The Millicent Protocol for Inexpensive Electronic Commerce

Steve Glassman, Mark Manasse, Martín Abadi, Paul Gauthier, Patrick Sobalvarro

Abstract

Millicent is a lightweight and secure protocol for electronic commerce over the Internet. It is designed to support purchases costing less than a cent. It is based on decentralized validation of electronic cash at the vendor's server without any additional communication, expensive encryption, or offline processing. The key innovations of Millicent are its use of brokers and of scrip. Brokers take care of account management, billing, connection maintenance, and establishing accounts with vendors. Scrip is digital cash that is only valid for a specific vendor. The vendor locally validates the scrip to prevent customer fraud, such as double spending. ***Keywords:*** *Electronic commerce, electronic cash, scrip, broker, authentication*

Electronic Commerce Background

There are a number of existing and proposed protocols for electronic commerce, such as those from DigiCash [2], Open Market [14], CyberCash [1], First Virtual [3], and NetBill [12]. They are all appropriate for medium to large transactions, $5 or $10 and up, because the costs per transaction are typically several cents plus a percentage. When these costs are applied to inexpensive transactions, 50 cents and less, the transaction costs become a significant or even dominant component of the total purchase price, thereby effectively creating a minimum price for goods and services purchased using one of these protocols.

Forcing online charges to be above some threshold reduces the options for service providers. Online services providing newspapers, magazines, reference works, and stock prices all have individual items that could be inexpensive if sold separately. The ability to purchase inexpensive individual items would make these services more attractive to casual users on the Internet. In addition, secure low-priced transactions support grass-roots electronic publishing. A user who is not likely to open a ten-dollar account with an unknown publisher may be willing to spend a few cents to buy an interesting-looking article.

In this section, we look at four existing options for Internet commerce: accounts, aggregation, credit cards, and digital cash, and discuss why they are not appropriate for inexpensive electronic commerce. In the next section, we describe our model for reducing costs and making lightweight electronic commerce feasible.

Accounts

The simplest model for electronic commerce is for customers to establish accounts with vendors. When a customer wants to perform a transaction with the vendor, the customer identifies himself (securely) and the vendor adds the cost of the transaction to the customer's account. Vendors maintain the account information and bill the customers periodically.

With accounts, transaction costs and prices can be fairly low, but there is a fair amount of overhead. An account may need to be established ahead of time and maintained over an extended period. This makes sense only when assuming a relatively long-standing relationship between a customer and a

vendor. There is often a minimum monthly charge associated with each account. The customer has separate accounts for each vendor, and the vendor needs to maintain accounts for every customer. All this overhead discourages casual users from making spur-of-the-moment purchases.

Aggregation

Aggregation amortizes billing charges over a sequence of less expensive transactions by accumulating transactions at the vendor until they exceed some threshold. Aggregation is another form of accounts and shares some of the problems of accounts. Although account setup is somewhat simplified, the vendor still has the problem of maintaining the accounts, accumulating enough transactions for a reasonable sized charge, and keeping transaction records for dispute resolution. Also, the customer must deal with separate charges from each vendor, minimum account charges, and the difficulty of contesting fraudulent charges.

Credit cards

Another simple model for electronic commerce is to use a credit card to pay for the purchase. Customers have credit cards; vendors register with credit card companies; customers give their credit card number to vendors; vendors contact their credit card companies for payment; the credit card companies handle the accounting and billing. There are established methods (like Netscape's SSL [13] based on RSA's public key encryption [16]) for ensuring secure transmission of the client's credit card number to the vendor.

Unfortunately, credit card transactions are (relatively) expensive since every purchase involves communication to a centralized credit card transaction service. In addition, credit card companies offer various features like individual item accounting, insurance, and fraud protection that add to the cost and aren't needed when purchasing inexpensive items.

Finally, customers may be unwilling to provide a credit card number to a vendor they don't know well. Although the credit card company insures the customer against any loss, there is still the inconvenience of clearing up any problems.

Digital cash

Digital cash is normally issued by a central trusted entity (like a bank). The integrity of digital cash is guaranteed by the digital signature of the issuer, so that counterfeiting digital cash is extremely hard. However, it is trivial to duplicate the bit pattern of the digital cash to produce and spend identical (and equally authentic) cash.

In an online digital cash scheme, when a vendor receives digital cash, he must contact the issuer to see if it is valid and not already spent. This extra communication makes the central site a bottleneck and adds cost to the transaction.

In an offline scheme (like one proposed by DigiCash [2]), the vendor authenticates the digital cash during the transaction and then later transmits it to the issuer to check for double spending. This scheme adds computational costs to the vendor for authenticating the digital cash, and adds messages and encryption to the protocol for pinpointing the source of the double spending.

Millicent

Our goal for Millicent is to allow for transactions that are inexpensive yet secure. We achieve this by using accounts based on scrip and brokers to sell scrip.

A piece of scrip represents an account the customer has established with a vendor. At any given time, a vendor has outstanding scrip (open accounts) with the recently active customers. The balance of the account is kept as the value of the

scrip. When the customer makes a purchase with scrip, the cost of the purchase is deducted from the scrip's value and new scrip (with the new value/account balance) is returned as change. When the customer has completed a series of transactions, he can "cash in" the remaining value of the scrip (close the account).

Brokers serve as accounting intermediaries between customers and vendors. Customers enter into long-term relationships with brokers, in much the same way as they would enter into an agreement with a bank, credit card company, or Internet service provider. Brokers buy and sell vendor scrip as a service to customers and vendors. Broker scrip serves as a common currency for customers to use when buying vendor scrip, and for vendors to give as a refund for unspent scrip.

Millicent reduces the overhead of accounts in a number of ways:

- Communication costs are reduced by verifying the scrip locally at the vendor's site; there are almost no Millicent-specific communication costs during a normal transaction. There is also no need for a centralized server or an expensive transaction-processing protocol.

