(53)

An Implementation of MicroMint

by

Jeffrey Burstein

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

[illegible]

Author _____
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by _____
Ronald L. Rivest
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

An Implementation of MicroMint

by

Jeffrey Burstein

Submitted to the Department of Electrical Engineering and Computer Science on May 22, 1998, in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis describes a prototype implementation of MicroMint, an Internet micropayment system designed to facilitate very small scale monetary transactions over the World Wide Web. By implementing a proposed system, we can determine its feasibility for real commercial applications, and identify advantages and disadvantages inherit in different systems. This prototype implementation pays special attention to details necessary for end users to adopt the payment system easily.

Thesis Supervisor: Ronald L. Rivest
Title: Professor

Table of Contents

Table of Figures

Chapter 1: Introduction

The explosion of the Internet into mainstream use has led to the creation of thousands of different web sites providing information services to millions of people around the world. Producing and maintaining a quality web site takes a great deal of time, dedication, and money. To help offset the monetary costs, web site producers turn to using their site as a source of revenue through advertising and subscriptions. An Internet micropayment system would allow web site producers access to an additional method for generating revenue: being paid for each time a user views a page on their site.

Micropayment systems allow users to spend small amounts of money at web sites in exchange for various content or services. Their design is usually quite different from their existing "macro-payment" counterparts, since micropayment systems must be very simple and efficient, with a very low cost per transaction.

The goal of this thesis is to implement a fully functional prototype of the MicroMint micropayment scheme [10]. By implementing all aspects of a working system, we can explore how each component interacts and determine how to design a micropayment system that is both secure and transparent. The primary emphasis will be to examine how

to design each component from the user's perspective. More information about MicroMint and this implementation can be found through the MIT/LCS Cryptography and Information Security Group home page, http://theory.lcs.mit.edu/~cis/.

1.1 Paying for Content

1.1.1 How We Currently Pay

The typical content providing web site such as an online magazine or information service uses either advertising or direct user subscriptions to generate revenue. Advertising supported sites typically show a single graphical banner advertisement per page, and are usually paid by their advertisers based on how many people view the ad and click on it to visit the advertiser's web site. Most content providing sites fall into this category.

While advertising may be an ideal revenue source for large content providers, it often falls short of supplying needed revenue to smaller, more specialized sites.[1] These sites face a choice – either close their doors or charge subscriptions.

A web site which is subscription based generates revenue by collecting money directly from their viewers for a limited-time subscription. Users typically pay somewhere between $10-$100 for a year's worth of unlimited access. Under this model, only those people who have purchased subscriptions can access the site by providing a name and password.

[1] Over the past year, there have been several well-publicized instances of advertising supported sites announcing that they cannot continue to be supported by advertising alone and have moved to a subscription-based revenue model. For an example, see Janelle Brown's article, "Premium Content Providers to Charge," http://www.wired.com/news/news/culture/story/9673.html.

However, subscriptions are far from the small web site's panacea. By charging subscriptions, content publishers are preventing vast numbers of users from visiting their site because they do not wish to purchase subscriptions. The subscription site gives users an all or nothing choice – either they pay for the ability to view everything, or they can view nothing. In addition, other sites cannot easily link to pages on subscription supported sites since their readers will generally not have a subscription to the destination site, thus eliminating one of the web's great advantages over other media.

1.1.2 Pay-Per-Click

The solution to the small web site's advertising-subscription quandary is to charge users individually for each page of content. This way, users can pay for only the pages they want to read, and producers don't alienate thousands of casual viewers. A small fee, from 1 to 10 cents per page, would cover the costs of producing a small web site.

1.2 Implementing Pay-Per-Click

In order for users to be able to enter a web site and pay for pages of content, there needs to be an underlying payment mechanism. Our goal is to create such a system with as little change as necessary made by the user.

1.2.1 Micro- vs. Macro-Payments

When contemplating how to implement a pay-per-click payment system, one must first wonder why we don't use existing payment systems. The answer is quite simple – the transaction costs are too high. Current "macro-payment" systems have extremely unwieldy transaction costs – the typical cost to process a credit card transaction is around

$0.25 [1]. As a result, it doesn't make sense to use existing systems for any transaction smaller than a dollar or two.

1.2.2 Micropayments Explained

An Internet micropayment system is a set of software, algorithms, and network protocols designed to allow users to make small sized monetary transactions across the Internet. These systems typically have extremely low overhead, and support transactions which fall below the acceptable range of existing systems – from fractions of a penny to at most a dollar or two. By limiting overhead, micropayment systems have very low transaction costs, which make them feasible for small transactions.

Chapter 2: Background

2.1 MicroMint

The MicroMint micropayment scheme is designed to be small and efficient without using any public-key cryptography. Since public-key cryptography can present a significant computational burden, this scheme does without it and relies on other mechanisms to accomplish authentication. Like many micropayment schemes, MicroMint is token-based: a "digital coin" is purchased by consumers from a central authority, spent at a vendor, and finally redeemed by the same central authority for currency.

In (basic) MicroMint, a coin is comprised of one or more sets of k-way hash collisions. These coins have the dual property of being very difficult to produce, yet trivial to verify. A MicroMint broker (the central authority, or bank) issues coins once a month, and unused coins are returned to the broker at the end of each month. Vendors redeem coins at regular intervals (such as once a day).

