A New E-Commerce Protocol:

Proposing a New Model for Facilitating Business to Consumer E-Commerce Transactions

General Design Documentation

Daryle Niedermayer,
B.A., B.Sc., M.Div., I.S.P.

Preface, Copyright & Disclaimer

Although this paper outlines the basic design issues involved in developing a secure e-commerce system that is competitive within the marketplace, it does not contain all the documentation necessary to implement such a model. If anyone would like to implement this model they can contact the author by phone at (306)757-4513 or by the contact information at http://www.niedermayer.ca. I would be very interested in engaging with someone to further this proposal but I do not have sufficient time or resources to devote to this project on my own.

There are a number of B2C merchant models available in the marketplace. Some are very costly and cumbersome for merchants or their technical resources to implement. Others do not have the flexibility that many merchants require. Still others depend on a proprietary model that confuses the customer by collecting the same information multiple times. Most offerings are not secure and are only waiting for a vulnerability to be exposed thereby bringing the entire industry into disrepute.

This document is shared to helping to bring about a consensus on developing the growing B2C E-Commerce segment. Although I currently reserve all rights to this document, I am also a strong advocate of the open standards and open source community. For this reason, I invite any comments or critiques and will consider turning this document over to a free source licensing model in the future.

Daryle Niedermayer

Contents

Overview

The marketplace forces and influences directing the emerging E-Commerce models are volatile and complex.

Merchants who decide to e-commerce enable their web sites often have an existing web site and relationship with an existing web hosting provider and wish to maintain this relationship. An important aspect in any proposed model is that is must be sufficiently flexible and expandable to allow a variety of hosting environments to connect to a payment acquirer. Important variables in this regard include the architecture, operating system, or development language or the merchant's host system environment.

Merchants implementing an e-commerce system may not want to manage the entire order-completion process, preferring not to SSL enable their own web site. For these reasons, any e-commerce model must enforce the encryption of sensitive cardholder and order fulfillment information whether that transmission occurs between the cardholder and the merchant, or the cardholder and the acquirer.

Security is always a concern of Internet applications. The transmission of sensitive cardholder information is even more of a security risk. At the same time, the marketplace will drive merchants to the acquirer who provides the most convenient system and the fastest and most economical time to market. These two goals--security and convenience—stand in paradox. The successful model will be one that puts the priority on security while providing a reasonable level of convenience for merchants and consumers.

Merchants are sensitive to costs. An effective e-commerce model will provide a lower cost to merchants. These costs include the development costs and licencing and setup fees to payment enable their site as well as the ongoing costs of maintaining their site and any per-month or per-transaction fees. For these reasons, off-the-shelf e-commerce software, which often involves large initial software costs and ongoing licencing fees, are difficult to justify.

High Level Approach

The design of a robust, flexible e-Commerce model is predicated on the following goals and objectives:

1. The system will be able to securely accept transactions from two types of merchants:

    a) Those without SSL encryption at the merchant storefront site. These merchants might use a simple buy button to allow a consumer to purchase a single item. Alternatively, they may use a shopping cart application but the actual exchange of cardholder and order fulfillment information is consummated by redirecting the consumer to an SSL encrypted web site hosted by the Payment Acquirer. The terminal status of each transaction sent by the acquirer for approval is communicated back to the merchant. Because cardholder information is not shared with the merchant storefront server, there is no need for encrypting any connections between the merchant and the acquirer. Only the consumer-payment acquirer connection needs encryption. These transactions will be known as "shopping cart transactions".

    b) Those with SSL encryption at the merchant storefront site. These merchants would typically use a shopping cart application and collect the cardholder and order fulfillment information. The necessary information is then transmitted to the acquirer. The terminal status of each transaction sent by the acquirer for approval is communicated back to the merchant. Because cardholder information is shared with

the merchant storefront, both the connections between the cardholder and the merchant and the connections between the merchant and the acquirer must be encrypted. These transactions will be known as “cardholder transactions”.

2. All communications between the merchants and the acquirer use Message Authentication Codes (MACs). MACs are used to verify the source of the transmission and verify that the message has not been modified enroute.

    a) Because the concept of a MAC is a security option not currently offered in the marketplace, the model allows the option to not include a MAC entity with messages. Such an option should only be made available if: the Merchant understands and agrees to accept the risks inherent in not using a MAC, and the acquiring system understands that this is the preferred option of this merchant.

3. Merchants will construct MACs using a MAC key generated by a certificate authority controlled by the acquirer. These keys will have a finite, predetermined life span. Merchants or their agents will use a Web interface to replace keys about to expire. The system will have to manage key chaining so that a merchant can have a transaction in transit using an old key while a newer key is still being enabled or installed.

4. All merchant authentication schemes will flow from a merchant id and merchant password. To request a new MAC key, a merchant will not only need this id and password but also some knowledge of the previous MAC key.

5. All valid message formats between the merchant and the acquirer will be explicitly specified.

6. Merchants can view transaction details, aggregate reports, and manage specific system configuration settings specific to their accounts using a web interface on a system controlled by the acquirer.

7. Where a message between a merchant and an acquirer does not contain an optional piece of information, the missing information will be retrieved from the merchant settings and configurations stored by the acquirer.

8. The acquirer system must be flexible and modular enough in design so that it can be modified to connect and communicate with multiple hosts, each with different transaction messaging formats.

9. Merchants can be notified of the terminal status of a transaction using a number of different methods: e-mail, secure web interface, or real-time HTTP communication.

10. XML will be used whenever possible to communicate data between independent systems.

Features in the Proposed Model

The Host Message Parser

One aspect of the design is the Host Message Parser. This module will provide the translation between XML and the native messaging format of the acquirer's host system. As such, one acquiring system can connect to more than one host by ensuring that a set of translation tables exists for each available host system. These tables also allow for an easy migration between host systems or to a different version of a host message format.

The construction of the Parser is a significant subject for the Detailed Design phase of this project.

Web Interface

The use of a web-enabled interface allows merchants unprecedented flexibility in configuring their settings or reviewing transactions. Some merchants may chose to use the acquirer's web site directly to fulfill orders from inventory rather than depending on confirmation messages from the merchant.

This interface can be leveraged into the marketing programs of the program or other applications to further extend the functionality offered to merchants.

Other Features

At the same time, additional enhancements are envisioned but not included in this design:

1. In the case of Shopping Cart Orders, the system can store the order fulfillment information of a cardholder and fill in that information when the cardholder places a subsequent order through the same acquirer. As well, this persistence can be leveraged into a cardholder authentication scheme. In such a scheme, a cardholder could enter a username and password to bring up their card information instead of having to enter their card information and address for every transaction they initiate. This feature could then subsequently be leveraged into a server-based wallet scheme.

2. Order fulfillment information can be compared to other orders on record. Using fraud detection algorithms, the merchant can be alerted regarding the trustworthiness of the order.

3. Demographic profiling can be combined with the buying patterns of customers to help identify potential market niches and promotional plans.

4. Recurrent billing would allow insurance companies, utility companies and other merchants who charge customers a monthly fee to automatically process transactions using the acquirer's system.

5. A merchant interface over a secure web connection to the acquirer's web server would allow event ticket vendors and other phone-order based vendors to immediately process transactions while requiring no or minimal shopping cart software on the merchant site.

6. Merchant based reporting would allow a merchant to reconcile their deposit account against the acquirer's records of processed transactions.

The Security Model of the Proposed Model

Principles

The e-commerce model being developed in this document is founded on a number of important assumptions:

1. Merchants and their Commerce Service Providers (CSPs) should be able to implement the system using any technology or implementation language of their own choosing. Such choices must be independent of the infrastructure selected by the acquirer. To facilitate this flexibility, transactions will be conducted using documents conforming to specified XML Document Type Definitions (DTD) over HTTP or SSL-encoded HTTP (HTTPS). Apart from these requirements, all other aspects of the acquiring or merchant systems can be independent of choices made by the other.

2. When encryption of the transaction is required, the SSL certificate of the acquirer will provide such encryption. The merchant should not be required to purchase or implement a Key Management System (KMS) or purchase keys or certificates to encrypt transactions with the acquirer. This is a service provided to the merchant by the acquirer.

3. The time to market for a merchant to bring their e-commerce site on line should be as short, economical and simple as possible. Very few requirements for additional software or hardware should be placed upon their Internet hosting provider.

For these reasons, the merchant has three possible options for using this system:

1. **The Buy Button option:** The merchant can insert a URL containing the entire XML encoded document in a static page. The XML encoded document would contain at a minimum, the purchase description, merchant identification, and order amount. The consumer clicks on the link to the URL, be redirected to the acquirer's SSL secured web-site, and enter their purchase card and order fulfillment information. The status of the order would normally be e-mailed back to the merchant. Optionally, the merchant can be informed of the status of an order by other means as well: an HTTP transaction, or a secure web site.

2. **The Shopping Cart Order option:** The merchant can create a dynamic URL based on some shopping cart application. This URL would include at a minimum, the purchase description, merchant identification, order number and order amount. The consumer clicks on the link to the URL, be redirected to the acquirer's SSL secured web-site, and enter their purchase card and order fulfillment information. Again, the status of the order would normally be e-mailed back to the merchant. Optionally, the merchant can be informed of the status of an order by other means as well: an HTTP transaction, or a secure web site.

3. **The Cardholder Order option:** The merchant can create a dynamic web page based on some shopping cart application and then ask the consumer for purchase card and order information. The merchant web site would necessarily provide SSL encryption for the user to supply this information. The merchant can then open a connection to an SSL-encrypted URL of the acquiring system and transmit all information for purchase authorization. In this case, the status of the order would normally be sent back to the merchant in an HTTP response over the SSL encrypted session.

Possible Vulnerabilities

The following vulnerabilities illustrate the need for a Message Authentication Code (MAC) to be part of the transaction processing system. Both vulnerabilities assume that there is no MAC included in the transaction. As such, they are similar to some of the competitive offerings already in the marketplace.

A Possible Vulnerability: Store and Forward

By placing an order with a merchant using either a Buy Button or a Shopping Cart Order, Eve can copy the URL with the XML encoded document and store it to her local machine. She edits the URL, changing the amount of the transaction to a lower amount, and then pastes this URL back into her web browser and executes the redirect. The transaction proceeds as normal. If the merchant does not validate the total amount of the transaction against the shopping cart or catalog value before fulfilling the order, Eve has effectively marked down the price of her purchase without the merchant's knowledge.