  In a centralized scheme, the central site is a bottleneck; the provider must have sufficient computing power to handle the peak transaction rate. In Millicent, there is no central server; there can be many brokers, a broker is only involved in a fraction of the transactions between a customer and a vendor, and the transactions involving a broker are lightweight.

- Cryptographic costs are reduced to keep them in line with the scale of transactions; we don't need strong or expensive cryptographic schemes because the value of the scrip is relatively low. We need only make the cost of breaking the protocol greater than the value of the scrip itself.

- Accounting costs are reduced by using brokers to handle accounts and billing. The customer establishes an account with a broker; the broker establishes its own accounts with the vendors. Using brokers allows us to split a customer-vendor account into two accounts: one between the customer and broker, and another between the broker and the vendor. This reduces the total number of accounts. Instead of many separate accounts for every customer-vendor combination, each customer has only one account with a broker (or, at most, a couple of brokers); and each vendor has long-standing accounts with just a few brokers.

  In most account-based schemes, the vendor maintains the account balance. In Millicent, the customer maintains the account balance—it is encoded in the scrip held by the customer. There is no risk for the vendor because a digital signature prevents the customer from modifying the scrip's value. Since the scrip contains the account balance and a proof of correctness for that value, the vendor does not need to look up the customer's balance, saving disk activity.

- The minimum monthly charges are not as much of a problem because they are amortized over more activity. The single customer-broker account supports transactions with all vendors, and so it is likely to have enough activity to cover a minimum charge. By prepaying the broker, even the monthly accumulation of charges can be avoided.

Millicent is best suited for a series of inexpensive, casual transactions. We will rely on other protocols for initial account establishment between brokers and customers, and brokers and vendors. Other higher-value protocols are also used for the funds transfers that occur when accounts are periodically settled.

Security and Trust

The security model for Millicent is based on the assumption that scrip is used for small amounts. People and businesses treat coins differently than they treat bills, and treat small bills differently than large bills. In Millicent, we imagine people treating scrip as they would treat change in their pocket.

Since people don't need a receipt when buying candy from a vending machine, they don't need a receipt when buying an item using scrip. If they don't get what they paid for, they complain and get a refund. If they lose a coin every now and then, they aren't too upset.

We expect users to have a few dollars of scrip at a time. We don't expect them to have hundreds, or even tens, of dollars of scrip. As a result, scrip is not worth stealing unless you can steal lots of it; and if you steal lots, you will get caught.

Trust Model

Millicent assumes asymmetric trust relationships among the three entities—customers, brokers, and vendors. Brokers are assumed to be the most trustworthy, then vendors, and, finally, customers. The only time customers need to be trusted is when they complain about service problems.

We believe that brokers will tend to be large, well-known, and reputable financial institutions (like Visa, MasterCard, and banks) or major Internet or online service providers (like CompuServe, NETCOM, or AOL). We expect there to be many vendors covering a full spectrum of size and trustworthiness, as in the real world. Finally, there will be large numbers of customers who are as trustworthy as people are in general.

Three factors make broker fraud unprofitable. First, customer and vendor software can independently check the scrip and maintain account balances, so any fraud by the broker can be detected. Second, customers do not hold much scrip at any one time, so a broker would have to commit *many* fraudulent transactions to make much of a gain, and this makes them likelier to be caught. Finally, the reputation of a broker is important for attracting customers and a broker would quickly lose its reputation if customers, have troubles with the broker. The repeat business of active customers is more valuable to a broker than the scrip that it could steal.

Vendor fraud consists of not providing goods for valid scrip. If this happens, customers will complain to their broker, and brokers will drop vendors who cause too many complaints. This acts as an effective policing mechanism, because vendors need a broker to easily conduct business in Millicent.

As a result, the Millicent protocol is skewed to prevent customer fraud (forgery and double spending) while providing indirect detection of broker and vendor fraud.

Security

The security of Millicent transactions comes from several aspects.

All transactions are protected
: Every Millicent transaction requires that the customer knows the secret associated with the scrip. The protocol never sends the secret in the clear, so there is no risk due to eavesdropping. No piece of scrip can be reused, so a replay attack will fail. Each request is signed with the secret, so there is no way to intercept scrip and use the scrip to make a different request.

Inexpensive transactions limit the value of fraud
: Inexpensive transactions can rely on inexpensive security: it's not worth using expensive computer resources to steal inexpensive scrip. In addition, it would take many illegal uses of scrip to acquire much money, and that raises the probability of getting caught.

Fraud is detectable and eventually traceable
: Fraud is detected when the customer doesn't obtain the desired goods from the vendor, or when the balance returned to the customer

doesn't match the balance due. If the customer is cheating, then the vendor's only loss is the cost of detecting the bad scrip and denying service. If the vendor is cheating, the customer will report a problem to the broker. When a broker notices a pattern of complaints from many customers against a vendor, it can pinpoint the fraud and cut off all dealings with the vendor. If a broker is cheating, the vendor will notice bad scrip coming from many customers, all originating from a single broker. The vendor can then publicize its complaint in an appropriate venue.

Scrip

The main properties of scrip are:

- It has value at a specific vendor.
- It can be spent only once.
- It is tamper resistant and hard to counterfeit.
- It can be spent only by its rightful owner.
- It can be efficiently produced and validated.

The next sections give more detail about scrip and its use, but the basic techniques to achieve these properties are outlined here:

- The text of the scrip gives its value and identifies the vendor.
- The scrip has a serial number to prevent double spending.
- There is a digital signature to prevent tampering and counterfeiting.
- The customer signs each use of scrip with a secret that is associated with the scrip.
- The signatures can be efficiently created and checked using a fast one-way hash function (like MD5 [15] or SHA [11]).