One can visualize a hash function as a process that tosses randomly numbered balls into one of a series of numbered bins. A hash collision occurs when more than one ball falls into the same bin. A MicroMint coin is a bin with k balls in it.

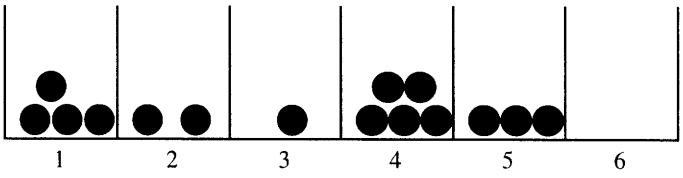

Figure 2-1: Tossing balls into bins. If this were part of a MicroMint broker with $k = 4$, bins 1 and 4 would have produced valid coins.

In order to prevent malicious users from spending coins multiple times, the coins should be made specific to both the user who purchased it and the vendor at which it should be spent. Because MicroMint does not use any digital signatures, coins must be able to be tied to certain user and vendor identities by other means. By modifying the means in which hash collisions are generated, we can create user and vendor-specific coins. A user specific coin can be generated by picking "balls" from different bins that correspond to the user's identity, instead of picking balls from the same bin to form a coin. The broker starts with one bin, and then chooses the next bin based on the user's identity.

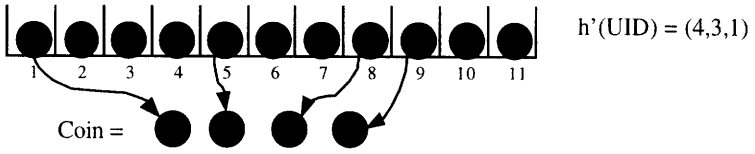

Figure 2-2: Creating a user specific coin

Vendor specific coins can be generated by dividing each bin's value into two parts: the superbin (user-specific part) and the sub-bin (vendor-specific part). The user purchases coins as k sets of superbins. When the user wants to spend a coin at a particular vendor, he picks balls from sub-bins which correspond to the vendor's identity. The superbin values were chosen by the broker to correspond to the user's identity, and the sub-bin

value corresponds to the vendor's identity. By using both techniques, most cases of double spending can be thwarted.

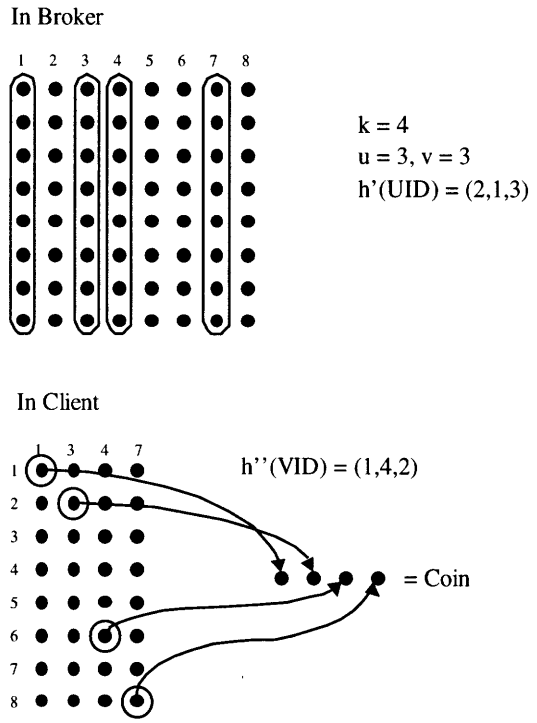

Figure 2-3: Producing user and vendor specific coins

Figure 2-3 illustrates how a broker and client work together to produce a user and vendor specific coin. The broker first tosses balls into bins, arranging them into 2^u columns – each column represents a superbin. When a client wishes to purchase a set of k superbins, the broker performs a hash of the client's user ID, h'(UID) = (uh_1, uh_2, …, uh_{k-1}). The broker then picks superbins such that $sb_{i+1} = sb_i + uh_i$. These superbins are then sold to the client. When making a purchase at a vendor, the client hashes the vendor's vendor ID in a similar manner to the broker's UID hash, h''(VID) = (vh_1, vh_2, …, vh_{k-1}). The client then chooses one ball from each of the k superbins purchased from the broker, such that $ball_{i+1} = sb_{i+1}[\text{Index}(ball_i) + vh_i]$. These k balls become one coin.

2.2 Other Micropayment Systems

There are a number of other micropayment systems in various stages of development, from proposals in academic literature to systems currently in commercial trials. While most micropayment systems share similar goals, the manner in which they achieve those goals can vary considerably. We will briefly examine two additional micropayment systems and highlight the major differences between these systems and MicroMint.

2.2.1 Millicent

Overview

Digital Equipment's Millicent [6] is one of the most widely publicized micropayment systems, and is currently undergoing public trials. The Millicent system is based on data packages called scrip, which can represent various amounts of money. Unlike MicroMint, Millicent scrip begins with the vendor, which either generates its own scrip or licenses the right to produce its scrip to a third-party broker. A particular vendor's scrip is only valid for payment at the same vendor. Customers purchase vendor scrip through brokers (who have either produced the scrip for the vendors or purchased scrip in bulk from the vendors), and then spend that scrip at vendors.

