Of the conventional payment instruments of cash, check, and card, the one most suited to low-value transactions is cash. Versatile as it is, it is limited in that no transaction can involve less than the value of the smallest coin (e.g., a penny). There are entire classes of goods and services where this poses a problem. Some examples include obtaining a quotation of the current price of a share on the stock market or making a single query of a database service. In conventional commerce, the solution to this has been to use a subscription mode of payment, where the buyer pays in advance and can avail of the product or service for a fixed period. While this ensures that the content provider can be paid for services rendered, it seals off what is in many cases a large customer base of people who may only wish to use a service very occasionally. It also restricts the ability of people to try out a service.

It is clear that the subscription model does not adequately solve the problem and that there is a need for a payment system that can efficiently transfer very small
amounts, perhaps less than a penny, in a single transaction. This implies that communications traffic, which in itself costs money, must be kept to an absolute minimum. A system in which the costs of conveying the payment are greater than the payment itself is unlikely to succeed. In many of the payment systems covered in previous chapters, a merchant validated each payment by having a real-time dialogue with a server on the network representing the payment systems provider, either to check that funds are available, or to complete the payment. This represents a very high per-transaction overhead and must be eliminated in the design of a micropayment system.

The low value per transaction also means that the profit made on each transaction must also be small. For a server to be viable under these conditions, it must be able to process transactions at a high rate. This gives rise to a further requirement that micropayment systems must be able to make the payment verification inexpensively. If a server is taking appreciable time to do public-key encryption or decryption, then its throughput, measured in transactions, cannot be very great. Consequently, a successful micropayment system must not involve computationally expensive cryptographic techniques.

The electronic payment methods outlined in earlier chapters have involved systems that mirror the properties of conventional payment instruments already in existence. Micropayments, however, have not been available in conventional commerce, and their introduction opens up many new areas of business. One can envisage network users paying to consult an on-line encyclopedia, purchasing a single song from an album, ordering just the business pages from a selection of daily newspapers, and so forth. The remainder of this chapter will outline the most influential systems available in this new field of electronic commerce.

7.1 Millicent

Millicent [1, 2] is a decentralized micropayment scheme that was developed at Digital Equipment Corporation (now Compaq) which is designed to allow payments as low as one-tenth of a cent (0.001) to be made. A Millicent payment can be efficiently validated at a vendor’s site without the need to contact a third party. This distributed approach, without any additional communication, expensive public key encryption, or off-line processing, allows it to scale effectively for repeated small payments.
The Millicent system uses a form of electronic currency called *Scrip*. Scrip can be thought of as the loose change you carry around in your pocket. It is fast and efficient to verify that it is valid, and if one loses a small piece of change by accident, it is not of great concern. Scrip is *vendor-specific* in that it has value at one particular vendor only. The security of the protocol is designed to make the cost of committing a fraud more than the value of a purchase. By using fast symmetric encryption the protocol can be both lightweight and secure.

7.1.1 The Millicent model

Figure 7.1 shows the three main entities in the Millicent system: *brokers*, *vendors*, and *customers*.

7.1.1.1 Brokers

A Millicent broker mediates between vendors and customers to simplify the tasks they perform.

*Aggregating Micropayments*  Typically, it might take a customer several weeks or months to make enough micropayments at a specific vendor to cover the cost of a standard macropayment financial transaction to that vendor. Thus it would not be efficient for customers to buy *vendor scrip* (scrip that can be spent at a specific vendor only) from every vendor from which they wish to buy. However, it is likely that a customer will make enough micropayments in total at different vendors to cover the cost of a

![Figure 7.1 The Millicent model.](image-url)
macropayment transaction. A *macropayment* is a transaction capable of handling payments worth several dollars or more, such as those systems described in previous chapters.

One of the functions of the broker is to provide all the different vendor scrip needs of a customer in return for a single macropayment. In other words, the broker sells vendor scrip to customers. The aggregation of the different vendor scrips justifies a macropayment transaction to purchase these pieces of scrip. A single vendor selling his or her own scrip would not normally justify this.

**Replacing Subscription Services** In a subscription service, a vendor usually maintains account information for customers who have paid to use the service for a set length of time. Customers have to maintain account information for each different vendor. Vendors have to create, maintain, and bill accounts for possibly a large number of users. The Millicent broker frees both the customer and vendor from these tasks, replacing a subscription service with a pay-per-access micropayment system.