Scrip Structure

There are three secrets involved in producing, validating, and spending scrip. The customer is sent one secret, the `customer_secret`, to prove ownership of the scrip. The vendor uses one secret, the `master_customer_secret`, to derive the `customer_secret` from customer information in the scrip. The third secret, the `master_scrip_secret`, is used by the vendor to prevent tampering and counterfeiting.

The secrets are all used in a way that shows knowledge of the secret without revealing the secret. To attest to a message, the secret is appended to the message, and the result is hashed to produce a signature. The message (without the secret) and the signature prove—due to the one-way nature of the hash function—knowledge of the secret, because the correct signature can only be derived if you know the secret.

Scrip has the following fields (Figure 1):

- `Vendor` identifies the vendor for the scrip.
- `Value` gives the value of the scrip.
- `ID#` is the unique identifier of the scrip. Some portion of it is used to select the `master_scrip_secret` used for the certificate.
- `Cust_ID#` is used to produce the customer secret. A portion of `Cust_ID#` is used to select the `master_customer_secret` which is also used in producing the customer secret.
- `Expires` is the expiration time for the scrip.
- `Props` are extra data describing customer properties (age, state of residence, etc.) to the vendor.
- `Certificate` is the signature of the scrip.

Validation and Expiration

Scrip is validated in two steps. First (Figure 2), the certificate is recomputed and checked against

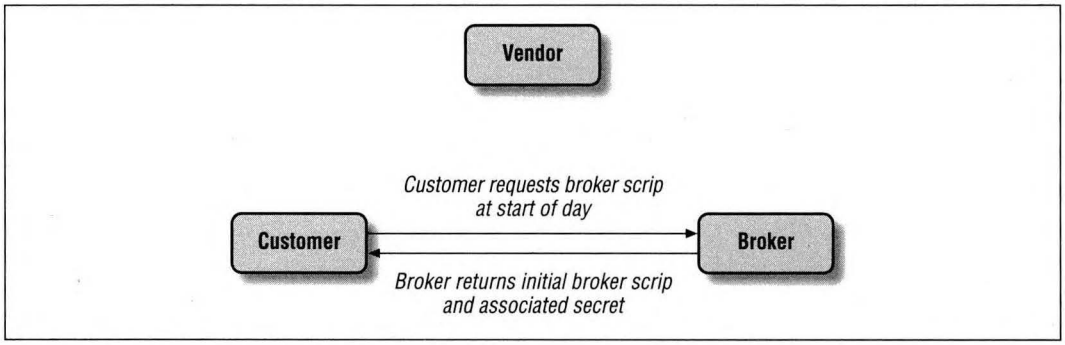

Figure 1: The certificate of a piece of scrip is generated by hashing the body of the scrip with a secret. The secret is selected using a portion of the scrip's ID#

the certificate sent with the scrip. If the scrip has been tampered with, then the two certificates will not match. Second, there is a unique identifier (ID#) included in the scrip body and the vendor can check for double spending by seeing if it has recorded that identifier as already spent. Generating and validating scrip each require a little text manipulation and one hash operation. Unless the secret is known, scrip cannot be counterfeited or altered.

The vendor records the unique identifier of every piece of scrip that is spent, so that it cannot be fraudulently respent. To save the vendor from maintaining this record forever, each piece of scrip is given an expiration time. Once the scrip expires, the vendor no longer has to worry about its being respent and can erase its record of the scrip.

Customers are responsible for renewing or cashing in scrip before it expires. The old scrip is submitted to the vendor, who returns new scrip with a later expiration time (and a new serial number). Vendors may choose to charge a small fee for this service, discouraging users from obtaining more scrip than they will need in the near future.

Properties

Scrip also has fields for storing properties, which are inserted by the vendor or broker when the scrip is produced. The exact property fields and their values will depend on an agreement between the brokers and vendors. The brokers

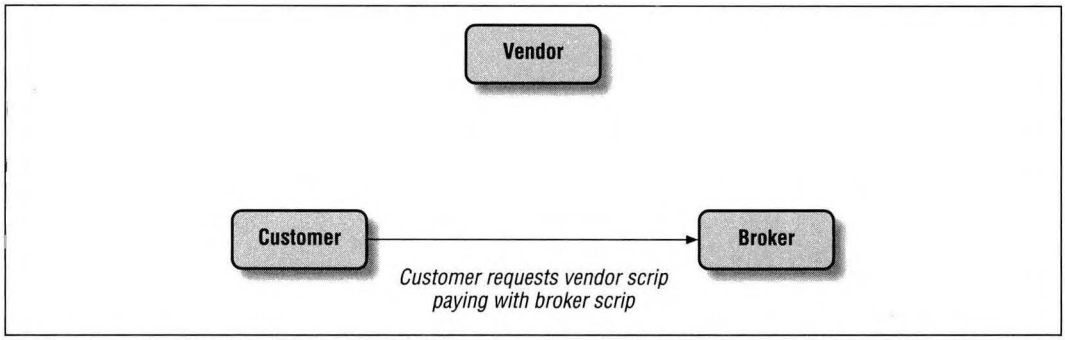

Figure 2: The received scrip is validated by regenerating the certificate and comparing it to the transmitted one. If they are identical, the scrip is valid

will get the information from customers when they create their account and enforce some set of rules when selling vendor scrip. Vendors, of course, are free to include whatever properties they desire in scrip they produce themselves.

Information such as the state of residence, or age of the consumer assists the vendor in making sales decisions. Adult material could only be bought if the scrip shows the customer is old enough. State sales tax charges can depend on a property included in the scrip.

Millicent Protocols

Scrip is the basis of a family of Millicent protocols. We will describe three of them and compare their simplicity, secrecy, and security. (A detailed description of the protocols is in the appendix.)

The first, "scrip in the clear," is the simplest and most efficient protocol. It is the basis for the other two protocols, but it may not be useful in practice because it is too insecure. The second, "private and secure," is secure and offers good privacy, but it is more expensive. The third, "secure without encryption," is also secure, but trades privacy for greater efficiency.

