Sean Evans
CS 6058
Data / Security and Privacy
Spring 2018
Project 2

Project 2: Symmetric-Key Encryption

Description

The AES-256 symmetric-key encryption tool has three major functions, including key generation, encryption, and decryption. The encryption and decryption functions both support ECB and CBC modes.

The keygen functionality outputs a random sequence of 256 bits in binary format to an output file.

The AES-256 CBC encryption function ingests plaintext data from an input file, and a 256-bit key in binary format from a key file, and outputs a random 128-bit IV and ciphertext data in binary format to an output file. The corresponding decryption function ingests a 128-bit IV and ciphertext in binary format from an input file, and a 256-bit key in binary format from a key file, and outputs the plaintext to an output file.

The AES-256 ECB encryption function ingests plaintext data from an input file, and a 256-bit key in binary format from a key file, and outputs the ciphertext in binary format to an output file. The corresponding decryption function ingests ciphertext data in binary format from an input file, and a 256-bit key in binary format from a key file, and outputs the plaintext to an output file.

Implementation Details

The AES-256 symmetric-key encryption tool was implemented using C++, and uses the C++ Standard Library and Standard Template Library (STL), OpenSSL, and Boost Libraries. Boost's lexical cast was used to convert a string representation of a number to a native integer type. OpenSSL was used to perform AES-256 encryption and decryption. The key generation functionality was implemented using the C++ Standard Library's random number generator device and uniform integer distribution filter to generate a sequence of 256 random bits. The build system for the tool was implemented using CMake. The tool was built and tested on a Windows 10 x64 system in the Cygwin x64 environment.

Comparison of Running Time Between AES-256 in ECB and CBC Modes

Test data from runs of the test program are included in the table below. From this test data we can draw some conclusions. First, we can see that CBC encryption performs only slightly worse than ECB when a fixed or predefined IV is used. We can also see that, on average, decryption and encryption operations of the same mode perform fairly similarly.

However, when a random 128-bit IV is generated for each plaintext, the performance of CBC falls off significantly. This shows us that CBC mode performs similarly, if slightly worse than, ECB mode with a predefined IV, meaning that precalculating a random IV may be a method of reducing latency when ECB mode is used. Additional testing is required in order to determine the source of the additional latency. Likely sources of latency may include additional overhead in generating random numbers, repeated memory allocation for the randomized IV, and differences in CPU caching and branch prediction performance depending upon a fixed or randomized IV.

	ECB ENCRYPTION	ECB DECRYPTION	CBC STATIC IV ENCRYPTION	CBC STATIC IV DECRYPTION	CBC ENCRYPTION	CBC DECRYPTION
ITERATIONS	5000	5000	5000	5000	5000	5000
MIN RUN TIME	200 ns	200 ns	500 ns	200 ns	30100 ns	29800 ns
MAX RUN TIME	29900 ns	3400 ns	3900 ns	2600 ns	53000 ns	57400 ns
MEAN RUN TIME	537 ns	557 ns	761 ns	579 ns	31565 ns	32266 ns
RUN TIME VARIANCE	2400 ns	4600 ns	3400 ns	600 ns	1800 ns	6600 ns
MEDIAN RUN TIME	600 ns	600 ns	800 ns	600 ns	30800 ns	32400 ns