**Selling Vendor Scrip** Brokers handle the real money in the Millicent system. They maintain accounts of customers and vendors. Customers buy vendor scrip for a specific vendor from their broker. The broker will have an agreement with each vendor whose scrip the broker sells. There are two main ways in which a broker gets the vendor scrip:

- **Scrip warehouse:** The broker buys many pieces of vendor scrip from the vendor. The scrip is stored and then sold piece by piece to different customers.

- **Licensed scrip production:** The actual broker generates the vendor scrip on behalf of that vendor. This is more efficient because:

  - The broker doesn’t need to store a large number of scrip pieces.
  - The vendor does less computation since the vendor doesn’t have to generate the scrip himself or herself.
  - The license, which can be granted and sent across the network, is smaller to transmit than large chunks of scrip.

The license will allow the broker to only generate a specific amount of vendor scrip. The license should be enforceable through normal
business practices. Brokers will typically be financial institutions or network service providers. They are assumed to be trusted by the other entities.

7.1.1.2 Vendors
Millicent vendors are merchants selling low-value services or information. A vendor accepts his or her own vendor scrip as payment from customers. The vendor can validate his or her own vendor scrip locally and prevent any double spending. The merchant sells vendor scrip at discount or a scrip-producing license to a broker. This discount or selling commission is how the broker profits from the scheme.

7.1.1.3 Customers
Users buy broker scrip with real money from their chosen broker, as shown in Figure 7.2. Broker scrip has value at that broker only. A macro-payment scheme such as SET or Ecash could be used to initially buy the broker scrip. Using this broker scrip, the customer buys vendor scrip for specific vendors. The vendor scrip can then be used to make purchases.

7.1.2 Purchasing with Millicent
Initially, the customer buys some broker scrip using one of the macro-payment systems, as shown in Figure 7.2. Typically, enough broker scrip to last a week might be bought, although more can be obtained at any time.

Start of week:

![Diagram](image)

1. Credit Card # (macropayment protocol)
2. $5.00 Broker scrip (Millicent protocol)

Figure 7.2 Buying broker scrip.
When a customer first encounters a new vendor, he or she must buy vendor scrip from the broker to spend at that vendor’s site. Figure 7.3 shows a customer buying 20 cents of vendor scrip using the $5 of broker scrip purchased earlier. Both the new vendor scrip and the change in broker scrip are returned. The same process will take place when a customer needs more vendor scrip, perhaps at the start of a new day.

The vendor scrip is sent to the merchant with a purchase request. The vendor will return a new piece of vendor scrip as change along with the purchased content. Remember, scrip is vendor-specific, and can be spent only at a particular merchant.

Figure 7.4 shows the customer buying from the same vendor again using the change. The customer already has valid vendor scrip for the vendor, so there is no need to contact the broker. Again, the scrip and purchase request are sent to the vendor who returns the item and the correct change. In this example the customer has bought an article costing 4 cents.

Repeated payments at a specific vendor are highly efficient in regard to network connections. If the customer already has valid scrip for that vendor, only a single network connection is required. Compare this with the number of network connections required in a secure macropayment scheme such as SET or Ecash. This increased communications efficiency is provided at the cost of slightly relaxing the security, as discussed later.

**New day or new vendor:**

1. $5.00 Broker scrip
2. $0.20 Vendor scrip
   $4.80 Broker scrip
3. $0.20 Vendor scrip + request
4. $0.19 Vendor scrip change + Purchased info/service

**Figure 7.3** Purchasing from a vendor.
Scrip is a piece of data used to represent microcurrency within the Milli-cent system. Scrip has the following properties:

- A piece of scrip represents a prepaid value, much like prepaid phone cards, fare cards, or coupons.
- Scrip can represent any denomination of currency. Expected values range from one-tenth of a cent up to about $5, although there is no defined upper- or lower-bound limits.
- The security of scrip is based on the assumption that it is only used to represent small amounts of money.
- It is vendor-specific and thus has value at one vendor only.
- It can be spent only once. Double spending will be detected locally by the vendor at the time of purchase.
- It can be spent only by its owner. A shared secret is used to prevent stolen scrip being spent, as discussed in Section 7.1.11.
- Scrip cannot be tampered with or its value changed.
- It is computationally expensive to counterfeit scrip. The cost of doing so outweighs the value of the scrip itself.
Scrip makes no use of public-key cryptography. It can be efficiently produced, validated, and protected using a one-way hash function and limited symmetric cryptography.

