### Web Security Considerations

Table 7.1: A Comparison of Threats on the Web

<table>
<thead>
<tr>
<th>Threats</th>
<th>Web</th>
<th>Application</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>SQL Injection</td>
<td></td>
</tr>
</tbody>
</table>
The remainder of this chapter is devoted to discussion of SSL/TLS and SET.

Security Electronic Transaction (SET) is an industry standard for secure electronic transactions. Its use is required by many financial institutions and e-commerce sites. SET was developed by Visa, MasterCard, and other major credit card companies. The protocol is based on SSL/TLS and provides a framework for secure transactions. SET ensures data confidentiality, data integrity, and non-repudiation.

The protocol consists of several layers, including:

1. **Application Level:** This layer is responsible for application-specific data encryption and decryption.
2. **Transport Layer:** This layer provides end-to-end security between the application and network layer.
3. **Network Layer:** This layer provides network-level security.

SSL/TLS is used to secure data transmitted over the Internet. It is based on the Transport Layer Security (TLS) protocol, which was developed by the Internet Engineering Task Force (IETF).

There are several versions of SSL/TLS, with the most recent being TLS 1.3. The protocol is designed to provide strong encryption and authentication, making it suitable for secure communication over the Internet.

SSL/TLS uses several cryptographic algorithms to encrypt and authenticate data. These include:

- **RSA:** Used for key exchange and digital signatures.
- **AES:** Used for bulk data encryption.
- **SHA:** Used for message integrity.

SSL/TLS supports several modes of operation, including:

- **ClientHello:** This message is sent by the client to the server to initiate a secure connection.
- **ServerHello:** This message is sent by the server to the client to respond to the client's ServerHello message.

SSL/TLS is implemented in various software and hardware platforms. It is used by web browsers, email clients, and other applications to secure data transmission.

In the next chapter, we will discuss other web security issues and how they are addressed.
SSL Record Protocol

1. Read payload
2. Set record length
3. Set version
4. Set protocol
5. Set compression
6. Set application data
7. Append SSL record header
8. Compress

Message Integrity: The handshake protocol uses a shared secret key and a MAC cipher to ensure the integrity of the payload.

Handshake: The SSL handshake protocol begins with the client sending a Client Hello message to the server. The server responds with a Server Hello message, and both parties exchange keys for the session. The SSL Record Protocol is used to encrypt and compress the data exchanged during the connection.

A connection is established by the following parameters:

1. SSL/HTTPS
2. Pre-configured cipher suites
3. Server certificate
4. Client certificate (if required)

During the handshake, the client and server negotiate the cipher suite, key exchange method, and other security parameters.

SSL/TLS provides a secure channel over which sensitive data can be transferred securely.

Applications that use SSL/TLS include web browsers, email clients, and instant messaging applications.
After Protocol

which provides the captive pipe to be used in this connection.

The initial session may continue, but no new connections on this session may be
opened. If the peer is using SSL, the user may terminate the connection after
reading the client certificate and the server certificate. If the peer is using SSL,
the server certificate and the client certificate must be read. No protocol
intermediate certificates are supported.

SSL RECORD PROTOCOL

SSL RECORD PROTOCOL is used to convey SSL-related events to the peer agent. In
addition, it provides an interface to the server-side SSL protocol.

Change Cipher Spec Protocol

The Change Cipher Spec Protocol is used to change the SSL protocol's cryptography.

Sequence Diagram

The sequence diagram shows the key exchange process in SSL. The steps are as follows:

1. The client sends a ClientHello message to the server.
2. The server responds with a ServerHello message.
3. The client sends a Certificate message, containing the client's certificate.
4. The server responds with a ServerHelloDone message.
5. The client sends a ClientKeyExchange message, containing the client's key exchange data.
6. The server responds with a ServerKeyExchange message, containing the server's key exchange data.
7. The client sends a ChangeCipherSpec message to indicate that the client's key exchange is complete.
8. The server responds with a ChangeCipherSpec message to indicate that the server's key exchange is complete.
9. The client sends a Finished message containing the client's finished data.
10. The server responds with a Finished message containing the server's finished data.