As an example, a typical Millicent transaction would proceed as follows: First, a client opens an account with a broker, and purchases some broker scrip with a macro-payment transaction. When the client wishes to make a purchase at a vendor, the client exchanges some broker scrip for vendor scrip, and then sends the vendor scrip to the vendor. Unused vendor scrip can be returned to the broker in exchange for broker scrip.

Each piece of scrip is comprised of two parts, a scrip body and an authenticator – a hash of the scrip body and a secret key. Since each broker and vendor has a different secret key, a particular piece of scrip is specific to a particular broker or vendor. When a client makes a purchase, the purchase is authenticated with a key shared by the vendor and client.

Comparison to MicroMint

The Millicent system shares some similarities with MicroMint:

- It does not use public key encryption. Scrip are authenticated by using a hash with a secret key, and messages are authenticated by hashing with a shared key.
- Coins are specific to both users and vendors. Since scrip are generated by vendors (or by brokers on the vendor's behalf) and are authenticated by a secret key known only to the vendor, they cannot be used at multiple vendors. Each piece of scrip has a unique ID which can be used to catch multiple users spending the same scrip.

Millicent differs from MicroMint in a few key places:

- Scrip are owned by vendors, not brokers. This decentralizes the scrip generation process, and results in a shorter start-up time than MicroMint.
- The broker needs to be involved in every purchase with a new vendor. Since a client can only spend a particular vendor's scrip at that vendor, she must first exchange her "generic" broker scrip for vendor scrip before making a purchase. This involves a few messages being sent back and forth between the client and broker. When the client no longer needs the vendor's scrip, additional messages must be sent back and forth to exchange it back for broker scrip. In an environment where a client only deals with a

few regular vendors, this is not much of a concern. However, if clients often make only a few purchases at many different vendors, this requirement can result in a large amount of network overhead. By comparison, MicroMint only requires one transaction between client and broker for the lifetime of the MicroMint coins, no matter how many vendors the client deals with.

- The client must establish a shared secret key with each vendor she deals with. MicroMint does not use shared secrets, and the client only needs the vendor's public vendor ID to produce vendor-specific coins.
- Clients cannot verify their coins. Since Millicent scrip is authenticated with a secret key known only to the scrip's issuer, clients have no way of verifying the scrip they hold. In MicroMint, the broker's hash function is public, so the clients can easily verify the coins they purchase from brokers.

2.2.2 PayWord

Overview

The PayWord micropayment system was proposed by Ron Rivest and Adi Shamir in the same paper that they proposed MicroMint [10]. In PayWord, users generate their own "coins," or paywords, which are sent to vendors and then verified by brokers. Each user has a "certificate," a block containing the user's identity and public key, digitally signed by a broker. When a user wishes to make a purchase at a vendor for the first time in a day, she first randomly picks a payword "seed," w_n, where n is a reasonable estimation of the number of purchases a user would make at a typical vendor in a day. The user then computes a payword "chain," by repeatedly hashing w_n: $w_{i-1} = h(w_i)$, where $i = 1, \ldots, n$.

Before making a purchase, (or along with the first purchase), the user sends the vendor a digitally signed commitment to w_0, the "root" of the payword chain. The user then sends the vendor w_1 through w_m, where m is the number of coins the user wishes to spend. The vendor can easily verify this chain by hashing w_m m times until he reaches w_0. At the end of each day, the vendor sends the broker the user's commitment to w_0, along with w_m, which the broker uses to credit the vendor's account for m coins.

Comparison to MicroMint

PayWord has several key similarities to MicroMint:

- It is off-line. The user only needs to contact the broker at the beginning of each certificate lifetime (i.e., once a month) in order to obtain a new signed certificate.
- Coins are user and vendor specific. Each payword chain is authenticated with a signed commitment, which contains w_0 along with the user's and vendor's certificates.

Differences between PayWord and MicroMint:

- PayWord uses public key cryptography. While somewhat expensive, vendors need only verify signatures once a day for each user who makes purchases. This helps amortize the cost over multiple purchases; however, if user spending patterns are such that only one or two purchases are made at a vendor in a day, the per-coin computational costs can be higher.[2]

[2] The authors of Mini-Pay [8] argue vehemently against the perceived computational burden associated with using public key cryptography for micropayments. While a complete analysis of this topic is beyond the scope of this thesis, they argue that network latency, and not public key operations, is the bottleneck to be avoided.

- Users generate their own coins. While in MicroMint, the broker generates the initial hash values to be used for coins, PayWord brokers only create user certificates.
- PayWord is credit based. MicroMint brokers send users a specific number of coins each month and debit their accounts upon initial purchase. In PayWord, however, the user's account is only debited after the vendor redeems payword chains she produces.
- Broker storage requirements. PayWord brokers only need to store user certificates, payword chain w_0 and w_m's, and other accounting information. MicroMint brokers, on the other hand, have enormous storage requirements, since they must keep every hash value (bin) produced each month.
- PayWord vendors need only keep the last coin received from users, since it signifies the number of paywords spent at that vendor (by counting the number of hashes necessary to reach w_0). MicroMint vendors must keep each coin spent.