Scrip in the Clear

In the simplest possible Millicent protocol, the customer just sends an unspent piece of scrip in the clear (i.e., not encrypted or protected in any way) along with each request to the vendor. The vendor returns the desired result along with a new piece of scrip (also in the clear) as change.

This protocol offers almost no security; an eavesdropping third party can intercept the scrip being returned as change and use it himself. When the rightful owner later attempted to spend the scrip, the vendor would have a record of its being previously spent, and would refuse the request.

Private and Secure

To add security and privacy to the Millicent protocol, we establish a shared secret between the two parties and then use the secret to set up a secure communications channel using an efficient, symmetric encryption method (such as DES [10], RC4 [17], or IDEA [6]).

In Millicent, scrip can be used to establish this shared key. When a customer buys an initial piece of scrip for a vendor, a secret is generated based on the customer identifier, and returned securely with the scrip (Figure 3). This requires either that the transaction be performed using some secure non-Millicent protocol, or that the scrip be purchased using a secure Millicent transaction.

The vendor does not directly record the secret associated with the piece of scrip. Instead, the customer identifier (Cust_ID#) field of the scrip allows rapid recalculation of the secret. The customer identifier must be unique whenever scrip is transmitted to a new customer, but it need not have any connection to the identity of the customer.

When the vendor receives the request, he derives the customer secret from the customer identifier in the scrip, derives the message key from the customer secret, and uses the message key to decrypt the request. The change scrip can be returned in the clear, while the response and any new secrets are returned to the customer encrypted by the message key.

In this protocol the request and the response are kept totally private; unless an eavesdropper knows the customer secret, he can't decrypt the messages. In addition, an eavesdropper can't steal the scrip because it can't be spent without knowing the customer secret.

Secure without Encryption

The previous section describes how the secret shared by the customer and vendor can be exploited to achieve security and privacy. But a

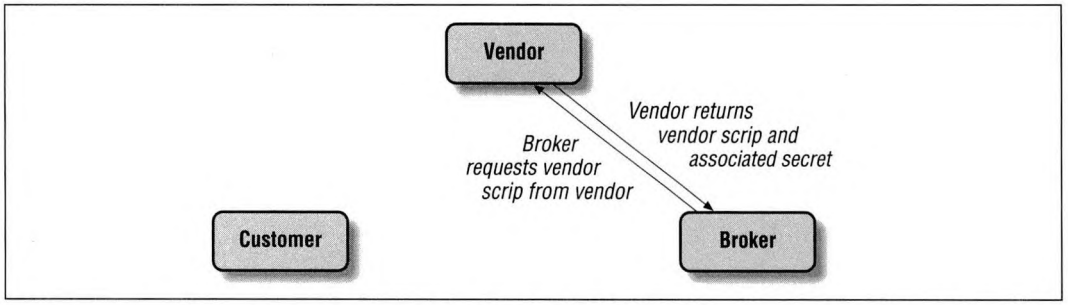

Figure 3: The customer secret is generated by hashing the customer identifier with a secret. The secret is selected using a portion of the customer identifier.

full-blown encrypted channel may be overkill for some Millicent applications. In this, our third variant of the protocol, we give up the privacy of the request and response to eliminate the use of encryption.

As in the previous protocol, the customer securely gets an initial piece of scrip and customer secret. To make a purchase, the customer sends the request, scrip, and a "signature" of the request to the vendor. The signature is produced in the same way that the certificate of the scrip is produced. The scrip and request are concatenated with the customer secret. The customer runs an efficient cryptographic one-way hash function over this string and sends the resulting hash as the signature.

When the vendor receives the request, he derives the customer secret from the scrip and regenerates the signature for the request. If the scrip or request have been tampered with in any way, the signature will not match (Figure 4).

The vendor now handles the request and returns a fresh piece of scrip as change. The change scrip shares the same customer identifier as the scrip submitted with the request, so that the original customer secret can be used to spend the change. There is no need to encrypt any of the response; an eavesdropper can't steal the scrip because the signature of the request can't be made without knowing the customer secret. The vendor may sign the response with the customer secret in order to prove authenticity to the customer.

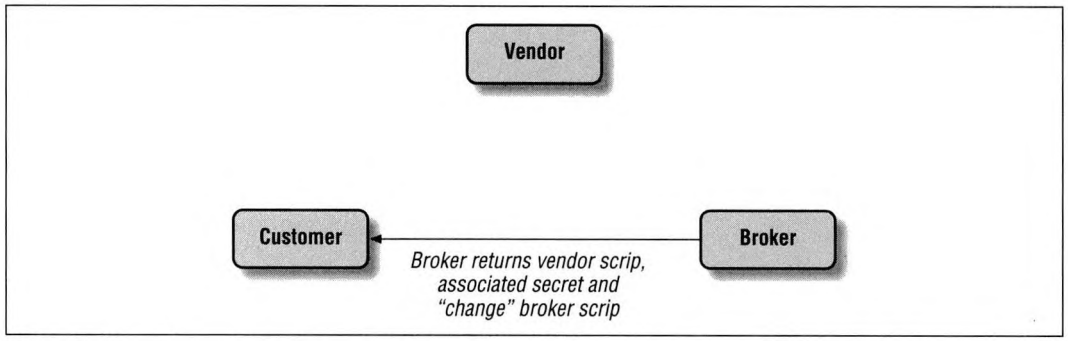

Figure 4: The request is validated by regenerating the request signature and comparing to the transmitted signature. If they match, the request is valid.

Thus, with only a few hashes, Millicent provides a lightweight and secure protocol.

Brokers

Brokers maintain the accounts of customers and vendors, and they handle all real-money transactions. The customer establishes an account with a broker by using some other method (like a credit card or a higher-security electronic commerce system) to buy some broker scrip. The customer then uses the broker scrip to buy vendor scrip.