If the merchant does notice that the amount is incorrect and does not fulfill the order, there are three possible scenarios:

1. The merchant discards the order and credits the original amount back to the cardholder, absorbing any transaction charges as the cost of doing business.

2. The merchant contacts the cardholder to request payment of the remainder of the amount owing. Eve would most likely claim that that was the price advertised on the web site and the merchant is obligated to fulfil the amount for this price. The merchant has no evidence to repudiate Eve's claims.

3. The merchant reports Eve's attempts as an attempted fraud to the authorities. Provided the fraudulent amount is large enough, and Eve resides under the same judicatory system as the merchant, this response might be reasonably successful. Eve, in her defense, could mount the same rebuttal as mentioned in the previous scenario. Alternatively, Eve could ensure that she does not live in the same province or country as the merchant, and that her fraudulent attempts are under the threshold amount to trigger a robust response from the authorities.

Another Possible Vulnerability: Arbitrary Transactions against a Card

Using the third merchant option: a Cardholder Order, exploitation is trickier. Because the transaction information does not necessarily need to be included in a URL, it can be sent by the merchant server directly to the acquirer using an HTTP POST directive. Eve would not normally have access to the Store and Forward attack.

Even if she was able to discern the XML structure used for communication between the merchant and the acquirer, she is unable to get the merchant system to send the transaction to the acquirer on her behalf. She could open a connection to the acquirer herself, pretending to be the merchant and executing a transaction against a card. However, the response would come back to her and not the merchant in the HTTP response. The only effect would be that the acquiring system would process the transaction against her card to the merchant's benefit but she would receive no order fulfillment.

With this knowledge, Eve has two possible exploits available: one a harassment exploit, and the other fraudulent.

In the harassment exploit, Eve acquires the card number of a victim that she does not like (perhaps an ex-spouse?) She then uses this number to charge up the card against a given

merchant knowing that the cardholder will never get fulfillment of these orders. When statements are reconciled by the merchant, and/or the cardholder, the resulting chargebacks will cause a great deal of confusion.

In the fraudulent exploit, Eve uses her own card to attempt to generate a credit against it. The merchant will only discover the fraud when reconciling its deposit account statements, normally at the end of the month. Eve may have taken a cash advance against the credit balance in her account by then.

A Third Vulnerability: Order Authorization Spoofing

Eve used to work for a merchant that used a Buy Button option. She has copies of the e-mail sent by the acquiring system to the merchant informing the merchant that an order request was sent and approved and asking the merchant to fulfill the order.

Eve uses these e-mails as a template from which she constructs new messages. She spoofs these e-mails so that they appear to come from the acquiring system, authorizing the merchant to fulfill the specified orders yet no transaction has ever been received or processed by the acquiring system.

Possible Responses to these Vulnerabilities

IP Filtering

By verifying the source address of all connections to the acquirer, or the value of all HTTP-REFERER values, some risk can be mitigated. This mitigation is not complete because the HTTP-REFERER value could be spoofed and is not cleanly implemented by many browsers[1].

As well, the management of all merchant IP numbers greatly complicates system management for the acquirer. Whenever merchants migrate or expand to a new machine with a new IP number, they would have to inform the acquirer to enable this new IP number. When merchants abandon an old IP number or change upstream Internet providers, they would have to inform the acquirer to remove the old numbers and insert new ones. There are also many times when a firewall performs an unexpected NAT for a server, thus remapping the server's own IP number to an unanticipated value.

Finally, IP Filtering offers the merchant no security that the e-mail confirming order authorization originated from the acquirer.

For all these reasons, IP filtering is not a sufficient mechanism for resolving the security issues posed by the scenarios above.

Message Authentication Codes

The presented design uses two types of communications from the merchant to the acquiring system: encrypted and unencrypted. Encrypted communications are only required in the case of Cardholder Orders. The merchant transmits all other acquirer-directed transactions via the cardholder's browser and does not require encryption by the merchant (although they will require encryption between the acquirer and the cardholder at some point).

However, all transactions from the merchant to the acquirer must ensure that the information was not altered between the time it was constructed by the merchant to the time it is received by the acquirer. Furthermore, should such an alteration be effected, the merchant must have the ability to repudiate the transaction and the acquirer must be able to confirm this

[1] Cf http://browserwatch.internet.com/news/story/news-980302-7.html

repudiation. Similarly, in the case of e-mail notification from the acquirer to the merchant of order authorization, the acquirer must be able to repudiate the response and the merchant must be able to confirm this repudiation.

Because encryption is provided by the acquirer's SSL certificate and SSL enabled web server, we do not need to concern ourselves with encryption of the transactions, but only with the authentication of the plaintext messages transmitted between merchants and acquirers. Such a scenario normally calls for a mechanism such as a Message Authentication Code (MAC).

Given a key and a plaintext message, MAC generation involves a one-way hashing algorithm so that a replicable bit sequence known as a MAC can be generated. However, given only the plaintext message, the recreation of the original key is very difficult[2], and given the plaintext message and the MAC, any change to the plaintext will invalidate the MAC.

In this way, the cardholder cannot effect a Store and Forward attack since any change to the amount of an order will invalidate the hash supplied by the merchant. Without the merchant's key used to create the hash in the first place, the cardholder cannot replace the MAC with a new hash consistent with the altered value.

Neither can the cardholder effect an arbitrary transaction against a card since any card transaction requires a valid MAC generated by the merchant's key.

Finally, if the MAC is also used to validate any e-mail from the acquirer to the merchant, the merchant can be assured that the acquirer sent the message. This is because any spoofing attempt would similarly require access to the merchant's own key to authenticate the e-mail message.

Technical Specification of the MAC Generation[3]

Because both the merchant and the acquiring systems will generate the MAC, it must be easy to implement:

- either through the incorporation of existing libraries into the implementation environment,
- or through a clear and efficient software algorithm that developers can write in the chosen implementation language.

A number of MAC generation schemes exist. Due to the nature of this implementation, any MAC scheme must be at least resistant to known-text attacks.

The Cipher Block Chaining MAC algorithm has been shown to be secure when a secure underlying encryption algorithm is employed[4]. CBC-MAC is very secure on messages of a

[2] Alternatively, a number of possible keys can be generated but only one of which is the correct key. In this way, the MAC algorithm is not collision-free, but because our data is so tightly structured by the XML DTDs, the odds of a collision successfully matching the XML DTD are too remote to be considered.

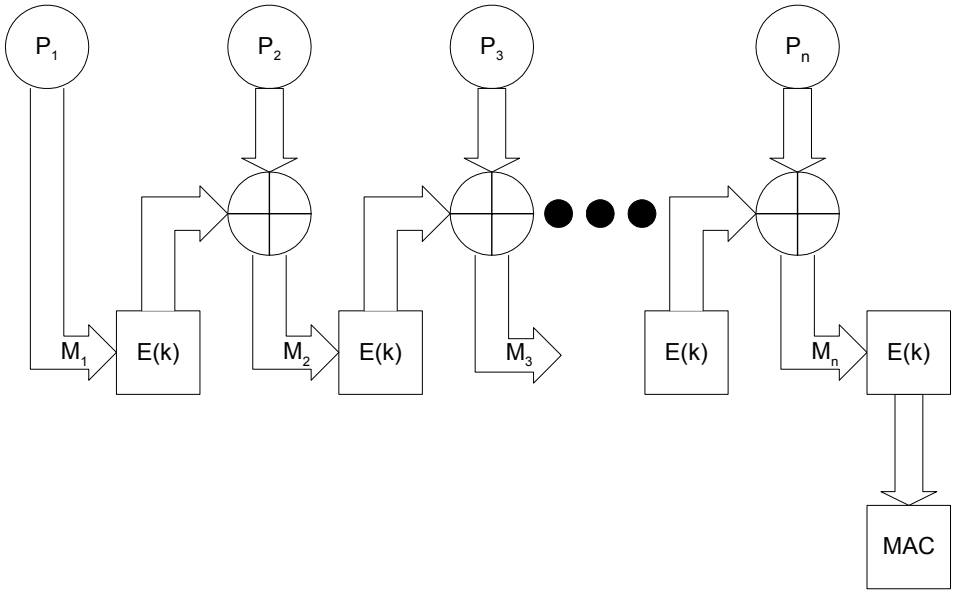
[3] See Menezes, A., P. van Oorschot, & S. Vanstone. *Handbook of Applied Cryptography*. CRC Press. 1996. Chapter 9 for a more complete discussion of the mathematics behind this issue.

[4] Bellare, M., J. Kilian, & P. Rogaway. "The Security of the Cipher Block Chaining Message Authentication Code". *Journal of Computer and System Sciences*. *61*:3. December 2000. pp. 362-399.

fixed length and very suitable for our purposes[5]. Because our messages use XML DTDs, they have a definite starting and ending point. A malefactor's attempt to add additional content at the end of the message will fail to match the XML specifications of the transmission.

CBC-MAC uses bitwise XOR operations to construct a hashed value out of the plaintext input "P" broken into blocks "P_1", "P_2", … "P_n" each of size *b* bytes, the specified key "k" and an encryption algorithm E(k). However, the CBC-MAC algorithm is only valid for a specified length of message and only if those messages are a multiple of *b* bytes in length. This first restriction is to prevent malefactors from appending new information to the end of the message until the resulting forgery computes to the existing hash resulting in a "collision" for the same hash value. Because our message format is based on a DTD, this problem is removed; additional content at the end of the message would result in a poorly formed and thus invalid XML document. The second restriction is also easy to remove: If $P_n < b$, then pad P_n with zero bits so that $P_n = b$.

The algorithm if very easy to implement as visualized in the following schematic:



[5] The only danger with CBC-MAC lies in the fact that the recipient necessarily holds the same key as the originator of the message. The recipient can use this key to decipher the hash in the reverse direction and generate a new message with the same hash value as the original message. Because a trusted relationship exists between the only two partners with a specific merchant key, and because of the way this design makes use of MACs and their corresponding messages, these concerns do not apply to this context.