Scrip cannot provide full anonymity. It has visible serial numbers that could be recorded and traced. Some limited anonymity could be maintained by buying broker scrip using an anonymous macropayment system.

### 7.1.4 Scrip structure

Figure 7.5 shows the data fields that make up a piece of scrip. The purpose of each is now briefly examined:

- **Vendor:** Identifies the vendor at which this scrip has value.
- **Value:** Specifies how much the scrip is worth.
- **ID#:** A unique identifier of the scrip, much like a serial number. It is used to prevent double spending of the scrip.
- **Cust_ID#:** An identifier used to calculate a shared secret (customer_secret) that is used to protect the scrip and any scrip issued as change. Cust_ID# need not have any connection to the real identity of the customer, but it must be unique to every customer. Scrip issued as change will have the same Cust_ID# as the original scrip used to make the payment.
- **Expiry:** The date on which the scrip becomes invalid. Used to limit the ID#s that must be remembered by a vendor to prevent double spending.

![Scrip data fields](Image)

**Figure 7.5** Scrip data fields.
Info: Optional details describing the customer to a vendor. They might include the customer’s age or country of residence. Such information could assist the vendor in making a sales decision on such matters as selling adult material and the levying of sales tax. The exact fields used, if any, will depend on an agreement between the brokers and vendors.

Certificate: The certificate field prevents the scrip being altered in any way and proves that it is authentic (but not already double spent). In this sense, it acts as a digital signature, although it is not created or validated using asymmetric key cryptography.

7.1.5 Scrip certificate generation
When a piece of scrip is generated the certificate field is created as a signature or “certificate of authenticity” for that scrip. The certificate is really a stamp of approval that cannot be forged and that prevents any of the scrip’s fields from being altered.

It is created by hashing the other fields of the scrip with a secret, as shown in Figure 7.6. Only the vendor (or trusted broker) who mints the scrip will know this secret, which is called a master scrip secret. The vendor will maintain a list of many different master scrip secrets, numbered from 1 to $N$, for the purpose of minting scrip. Which master scrip secret is used with a particular piece of scrip depends on some part of the scrip’s ID#. As a simplified example, if the last digit in the ID# was 6, then master scrip secret 6 might be used.

Since the certificate is the product of a one-way hash function, such as MD5, it prevents the scrip’s fields from being altered successfully. Any change will result in a recomputed certificate not matching the original one. Only the party who knows the master scrip secret can generate scrip. Thus the scrip certificate prevents both tampering and counterfeiting.

7.1.6 Scrip validation
At the time of purchase, a vendor must be sure that the scrip the vendor is accepting is valid. It must be:

- Authentic scrip produced by the vendor or licensed broker;
- Not already spent (double spending).
Figure 7.6  Scrip certificate generation.
The merchant recalculates the certificate and compares it with the scrip certificate from the customer. This is shown in Figure 7.7. Both certificates will match if the scrip has not been tampered with.

### 7.1.7 Preventing double spending

To prevent double spending, the vendor checks that the ID# has not already been spent. The vendor maintains bit vectors (data structures where one bit is used to represent each ID#) corresponding to the issued serial numbers (ID#s) to keep track of spent scrip. Vectors covering ranges that have been fully spent or expired can be discarded. This will allow the vendor to keep the database of valid scrip ID#s in memory, which will speed up transactions.

### 7.1.8 Computation costs

Table 7.1 shows the computations required in a Millicent purchase. Compared to macropayment systems examined in earlier chapters, accepting a Millicent micropayment is cheap and efficient.

### 7.1.9 Sending scrip over a network: the Millicent protocols

When sending scrip over a network, different levels of efficiency, security, and privacy may be required. For example, on an internal network within an organization, there may be little need for privacy or security. However, on the public network these may be more important.

There are three main Millicent protocols that provide different levels of these requirements. Table 7.2 compares their characteristics. Each is now examined in turn.

#### 7.1.10 Scrip in the clear

In the simplest protocol, the customer sends the scrip unprotected across the network to the vendor. The vendor will also return the purchased content and change in the clear. No network security is provided in this protocol. An attacker can intercept the scrip or the change and use it himself or herself. Remember, the stolen scrip can only be spent at one particular vendor.