Chapter 3: Design and Implementation

The MicroMint system consists of three major components, the broker, the vendor and the client. These modules interact with each other to enable a user to purchase web pages from a vendor using coins obtained from a broker.

3.1 The MicroMint Protocol

Each MicroMint module communicates with other modules through a standard communications protocol. The basic protocol is as follows:

```
MicroMint-1.0
<Message Type>
[<Header>: <Value>]…
[Data-Size: <Value>
Base64-Encoded Data]
End-MicroMint
```

There are several different types of MicroMint protocol messages, listed below:

Message Type	Sender	Recipient	Meaning
Funds-Withdrawn	Broker	Client	Contains coins the client purchased from the broker
Send-Expired	Broker	Client	Instructs the client to return expired coins
Request-Credit	Client	Broker	Client is returning coins to the broker for credit
Request-Payment	Vendor	Client	Client must pay for the content requested
Payment	Client	Vendor	Payment to the vendor for the client's request
Invalid-Payment	Vendor	Client	Payment sent to vendor was rejected

Table 3-1: MicroMint Protocol Messages

The Funds-Withdrawn, Request-Credit, Payment and Invalid-Payment MicroMint protocol messages contain embedded binary data, such as coin parameters or actual coins. In order to comply with certain restrictions imposed by the lower-level protocols carrying MicroMint messages, all MicroMint protocol messages must be 7-bit ASCII text. Any binary data stored in a MicroMint protocol message is sent as a Base64 encoded block [5].

3.2 The Broker

The MicroMint broker is the heart of the system. It is the central clearinghouse for all transactions, arbiter of disputes, and is the bridge between MicroMint coins and real currency. The broker consists of three main components: `CoinMint`, the MicroMint coin minter, `MMBroker`, the interface between the broker and clients, and `MMRedeem`, the back-end interface between the broker and vendor. Each of these components have very separate design requirements, and as such are implemented using different technologies.

3.2.1 `CoinMint` - Coin Minting

The `CoinMint` module is the workhorse of the MicroMint system. Before any transactions can take place, the `CoinMint` module must run, producing the hash value pairs which will be used later to generate coins. `CoinMint` is configured with a set of minting parameters, and it iterates a particular hash function on a large range of values – tossing balls into bins. These bins are grouped into superbins, which are later used by `MMBroker` to create user-specific coin sets.

Parameter	Description
coinfile	`CoinMint` output file
hashfn	Hash function name
iv	Hash function initialization vector, 64-bits
input-size	Number of total bits in input
output-size	Number of total bits in output (n)
num-fixed-bits	Number of bits which are fixed in output (t)
fixed-bits	Fixed bits in output
subbin-size	Number of bits in vendor-specific part of output (v)
expiration	Expiration date of these values

Table 3-1: `CoinMint` **Parameters**

The fundamental way the MicroMint system prevents forgery is by giving the broker computational resources which are far greater than that of a reasonable adversary. This would usually involve large, expensive hardware, often with special chips designed to perform a particular hash function at top speed. Alternative methods for coin minting could harness the power of massively parallel supercomputers, or distributed networks of computers, similar to Internet crypto-key crackers.[3] Exploring potential methods for efficiently hashing billions of numbers could be its own thesis project, and the goal of this project is to implement all of the essential components for a working prototype system.

[3] For more information about Internet crypto-key crackers, see http://www.distributed.net

As a result, experimenting with custom hardware, massively parallel supercomputers and distributed Internet hashing clients will be left to future work.

The Hash Function

`CoinMint` is designed to be hash function independent, however for this implementation, we will use the DES encryption system [8] to generate coins. To use DES as a MicroMint hash function, we use a fixed plaintext (the initialization vector from `CoinMint`'s parameters), and iterate the keys as the inputs. The ciphertext represents our output.

The `CoinMint` *Output File*

The `CoinMint` output file is depicted in Figure 3-1. It starts with a file header, which contains information similar to the `CoinMint` configuration file, along with a pointer to the next superbin to be sold to clients. It is then followed by 2^u superbins, each of which has a header containing the number of balls in that superbin, whether the particular superbin was sold, and if so, to whom. Each superbin has 2^v bins, which has each x and y value, sorted by y in increasing order. These bins have four "status" bits: one for whether a ball exists in the bin, one for whether the ball has been redeemed by a vendor, one for whether the ball has been returned unspent by the client, and one bit for whether the ball has been deemed unusable, or compromised in any way.