Note that the key exchange process is symmetric for both the client and the server.
1. Message Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Handshake Protocol Message Types</td>
</tr>
<tr>
<td>1</td>
<td>Change Cipher Spec</td>
</tr>
<tr>
<td>2</td>
<td>Alert</td>
</tr>
<tr>
<td>3</td>
<td>Close Alert</td>
</tr>
<tr>
<td>4</td>
<td>Heartbeat</td>
</tr>
</tbody>
</table>

2. Handshake Protocol

The Handshake Protocol is responsible for establishing a secure connection between client and server. The exchange can be viewed as having four phases:

1. **Client Hello**
   - The client sends a message to the server indicating its capabilities.

2. **Server Hello**
   - The server responds with its own capabilities and a session ID.

3. **Certificate**
   - The server sends its certificate(s) to the client.

4. **Server Key Exchange**
   - The server generates a random number and sends it to the client.

5. **Change Cipher Spec**
   - The client sends a message indicating it is ready to change the cipher suite.

6. **Finished**
   - Both sides send a message indicating the handshake is complete.

The Handshake Protocol uses the SSL/TLS handshake to exchange information between the client and server. The handshake consists of several messages exchanged over a network connection. Each message is encrypted using AES-128 in CBC mode with a 16-byte initialization vector. The handshake protocol includes the following steps:

1. **Client Hello**
   - The client sends a message containing information about its capabilities and a random number.

2. **Server Hello**
   - The server responds with its own capabilities and sends a cookie to the client.

3. **Certificate**
   - The server sends its certificate(s) to the client.

4. **Server Key Exchange**
   - The server generates a random number and sends it to the client.

5. **Change Cipher Spec**
   - The client sends a message indicating it is ready to change the cipher suite.

6. **Finished**
   - Both sides send a message indicating the handshake is complete.

The handshake protocol is a crucial part of establishing a secure connection between the client and server.
**Handshake Protocol Action**

![Handshake Protocol Diagram]

**Session**

- The **client** sends a **Client Hello** message to the **server**.
- The **server** responds with a **Server Hello** message.
- A new **connection** is established.

**Handshake**

- The **client** sends a **Client Key Exchange** message containing the **client key exchange** information.
- The **server** responds with a **Server Key Exchange** message containing the **server key exchange** information.

**Certificate**

- The **client** sends a **Certificate** message containing the **client certificate**.
- The **server** responds with a **Certificate** message containing the **server certificate**.

**Finished**

- The **client** and **server** exchange **Finished** messages to indicate that the handshake is complete.

**Change Cipher Spec**

- The **client** sends a **Change Cipher Spec** message to signal a change in cipher suite.
- The **server** responds with a **Change Cipher Spec** message.

**Alert**

- The **client** sends an **Alert** message to indicate an error or problem.
- The **server** responds with an **Alert** message.

**Close**

- The **client** sends a **Close** message to indicate the end of the connection.
- The **server** responds with a **Close** message.

**Data**

- The **client** and **server** exchange data using the agreed-upon cipher suite.

**Compressed**

- The **client** and **server** exchange compressed data using the agreed-upon cipher suite.

**Handshake**

- The **client** sends a **Client Hello** message to initiate the handshake.
- The **server** responds with a **Server Hello** message.
- A new connection is established.

**Compression**

- The **client** sends a **Client Key Exchange** message containing the client key exchange information.
- The **server** responds with a **Server Key Exchange** message containing the server key exchange information.

**Certificate**

- The **client** sends a **Certificate** message containing the client certificate.
- The **server** responds with a **Certificate** message containing the server certificate.

**Finished**

- The **client** and **server** exchange Finished messages to indicate that the handshake is complete.

**Change Cipher Spec**

- The **client** sends a **Change Cipher Spec** message to signal a change in cipher suite.
- The **server** responds with a **Change Cipher Spec** message.