#### 7.1.11 Encrypted network connection

To prevent scrip being stolen, and to prevent an eavesdropper gaining any information from the transaction, the network connection can be
Validating scrip at the time of purchase.
encrypted. This can be done using a shared symmetric key, called the \textit{customer_secret}, between the customer and vendor. The customer_secret is used to secure the communications channel using an efficient symmetric algorithm such as Rijndael, RC6, or DES. Figure 7.8 shows a purchase using the customer_secret to encrypt the network connection. The protocol is both secure and private.

Scrip cannot be stolen and an eavesdropper cannot see the purchase or scrip details. Vendor_ID and Cust_ID# are sent in the clear in both messages so that the recipient can calculate customer_secret. Section 7.1.11.1 describes how the customer_secret is generated. The original scrip certificate is included in the response to show that it is the correct response to the request.

### 7.1.11.1 The \textit{customer_secret}

Figure 7.9 shows how the customer_secret is generated when the scrip is created. It is formed by hashing the customer identifier with another secret, called the \textit{master_customer_secret}. Only the vendor (or trusted broker) will know this secret. As with the master_scrip_secret, the vendor

---

**Table 7.1**

<table>
<thead>
<tr>
<th>Action</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recalculate certificate</td>
<td>One hash function</td>
</tr>
<tr>
<td>Prevent double spending</td>
<td>One local ID# database lookup (in memory)</td>
</tr>
<tr>
<td>Making purchase across network</td>
<td>One network connection</td>
</tr>
</tbody>
</table>

**Table 7.2**

<table>
<thead>
<tr>
<th>Millicent Protocol</th>
<th>Efficiency Ranking</th>
<th>Secure</th>
<th>Private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrip in the clear</td>
<td>1</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Encrypted connection</td>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Request signatures</td>
<td>2</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

---
maintains a list of many different master_customer_secrets, numbered from 1 to $N$. Part of the Cust_ID# is used to select the master_customer_secret.

The vendor can recalculate the customer_secret at any time from the piece of scrip. The customer must also obtain the customer_secret. It is returned to the customer when the vendor scrip is purchased from a broker, as shown in Figure 7.10. To protect the customer_secret as it passes from broker to customer, the transaction could be performed using a secure non-Millicent protocol. Alternatively, a secure Millicent transaction could be used, where a customer_secret exists for the broker scrip being used by the customer. The customer_secret would be used to encrypt the connection in much the same way as in Figure 7.8. The customer_secret for the broker scrip must be obtained using a secure protocol outside the Millicent system.
Table 7.3 summarizes the purpose and usage of the three different types of shared secret used in Millicent.

### 7.1.12 Request signatures

A fully encrypted network connection might be more than is required, especially if privacy is not important. A third Millicent protocol removes the encryption but maintains a level of security that prevents scrip being stolen.

The `customer_secret` is used to generate a request signature instead of being used for encryption. It is similar to the certificate field of a piece of scrip in that it is a hash of other fields. The request signature is generated by hashing the scrip, `customer_secret`, and request together, as shown in

![Figure 7.10 Buying vendor scrip.](image)

#### Table 7.3

Secrets Used in Producing, Validating, and Spending Scrip

<table>
<thead>
<tr>
<th>Secret</th>
<th>Shared by</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master_scrip_secret</td>
<td>Vendor, minting broker</td>
<td>Prevents tampering and counterfeiting of scrip. Used to authenticate scrip.</td>
</tr>
<tr>
<td>Customer_secret</td>
<td>Customer, vendor, minting broker</td>
<td>Proves ownership of the scrip. May be required to spend the scrip.</td>
</tr>
<tr>
<td>Master_customer_secret</td>
<td>Vendor, minting broker</td>
<td>Derives the <code>customer_secret</code> from <code>customer_information</code> in the scrip.</td>
</tr>
</tbody>
</table>
Figure 7.11. It is created by the customer and sent along with the scrip and request to the vendor. This is shown in Figure 7.12.

The vendor verifies the request signature by recomputing it, as shown in Figure 7.13. Remember, the vendor can compute the customer_secret using the scrip and a master_customer_secret (Figure 7.9). If the request has been tampered with, then the two request signatures will not match and the vendor will refuse to process the transaction.

For a valid request, the vendor returns the purchase reply, change in scrip, and a reply signature. The reply signature is generated in the same way as the request signature, using the same customer_secret. The change cannot be stolen by an attacker because it cannot be spent without knowledge of the customer_secret. This is because without the customer_secret, a valid request signature cannot be calculated, and the merchant will refuse the transaction.

