq\_minus) / 2

We are given that :

n = 898607526590969848863184322603417866026220164569611859928589

phi(n) = 898607526590969848863184322601520436048324820954773803576156

It then follows, that:

p = 986828026913848038774774981167

q = 910601950981495576063281371267

Your answer will be the values of p and q in a text block with the format:

=PQ=

Cohn, Clayton

p 986828026913848038774774981167

q 910601950981495576063281371267

**Meet in the middle attack**. For this problem, I did the work in the attached Python file via my code from assignments 4 and 5.

=MITM=

Cohn, Clayton

DESkey [152,49,315,271]

Shift [0,1,5,11]