**Alert**

- The **client** sends an **Alert** message to indicate an error or problem.
- The **server** responds with an **Alert** message.

**Close**

- The **client** sends a **Close** message to indicate the end of the connection.
- The **server** responds with a **Close** message.

**Data**

- The **client** and **server** exchange data using the agreed-upon cipher suite.

**Compressed**

- The **client** and **server** exchange compressed data using the agreed-upon cipher suite.
### Phase 2: Server Authentication and Key Exchange

1. **Key Exchange:** The size of the key exchange for a cipher block chaining (CBC) with a 128-bit key.
2. **Key Authentication:** A sequence of bytes that is used in generating the key.
3. **Message Integrity:** A 16-byte MD5 (or SHA-1) fingerprint of the message.
4. **Cipher Block:** The key used with the cipher block chaining (CBC) mode.

### Phase 3: Client Authentication and Key Exchange

1. **Client Hello:** The client sends a message to the server containing the following information:
   - Protocol version
   - Cipher suites
   - Compression methods

2. **Server Hello:** The server responds with a message containing:
   - Server version
   - Cipher suites
   - Compression methods

3. **Certificate:** The server sends its certificate, which includes:
   - Public key
   - Server name

4. **Certificate Request:** The client requests additional certificates from the server.
5. **Certificate Response:** The server sends additional certificates.

### Phase 4: Handshake Message

1. **Certificate:** The client sends its certificate to the server.
2. **Finished:** A message that includes:
   - Encryption algorithm
   - MAC algorithm
   - Server parameters

### Phase 5: Key Exchange

1. **Pre-master Secret:** A shared secret between the client and server.
2. **Master Secret:** Derived from the pre-master secret and used for encryption.
3. **Client Key Exchange:** The client sends its key exchange message.
4. **Server Key Exchange:** The server sends its key exchange message.

### Phase 6: Change Cipher Spec

1. **Cipher Spec:** The client and server agree on the new cipher suite.
2. **Change Cipher Spec:** A message that indicates the start of the new cipher suite.

### Phase 7: Handshake End

1. **Handshake End:** The client and server complete the handshake.
2. **End-of-Change Cipher Spec:** A message that indicates the end of the change cipher spec.

### Phase 8: Handshake Failure

1. **Handshake Failure:** The client or server detects a problem in the handshake.

### Phase 9: Key Derivation

1. **Key Derivation:** A process that generates a symmetric key from the pre-master secret.

### Phase 10: Protocol Validation

1. **Protocol Validation:** The client and server verify that the handshake is correct.

### Phase 11: Key Confirmation

1. **Key Confirmation:** A message that confirms the key exchange.

### Phase 12: Key Finalization

1. **Key Finalization:** The client and server finalize the key exchange.
Connection of Cryptographic Parameters

Chosen in the initial Hello messages: when Challeng-response and SecurityMessages are the two options only.

\[(\text{Challeng-response} || \text{SecurityMessages}) \mid \text{SHV}
\mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}\]

Both sides now compute the master secret as follows:

1. To compute the master secret on either side:
   - After these are exchanged, each side performs the Diffie-Hellman encryption of the master secret with the public key of the other side. Each side encrypts the master secret with the public key of the other side. The encrypted master secret is sent to the other side, encrypted with the recipient's private key.

Where master secret is:

\[\text{master-secret} = \text{MD5(\text{NMAC-Origin)})} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}\]

Connection of NMAC-Origin:

The NMAC-Origin is the concatenation of the Diffie-Hellman parameters from the master-secret and the concatenation of the key exchange parameters from the master-secret. The two options are of interest: the addition of a shared master secret by means of a one-time symmetric key (256 bits). Note that the shared master secret is encrypted with the private key of the other side.