Thus, while an eavesdropper can see all parts of the transaction (no privacy), the purchase request cannot be altered and the scrip cannot be stolen. Security has been provided more efficiently than using encryption, but at the cost of losing the privacy.
7.1.13 Performance

It is desirable that both vendors and brokers can process a large number of transactions a second in order to make small micropayments viable. Initial tests of a Millicent implementation on a Digital AlphaStation 400 4/233 [1] produced the following results:

- 14,000 pieces of scrip can be produced per second;
- 8,000 payments can be validated per second, with change scrip being produced;
- 1,000 Millicent requests per second can be received from the network and validated.

The bottleneck appears to be in handling the network connection, which in this case was TCP, the transport protocol used on the Internet. Thus, Millicent is capable of handling the maximum number of micropayment purchases that can be received from the network per second.

7.1.14 Millicent with the Web

Millicent is well suited for paying for Web content. The Millicent protocol can be implemented as an extension to the Web’s HTTP protocol. A software implementation consists of a user wallet, a vendor server, and a broker server.
Since Millicent supports small micropayments, users may not be so tempted to steal or copy content worth only a cent. The designers feel that users will consider it foolish to steal if the price is already so low.

7.1.15 Extensions
Millicent can securely handle transactions from one-tenth of a cent up to a few dollars. This makes it suitable for micropayments, such as paying for information content, database searching, or access to a service. However, it could also be used with a broad range of other applications that may or may not involve payments, both on the Internet and private networks. These might include:

- **Authentication to distributed services**: Scrip could be used to provide Kerberos-like authentication (see Chapter 3) for access to network services. At the start of the day, a user obtains authentication scrip from a broker. This authentication scrip is then used to buy scrip for access to particular network services. Access is dynamically provided based on a user having scrip for that system.

- **Metering usage**: Millicent could be used with accounting and metering applications inside private networks. The organization will act as a broker, with employees as the customers. The vendors will be the servers to which the employees have access.

- **Usage-based charges**: Millicent could be used for per-connection charges for such services as e-mail, file transfer, Internet telephony, teleconferencing, and other on-line services. However, it would not be efficient enough for charging per packet for these services.

- **Discount coupons**: Further fields could be added to scrip to provide discounts for certain content. For example, having bought the first half of an article, the change scrip could contain a discount for buying the second half of the article, provided it was bought the same day with that scrip.

- **Preventing subscription sharing**: By using scrip to access a prepaid subscription service, sharing of that subscription account can be prevented. The scrip acts as an access capability to the service, with the scrip change giving access the next time. However, trying to gain access with an already used piece of scrip (such as shared scrip would be) will fail.
7.1.16 Summary

The Millicent protocol was first published in 1995 [1, 2], and in the following years a number of Millicent pilot experiments took place around the world. However, it wasn’t until June 1999 that the first commercial application of Millicent went live in Japan. This first deployment based on real currency, the Japanese yen, is a cooperation between Compaq and KDD Communications. Japanese users are able to buy Millicent scrip using a credit card, and are able to make purchases as small as 0.1 yen (less than $0.001). At the time of writing, there are plans to make Millicent commercially available in North America and Europe, although deployment schedules are likely to depend on the success of the Japanese venture.

Millicent is an efficient, lightweight, flexible micropayment system. It can support multiple brokers and vendors and can be extended for use with many applications.

One drawback of the scheme is that both the broker and vendor must be trusted to issue the correct change, as there is no way to prove that change scrip is owed. Similarly the user cannot independently verify the validity of a piece of scrip, since the user cannot regenerate the scrip certificate. For the small amounts involved, vendors are unlikely to try to defraud customers, especially since the anomaly will be revealed when the user later returns to spend the change. Lightweight hash functions are used to achieve this relaxed level of security, which is suited to low-value micropayments, and Millicent significantly reduces the computational and communications overhead, compared to macropayment systems, for multiple accesses to the same vendor within a short time.

7.2 SubScrip

SubScrip [3] is a simple micropayment protocol designed for efficient pay-per-view payments on the Internet. It was developed at the University of Newcastle, Australia, and is a prepaid system with no need for user identification.

In essence, it works by creating temporary prepaid accounts for users at a specific vendor. The user makes micropayment purchases from the vendor against this account. Since the account is temporary and prepaid, it does not carry the normal overhead associated with subscription services. The system does not require its own billing or banking hierarchy.