IDEA with CTR

Richard Kirchofer

April 17, 2015

1 IDEA

IDEA stands for International Data Encryption Algorithm. It was published in 1991 by Xuejia Lai and James L. Massey. IDEA is a block cipher encryption algorithm that is capable of encrypting 64 bits at a time and uses a 128 bit key. With each round, IDEA generates new keys from the master key. The 64 bit block of input is run through 17 rounds of idea encryption. At this point, the result should be completely dissimilar to the original.

2 Key Generation

2.1 In Theory

IDEA takes a 128 bit initial key. From this, it generates 52 round keys. There are 17 total rounds of IDEA. Every odd round uses four round keys and every even round uses two round keys. This means that there are nine rounds which need four keys and eight rounds which need two keys. In total, IDEA needs 52 round keys. Each round key is 16 bits. All of the 52 round keys are pulled from the initial 128 bit master key. IDEA creates these round keys by pulling chunks of 16 contiguous bits from the master key starting with different offsets. The offset starts at 0 and eight, 16 bit keys are made. The offset advances by 25 and eight more keys are made. The round key wraps around from the end of the master key to the beginning. This continues until 52 round keys have been made.

2.2 In Practice

IDEA has a function called key_expansion that expects a list of strings and returns a list of integers. The list of strings that is passed to the key_expansion function represents the 128 bits of the master key. Each of the strings is only one character long and is either a '1' or a '0'. In this way, the master key can easily be sliced into 16 bit chunks. When the chunks of 16 bits have been cut out of the master key, they are saved as a list of lists. At this point, there are actually more than 52 round keys because eight are added with each loop but only four are needed on the last iteration. As a solution, last four round keys are discarded leaving just 52 round keys in the list. Each of the sublists are then joined together to make just a single list of strings. Each of the strings are 16 bits long and there are 52 of them. Now each of the round key strings are converted to integers. Python provides an int() function that accepts a string of characters and an integer representing the current base of the string. The function returns the base ten decimal value of the given string. All of the strings are converted to integers and then as a last step, keys 49 and 50 are swapped. These are keys 49 and 50 when the first key is addressed as zero and the last key is key number 51.

3 Operations

There are three basic operations for manipulating the data in IDEA.

3.1 Exclusive Or

The exclusive or operation is a popular operation among encryption algorithms. It it a bit-wise operation applied between two bits at a time. In this implementation of IDEA, all of the operands are 16 bits long and can be XORed with any of the other operands. The result of an XOR operation can easily be reversed by XORing the output with either of the original operands. In this implementation, 16 bit segments of data are represented by integers in the range of 0 to 2**16. These integers may be XORed with each other using Python's built in bit-wise XOR operation. The result of an XOR operation will never require more bits than that of its operands because it works bit per bit.

3.2 Addition

IDEA uses addition as another of its operations to manipulate data. Because the data may not exceed 16 bits in length, any values that carry over 16 bits are discarded. This is achieved by calculating the modulus of the result against the value of 2**16. Besides this, IDEA addition is the same as regular addition.

3.3 Multiplication

Multiplication in IDEA also has a modification to ensure 16 bit results. While the full multiplication is calculated between the two values, only the remainder when divided by 2**16+1 is saved as a result. This serves two purposes. The modulus of the result guarantees that the result can be represented by 16 bits. This also guarantees that the operation is reversible. 2**16+1 is a prime number and any number modulus a prime number has an inverse. All of the modulus multiplication is reversible.

4 Odd Round

4.1 In Theory

The odd rounds of IDEA use multiplication and addition. It takes the 64 bit block being encrypted and four round keys. The 64 bit block is treated as four 16 bit blocks to match the round keys. The first and last blocks of the message are multiplied by the first and last round keys. The second and third blocks of the message are summed against the second and third round keys. These two results are also swapped before being passed on to the next round.