The NEC Protocol (NECP)

The proposed E-Commerce model is founded on a new message transmission protocol dubbed the "New Electronic Commerce Protocol" or "NECP". NECP defines the following characteristics of the E-Commerce model:

- The valid message formats available to merchants for transmitting orders to the acquiring systems using XML DTD definitions;
- The valid message formats available to the acquirer in communicating the status of a specific order back to the merchant;
- The algorithms to be used in constructing the MAC to validate these messages;
- The procedures to be employed in distributing the authentication keys used in constructing the MACs;
- The nature and list of optional parameters for any specific merchant to be stored as message defaults by the acquirer;
- The transmission formats available to the merchant and acquirer in sending messages to each other.

Merchant Originated Message Formats

The message format of shopping cart transactions will be formatted using XML or URL encoded XML. The Data Type Definitions (DTDs) of these message formats are included in the Appendices.

There are two types of messages sent by a merchant: **Shopping Cart Orders** and **Cardholder Orders**.

Shopping Cart Orders allow the customer to fill a "shopping cart" on the merchant's site. At checkout time, the merchant site redirects the customer to the acquirer site and passes along a summary of the shopping cart along with the order amount. Once redirected, the customer then supplies card and order fulfillment information using the acquirer's SSL encrypted web site. With this method, the merchant does not need an SSL encrypted web-site.

In the case of a merchant that chooses the Cardholder Orders option, the locus of control for collecting cardholder information passes to the merchant. The merchant must have an SSL encrypted web-site. The merchant thus gathers the information from the cardholder and controls the customer's entire shopping experience. The merchant then passes the transaction information along to the acquiring system using a Cardholder Order XML document.

Shopping Cart Orders

Essentially, a Shopping Cart Order message combines an order identifier, details about an order and a MAC.

At a minimum, the order identifier must include the merchant identity. If no order number is supplied, then it is created by the acquiring system, pre-incrementing the highest order number currently on record for that merchant. Similarly, if no timestamp value is supplied, it is created using the acquiring system's clock. In this way, a merchant can use a URL on the catalog web-site to embed the entire XML document without having to dynamically generate

a new XML document. The merchant thus has a quick and efficient “Buy Button” for products that do not require a full shopping cart experience. Because all other aspects of this order are static, the MAC can also be pre-generated. In this way the URL does not need to change until the product description or price changes or the merchant’s MAC key expires.

The order details element includes at a minimum, the transaction amount. It can also include the type of transaction. Available options include a purchase, a pre-authorization, a capture of a previously authorized amount, or a credit. If this field is not supplied, a “purchase” is assumed. The order details can also include optional attributes such as currency type and decimal precision. If these are not supplied, they are derived from the system or merchant defaults in the acquiring system.

A Shopping Cart Order normally includes a calculated MAC using the merchant’s current MAC key and an optional description attribute.

If the merchant has decided to opt out of generating MAC values and expressly informed the acquirer of this decision, then the MAC entity should be set to the value of “0”. Upon receiving this MAC value, the acquiring system will verify that the merchant has chosen this option from its own records of the specific merchant’s preferences.

A final issue involved in Shopping Cart Orders is the calculation of shipping or delivery options. The default configuration has the merchant calculating the shipping costs and adding it to the total cost of the bill before redirecting the customer to the acquirer. However depending on the merchant’s business practices, this could require the cardholder to enter duplicate information. For example, the cardholder might be prompted to provide their state or province twice: once to the merchant as part of the shopping cart creation process and again when entering the order fulfillment information on the acquirer’s site.

This redundancy shouldn’t be a problem for most cardholders if it kept to a minimum, however the DTD has two attributes designed to manage this redundancy. By default, “shipping_included” is set to ‘Y’ meaning that the shipping costs of fulfilling the order are included in the price. If this field is set to ‘N’, it will be expected that the order amount sent to the acquirer does not including shipping costs. These costs will then be calculated by the Cardholder Interface module based on merchant specified configurations. To aid this configuration, a “shipping_code” value is included in the DTD to supply any parametric values to pass to this calculation. Examples of these values could include total weight calculation of the order, or shipping method. This acquiring system would then use the shipping code along with the postal code or zip code of the destination address and some template configurations to calculate the shipping cost to add to the order.

Cardholder Orders

A Cardholder Order is more detailed than a Shopping Cart Order. It requires a cardholder element. A cardholder element in turn requires a card and some minimum attributes of the cardholder’s identity.

At a minimum, the card entity includes the card number and the expiry date on the card. The definition also includes the ability to incorporate debit card and CVC verification as options.

Including additional information of the cardholder in the message can be used for demographic profiling and fraud detection. The question of how much cardholder data should be required is a business case question. These same arguments hold for the collection of order fulfillment attributes such as shipping addresses. Any decision would also involve legal implications for the sharing of consumer data between independent corporations.

The Cardholder Order message also includes a shopping_cart element almost identical to a Shopping Cart Order message except that in a Cardholder Order, the timestamp and order

numbers are required elements. The inclusion of these requirements provides idempotency checks to ensure that a consumer did not make multiple purchases within a given transaction window because of some network error. This removes the possibility of incurring inadvertent duplicate charges.

Again, a MAC ensures that the transaction originated from the merchant and was unaltered during transmission.

Acquirer Originated Message Formats

Transaction Response Document Type Definition

The acquiring system has a number of options to provide a response back to the merchant concerning the status of an order. At least three options are envisioned: e-mail notification, an acquirer-hosted merchant authenticated web site, and a server "push" by which the acquiring system responds to a transaction by sending a response to the merchant's server in real time.

The system as described also incorporates a wide spectrum of information already held by the merchant. In the case of a Cardholder Order transaction, the merchant only requires the response status and authorization code from the acquirer to go ahead and fulfill the order. For most Shopping Cart Order transactions, the merchant may know that an order is being made, but may have no customer or order fulfillment details. For the simplest type of transaction, that of a "Buy Button", the merchant may not even know that an order is made until informed of a transaction by the acquirer. For these reasons, any response must be flexible concerning the information that can be returned to the merchant following complete processing of a transaction by an acquirer.

However, regardless of the method chosen, a Transaction Response XML document describes the nature and construction of the data required by the merchant. A number of the elements of this response mimic the values available in the initial transaction request: order, destination and customer elements return to the merchant the values supplied by the customer in the case of a shopping cart transaction. They are redundant for cardholder transactions but can be supplied anyway. The MAC element is calculated by the acquiring system using the symmetric key shared by the acquiring system and the merchant. This MAC should always be included in an acquirer-generated response regardless of whether the merchant system chooses to validate it or not.

At a minimum, the response element includes the status keyword of the transaction--one of captured, authorized, denied, or credited--and timestamp of the response. Any other status indicates an untrapped error. If the transaction is in some sort of approved state, it will also include an auth_code from the cardholder's issuing system.

A response can also include a number of other attributes: a primary and secondary return code can be used to supply more information on error conditions to the merchant for problem identification. A response message can supply an English expression of the interpretation of these return codes. An optional confidence indicator could be used to identify what level of confidence can be placed on the transaction. Orders destined for high-risk countries, locales or addresses could be flagged in this way. Such a field might be a useful value-added feature for merchants, but could expose the acquirer to some liability. As such, other issues need to be considered before deciding whether to use such a field.

MAC Generation Algorithm

The specified MAC encryption function to be used with NECP is the HMAC with MD5 hashing. The specification for HMAC, authored by Krawczyk, Bellare and Canetti is found in

RFC2104.[6] While HMAC describes the procedure for message authentication using a cryptographic hash function, it does not define the specific cryptographic hash function. NECP uses MD5 as its hashing algorithm because MD5 has been well researched, has benefited from successfully identified weaknesses in its predecessors (notably MD4), and has been rigorously tested and yet appears secure. It is readily available as a library function on many platforms and in many languages and results in a 16-byte hash.

One important question is whether the MD5 hashes, produced by different libraries for different languages on different platforms are consistent. Because the MAC values for NECP will be generated on one machine and then verified on another, this is an important concern. Because the hashing algorithm operates on a byte-level representation of the data to be hashed, the actual representation of the data can affect the resulting hash value. For these reasons, NECP resolves these ambiguities by invoking the following specifications:

1. The data to be encoded must be in Unicode representation.[7] Alternatively, the resulting hash must be identical to a hash generated from the same data in Unicode representation.

2. All encryption keys will be 16 byte keys. If the first byte of an encryption key is 0, then the first byte of resulting key array will be stored as zero. If the first byte of an encryption key is less than 16 decimal (f hex), but greater than zero, then the first hexadecimal digit stored in the key's byte array will be zero.[8]

3. All keys will be distributed as hexadecimal digits and all MACs will be represented as hexadecimal digits.

   The algorithm to compute the HMAC value as specified in RFC2104 is as follows:

   MD5(K XOR opad, MD5(K XOR ipad, text))

   Where ipad is the byte 0x36 repeated B times and opad is the byte 0x5C repeated B times.

4. append zeros to the end of K to create a B byte string (e.g., if K is of length 20) bytes and B=64, then K will be appended with 44 zero bytes 0x00)

5. XOR (bitwise exclusive-OR) the B byte string computed in step (1) with ipad

6. append the stream of data 'text' to the B byte string resulting from step (2)

7. apply H to the stream generated in step (3)

8. XOR (bitwise exclusive-OR) the B byte string computed in step (1) with opad

9. append the H result from step (4) to the B byte string resulting from step (5)

10. apply H to the stream generated in step (6) and output the result

[6] http://asg.web.cmu.edu/rfc/

[7] http://www.unicode.org/charts/

[8] This step removes the ambiguity of what to do if the key is something like "0x00d7…." Some algorithms will truncate any leading zeros from a number regardless of its radix base, thereby rendering the number as a 15 byte key instead of 16 bytes. This step prevents such truncation.

Key Management Processes

NECP only requires keys to generate MACs. Encryption is not a function of NECP. In the case of Shopping Cart Orders, encryption is not required. In the case of Cardholder Orders, the SSL encryption layer of the acquirer's web server is relied upon to provide the data encryption function.

NECP uses a 16 byte symmetrically key generated by a cryptographic grade pseudo random number generator. A key of this size allows for 3.4E38 possible combinations. A brute force attack testing 1 million keys a second would still require 1.1E22 millennia to try all combinations.

