Details about encrypted connect

SHA-AES Security Method
This post provides a description of the SHA-AES method so as to reassure users of its security (and allow them to identify any defects). The SHA-AES security method is the most secure and the fastest security method provided by the cubeSQL. It employs standard algorithms published and certified by the US Government.

Cryptographic Algorithms

It is foolish to attempt to devise and trust new cryptographic algorithms for particular products because the strength of an algorithm is usually derived solely from its reputed uncrackability, and this property can only be built up over time. For this reason, the cubeSQL network protocol only uses well-known and trusted cryptographic algorithms. Here are the algorithms the protocol employs:

SHA-1: This is NIST's secure hash function. It reads a block of bytes (of any length) and produces a cryptographically strong 160-bit (20 byte) hash of the block. In this page, the SHA-1 of data block X is written as H(X).

AES-CBC: This is NIST's AES (Advanced Encryption Standard) algorithm. On January 2nd, 1997, NIST made a call for algorithms to replace the aging DES (Data Encryption Standard) algorithm. Out of fifteen AES candidate algorithms, Rijndael was selected on October 2nd, 2002. This algorithm is considered more secure than the Triple-DES algorithm. cubeSQL uses the algorithm with a 128/192/256 bit key and in Cipher Block Chaining Mode (CBC). cubeSQL's implementation comes from Dr Brian Gladman. The algorithm is reputedly uncracked. In this page, the AES-CBC encryption of data block X with key K is written as AESCBC(X,K).

These algorithms are combined in various ways to create the protocol.

Step By Step: The Protocol Conversation

In the following, A;B means the data block A followed by the data block B.

The protocol is based on the client and server sharing knowledge based to a secret password P that is specific to the server. The password consists of a sequence of one or more characters taken from the set of decimal digits and upper and lower case letters. The password is represented as ASCII bytes for processing.

The server's configuration file contains H(H(P)). This is expressed in hexadecimal following the SHA-AES: method keyword. Because the server only stores H(H(P)), it is not possible to determine the password of a virtual server from its configuration file entry. Connections proceed as follows and the conversation is aborted at each stage if any check fails.

Client sends random X: If the client is happy with the use of the SHA-AES security method, it generates a 20-byte random number X and sends AESCBC(X;H(X),H(H(P))) to the server.

Server checks X: The server uses H(H(P)) to decrypt the message from the client and retrieve X and H(X). It then calculates H(X) from X and compares it to the H(X) sent by the client. If the two match, then this proves that the message was encrypted by H(H(P)) and this proves that the client possesses H(H(P)) which is enough authentication for the server to proceed to the next step.

Server sends random Y: Now it's the server's turn to generate a random number. The server generates a 20-byte random number Y and sends AESCBC(Y;H(Y),H(H(P))) to the client. The server guarantees to never generate Y=X.

Client checks Y: The client uses H(H(P)) to decrypt the message from the server and retrieve Y and H(Y). If Y is equal to the random number X previously sent, the client assumes that it is talking to a spoof server that is conducting a replay attack and it terminates the conversation. The client then calculates H(Y) from Y and compares it to the H(Y) sent by the server. If the two match, then this proves that the message was encrypted by H(H(P)) and this proves that the server possesses H(H(P)) which is enough authentication for the client to proceed to the next step.

Calculation of session key S: The client and server then each independently calculate 160-bit session key S as S=H(H(H(P));X;Y). The session key includes the two random numbers which ensure that each session key is likely to be unique, along with the secret information H(H(P)) shared by the client and the server. The secret information is a

good backup in case the client and server's random number generators are predictable for some reason. All this information is hashed to yield the session key so that if someone somehow uses the streamed data to determine the session key, they still cannot determine H(H(P)). The session key is not transmitted by either the client or the server.

Client sends H(P);PADx12: To prove that it knows P, the client sends the server AESCBC(H(P);PADx12,S). PADx12 refers to 12 bytes of padding, this is necessary since the AES algorithm encrypts in blocks of 16 bytes. The server decrypts H(P) using the session key. It then calculates H(H(P)) from the H(P) just sent and compares the calculated H(H(P)) with the H(H(P)) provided by the configuration file. If the two match, then this proves that the client knows H(P) (and therefore almost certainly P). This is the final piece of authentication required by the server and an important piece because by providing H(P), the client proves that it did not merely get a look at H(H(P)) in the configuration file. This step of the protocol prevents those who see H(H(P)) in the configuration file from connecting. To connect, you must know H(P) and hence effectively P.

AES-CBC encryption: Once the exchanges described above have taken place, the remainder of the TCP connection is encrypted using AES-CBC encryption with first 128 bits of the session key S.

Protocol Properties

The protocol has the following properties:

Password P is never transmitted: The password P is never transmitted through the network, not even in encrypted form. Only the hash of the password is transmitted and even that is heavily AES-CBC encrypted first. Furthermore, the protocol does not transmit H(H(P)) in plain or encrypted form, and it does not transmit H(P) in plaintext.

Config file does not reveal password: The configuration file does not contain the password. It only contains H(H(P)).

Reading config file does not allow access: The configuration file contains H(H(P)) only. But H(P) is required to connect. This means that read-access to the server's configuration file does not allow access to the server.

Reading config file enables eavesdropping: If an intruder can read the configuration file AND can sniff packets, then the intruder can decrypt any conversation with the server. If the intruder can decrypt a conversation, it can determine H(P) and thereafter connect to the server himself at will. Therefore it is important either to keep the server configuration file secret or to ensure that no one can read the packets of connections to the server.

Reading config file enables spoofing: If you can read the configuration file and can divert packets headed for the server, it might be possible to create a spoof server. Therefore if packet diversion is a possibility at your site, it is important to keep the configuration file secret.

Server crypto-silent until it sees H(H(P)): The protocol is designed so that the server does not provide any crypto-related information to the client until the client sends it H(H(P)).

Two-way authentication with zero leakage: In addition to the server initially authenticating the client by requiring evidence of knowledge of H(H(P)), the client authenticates the server by requiring evidence of knowledge of H(H(P)). Neither proof conveys H(H(P)) to the other party, so the failing party does not obtain any information from the failed transaction.

H(P) transmission protected: The client only hands over H(P) after the client has authenticated the server and has set up a session key with which to encrypt H(P).

Protocol Design Notes

The following design notes may be of interest:

Two levels of hash needed: If the server stored H(P) in its configuration file instead of H(H(P)), then the client would have to send P to prove that it knows P. This would involve transmitting P (admittedly encrypted) which seemed like a bad idea. The current protocol does not ever transmit P, either encrypted or unencrypted. By eliminating P itself from all storage and communication, the protocol reduces the chance of a human getting their hands on P.

Dual contribution to session key: Both the client and the server contribute to the randomness of the session key.

This is important because if the random component of the session key were generated entirely by one party, the other party might worry that the first party might be compromising the security of the connection by using a random key that wasn't sufficiently random.

Playback attacks eliminated: The protocol eliminates playback attacks by either party because both sides generate part of the session key and the communication can only be understood by an entity that possesses H(H(P)).