The vendor and the broker have a long-term business relationship. The broker sells vendor scrip to customers and pays the vendor. There can be different business models for the way the broker gets vendor scrip, for example, pay in advance, consignment sale, or licensed production. In all models, the broker can make a profit selling scrip because he pays the vendor (at a discount) for scrip in bulk and sells individual pieces to customers.

When a customer wants to make a purchase, the customer contacts the broker to obtain the necessary vendor scrip. The customer uses his broker scrip to pay for the vendor scrip using the Millicent protocol. The broker returns the new vendor scrip along with change in broker scrip.

We will examine three ways in which the broker gets the vendor scrip. The "scrip warehouse" model assumes a casual relationship between the broker and vendor. The "licensed scrip producer" model assumes a substantial and long-lasting relationship between the broker and vendor. The "multiple broker" model assumes a relationship between brokers, but requires no relationship between the vendor and broker.

Scrip Warehouse

When the broker is acting as a scrip warehouse, the broker buys multiple pieces of scrip from a vendor. The broker stores the scrip and sells the pieces one at a time to customers (Figure 6-8).

This model assumes no special relationship between the vendor and broker. It works best when the broker's customers have a light to moderate demand for that vendor's scrip. The broker uses the Millicent protocol to buy the scrip from the vendor in the same way a customer would. Selling scrip in large blocks is more efficient for the vendor since the communication and financial transaction costs are amortized over all the pieces of scrip. We presume that the vendor offers some sort of volume discount to encourage brokers to buy large blocks of scrip. The broker makes a profit when it resells the scrip to customers at full price. The vendor depends on the broker to ensure any customer properties encoded in the scrip.

Licensed Scrip Production

If a broker's customers buy a lot of scrip for a specific vendor, it may be desirable for a vendor to "license" the broker to produce vendor scrip. This means that the broker generates scrip that the vendor can validate and accept. The vendor sells the broker the right to generate scrip using a given `master_scrip_secret`, series of scrip ID#'s, `master_customer_secret`, and series of customer identifiers. The vendor can validate the licensed scrip because the `master_scrip_secret` is known from the series of the scrip ID# and the `master_customer_secret` is known from the series of the customer identifier.

Brokers produce the scrip and collect money from customers; vendors record the total value of scrip originating from a particular broker. When all the scrip produced under a particular contract has expired, brokers and vendors can settle up. The broker presumably takes some commission for producing the scrip.

A license covers a specific series (unique range of identifiers—ID#'s) of scrip for a given period of time, and the secrets shared between the broker and vendor only apply to that series. A vendor can issue licenses to different brokers by giving out different series and secrets to each one.

Of course, a vendor can produce its own scrip using its own private series and secrets.

Licensing scrip production is more efficient for the vendor and broker than the scrip warehouse model. There is less communication because the license is smaller to transmit than a few pieces of scrip. The vendor does less computation since it does not have to generate the scrip itself. The broker does not have to store large blocks of scrip, since it can generate the scrip on demand. Additionally, it allows the broker to encode specific user properties into each piece of scrip it generates.

Multiple Brokers

In an environment where there are multiple brokers, a customer of one broker may want to make a purchase from a vendor associated with another broker. If the vendor only wants to have an account with its own broker (perhaps to simplify accounting), the customer will have to go through the vendor's broker to buy vendor scrip.

The entire transaction will go like this:

- The customer asks his broker for vendor scrip.
- The customer's broker tries to set up an account with the vendor.
- The vendor tells the customer's broker his broker's name.
- The customer's broker buys broker scrip from the vendor's broker.
- The customer's broker returns the vendor's broker's scrip to the customer.
- The customer buys vendor scrip from the vendor's broker.
- The customer uses the vendor scrip at the vendor.

The idea of licensed scrip production can be extended so that brokers can generate broker scrip for other brokers.

Customer, Broker, and Vendor Interactions

The following diagrams (Figures 5-10) present the steps for a complete Millicent session (including the broker buying scrip from the vendor). The initial step (Figure 5) happens only once per session. The second step (Figure 6) happens each time the customer has no stored scrip for a vendor. Step three (Figure 7) happens only if the broker must contact the vendor to buy the scrip. It is not needed for licensed scrip production. The fourth step (Figure 8) shows the broker returning the vendor scrip to the customer. The fifth step (Figure 9) shows the customer using the scrip to make a purchase from the vendor.

The last step (Figure 10) shows a typical Millicent transaction. The customer already has vendor scrip and uses it to make a purchase. There are no extra messages or interactions with the broker.

Status

We have produced an initial implementation of Millicent [9] consisting of a set of libraries, and a vendor and broker written using the libraries for Millicent transactions across a network using TCP/IP. Our measurements show that the Millicent protocol is efficient enough for sub-cent purchases. Our untuned vendor implementation can validate about 1000 Millicent requests per second (on a Digital AlphaStation 400 4/233) and, of that, most of the time goes into the TCP connection handling.

Using zero-cost transactions, Millicent scrip can be used as a distributed capability. Using this aspect of scrip, our first application of Millicent is in a Kerberos-like [5] authentication suite for our network firewall services. We have modified a SOCKs [7] based TCP relay, rlogin daemon, FTP daemon, and rlogin, telnet, and FTP clients to use Millicent scrip to convey authentication information. A user does one cryptokey (cryptographic challenge/response) authentication to get scrip

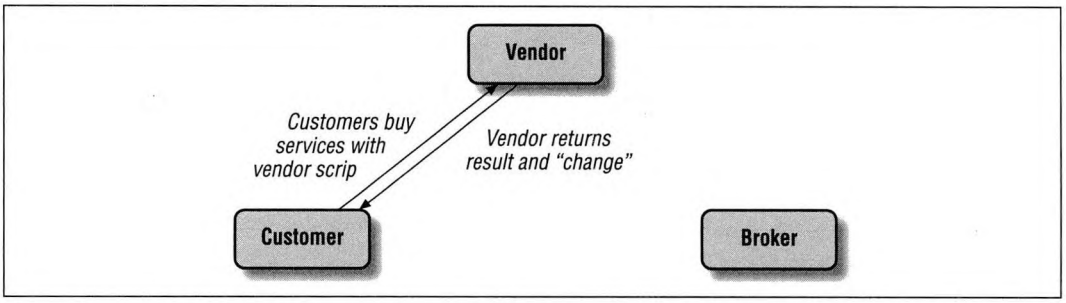

Figure 5: The client makes a secure connection to the broker to get some broker scrip