The management of key generation is important to the security of the protocol. This importance is best illustrated given some hypothetical scenarios:

- A key is used to generate MACs for a merchant. This merchant is a particularly active one and a malefactor is able to sniff all transactions from the merchant to the acquirer. Given sufficient volumes and time, the malefactor is able to reverse engineer the key used to generate the MACs. Replacing MAC keys on a regular basis would mitigate this form of attack.

- An employee of the merchant's hosting company terminates her employment. She has knowledge of the MAC generation key. She then takes this key and sets up a spoofing site to embarrass her former employer. Allowing the employer to "sunset" a current key and obtain a fresh key would foil this form of attack.

- The merchant moves its electronic storefront to a new hosting provider. The merchant wants to severe all ties with its former hosting provider. Allowing the merchant to sunset a current key and obtain a fresh key would protect the merchant.

The key generation function is supplied by the acquirer using an SSL encrypted web enabled interface. To obtain a key on behalf of a merchant, the merchant's username and password must be supplied to the key generator.

If this is not the first key generated for the merchant, then the requester must supply some knowledge of the most recent former key, such as the first four bytes. In this way, the requester must demonstrate knowledge available only to the merchant as well as demonstrate knowledge of the merchant's current environment.[9]

At the time of requesting a new key, the requester can also request the period to expire the previous key. If this were a routine key change, then some default period such as one week would allow for the merchant to operate using either key during this interim period. If this is a security-related change, then the current key can be expired immediately.

The new key value and its serial number along with the expiration date of the current key are stored by the acquiring system in its database. The new key's value and serial number and a confirmation of the expiration of the old key are returned to the requester.

Merchant Specific Defaults

NECP allows considerable latitude in the content sent with a transmission. When parameters are omitted from a transmission, the acquiring system constructs these parameters from default values common to all merchants or default values specific to a particular merchant.

[9] Hopefully, these two values will not be available to someone looking under a desktop blotter pad.

In addition, some configurations are not specified in the XML DTDs but are needed for the end-to-end shopping experience of the customer.

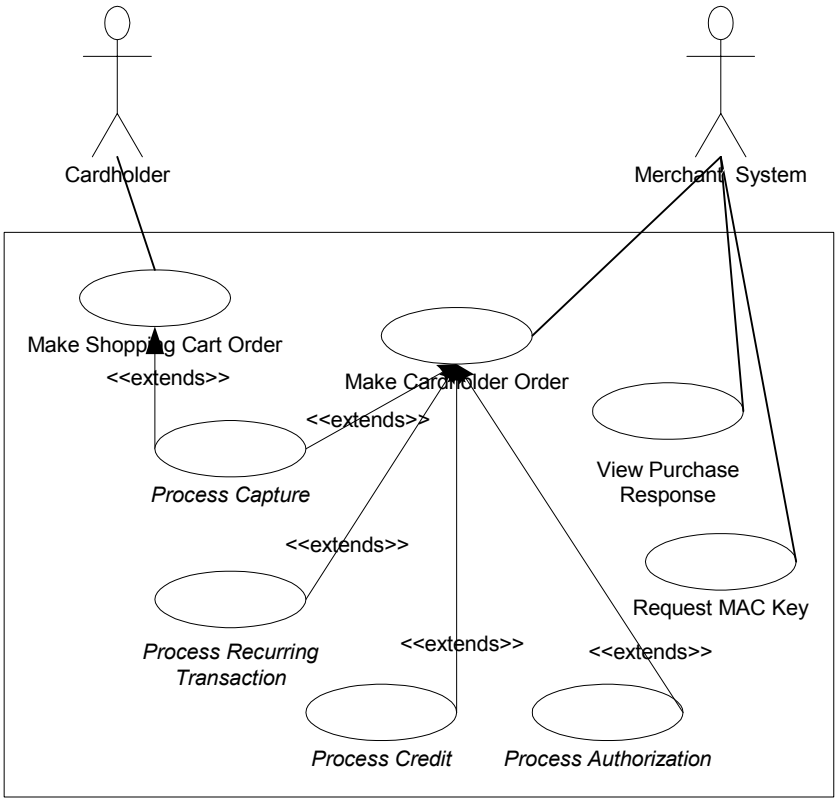
Available default values include:

Name	Description	Acceptable Values	Default	Merchant or System Value?
order_no	An order number unique to this merchant	Unique integer	Next largest integer	M
timestamp	The time an order is submitted to the acquirer	YYYYMMDD HHMMSS	Acquirer's timestamp	S
trans_type	The type of transaction being requested	purchase, authorize, capture, credit	purchase	S
description	The description of the shopping cart contents	Variable length string	none	M
currency	The national currency in which the transaction is denominated	CA, US	CA	M
precision	The number of decimal places used by the de-nominated currency	2, 3, 4	2	M
uses_MAC	Flag specifying those merchants who do not want the security of MAC authentication	Y, N	Y	M
redirect_ successURL	The URL on the merchant's site that the cardholder should be redirected to upon a successfully approved transaction	URL string	none	M
redirect_ failureURL	The URL on the merchant's site that the cardholder should be redirected to upon a failed transaction	URL string	none	M
redirect_ errorURL	The URL on the merchant's site that the cardholder should be redirected to upon an unexpected error	URL string	none	M
shipping_ calculation	The algorithm/equation to use in calculating shipping costs	Reference	none	M

System Design

Use Case Diagrams

Use Case diagrams show the associated roles and accesses available to actors and other systems external to the system. They show how users can interact with the system while avoiding the complications of diagramming the internal workings of the system. There are three primary use cases for this system: shopping cart transactions, cardholder transactions and key update transactions.



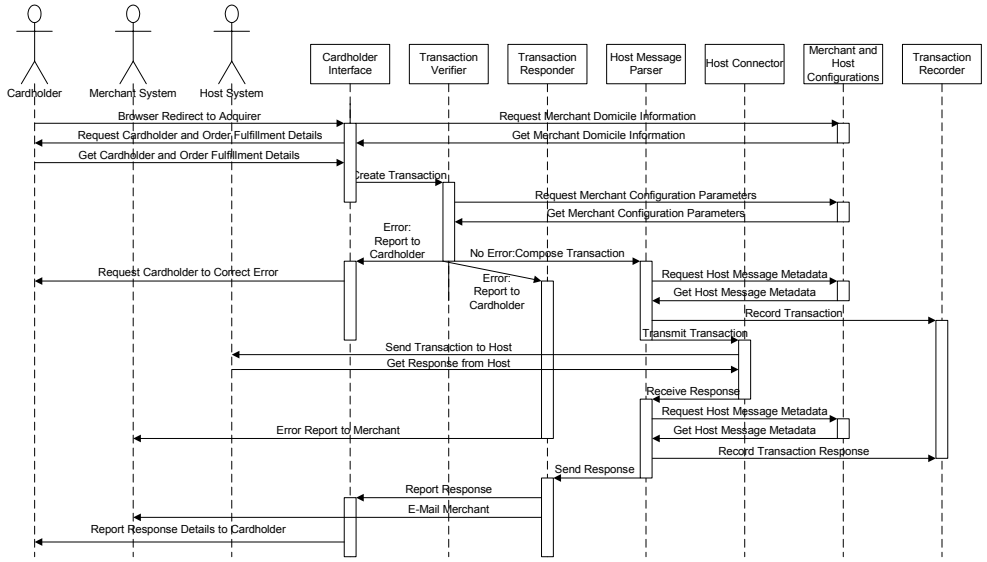
Shopping cart transactions have the Merchant system create a Shopping Order that is passed on to the Acquiring system. Cardholders are then redirected to the acquiring system as well, where they supply their cardholder information and confirm the purchase details. The system then processes the purchase by parsing the details and constructing a message to send to the host. The response from the Host System is then interpreted and a Purchase Response created in the system. This Purchase Response can be pushed back to the merchant using a number of methods: e-mail, HTTP or a web interface.

Cardholder transactions function in the same manner except here, the cardholder has supplied their cardholder and order fulfillment to the merchant through a secure interface on the merchant's web site. In this case, the merchant creates both the Shopping Order and the Purchase Order. All other aspects of the transaction are unchanged.

The final scenario is the creation of a new MAC key for a merchant. To request a MAC Key, the merchant or its agent will use a secure web interface to request a new key. Since the system uses symmetric keys for its MACs, this key will returned to the merchant and also be retained by the system for MAC verification.

Sequence Diagrams

Make Shopping Cart Order Sequence



The Make Shopping Cart Use Case sees the merchant web site redirect the customer's browser to the acquirer's web site. The acquirer's web site uses a Cardholder Interface to display a form requesting the customer to supply cardholder and order fulfillment information. In order to display this information, the Cardholder Interface must parse the embedded XML document to extract the merchant identity, and order details. This data is used to populate the interface screens presented to the user from information provided in the Merchant Configuration database.

Once all required information is obtained from the cardholder, a transaction is created and handed off to the Transaction Verifier. The Transaction Verifier checks that all data supplied are valid and that any missing information can be obtained from the Merchant Configuration database. At this point, two types of errors can be generated: those that originated with the cardholder, and those that originated with the merchant. If the former is the case, the Cardholder Interface can request the customer to correct the supplied data and the transaction can be reprocessed during this session. In the latter case, the error is not recoverable during the current session; the customer must create a new shopping cart on the merchant system before proceeding. In either case, the error is reported back to the Cardholder Interface for display to the cardholder. An additional error report may be generated for the merchant or system administrator. If there is no error, then the completed and verified Transaction object is handed off to the Host Message Parser. A reference to the Transaction object is also handed to the Transaction Recorder for insertion into the database.

The Host Message Parser provides encapsulation of host messaging details. In this way, the acquiring system can reroute transactions to different host systems depending on predefined criteria. The Host Message Parser obtains message metadata from the Host Configuration database and then constructs a Host message from the supplied Transaction object. The resulting Host Message is then sent to the appropriate Host Connector for transmission to the host system.