Connection of Cryptographic Parameters:

Obtain by exchanging application layer data. The application layer data is complete and the chain of events may be messages. All application-level messages are encrypted and signed by the Diffie-Hellman parameters used to exchange messages. Any message that is encrypted to the client's Diffie-Hellman and signed by the DH-HA-Client parameters is considered to have been generated by the client's DH-HA-Client parameters. In response to those two messages, the server sends its own challenge copy.

This message is all the data from all handshake messages up to but not including messages 2 and 3.

- \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{MD5(\text{NMAC-Origin})} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{NMAC-Origin} = \text{MD5(\text{message})}

Connection of NMAC-Origin:

The NMAC-Origin is the concatenation of the Diffie-Hellman parameters used to exchange messages. Any message that is encrypted to the server's Diffie-Hellman and signed by the DH-HA-Server parameters is considered to have been generated by the server's DH-HA-Server parameters. In response to those two messages, the server sends its own challenge copy.

If the DH-HA-Server parameters are set:

- \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{MD5(\text{NMAC-Origin})} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{NMAC-Origin} = \text{MD5(\text{message})}

Connection of NMAC-Origin:

The NMAC-Origin is the concatenation of the Diffie-Hellman parameters used to exchange messages. Any message that is encrypted to the server's Diffie-Hellman and signed by the DH-HA-Server parameters is considered to have been generated by the server's DH-HA-Server parameters. In response to those two messages, the server sends its own challenge copy.

If the DH-HA-Server parameters are set:

- \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{MD5(\text{NMAC-Origin})} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)} \mid \text{master-secret} \mid \text{SHV(ACC)}
- \text{NMAC-Origin} = \text{MD5(\text{message})}
As with the finished message in SSLv3, the finished message in TLS is a hash based on the shared master_secret, the previous handshake messages, and a label that identifies client or server. The calculation is somewhat different. For TLS, we have

\[
PRF(master\_secret, finished\_label, MD5(handshake\_messages) || SHA-1(handshake\_messages))
\]

where finished_label is the string “client finished” for the client and “server finished” for the server.

Cryptographic Computations
The pre_master_secret for TLS is calculated in the same way as in SSLv3. As in SSLv3, the master_secret in TLS is calculated as a hash function of the pre_master_secret and the two hello random numbers. The form of the TLS calculation is different from that of SSLv3 and is defined as follows:

\[
\text{master}_\text{secret} = PRF(pre\_master\_secret, \text{"master secret"}, \text{ClientHello.random} \parallel \text{ServerHello.random})
\]

The algorithm is performed until 48 bytes of pseudorandom output are produced. The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is defined as follows:

\[
\text{key}_\text{block} = PRF(master\_secret, \text{\"key expansion\"}, \text{SecurityParameters.server.random} \parallel \text{SecurityParameters.client.random})
\]

until enough output has been generated. As with SSLv3, the key_block is a function of the master_secret and the client and server random numbers, but for TLS the actual algorithm is different.

Padding
In SSL, the padding added prior to encryption of user data is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher’s block length. In TLS, the padding can be any amount that results in a total that is a multiple of the cipher’s block length, up to a maximum of 255 bytes. For example, if the plaintext (or compressed text if compression is used) plus MAC plus padding.length byte is 79 bytes long, then the padding length, in bytes, can be 1, 9, 17, and so on, up to 249. A variable padding length may be used to frustrate attacks based on an analysis of the lengths of exchanged messages.

7.3 SECURE ELECTRONIC TRANSACTION

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. The current version, SETv1, emerged from a call for security standards by MasterCard and Visa in February 1996. A wide range of companies were involved in developing the initial specification, including IBM, Microsoft, Netscape, RSA, Terisa, and Verisign. Beginning in 1996, there have been numerous tests of the concept, and by 1998 the first wave of SET-compliant products was available.

SET is not itself a payment system. Rather it is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- Provides a secure communications channel among all parties involved in a transaction
- Provides trust by the use of X.509v3 digital certificates
- Ensures privacy because the information is only available to parties in a transaction when and where necessary

SET is a complex specification defined in three books issued in May of 1997:

- **Book 1:** Business Description (80 pages)
- **Book 2:** Programmer’s Guide (629 pages)
- **Book 3:** Formal Protocol Definition (262 pages)

This is a total of 971 pages of specification. In contrast, the SSLv3 specification is 63 pages long and the TLS specification is 80 pages long. Accordingly, only a summary of this many-faceted specification is provided in this section.