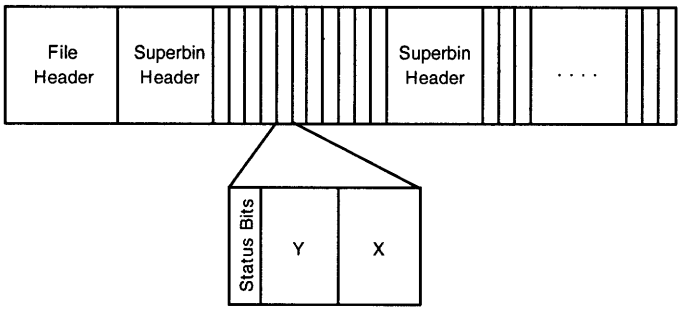

Figure 3-1: The `CoinMint` Output File Format

`CoinMint` *Operation*

Upon startup, the `CoinMint` program first reads its configuration file, which specifies the parameters in Table 3-1. Next, it creates its output file, as described above. Finally, it begins tossing balls into bins, by sequentially iterating the chosen hash function across the input space. For each hash value pair, the output is compared to the fixed bits specified in the `CoinMint` configuration file. If the upper t bits match, then the pair is recorded in the output file, placing the pair in the proper bin (by looking at output bits t through n-v for determining the superbin, and bits n-v through n for determining the bin). Existing balls are not overwritten, by first checking the bin's "exists" bit. This process is repeated until the entire input space is exhausted.

3.2.2 `MMBroker` - Coin Distribution

The `MMBroker` module provides coin distribution services for the MicroMint broker. Using `MMBroker`, a client can log on to the broker's web site and purchase coins. `MMBroker` is implemented as a CGI program, which is fed input from an HTML form filled out by the client. This form contains the client's user ID and the number of superbin sets desired. As described in section 2.1 above, clients can only purchase coins as

"superbin sets," which contain *k* superbins. These superbin sets are read from the broker's coin database (the output file from `CoinMint` – see 3.2.1 above) and packaged into a MicroMint protocol message of type Funds-Withdrawn. This message is sent to the client, which stores the coins locally. (For more detail about client operations, see 3.4.1 below.) On the broker side, `MMBroker` marks the first superbin of each set sent to the client as "sold," records the client's user ID in each superbin header, and updates the "next superbin" field in the file header. The file is locked during this operation to handle concurrency issues.

In a true commercial implementation, this module would also handle creating and maintaining client accounts. Clients would log on to the broker's web site, enter contact and billing information, and then purchase coins. With each purchase (or after aggregating a certain number of purchases), the broker would collect actual currency, by charging the client's credit card or through some other billing system.

3.2.3 `MMRedeem` - Coin Redemption

After clients spend coins at vendors, the vendors need to exchange the MicroMint coins they receive for actual currency. Although this implementation does not deal with real money, the `MMRedeem` module is meant to simulate this process. At the end of each day (or some other appropriate time interval), each vendor sends the coins they have collected from clients back to the broker. The `MMRedeem` module reads these files, checking that each coin is valid for both the vendor which received it and the user who purchased it. If the coin is valid, the first value (out of the *k* values in each coin) is marked "spent" in the

broker's coin database. In a commercial implementation, an appropriate credit would be made to the vendor's account at this point. Invalid coins are set aside for later processing.

3.3 The Vendor

The MicroMint vendor is responsible for selling goods to clients and sending coins collected from clients back to the broker for redemption. These two functions are separated into two modules, `MMVendor` and `MMVendorRedeem`, which are described in detail below.

3.3.1 `MMVendor` - Purchasing Pages

The `MMVendor` module is the main interface between the vendor and clients. It is responsible for receiving client requests and returning with the appropriate response. Since a micropayment system is only as good as the content available, the MicroMint vendor module is designed to be as easy to set up as possible. This leads to several design criteria:

- Vendors should not be required to change their content. Many web sites contain hundreds of links, and requiring a vendor to modify every single page on their site would be an unnecessary barrier. URLs should work just as they did before the imposition of a payment system.
- Vendors should not be required to change their web server software. Just as the point above, many sites have made extremely complicated customizations to their existing server software – abandoning these changes would often have unreasonable financial repercussions.

- Clients should not have to change their browser, or use an external program to view payable content.

There are several different options for implementing the `MMVendor` module. To make the argument easier, the author decided to consider those options which were implementable using the Netscape Enterprise Web Server.[4] (The Apache server[5] was also considered, however its support for Windows NT is not as complete as Netscape's, and NT was the preferred platform.) The Netscape server supports three main programming interfaces, each of which have advantages and disadvantages for implementing a micropayment system:

- CGI – By far the simplest of programming interfaces, CGI has one major drawback: it requires a specific URL prefix. One can only reference a page through a CGI program by first referencing the CGI program itself. For example, suppose the client wishes to purchase foo.html from our server. With a CGI vendor module, the client would need to request http://www.ourserver.com/cgi-bin/`mmvendor`?/foo.html. This clearly violates our first design criterion.

- WAI – Netscape's Web Application Interface is a CORBA compliant Java programming interface for Netscape Web Servers. While much more powerful than CGI, it shares a similar drawback: fixed URL prefixes.

- NSAPI – The Netscape Server API is the most powerful way to extend the web server, but also the most complicated. Despite its complexity (or perhaps, because of it) it

[4] For more information on the Netscape Enterprise Web Server, see http://www.netscape.com.
[5] For more information on the Apache web server, see http://www.apache.org.

does not require any URL prefixes or changes to existing links. NSAPI is described in further detail below.