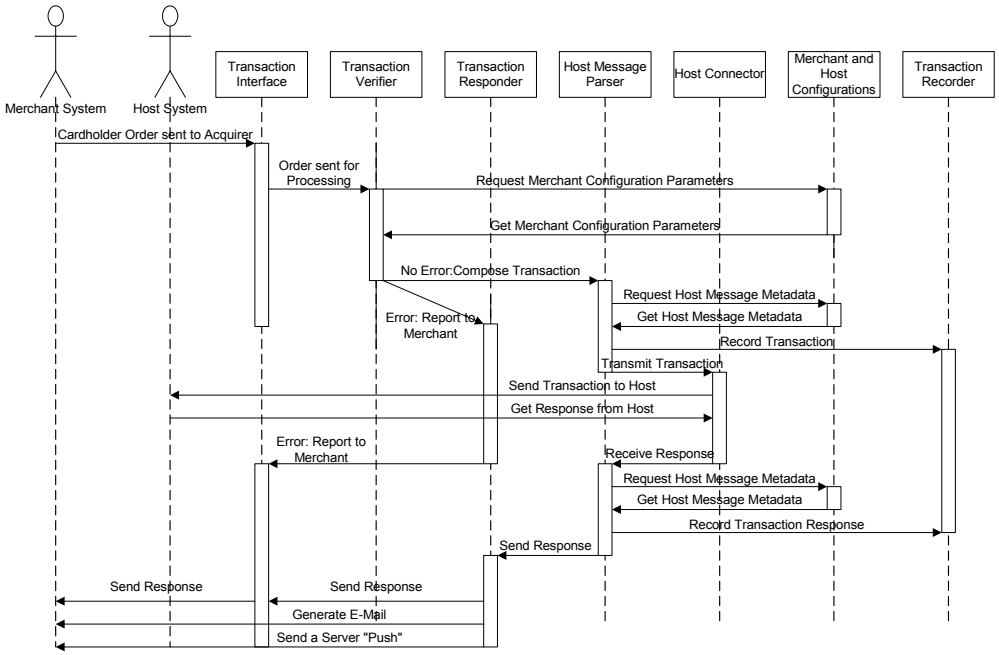
The Host Connector sends a transaction to the host and awaits the response from the host. In the event of a failure of the host system to send a response within a predefined timeout setting, an error will be generated, recorded in the system database, and sent to a system administrator. Upon a successful response, the Host Connector constructs a new Host

Message containing this response and returns this new Host Message to the Host Message Parser.

The Host Message Parser receives the Host Message and constructs a TransactionResponse object using metadata from the Host Configuration database. This TransactionResponse is handed to the Transaction Recorder for inclusion in the archival database. A reference of the TransactionResponse is then handed off to the Transaction Responder.

The Transaction Responder parses the appropriate information from the TransactionResponse and sends this to the Cardholder Interface. The Cardholder Interface uses this information along with the initial Transaction object to construct a return screen for the cardholder's web browser reporting the final status of the order and supplying the final sales invoice, merchant contact information, and order confirmation data. The Transaction Responder also consults with the Merchant Configuration database and e-mails the order details to the merchant and/or pushes the transaction object back to a merchant defined URL using XML.

Make Cardholder Order Sequence

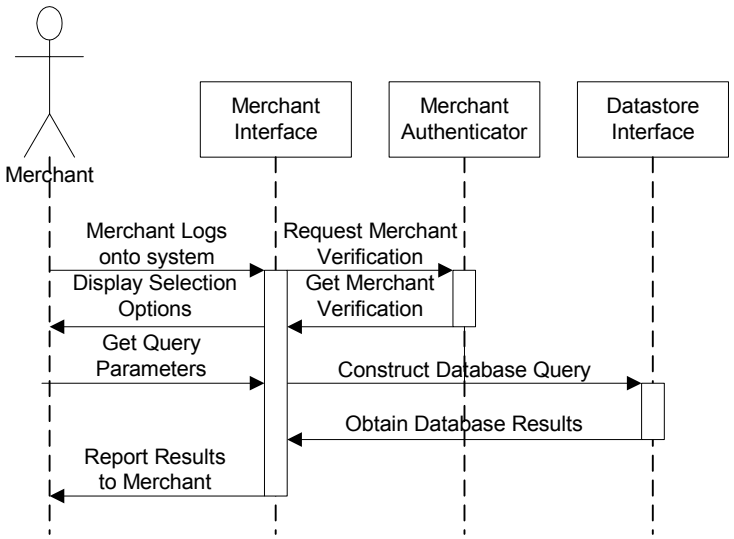


A cardholder order is a much simpler sequence because no interaction with the cardholder is required. Instead, the merchant is responsible for gathering all this information and passing it directly to a URL directly tied to the Transaction Verifier. The Transaction Verifier then does initial processing of the order to ensure that all required data is present and then performs some preliminary data validation. For example, it checks that the card number is a correctly formed number with a valid expiry date. Missing information is obtained from the Merchant Configuration database. If required information is still missing or any error is detected, program flow passes to the Transaction Responder, which creates a Response object documenting the error. This object is then used to return an XML message back to the merchant over the HTTP connection that initially sent the transaction.

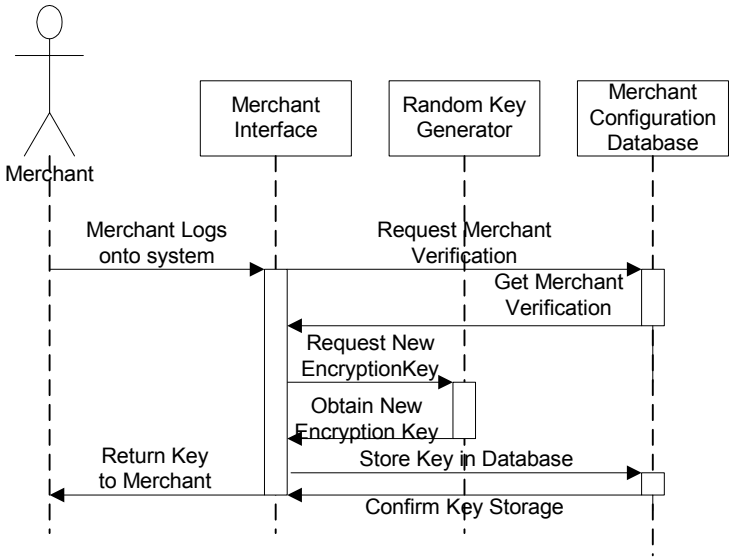
If no error is generated, the transaction processing then proceeds as in a Shopping Cart Order use case. The exception is that no response to the user is required and the merchant response can be sent back over the same HTTP connection that initiated the transaction in the first place.

View Purchase Response Sequence

The Purchase Response sequence is a web interface for merchants. After authenticating the merchant identity, it provides an application middle-tier to effect backend database calls on the merchant's transactions.



Request MAC Key



The Request MAC Key sequence allows the merchant to request a new key for computing MAC values. Required inputs are the merchant id, the merchant passphrase and the value of the current MAC Key against which to issue a renewal. Merchants can request a renewal at any time such as when an employee with access to the current key terminates their employment with the merchant. Because the information used in order transactions is very structured and highly predictable, it is expected that MAC keys will naturally expire within a predefined time interval. At most, keys should expire from one year of their issue. A better practice would have them expire at least every six months. Merchants generating a high volume of traffic might benefit from the security of having keys expire every three months.

When a new key is requested, the request is matched against the known merchant values of merchant id, passphrase and current key value. If this authentication is successful, then a new cryptographically strong randomly generated key is issued to the merchant over a secure web connection. This key is a four byte key with a specified sequential serial number. Upon request, the key is returned to the merchant over the secure web connection as a hexadecimal value and simultaneously stored within the merchant configuration database.

Other Use Cases

The sequence diagrams for the Process Recurring Transaction, Process Credit, Process Capture, and Process Authorization are identical to the Make Cardholder Order sequence diagram. The determination of which extension to use will be flagged in the XML document sent to the Transaction Verifier. A Make Shopping Cart Order always uses the Process Capture extension.

Class Diagrams

Shopping Cart Order Class Diagram

In the Shopping Cart Order Use case, a great deal of effort is expended on gathering and verifying data. The information-gathering task is centered in the Cardholder Interface. This module is responsible for catching the Shopping Cart Order XML document and then querying the customer for their card and order fulfillment information. This information is then used to construct a Transaction object, which is in turn a composite of ShoppingCart, Address, and Cardholder objects. The Transaction object is then passed to the Transaction Verifier. If the Transaction Verifier reports an error back to the Cardholder Interface, the Interface must determine if this is an error correctable by the cardholder (such as a bad card number) or an unrecoverable error (such as an invalid MAC). In either case, a message should be displayed to the cardholder with possibly a new form to correct any missing or inaccurate cardholder information. If the error is merchant-generated, then it must be reported to the system for referral to the merchant's technical personnel. For this reason, merchant-generated errors cause program flow to be routed immediately to the Transaction Responder. The Responder creates a Response object identifying the type of error and reports this error to the system administrator and/or merchant.

The Transaction Verifier is responsible for verifying that all the required information is present. Any missing information must be supplied by the Merchant Configuration settings or an error will result. The Transaction Verifier also ensures that the MAC is valid for the XML document supplied. Validating the MAC involves a number of tests:

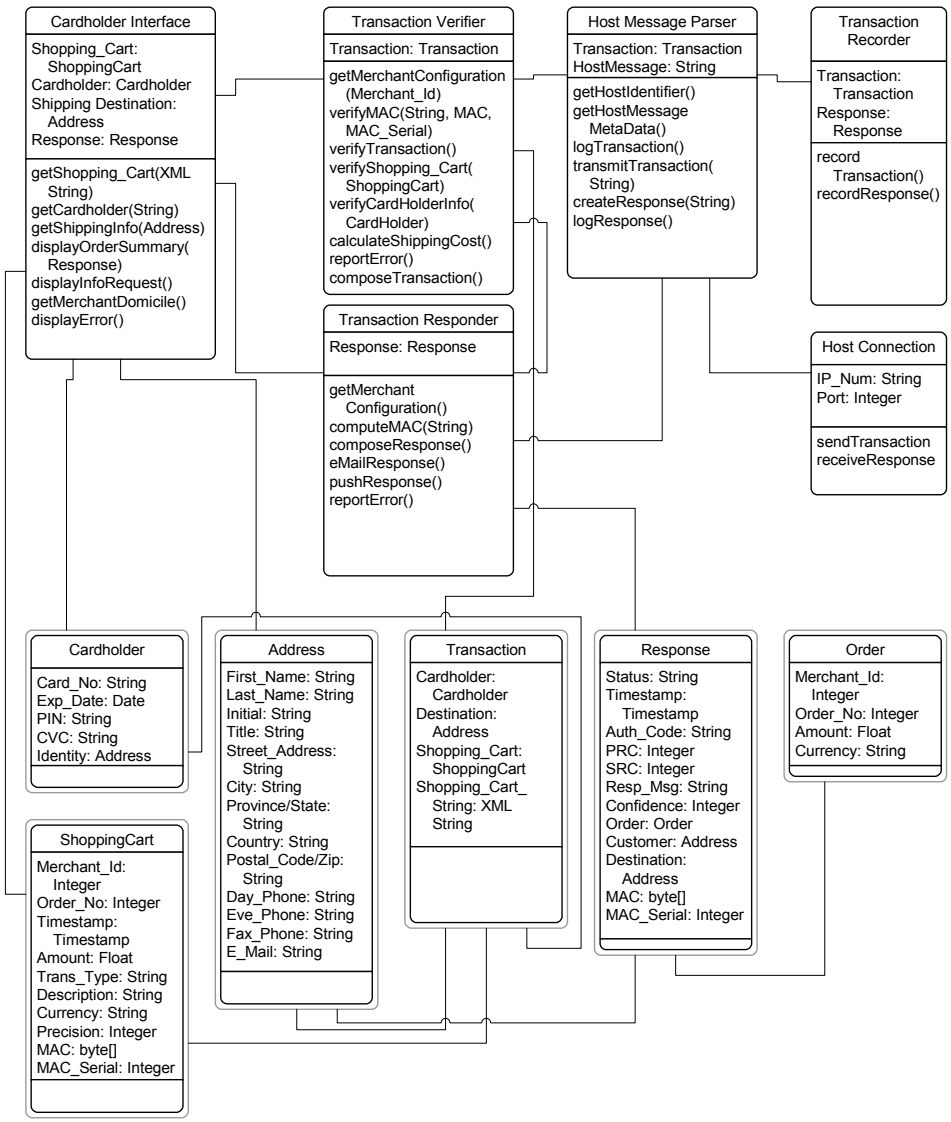
- The MAC must be validated against the specified merchant's key.
- The MAC must be verified to be using the current key.
- If the MAC is not using the current key, then it must be verified to be using a key that has not yet expired.

The Transaction Verifier also provides a number of additional functions:

- It verifies the Cardholder information. This could involve the mundane tasks of ensuring that the expiry date has not passed and the card number has a valid check digit. It could also involve some address verification procedures or fraud assessment. These latter two functions would most likely be passed to another module for processing.
- It can calculate the shipping value and add this to the total value of the order if the merchant has opted to request the acquirer to perform this function.

- Idempotency checking.

If the Transaction Verifier detects no error, it passes the Transaction object on to the Host Message Parser. The Host Message Parser determines where the transaction is to be sent for back-end acquiring. It then uses a table of translation values to translate the required elements from the Transaction object into the host-specific string for the type of transaction to be processed.

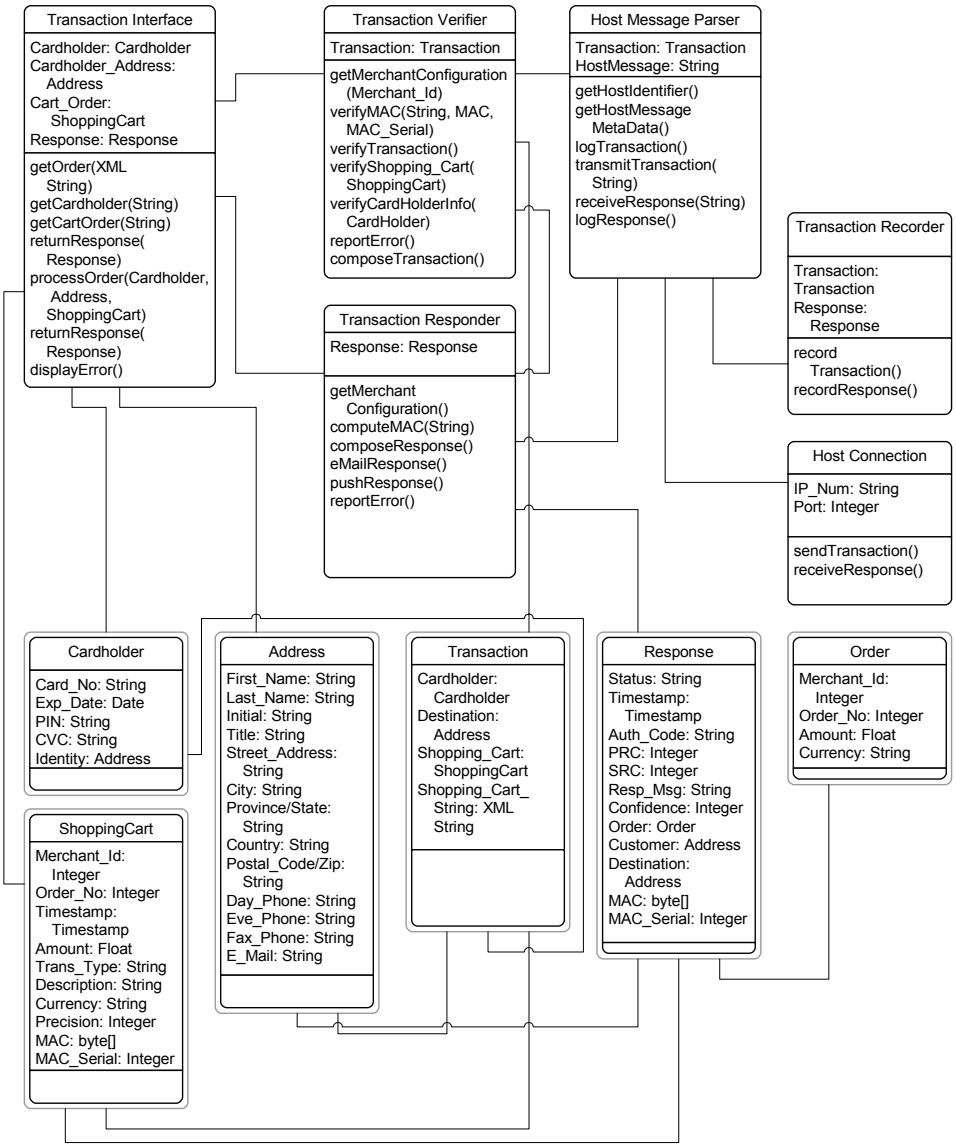


The Host Message Parser then opens a connection to the specific host and sends the transaction string, then waits for a response. Upon receiving a response, it parses the response string into a Response object and passes this object to the Transaction Responder for further processing. Both the transaction and the response are logged to a database.

The Transaction Responder creates a Transaction Response XML document and calculates a MAC for this document using the merchant's key. The Response object is optionally constructed into an e-mail document and delivered to the merchant and/or pushed to the merchant's server. The Response object is also returned to the Cardholder Interface so that a web page can be constructed for the cardholder informing them of the status of their order.

Cardholder Order Class Diagram

Cardholder Orders are similar to Shopping Cart Orders except that no HTML interface is required. The Transaction Interface accepts an XML document and parses it into the component objects belonging to a Transaction object: Cardholder, Address, and Shopping Cart. These components are then passed on to the Transaction Verifier for creation of a Transaction Object. As in the Shopping Cart Order, the Transaction Verifier checks all aspects of the order with the exception of calculateShippingCosts. In the Cardholder Order, the price passed to the acquirer is always the final price.

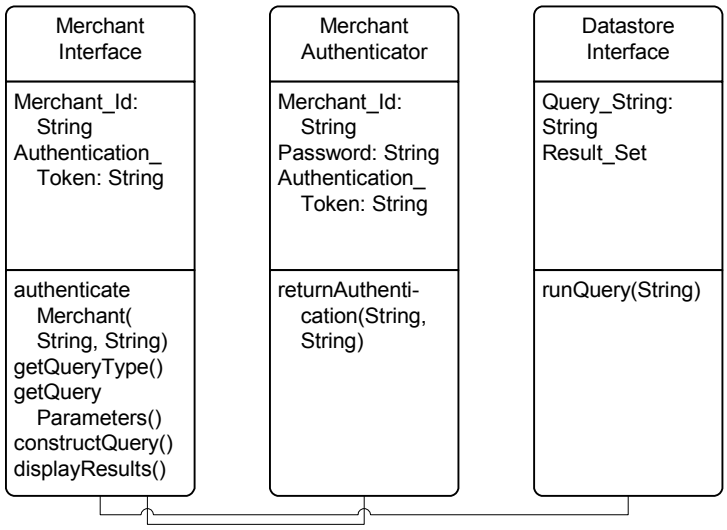


If the Transaction Verifier discovers an error, processing flow is passed to the Transaction Responder. The Responder then generates a Response object documenting the details of this error and then passes the Response object back to the Transaction Interface as well as reporting the error to a system administrator and/or merchant. Otherwise, processing

continues with the Host Message Parser. Order processing continues as per the Shopping Cart Order model.

Purchase Response Class Diagram

The Purchase Response Class diagram is indicative of a general group of class diagrams where a merchant seeks to query the acquirer's database concerning the status of a transaction. In some cases, these queries will be for a single transaction and in other cases, the query will seek an aggregate value such as the net or gross value of all purchases processed within a given period.



In all these cases, the class structure is comparable: a merchant points a browser to an SSL encrypted URL and logs onto the acquirer system with a username (or merchant id) and password. The Merchant Interface sends this data to the Merchant Authenticator for validation and upon a successful response, sends a cookie back to the merchant's browser with a time-limited authentication token. This token is also stored by the acquirer system. As long as the two tokens match and the token has not expired, the merchant is deemed to be authenticated for all future connections. The token is also "freshened" with a new expiry date after every HTTP exchange.

With an authenticated session, the Merchant Interface will construct a query and send it to the Datastore Interface. The Datastore Interface will then run the query against the backend database and return the results.