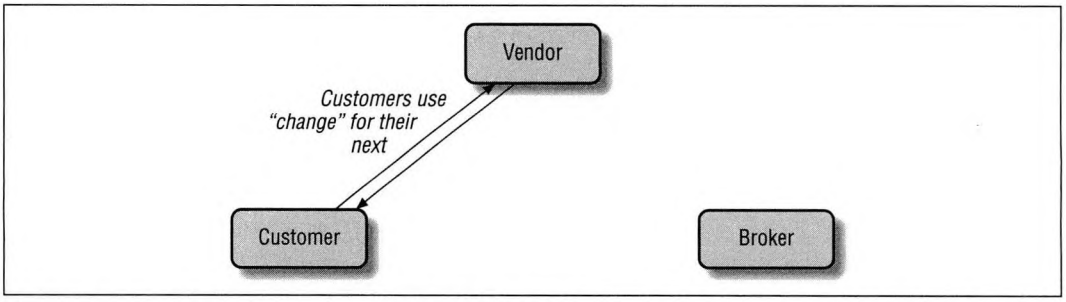

Figure 6: If the client doesn't already have scrip for a particular vendor, he contacts the broker to buy some using his broker scrip

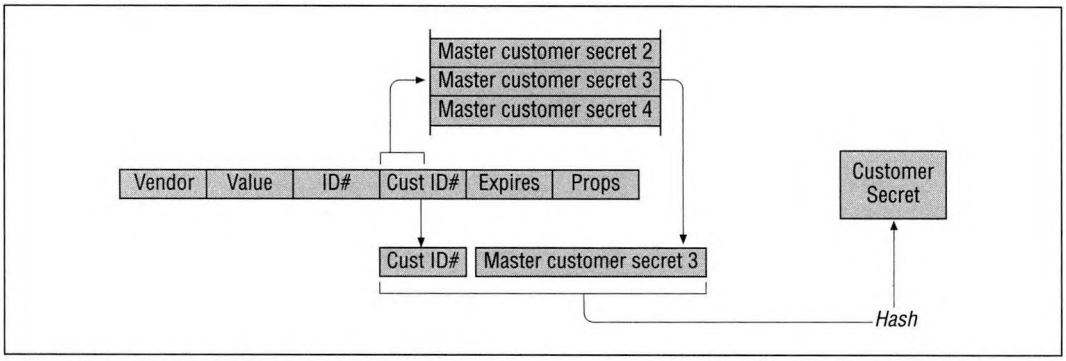

Figure 7: If the broker doesn't already have scrip for that vendor, he buys some from the vendor

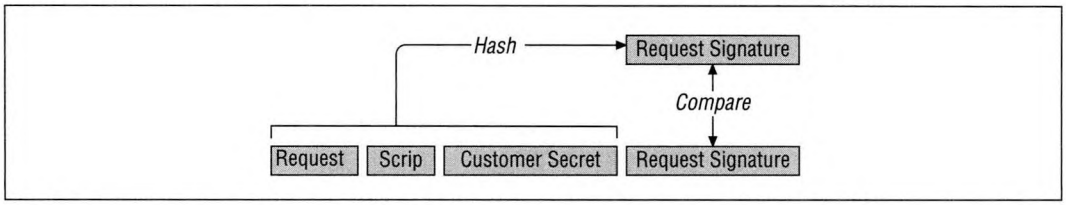

Figure 8: The broker returns vendor scrip and change (in broker scrip) to the client

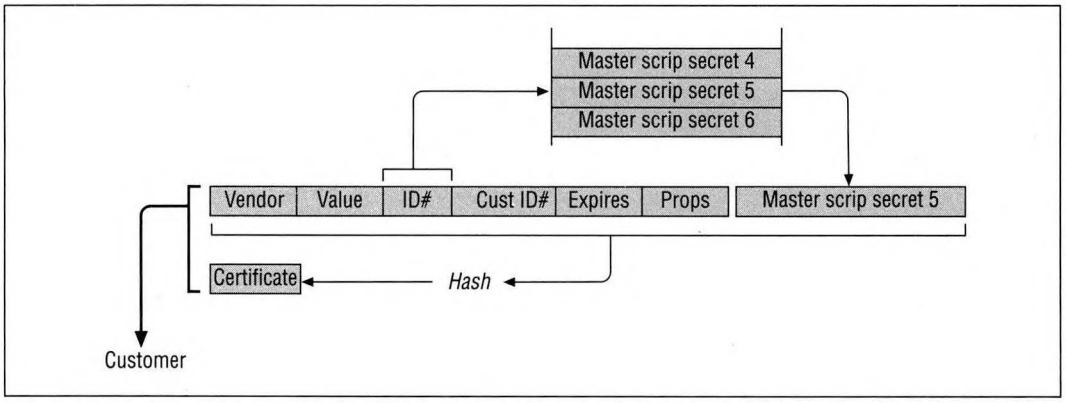

Figure 9: The customer uses the vendor scrip to make a purchase from the vendor. The vendor returns change (in vendor scrip) to the client