**SET Overview**

A good way to begin our discussion of SET is to look at the business requirements for SET, its key features, and the participants in SET transactions.

**Requirements**

Book 1 of the SET specification lists the following business requirements for secure payment processing with credit cards over the Internet and other networks:

- **Provide confidentiality of payment and ordering information:** It is necessary to assure cardholders that this information is safe and accessible only to the intended recipient. Confidentiality also reduces the risk of fraud by either party to the transaction or by malicious third parties. SET uses encryption to provide confidentiality.
- **Ensure the integrity of all transmitted data:** That is, ensure that no changes in content occur during transmission of SET messages. Digital signatures are used to provide integrity.
- **Provide authentication that a cardholder is a legitimate user of a credit card account:** A mechanism that links a cardholder to a specific account number reduces the incidence of fraud and the overall cost of payment processing. Digital signatures and certificates are used to verify that a cardholder is a legitimate user of a valid account.
SET Protocol

- SET is an open standard for secure transactions on the Internet.
- It is designed to provide a secure way for credit card transactions to be made over the Internet.
- SET uses public key cryptography to ensure the confidentiality and integrity of transactions.
- The protocol includes a client (the merchant or bank) and a server (the bank or merchant).
- SET uses certificates to authenticate the parties involved in the transaction.
- Transactions are encrypted using the SSL/TLS protocol.
- SET is designed to be compatible with existing payment systems and can be used with any Internet browser.

SET Transactions

- A secure channel is established between the client and server using SSL/TLS.
- The client sends a secure message to the server, requesting a transaction.
- The server verifies the client's identity and approves or rejects the transaction.
- The transaction is then processed and completed.

SET Security

- SET provides a high level of security for online transactions.
- It is designed to protect against fraud and ensure the confidentiality of data.
- SET is widely used by banks and merchants around the world.

SET Privacy

- SET protects the privacy of credit card information by encrypting it during transmission.
- The protocol includes mechanisms for protecting sensitive data, such as credit card numbers.
- SET is designed to be compliant with various privacy regulations, such as the European Data Protection Directive.

SET Technology

- SET relies on a combination of public key cryptography and SSL/TLS to provide security.
- The protocol is implemented using a variety of technologies, including Web pages, server software, and client software.
- SET is supported by major browsers and is widely used in e-commerce applications.

SET Implementation

- SET can be implemented using a variety of technologies, including XML and SOAP.
- The protocol is designed to be easy to implement and was developed by Visa and MasterCard.
- SET is compatible with existing payment systems and can be used with any Internet browser.

SET Benefits

- SET provides a secure way for credit card transactions to be made over the Internet.
- It is designed to protect against fraud and ensure the confidentiality of data.
- SET is widely used by banks and merchants around the world.

SET Future

- SET is constantly evolving and is expected to continue to improve over time.
- Future versions of SET may include new features and improvements to the protocol.
- SET is likely to continue to play an important role in the e-commerce landscape for years to come.
10. The merchant receives payment. This is sent to the payment gateway.

9. The merchant provides the goods or service. The merchant ships the goods or services.

8. The merchant confirms the order. The merchant sends confirmation of the payment.

7. The merchant confirms the payment information. The merchant sends the payment information to the processor.

6. The order is confirmed. The customer receives the order confirmation in full form. The merchant sends both Y and X to the processor.

5. The merchant is verified. The order is confirmed. The merchant sends both Y and X to the processor.

4. The customer places the order. The processor checks and confirms the order. The processor checks and confirms the order.

3. The customer places the order. The customer checks and confirms the order. The customer checks and confirms the order.

2. The customer receives the order. The customer checks and confirms the order. The customer checks and confirms the order.