NSAPI Overview

NSAPI is the C programming interface available to extend the functionality of the Netscape web server [13]. It provides the programmer with the ability to modify the server's handling of every step in the HTTP request-response process. For each step, the web site administrator can call either a built-in server function or a user defined function (often both). These functions have complete control over the server's processing, and can read data from the client, send data back to the client, or modify the server's later response. NSAPI functions allow the server administrator to create a custom-designed web server, which is exactly the functionality needed to make a MicroMint vendor run transparently.

The MicroMint vendor NSAPI module defines two externally accessible functions, `check-micromint` and `micromint-error402`. Both of these functions are essential to the vendor's processing. The vendor's functionality is split between two functions because the vendor needs to be involved in three separate steps of the web server's request-response process: initialization, access control and error handling.

`init-micromint`

The `init-micromint` function initializes some global data structures in the vendor module. This function reads the vendor's configuration file, which includes MicroMint's operating parameters, along with information about where the vendor should store its daily logs.

check-micromint

The `check-micromint` function serves as the access control portion of the vendor module. `Check-micromint` is called during the server's PathCheck phase, which is designed to determine whether the user making the request has permission to access the data requested. Although a micropayment system is different from an authentication and authorization system, it bears many similarities. Proper and valid payment in a micropayment system is analogous to proper credentials in an authorization system. Put simply, if a given client supplies the right payment with her request, she will be sent the data requested. If payment is missing or invalid, the request is denied.

When the `check-micromint` function is called, it first determines if the file requested from the web server requires payment. To indicate payable content, the vendor's webmaster creates a file with ".mm" at the end of its name corresponding to an existing file on the site. This ".mm" file has one line with the price of its corresponding content file. For example, if we want to charge two coins for /foo/bar.html, we create a file /foo/bar.html.mm, and put one line with "2" in it. The `check-micromint` function reads this file and uses it to determine if a given web server request is for payable content.

Next, `check-micromint` looks to see if the request contains payment. The client supplies payment for a request as an HTTP POST request.[6] This data is a MicroMint protocol message of type Payment. `Check-micromint` reads this data into a temporary buffer.

Now, `check-micromint` has four options:

[6] There are three types of HTTP requests, GET, HEAD, and POST. GET is a normal request, HEAD is a request for only header information, and POST is used to send data from the client to the server.

1. The content is not payable, and no payment was supplied: `Check-micromint` informs the web server to proceed as normal with the request.

2. The content is not payable, and payment was supplied: `Check-micromint` would return the payment to the client, and indicate that the supplied coins were not recorded and should be reused for another request. Currently, this option is not implemented.

3. The content is payable, and no payment was supplied: `Check-micromint` sets the server's response to HTTP error 402, "Payment Required." It records the content's price in an internal server data structure and exits.

4. The content is payable, and payment was supplied: `Check-micromint` first checks the coins to ensure that they are valid by hashing each value and comparing the lower v bits of the output to the vendor hash of its vendor ID. If the coins are valid, they are recorded, and the function indicates to the web server that the request should proceed normally. If the coins are not valid, `check-micromint` would return the coins to the client, indicating that they are invalid and the request was denied. Currently, the coin return functionality is unimplemented.

micromint-error402

The `micromint-error402` function is designed as a web server error handling routine. Normally, when a built-in or user-defined server function sets the server's response to an error condition, the error code is returned to the client and some generic message is sent. Error handling routines allow programmers to customize this functionality. When `check-micromint` sets the server's response to error 402, the

web server calls `micromint-error402`, which is registered to handle error 402 conditions.

A Word About HTTP Error 402

The current HTTP specification [3] defines several server error responses. These error responses typically inform users that the URL they requested is unavailable, they are not authorized to view the data, or there was an internal server error. HTTP error 402 is set aside in the spec to indicate that payment is required for the requested URL. While it is present in HTTP/1.1, it is designated "reserved for future use," and handling for it is currently unimplemented in all major web browsers. A future micropayment-enabled web browser would be able to properly respond to an error 402 and supply payment with requests. However, since error 402's are currently ignored by web browsers, this MicroMint system uses a work-around by capturing error 402's on the server and changing them to something which can be passed on to the browser (and browser add-ons).

`micromint-error402` Operation

`Micromint-error402` is called when a user requests payable content from the web server but does not supply payment with the request. It is `micromint-error402`'s responsibility to inform the user that the request was denied and to try again by supplying a certain amount of MicroMint coins. `Micromint-402` first reads the URL and price fields from the server's request data and then generates a MicroMint protocol response of type Request-Payment. This message contains the URL requested, its price, and the vendor's Vendor ID. After the message is sent to the client, the function exits.

3.3.2 MMVendorRedeem - Redeeming Coins

At the end of each day, vendors must send the coins collected in that day to the broker for redemption. While not currently implemented, `MMVendorRedeem` would perform this action by sending the vendor's coin log to the broker. The broker then performs processing offline, notifying the vendor upon completion that its account is updated, and also notifying it of any coins which were rejected as invalid or previously redeemed.

3.4 The Client

The MicroMint client module serves two main purposes: to move coins back and forth over the network, and to interact with the user about these coin transactions. The client module is designed to be seamlessly integrated with the user's existing browser.

