Overview

Client to ePay Server Encryption

Database Store Encryption and Management

PAN Handling in Memory

Assumption

In this document '**Client**' refers to any one of DSIClientX.ocx, dsiPDCX.ocx or dsiEMVX.ocx and '**Serve**r' refers to a NETePay5 V5.06 application.

Client / Server Communication

1. Client is requested by POS to perform a transaction.
2. Client acquires a Crypto Context (128-bit).
3. Client takes predefined embedded obfuscated key and hashes it using MD5 Hashing algorithm.
4. It uses this hash to generate a 128-bit key and destroys the hash in memory.
5. Client generates a chunk of random data (using MS Crypto API) and uses the newly generated key to encrypt the random data using an RC4 Cipher.
6. Client Connects to Server TCP/IP Stream Socket
7. Server Accepts Connection
8. Server acquires a Crypto Context (128-bit).
9. Server takes predefined embedded obfuscated key and hashes it using MD5 Hashing algorithm.
10. It uses this hash to generate a 128 bit key and destroys the hash in memory.
11. Server generates a chunk of random data (using MS Crypto API) and uses the newly generated key to encrypt the random data using an RC4 Cipher.
12. Server transmits the clear text random data as a 'challenge' to the client.
13. Client encrypts this random data sent by the server
14. Client sends the encrypted data back as a 'response' to the challenge. It also sends its random data as a 'challenge' to the server.
15. The server compares the encrypted data to what the client sent back. If it does not match the client socket is gracefully disconnected. If it does, processing continues.
16. Server Sends back its response to the Servers Challenge and tells the client it liked the response.
17. The Client performs a similar check. The Client compares its encrypted random data to what the Server sends back. If it does not match, the socket is gracefully disconnected; if it does, processing continues.
18. At this point The Server knows it is speaking to one of its clients and the client knows it is speaking to one of its servers. There is a password option available to make this check even more granular. The previous has no bearing on the encryption of the actual financial transaction just client server authentication. Even though the software is designed to run at the local level, this extra authentication helps defend against a man-in-the-middle attack, and adds a layer of security not found in other models.

19. The Client and Server are connected at this point. When the server starts for the first time, it creates a Keyset. This Keyset is created using RSA Full strength 1024 bit keys. This Keyset is a public / private key pair.
20. As part of the initial dialogue, the server transmits its public key the client.
21. The client takes this public key, imports it into its Crypto API context, and generates a session key.
22. The Client uses the generated session key to encrypt the data it is going to send to the server.
23. It then exports the session key blob from the Crypto API.
24. This key blob is then transmitted to the server along with the encrypted data.
25. The Server takes the key blob and imports it into the Crypto API. It then uses this and its private key to decrypt the request.
26. It processes the request and uses the session key to encrypt the response.
27. Even though there is no sensitive data in the response, the response is encrypted using the session key generated by the client uniquely on each request. Some error responses are not encrypted. For instance, a failure to encrypt due to insufficient crypto libraries installed.

Database Store Encryption and Management

1. When the Server starts for the very first time, it generates a unique session code. This code is stored in the encrypted license file that is used for software activation. Each Server uses this value along with several other pieces of fixed data (different for each product) to encrypt sensitive data in the database.
2. Starting with PCI 2.0 the server now also uses the EPOCH of the machine on each transaction – this makes each record uniquely encrypted. Before submitting this fixed and variable data this unique string is put through a prime number modification algorithm.
3. The encryption starts by taking this value and hashing it using SHA Hashing Algorithm.
4. It uses this hash to generate a 128-bit key and destroys the hash in memory.
5. Server uses the generated key to encrypt the random data using an 3DES Cipher.
6. The Server encrypts any column that needs to store an account number or expiration date. Some Servers never need to encrypt because they never store the account number or expiration date.
7. The Servers that need to encrypt account numbers do so because this information is required to transmit the batch at the end of the day.
8. After successful settlement, the Server updates all of the rows with any encrypted columns to make the data invalid. Usually this is done by writing an ‘X’ over any encrypted data.
9. On Servers that are forced to require encryption, batch sizes are limited. If a merchant forgets to settle his/her batch and exceeds the batch size limit, processing new transactions is halted until the current batch is settled.

PAN Handling in Memory

When NETePay receives a payment request from the POS, it creates a request object. The message from the POS is parsed, extracting the PAN, purchase amount,

etc.

Once the message is parsed a request packet is built for the given payment service. The request packet includes the PAN.

The request packet is sent to the payment service to request an authorization.

When the response from the payment service is received, it is parsed to determine if the transaction is approved or declined.

An XML message is built with the response for the POS. The XML message does not include the PAN. Once the XML response message is returned to the POS, the original request object is destroyed.