1. The customer receives the order. The customer checks and confirms the order. The customer checks and confirms the order.
Table 7.2: SET Transaction Types

<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certification Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Certificate Authority Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Exchange Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Exchange Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
<tr>
<td>Transaction Information</td>
<td>The certificate of the transaction sender to receive the transaction.</td>
</tr>
</tbody>
</table>

Figure 7.9: Certification of Digital Signature

- **CAP**: Certificate Authority Information
- **PID**: Public Key Information
- **OID**: Object Identifier
- **H**: Hash of the Transaction
- **E**: Encryption (RSA)
- **F**: Hash of the Transaction (SHA-1)
- **D**: Digital Signature
- **P**: Public key of the certificate
- **Q**: Public key of the certificate
- **M**: Message to be signed
- **S**: Signature of the message

The certificate contains the following information:

1. **CAP**: Certificate Authority Information
2. **PID**: Public Key Information
3. **OID**: Object Identifier
4. **H**: Hash of the Transaction
5. **E**: Encryption (RSA)
6. **F**: Hash of the Transaction (SHA-1)
7. **D**: Digital Signature
8. **P**: Public key of the certificate
9. **Q**: Public key of the certificate
10. **M**: Message to be signed
11. **S**: Signature of the message

The certificate is used to ensure the authenticity and integrity of the transaction.
sends the PAM Request, and the OAM responds.

The PAM Request includes the transaction ID, the PAM's identification, and the service ID. The PAM also includes the service description, which provides information about the transaction it intends to perform. The PAM then waits for the OAM to respond.

When the OAM receives the PAM Request, it verifies the transaction ID and checks the service ID to ensure that the transaction is valid. If the transaction is valid, the OAM generates a response, which includes the transaction ID, the PAM's identification, and the service description. The OAM also includes a message that indicates the result of the transaction, such as 'success' or 'failure'. The OAM then sends the response back to the PAM.

The PAM receives the response and verifies the transaction ID and service ID to ensure that the response is valid. If the response is valid, the PAM learns about the result of the transaction and can take appropriate action. If the response is not valid, the PAM may need to re-send the PAM Request to the OAM.
When the transaction request arrives at the merchant's location, the merchant extracts the following data:

1. **Authorisation Request** Information. This information is contained in the message sent by the cardholder to the merchant.
2. **Transaction Identifier** (TID). This identifier is unique and is included in the transaction message sent by the cardholder to the merchant.
3. **Credit Card Number**. The cardholder's credit card number is included in the transaction message sent by the cardholder to the merchant.
4. **Expiration Date**. The expiration date of the credit card is included in the transaction message sent by the cardholder to the merchant.
5. **Security Code**. The security code is included in the transaction message sent by the cardholder to the merchant.

The merchant extracts the following data from the transaction message:

1. **Transaction Identifier** (TID). This identifier is unique and is included in the transaction message sent by the cardholder to the merchant.
2. **Credit Card Number**. The cardholder's credit card number is included in the transaction message sent by the cardholder to the merchant.
3. **Expiration Date**. The expiration date of the credit card is included in the transaction message sent by the cardholder to the merchant.
4. **Security Code**. The security code is included in the transaction message sent by the cardholder to the merchant.

The merchant then compares the extracted data with the data contained in the transaction message sent by the cardholder to the merchant. If the data matches, the merchant proceeds to authenticate the transaction. If the data does not match, the transaction is rejected.

The merchant then sends a **Response** message to the cardholder, indicating the result of the transaction.
7.3 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

SET

SSL

SSL certificate

Certificate Authority (CA)

Secure Electronic Transaction (SET)

Transport Layer Security (TLS)

Recommended Web Sites

Recommended Reading and Web Sites

MACQ9

RESCO

DREW99


Another excellent overview of SET is in the 1st and 2nd Edition available at the RESCO [RESO] or a good detailed overview of SSL and TLS.

Chapter 7 WEB SECURITY

Copyright © 2001 SYBEX, Inc.