3.4.1 NPMM32 – The Client Browser Plug-in

The MicroMint client acts as the user's agent in all MicroMint transactions. It receives MicroMint protocol messages from brokers and vendors, and stores and spends coins from a file on the user's hard drive – the MicroMint wallet.

Browser Plug-in Overview

The heart of the client module is `NPMM32.DLL`, a Netscape browser plug-in [9]. Browser plug-ins are shared objects (or DLLs) which have access to both the client's local disk and to certain requests and responses sent to the browser. These plug-ins let programmers extend the browser's functionality by allowing users to view different types of data within the browser, instead of invoking an external program.

Each browser plug-in has an associated MIME type [5] for the type of data it handles. MIME types are a standard way of specifying the format of a particular stream of data. Web browsers use these type designations to determine how to handle data received from a server. For example, data of type text/html gets processed as an HTML file, image/jpeg data gets processed as a JPEG image, and application/pdf data gets processed as an Acrobat PDF file. Browser plug-ins act as extensions to the browser's built-in display mechanism. They register themselves as being able to handle data with a certain MIME type, and then get called by the browser when it receives data with that type. For example, the Adobe Acrobat plug-in registers itself for the application/pdf MIME type; the browser invokes the plug-in whenever it receives a PDF file.

Netscape browser plug-ins can be invoked by one of two methods, either by browsing a page with the plug-in's MIME type, or by browsing an HTML page with an EMBED tag which references the plug-in's MIME type. These two methods for invoking the plug-in give us a choice in how the MicroMint client can act, as discussed below.

Micropayment systems present a particular quandary when trying to integrate with existing browsers. The Netscape browser plug-in API is designed for plug-ins to display data which the browser does not handle by default. But micropayment system messages are not meant to be displayed, they are meant to be handled internally by the browser and

for the browser to handle the display of data.[7] However, since the plug-in API is all that is available for extending the Netscape browser's functionality, it will have to do.

Plug-in Initialization

When the Netscape browser starts, it scans a particular directory for its available plug-ins. Each of these plug-ins provide the browser with their registration information, including the MIME type it handles. The MicroMint client plug-in registers itself for the MIME type application/x-micromint.

When the plug-in is first invoked, the browser calls an initialization routine. This function reads the client's configuration from the Windows registry (a system-wide database for storing small pieces of information, such as system configuration, user preferences, etc.). There are three user-settable options for the MicroMint client: the user's User ID, the path to the MicroMint wallet, and the "autopay threshold." The autopay threshold is the maximum number of coins which can be spent in a single transaction without user intervention. For example, if the autopay threshold is 5, a page which costs 1-5 coins will be paid for automatically, while a page which costs 6 or more coins will require the user to confirm payment before the coins are transferred.

Receiving Coins

When the user visits a broker, she can purchase coins to be spent at vendors. After entering data into the broker's input form, the `MMBroker` CGI program returns a

[7] While this might seem like a simple technicality, this intention presents problems with the browser's page history when dealing with MicroMint. Since the browser expects the MicroMint plug-in to display the data, it adds a new page in its history for the MicroMint plug-in. After the plug-in's request for the payable content is fulfilled, the browser creates another page in its history. Consequently, there is a "blank" page in the browser's history after the payable content is displayed. This causes problems when the user clicks on the browser's "back" button.

MicroMint protocol message with the newly purchased superbins. Because MicroMint protocol messages have the MIME type application/x-micromint, the MicroMint browser plug-in is called when the browser receives `MMBroker`'s output. The plug-in reads the Funds-Withdrawn message and stores the superbins in the user's wallet, updating the wallet's balance.

Spending Coins

The entire purpose of a micropayment system is to allow users to spend money at a vendor's web site and receive content in return. This action should be smooth and efficient, and should function just as if the user were clicking on a normal link (with the possible exception of a slightly different look to signify that the link costs money). After the user has purchased superbins from a broker, the client module is able to use those superbins to create coins for a vendor. However, in order to spend coins at a vendor site, the client plug-in must have three essential pieces of data: the URL to request, the price, and the vendor's Vendor ID. Because Netscape browser plug-ins can be invoked by two separate methods, there are two different techniques for implementing this "click and pay" action.

Click and Pay Technique 1 – Hidden Plug-in

The first method for implementing "click and pay" is through a hidden plug-in. This technique is somewhat similar to how Millicent [6] and CyberCoin [1] are implemented.

1. First, the user clicks on a normal link on a web page, which references a page of payable content.

2. Next, the server calls the MicroMint vendor plug-in function, `check-micromint`. As described above, if the requested file is not payable, the server processes it as a regular request.

3. If the file is payable, `check-micromint` notes that the client did not include payment with the request, and thus sets the server's response to HTTP error 402. The web server then calls the vendor function `micromint-error402`, which creates a MicroMint protocol message requesting payment. This request is given the MIME type "application/x-micromint," and is sent back to the client as a regular server response.

4. The client browser receives this response and routes it to the MicroMint plug-in, since it has the appropriate MIME type. The plug-in checks that the user has a sufficient amount of coins available to pay, and that the price is not over the user-specified autopay threshold (if it is over the threshold, a box pops up asking the user if the expenditure is okay). The plug-in then generates vendor specific coins for the request, and resends the request as an HTTP POST, with a MicroMint protocol message of type Payment.