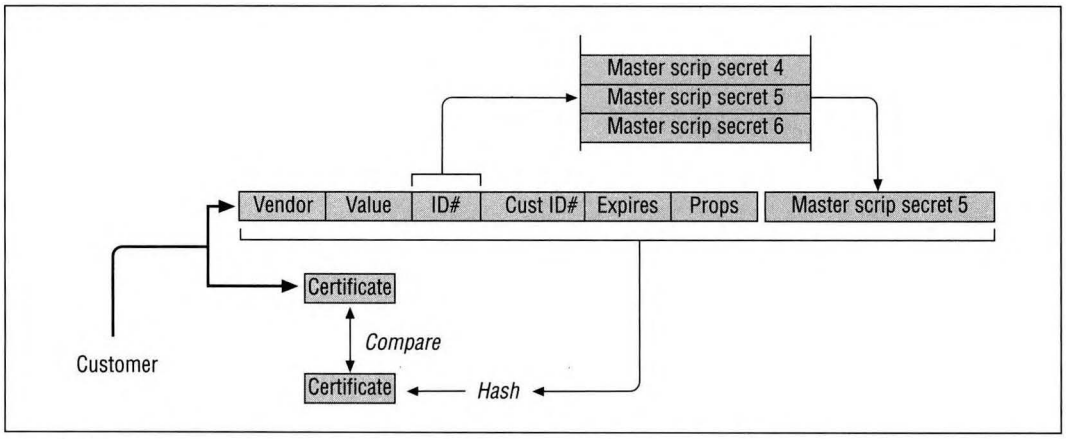

Figure 10: The customer continues using the change to make more purchases

from an authentication broker. Then, for the rest of the day, the user can use the authentication scrip to buy scrip for particular firewall services.

We are also working on Millicent-based World Wide Web (WWW) services. We have developed a local *pseudo-proxy* that intercepts all requests from the client's WWW browser and modifies the HTTP header to add scrip as necessary. The WWW server checks the HTTP request for sufficient scrip to buy the page and returns the page with change in the HTTP response. The pseudo-proxy extracts the change before forwarding the response to the browser. When Millicent becomes popular, the functionality of the pseudo-proxy can be integrated in with the browser.

Future and Applications

The range of potential applications for Millicent is quite broad. With current technology, Millicent is appropriate for transactions from a few dollars to as little as one-tenth of a cent. The upper bound comes from the trust model for brokers and the availability of alternative protocols appropriate for transactions above a few dollars,

while the lower bound comes from a conservative estimate based on the computational costs of a broker. This price range covers most print and information services that will be available in an online format—magazines, newspapers, encyclopedias, indices, newsletters, and databases.

MacKie-Mason and Varian [8] argue that as the Internet develops there will be increasing pressure for usage-based charges. Current free Internet services like email, file transfers, the Internet telephone, and teleconferencing will have to be paid for. At the lowest level, they estimate that the cost of transmitting one packet on the Internet backbone is one six-hundredth of a cent. We don't believe that Millicent is quite efficient enough for such packet-level charges; for these there are proposals like the noncryptographic *Digital Silk Road* [4]. We do believe that Millicent can be used for per-connection charges for these services.

Conclusion

We see growing opportunities for inexpensive Internet services. These services need an appropriate electronic commerce protocol. We believe that the Millicent protocol is a good candidate to be that protocol. ■

Appendix

The following is a more precise description of the Millicent protocol for customer and vendor. The interaction between customer and broker relies on the same protocol, as explained in the main body of the paper.

We use the following notations:

`X, Y, Z`
: Represents the string encoding the tuple `X, Y, Z`

`H(X)`
: Is the result of hashing `X` with a cryptographic hash function, such as MD5

`{X}Y`
: Is the result of encrypting `X` with a cryptographic function, such as DES, under the key `Y`

`A->B: X`
: Means that `A` sends `X` to `B`

Before giving the protocol, we describe its fields. First, we list some of the ingredients of scrip; scrip includes the name of the vendor, some properties of the customer, a value, and an expiration time:

`vendor_id:`
: A unique identifier (or name) for the vendor

`props:`
: Any data describing customer properties (possibly including a name)

`value:`
: The value of the scrip

`exp:`
: The expiration time for the scrip

The customer and the vendor generate a request and a reply. Both of these are arbitrary strings.

`request:`
: The request from the customer

`reply:`
: The reply from the vendor. (We assume that, by its format, `reply` is distinguishable from `request`.)

In addition, the protocol relies on various secrets, and on corresponding identifiers for those secrets:

`master_scrip_secret:`
: A secret used for certifying scrip. The `master_scrip_secret` is known only to the vendor (when the vendor produces his own scrip) or only to the vendor and to the broker (when the broker may produce scrip).

`id_series#:`
: An identifier for `master_scrip_secret`.

The vendor can map id_series# to master_scrip_secret.

id_sequence#:
A unique identifier, such as a sequence number. The vendor accepts id_sequence# for at most one transaction in conjunction with id_series#.

id# = id_series#, id_sequence#

master_customer_secret:
A secret used for producing customer secrets. The master_customer_secret is known only to the vendor and to the broker.

cust_id_series#:
An identifier for master_customer_secret. The vendor can map cust_id_series# to master_customer_secret.

cust_id_sequence#:
A unique identifier, such as a sequence number. Together with cust_id_series#, it identifies the customer.

cust_id# = cust_id_series#, cust_id_sequence#

customer_secret = H(cust_id#, master_customer_secret)
A secret that the vendor or the broker sends to the customer. The customer gets this quantity along with cust_id# (but not master_customer_secret); the customer can map vendor_id and cust_id# to customer_secret. Both the vendor and the broker can generate customer_secret from cust_id# and master_customer_secret. No one else knows customer_secret.

Scrip is generated by combining all of the fields listed above, as follows:

```
id_material = vendor_id, id#, cust_id#
cert_material = props, value, exp
scrip_body = id_material, cert_material
```

cert = H(scrip_body, master_scrip_secret)
This is a certificate that proves the authenticity of scrip_body for the customer associated with cust_id#.

scrip = scrip_body, cert
The vendor or the broker gives scrip to the customer; the customer presents it to the vendor along with a request.