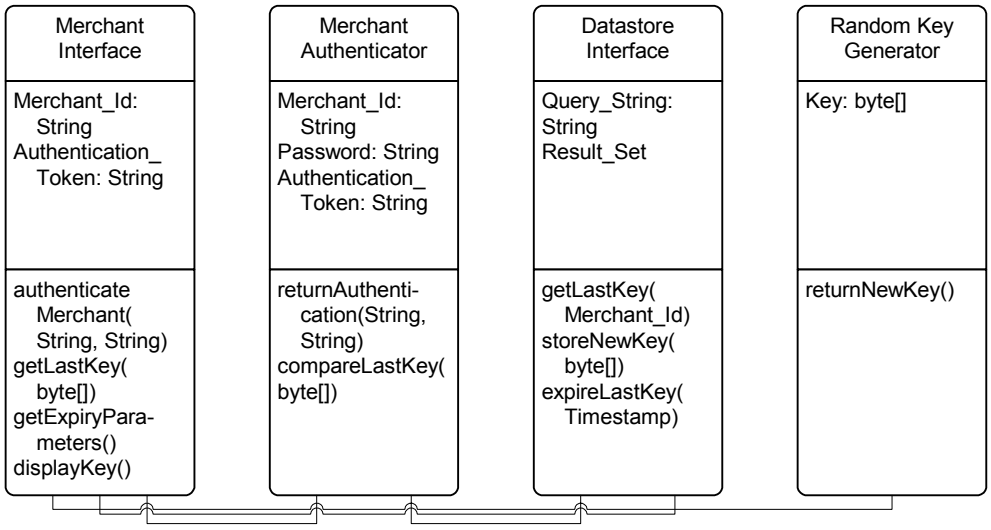
It is important to note that the Datastore Interface must be protected from the end-user. Without this protection, there is a danger that the end-user could run a custom query against the database returning data that does not belong to the originating merchant.

Request MAC Key Class Diagram

Requesting a new MAC Key involves the Merchant authenticating themselves as in the previous example. At this point, there are two possible scenarios:

1. This is an existing merchant who is renewing their MAC generation key. In this case, the merchant must also supply their last key to verify their identity.

2. There is no existing MAC Key for a new merchant. A new MAC generation key will be generated without any additional authentication criteria.


Merchant Interface
Merchant_Id: String
Authentication_ Token: String
authenticate Merchant(String, String)
getLastKey(byte[])
getExpiryPara- meters()
displayKey()
Merchant Authenticator
Merchant_Id: String
Password: String
Authentication_ Token: String
returnAuthenti- cation(String, String)
compareLastKey(byte[])
Datastore Interface
Query_String: String
Result_Set
getLastKey(Merchant_Id)
storeNewKey(byte[])
expireLastKey(Timestamp)
Random Key Generator
Key: byte[]
returnNewKey()

Network Architecture

Network Topology

The network architecture of the system is based on TCP/IP. Where the environment is hostile and sensitive information is being transmitted between entities, encryption is always used. From the customer's perspective, sensitive information includes cardholder information such as card numbers, cardholder names and expiry information. It also includes order fulfillment information such as shipping address, shipping date or method, or other delivery instructions. For merchants, sensitive information includes MAC generation keys, viewing on-line reports, and managing their account configuration.

For this reason, almost all Internet originated traffic to the acquirer system would be considered sensitive. Consequently, extensive use of SSL-enabled HTTP would be used by the acquirer system.

When sensitive information is sent over dedicated or secure channels, the option exists to allow the underlying network management to provide the requisite security. For this reason, if the acquirer has a dedicated connection to the Banknet system, then the transmission is done in the clear. The option exists to use VPN between the acquiring system and backend host system. Other options can be easily constructed.

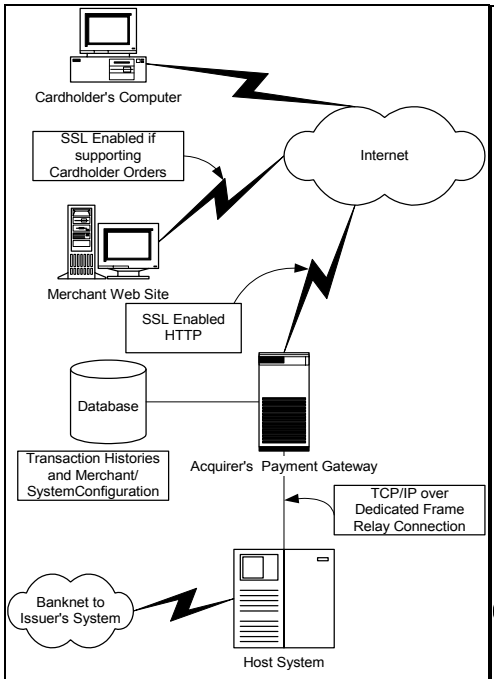


Figure 1: Dedicated connection to back-end Merchant Acquiring System

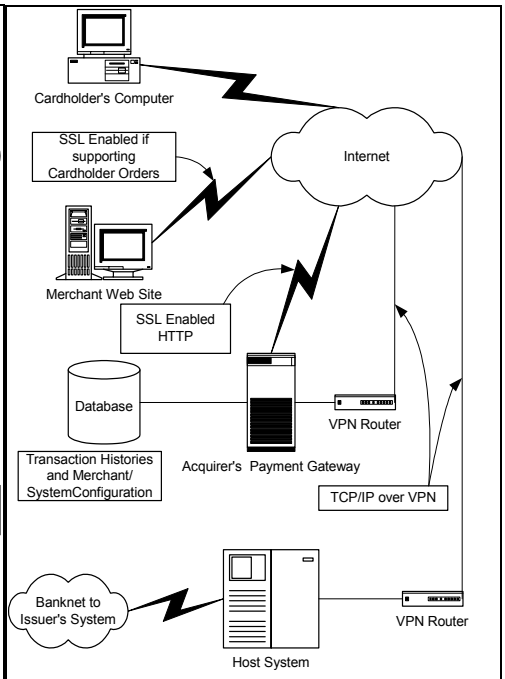


Figure 2: VPN connection to back-end Merchant Acquiring System over Internet or WAN

The architecture specified above is very scalable.

General System Requirements

The specifications for servers involved in the Payment Gateway function are quite generic. Any server grade hardware including Intel, RS/6000, Sparc, HP-UX, or other hardware platforms would be sufficient. Operating systems such as Linux, Solaris, AIX, or Windows would be acceptable. This generality allows for wide latitude in scalability and functionality.

Database Requirements

Normally, the database would be on a separate server networked with one or more Payment Gateways in a cluster. This allows for load balancing, high availability clustering, and fault tolerance.

The database management system (DMS) needs to support transactional integrity, but can be any JDBC compliant product such as Oracle, DB2, MS-SQL or MySQL. For extremely high load systems, additional scalability in the DMS may be called for, but many vendors can support this requirement.

Payment Gateway Requirements

The Payment Gateway function would normally be met by a Web Server/Application Server capable of supporting Java Servlets. Other configurations are certainly acceptable. Mod-perl and fast-CGI enabled Apache servers should have very good performance.

Since most of the work of the Payment Gateways will be done using SSL, the CPU load created by SSL encryption may be significant. This load can be minimized with dedicated SSL-encryption co-processors installed in the server.

Firewalls and Load Balancing

A firewall in front of the Payment Gateway is an essential feature of any network architecture. However, the firewall, possibly in conjunction with a round-robin DNS can provide some levels of load balancing between multiple Payment Gateway servers. It also insulates the Database server from any attempted connection by external entities.

Transaction Flow

The nature of transmissions between the cardholder, the merchant and the acquirer depends on whether the transaction is a Shopping Cart Order or a Cardholder Order. In the latter case, the cardholder is completely insulated from the acquirer. The merchant controls the entire transaction with the cardholder.

Shopping Cart Orders are more complicated. Because the merchant redirects the cardholder to the acquirer for checkout processing, additional issues surface in managing the shopping experience for the cardholder. In particular, after checkout, the cardholder should be redirected to the merchant site. To this end, the merchant can specify the redirection URLs to be used by the acquirer following processing of a transaction. These three URLs include where to send the customer following a successfully captured transaction, an unsuccessfully captured transaction, and an unspecified error (such as the backend host system does not respond within a timeout period).

Should the merchant require more texture in redirections than this, the option exists for the merchant to set a cookie on the cardholder's browser prior to the redirection to the acquirer payment gateway. Upon redirection back to the merchant site, the merchant can then retrieve this cookie and based on its value, redirect the cardholder to the specific page on the merchant site.