4.2 In Practice

There is a function called odd_round that takes a list of four integers as the message and another list of four integers as the round keys. The function has within it two functions declared inline. The mult function accepts two numbers and returns the product of the two. If either of the numbers is equal to 2**16 then that number is reduced to 0 before the multiplication. The other inline function is add. This function accepts two numbers and computes the sum. It then returns the remainder of division by 2**16. The two lists passed in may be indexed to access the integers. The actual calculations are done within the return statement. The function returns a list of four integers where each of the integers is obtained by computing the result of two operands. In the first and last place, mult is called on the first and last values of the message and the key. Add is called in the middle with the third value returned in place of the second and the second returned in place of the first.

5 Even Round

5.1 In Theory

The even round of IDEA has more operations than the odd round. It is also where the mangler function resides.

5.2 In Practice

With the mult and add operations defined the same as in the even round, here is the expanded calculation for the first value in the list where X represents the message and K represents the key. X[0] XOR mult(add(mult(X[0] XOR X[1], K[0]), X[2] XOR X[3]), K[1]) Each of the other three values returned are calculated in a similar way as this.

6 Block Chaining

IDEA only works on 64 bit blocks of information. This is common for encryption and the encryption algorithm is then referred to as a block cipher. In order to encrypt longer messages there needs to be a way to connect the blocks together.

6.1 CTR

This implementation uses counter mode chaining. The encryption algorithm generates a stream of information as long as the message and then the message is XORed against the data stream to produce the cipher text. The encryption algorithm takes a key and an initialization vector. The initialization vector is a substitute for the message. The initialization vector should be saved with the key because both are needed to decrypt the cipher text. Basically, the algorithm is encrypting the initialization vector each time and is using the master key to do so. In order to produce multiple unique blocks for the data stream, the initialization vector is incremented by one each time it is encrypted. If the algorithm is a good one, there should be no dicernable connection between succesive blocks of the stream. This is important because the data stream is exposed if the message to be encrypted contains all 0s. The data stream that is XORed with the message to create the cipher text is called a pad and is essentially a one time pad and IDEA in CTR mode acts as a stream cipher.

6.2 Future Improvment

This implementation does not feature any padding and instead encrypts the information byte by byte. This is undesierable as it may leak information about the message. A future improvment on this implementation would be to pad the message with the rest of the stream or append random information. The challange with this is to strip the garbage data off durnig decryption.

7 Profile and Performance

7.1 Time

The program was benchmarked against a one megabyte (1M) file of random data. The file was created using the command 'dd if=/dev/urandom of=filename bs=1M count=1.' The type of data should not affect the performance. The kernprof Python line profiler slows down the program but shows the relative speed of the program's operations. An even round takes longer than an odd round. Despite having an extra odd round, the total time spent in even rounds was 9.8s while the total time spent in odd rounds was 6.3s. The profile of the encrypt function also shows that 54% of the time was spent on the even rounds while only 37% of the time was spent on the odd rounds. The total time for the program to encrypt the test file was 59s. Almost 70% of the time is spent on actually generating the data for encryption while the rest is spread though formating the data, reading the data, and writing the data.

7.2 Future Improvment

Many of the computations are stacked onto one line. The profiler only displays the sum time for the line. Spreading out the operations accross multiple lines would add some distinction. For example, it isn't clear whether writing the data or XORing the data takes time because these two operations appear on the same line.

8 Running the Program

To encrypt a file named foo using a file named bar as the key file. Run 'python idea-ctr-file.py foo bar' This will create a file called 'foo.idea' that is the incrypted version of the file. To decrypt this file, run 'python idea-ctr-file.py foo.idea bar' This will create a file called 'foo.idea.idea' that is the original file. If you just run 'python idea-ctr-file.py foo' then two new files will be created. The program will create 'foo.idea' and 'foo.key' where 'foo.key' contains the 24 random bytes used to encrypt the file. To decrypt, run 'python idea-ctr-file.py foo.idea foo.key' In the file 'foo.key' the first 16 bytes are the key and the next 8 bytes are the initialization vector.