For change returned from a transaction, the vendor issues new scrip, with a new certificate. The new quantities may differ from the previous ones in all their components except for cust_id# and vendor_id. If the broker initially knows master_scrip_secret and this quantity remains the same, then the broker is in principle capable of producing change instead of the vendor. This may not be desirable, since it implies unnecessary trust from the vendor to the broker; hence, when the vendor makes change, it is sensible for the vendor to use a new master_scrip_secret not known to the broker. In any case, the vendor should pick a new value for id# as a protection against replays. We write scrip' for the new scrip.

- In the clear (insecure)
```
customer -> vendor: scrip, request
vendor -> customer: scrip', reply
```
- Authentic and private
```
customer -> vendor: vendor_id, cust_
    id#, {scrip, request} customer_
    secret
vendor -> customer: vendor_id, cust_
    id#, {scrip', cert,
    reply}customer_secret
```

  Most of the communication is under customer_secret for authenticity and privacy. It is possible to encrypt less, with a gain in efficiency. For example, some parts of scrip' are not sensitive and could be sent in the clear.

  The response includes cert in order to allow the customer to check that the

response received is in fact a response to the request.

Both messages include `vendor_id` and `cust_id#` in the clear in order to allow the recipient to generate `customer_secret`.

- Authentic but not private

```
customer->vendor: scrip, request,
    H(scrip, request, customer_
    secret)
vendor->customer: scrip', reply,
    H(scrip', cert, reply, customer_
    secret)
```

All messages are sent in the clear, but they are protected by the signatures (based on `customer_secret`).

No encryption is used and only five hashes are necessary at the server to handle a request. The hashes are: (1) for checking the old scrip, (2) for regenerating `customer_secret`, (3) for checking the customer's signature, (4) for generating the new scrip, and (5) for signing the response.

The response includes `cert` in order to allow the customer to check that the response received is in fact a response to the request.

Both messages include `vendor_id` and `cust_id#` in the clear (in `scrip` and `scrip'`) so the recipient can generate `customer_secret`.

References

1. CyberCash Inc., URL: *http://www.cybercash.com/*
2. DigiCash Inc., *http://www.digicash.com/*
3. First Virtual Holdings Inc., URL: *http://www.fv.com/*
4. Norman Hardy and Eric Dean Tribble, *The Digital Silk Road, ftp://ftp.netcom.com/pub/jo/joule/DSR/DSR1.txt.gz* or *http://web.gmu.edu:80/bcox/Bionomics/Extropians/HardyTribbleSilkRoad.html*
5. J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5), IETF RFC 1510, gopher://ds2.internic.net/00/rfc/rfc1510.txt*
6. Xuejia Lai, *On the Design and Security of Block Ciphers*, Institute for Signal and Information Processing, ETH-Zentrum, Zurich, Switzerland, 1992.
7. M. Leech, M. Ganis, Y. Lee, et al., *SOCKS Protocol Version 5*, draft-ietf-aft-socks-protocol-v5-04.txt, *http://src.doc.ic.ac.uk/computing/internet/internet-drafts/draft-ietf-aft-socks-protocol-v5-04.txt.Z*
8. Jeffrey K. MacKie-Mason, and Hal R. Varian. *Some FAQs about usage based pricing.* University of Michigan, September 1994, *ftp://gopher.econ.lsa.umich.edu/pub/Papers useFAQs.html*
9. Mark S. Manasse, *A Method for Low Priced Electronic Commerce*, Patent pending, *http://www.research.digital.com/SRC/personal/Mark_Manasse/uncommon/ucom.html*
10. National Institute for Standards and Technology (NIST), *Data Encryption Standard (DES), Federal Information Processing Standards Publication 46-2*, December 1993, *http://www.ncsl.nist.gov/fips/fips46-2.txt*
11. National Institute for Standards and Technology (NIST), *Secure Hash Standard, FIPS PUB 180-1: Secure Hash Standard*, April 1995, *http://csrc.ncsl.nist.gov/fips/fip180-1.txt*
12. NetBill, Carnegie Mellon University, *http://www.ini.cmu.edu/netbill/*
13. Netscape Inc., *SSL Protocol, http://home.netscape.com/newsref/std/SSL.html*
14. Open Market Inc., *http://www.openmarket.com/*
15. R. Rivest, *The MD5 Message-Digest Algorithm, IETF RFC 1321, gopher://ds2.internic.net/00/rfc/rfc1321.txt*
16. RSA Inc., *PKCS#1: RSA Encryption Standard, http://www.rsa.com/pub/pkcs/ps/pkcs-1.ps*
17. RSA Inc., *RSA's Frequently Asked Questions About Today's Cryptography, http://www.rsa.com/rsalabs/faq/faq_misc.html#misc.6*

About the Authors

Martín Abadi

[*http://www.research.digital.com/people/Martin_Abadi/bio.html*]

Systems Research Center, Digital Equipment Corporation

ma@pa.dec.com

Paul Gauthier

[*http://www.cs.berkeley.edu/~gauthier/*]

University of California Berkeley

gauthier@cs.berkeley.edu

Paul Gauthier is a Ph.D. student at the University of Califoria, Berkeley. He was a summer intern at SRC in Summer, 1995.

Steve Glassman
[*http://www.research.digital.com/people/Steve_Glassman/bio.html*]
Systems Research Center, Digital Equipment Corporation
steveg@pa.dec.com

Mark S. Manasse
[*http://www.research.digital.com/people/Mark_Manasse/bio.html*]
Systems Research Center, Digital Equipment Corporation
msm@pa.dec.com

Patrick Sobalvarro
[*http://www.psg.lcs.mit.edu:80/~pgs/*]
Massachusetts Institute of Technology
pgs@lcs.mit.edu
Patrick Sobalvarro is a Ph.D. student at the Massachusetts Institute of Technology. He was a summer intern at SRC in Summer, 1995.