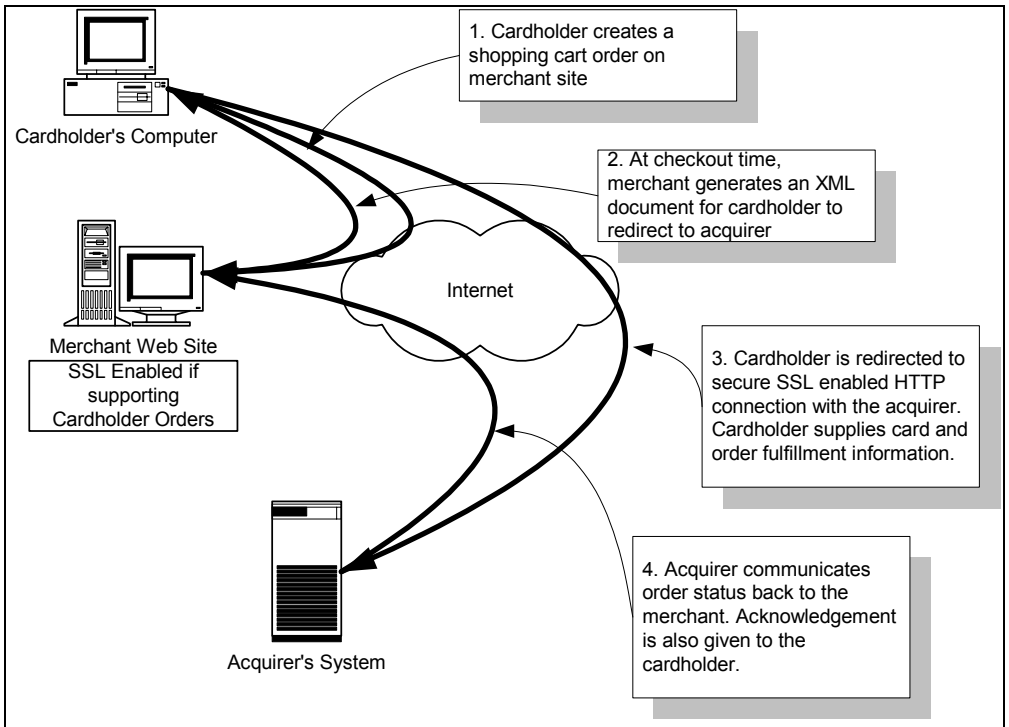


Figure 3: Chronology of Shopping Cart Orders

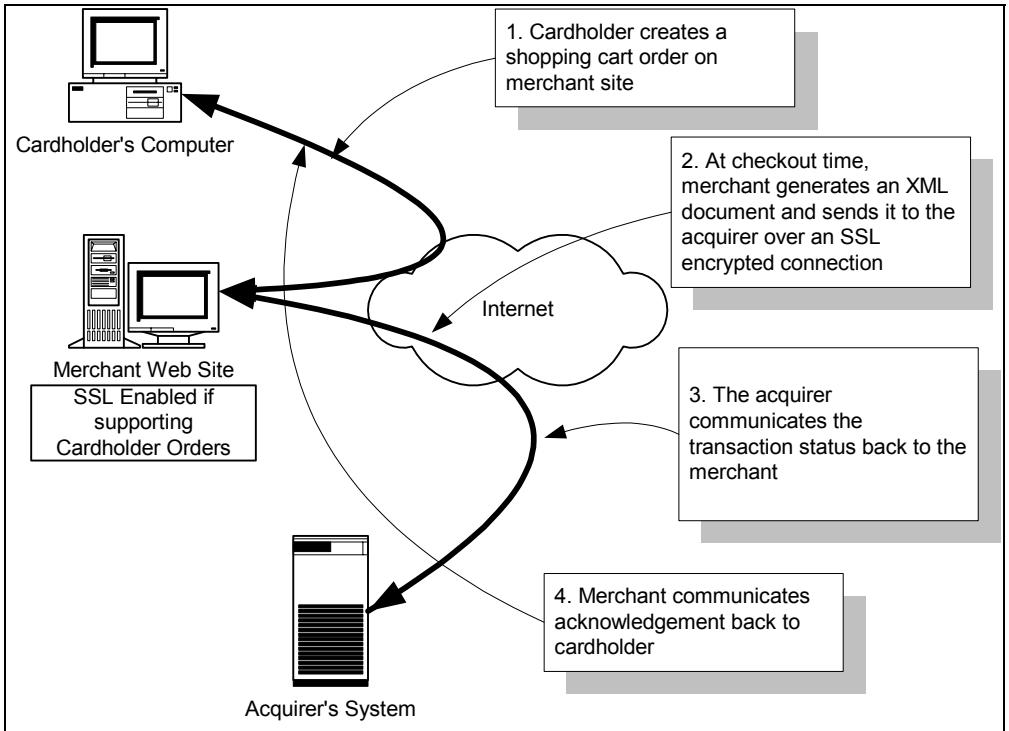


Figure 4: Chronology of Cardholder Orders

Final Comments

From the design proposed above, the subsequent step to a detailed design phase and the resulting cost estimates toward implementing the model are relatively straight forward.

Two issues are worth noting. First, in preliminary tests using the MD5-MAC algorithms included in existing in Java and PERL libraries, results were not always consistent or reproducible. This may also be true of other common implementation languages such as Visual Basic, C#, C, C++. In detailing the steps towards implementation, it might be necessary to create a new set of MD5-MAC libraries for all common implementation languages and to distribute these languages to merchant developers and software vendors manufacturing shopping cart software so as to ensure a consistent MAC is generated regardless of the underlying operating platform or implementation language.

Secondly, the Host Message Parser will require some further design work. It is envisioned that this translator will use a number of database tables in constructing the translation algorithm. One will define the acceptable limits or values for a field, another will contain the ordered list of how these values must be sequenced into the host messaging format.

Appendix A: NECP Message Format Specifications

Merchant Originated Message Formats

The following XML definitions are proposed for merchant-originated messages:

Shopping Cart Order Document Type Definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE shopping_cart [
   <!ELEMENT shopping_cart (order_id, order_details, MAC)>
   <!ELEMENT order_id      (merchant_id, order_no?,
                            timestamp?)>
   <!ELEMENT order_details (EMPTY)>
   <!ATTLIST order_details amount CDATA #REQUIRED>
   <!ATTLIST order_details shipping_included (Y|N)
                           #DEFAULT "Y">
   <!ATTLIST order_details shipping_code #IMPLIED>
   <!ATTLIST order_details trans_type
                           (purchase|authorize|capture|credit)
                            #DEFAULT "purchase">
   <!ELEMENT merchant_id   (#CDATA)>
   <!ELEMENT order_no      (#CDATA)>
   <!ELEMENT timestamp     (#PCDATA)>
   <!ATTLIST shopping_cart description PCDATA #IMPLIED>
   <!ATTLIST order_details currency (CA|US) #IMPLIED>
   <!ATTLIST order_details precision (2|3|4) #IMPLIED>
   <!ELEMENT MAC           (#CDATA)>
   <!ATTLIST MAC           serial_number CDATA #REQUIRED>
]>
```

Cardholder Order Document Type Definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE cardholder_order [
   <!ELEMENT cardholder_order    (cardholder, shopping_cart,
                                  MAC)>
   <!ELEMENT cardholder          (card, identity?,
                                  destination?)>
   <!ELEMENT card                (card_number, exp_date, PIN?,
                                  CVC?)>
   <!ELEMENT card_number         (#CDATA)>
   <!ELEMENT exp_date            (#CDATA)>
   <!ELEMENT PIN                 (#CDATA)>
   <!ELEMENT CVC                 (#CDATA)>
   <!ATTLIST card                brand CDATA #IMPLIED>
   <!ELEMENT identity            (name, address?, contact?)>
   <!ELEMENT name                (title?, surname, firstname,
initial?)>
   <!ELEMENT title               (Mr.|Mrs.|Ms.|Dr.|Rev.)>
   <!ELEMENT surname             (#CDATA)>
   <!ELEMENT firstname           (#CDATA)>
   <!ELEMENT initial             (#CDATA)>
   <!ELEMENT address             (street_address, city,
                                  province, country,
                                  postal_code)>
   <!ELEMENT street_address      (#PCDATA)>
   <!ELEMENT city                (#CDATA>
   <!ELEMENT province            (#CDATA)>
   <!ELEMENT country             (#CDATA)>
   <!ELEMENT postal_code         (#CDATA)>
   <!ELEMENT contact             (day_phone*, eve_phone*,
                                  fax_phone*, email*)
   <!ELEMENT day_phone           (#CDATA)>
   <!ELEMENT eve_phone           (#CDATA)>
   <!ELEMENT fax_phone           (#CDATA)>
   <!ELEMENT email               (#CDATA)>
   <!ELEMENT destination         (name, address)>
   <!ELEMENT shopping_cart       (order_id, order_details)>
   <!ELEMENT order_id            (merchant_id, order_no,
                                  timestamp)>
   <!ELEMENT order_details       (EMPTY)>
   <!ATTLIST order_details       amount CDATA #REQUIRED>
   <!ATTLIST order_details       trans_type
                                 (purchase|authorize|capture|
                                  credit)
                                 #DEFAULT "purchase">
   <!ELEMENT merchant_id         (#CDATA)>
   <!ELEMENT order_no            (#CDATA)>
   <!ELEMENT timestamp           (#PCDATA)>
   <!ATTLIST shopping_cart       description PCDATA #IMPLIED>
   <!ATTLIST order_details       currency (CA|US) #IMPLIED>
   <!ATTLIST order_details       precision (2|3|4) #IMPLIED>
   <!ELEMENT MAC                 (#CDATA)>
   <!ATTLIST MAC                 serial_number CDATA REQUIRED>
]>
```

Acquirer Originated Message Formats

The following XML definitions are proposed for merchant-originated messages:

Transaction Response Document Type Definition

```
<!DOCTYPE transaction_response [
   <!ELEMENT transaction_response (response, order, customer?,
                                   destination?, MAC)>
   <!ELEMENT response             (status, timestamp,
                                   auth_code?)>
   <!ELEMENT status
         (captured|authorized|denied|credited|error)>
   <!ELEMENT timestamp            (#PCDATA)>
   <!ELEMENT auth_code            (#CDATA)>
   <!ATTLIST response             prc         CDATA #IMPLIED>
   <!ATTLIST response             src         CDATA #IMPLIED>
   <!ATTLIST response             resp_msg    CDATA #IMPLIED>
   <!ATTLIST response             confidence  CDATA #IMPLIED>
   <!ELEMENT order                (merchant_id, order_no,
                                   amount)>
   <!ELEMENT merchant_id          (#PCDATA)>
   <!ELEMENT order_no             (#PCDATA)>
   <!ELEMENT amount               (#PCDATA)>
   <!ATTLIST amount               currency    CDATA #IMPLIED>
   <!ELEMENT customer             (name, address, contact?)>
   <!ELEMENT name                 (title?, surname, firstname,
initial?)>
   <!ELEMENT title                (Mr.|Mrs.|Ms.|Dr.|Rev.)>
   <!ELEMENT surname              (#CDATA)>
   <!ELEMENT firstname            (#CDATA)>
   <!ELEMENT initial              (#CDATA)>
   <!ELEMENT address              (street_address, city,
                                   province, country,
                                   postal_code)>
   <!ELEMENT street_address       (#PCDATA)>
   <!ELEMENT city                 (#CDATA>
   <!ELEMENT province             (#CDATA)>
   <!ELEMENT country              (#CDATA)>
   <!ELEMENT postal_code          (#CDATA)>
   <!ELEMENT contact              (day_phone*, eve_phone*,
                                   fax_phone*, email*)
   <!ELEMENT day_phone            (#CDATA)>
   <!ELEMENT eve_phone            (#CDATA)>
   <!ELEMENT fax_phone            (#CDATA)>
   <!ELEMENT email                (#CDATA)>
   <!ELEMENT destination          (address)>
   <!ELEMENT MAC                  (#CDATA)>
   <!ATTLIST MAC                  serial_number CDATA
                                   #REQUIRED>
]>
```