5. When the server plug-in receives the request, it again forwards it to `check-micromint`, which checks that the coins are valid for both the user and the vendor. Then it records the payment in a local file, and forwards the actual request to the web server, which sends it along to the client.

6. If the user's payment is invalid (bad coins, too few, etc.), the server responds with an appropriate error message to the client.

Click and Pay Technique 2 – EMBEDded Plug-in

An alternative click and pay technique is also possible by using an EMBED tag to call the client plug-in.[8] This technique has the advantage of requiring fewer communication steps between the client and server, but sacrifices some convenience in eliminating the ability to reference payable files through normal links.

1. First, the client requests an HTML page with links to payable content. These links are in the form of EMBED tags which contain all of the required parameters for the client to generate payment information (URL, price, vendor ID).
2. When the client loads the page with EMBED tags, the browser invokes the MicroMint client plug-in, which displays an icon representing the link. When the user clicks on this icon, the client plug-in sends a MicroMint protocol message in an HTTP POST to the server, similar to step 4 above.
3. The process follows similarly to the technique above.

Click and Pay Implemented

Currently, click and pay technique #1 is implemented in the MicroMint client plug-in. This technique allows sites to support payment without rewriting any HTML pages. While click and pay technique #2 requires only one request-response communication, it requires very careful modification of HTML pages. An EMBED tag can invoke a client plug-in in two ways, through its SRC attribute or its TYPE attribute. If the tag has a SRC attribute, the browser requests the referenced file from the web server, and sends the response to the client plug-in. If the tag has a TYPE attribute, the browser simply invokes

[8] This is the technique was inspired by IBM's MiniPay [8].

the client plug-in and sends it the remaining attributes in the EMBED tag. Obviously, if the EMBED tag has a SRC attribute which references a remote file containing MicroMint price, URL and Vendor ID information, the entire advantage of using EMBED tags is lost. As a result, every page which references payable content through an EMBED tag with a TYPE attribute must have accurate price information. If a vendor wishes to change the price of a file, all of the pages which reference that file must also be changed. All hope is not lost, however, since server-parsed HTML[9] can automate this task; however, this adds an additional processing burden on the web server. Using an additional NSAPI function can help eliminate the overhead associated with creating a new process every time a file is served, but it cannot overcome the additional work required to scan the file in the first place.

Unfortunately, the vendor has no control over external links to its pages. In this case, any implementation of click and pay technique #2 would have to include the functionality of technique #1 as well. Thus, when a user clicks on a link to a page of payable content from an external site (which doesn't have an EMBED tag), the process described in technique #1 would ensue.

Returning Coins

At the end of each month, all of a user's MicroMint coins expire, and are no longer valid for purchases. As a result, the user must return these coins back to the broker for credit, and then download new ones. Currently, this feature is unimplemented, however it can easily be supported by extending the client plug-in and `MMBroker`. To return expired

[9] In server-parsed HTML, the web server parses HTML pages before they are sent to clients and replaces special tags with data from either built-in server functions, or output from an external program.

coins, the user would log on to the broker's site, and click a link which tells `MMBroker` that she wishes to return her unspent coins. `MMBroker` would respond to the client with a MicroMint protocol message of type Send-Expired, which informs the client that it is ready to receive the expired coins. The client plug-in, invoked by the `MMBroker`'s response, would package the remaining values from the first superbin of each set and return them to the broker in a Request-Credit MicroMint protocol message.

3.4.2 printwallet – Client Wallet Viewer

The printwallet utility is a simple console application which displays the contents of the user's wallet. It displays the current balance, MicroMint parameters, and the number of superbins present.

Chapter 4: Conclusions

4.1 Designing a Micropayment System from the User's Perspective

The process of implementing a proposed system allows us to further examine the details involved in implementation, and lets us highlight specific features which are desirable in a real, working system. In developing a prototype implementation of MicroMint, several design guidelines for each major component's user interface have surfaced which can help us build more useful micropayment systems in the future.

4.1.1 Broker Guidelines

The MicroMint broker is the component with the least interaction with end users, and as a result, leaves us with the fewest guidelines for future implementations. As with any user interface, the client's interactions with the broker should be through a clear, intuitive interface. While not implemented in this prototype, there are several desirable features for the client-broker interface:

- Account interaction. Clients should be able to view the status of their broker account online, along with some form of purchase history.

- Vendor feedback. Since the MicroMint system does not enable users to request refunds for poor (or nonexistent) service from vendors, the user's only recourse is to complain to the broker. If the broker logs enough complaints about a particular vendor, that vendor's account can be disabled.

4.1.2 Client Guidelines

The main goal in implementing a micropayment client is to make the user's interactions with the client software as transparent as possible. Habits are hard to break, and forcing the user to interact with her browser in a different way will cause the user to disregard pages of payable content. Jacob Nielsen's sidebar to [12], "User Interfaces for Internet Payments," makes several excellent suggestions for client user interfaces. In this article, he suggests that very inexpensive links should be invisible in the user interface – a link which costs less than a cent should appear the same as a normal link. Moving up the price scale slightly should cause links to have a subtle user interface reminder of their cost. Possibilities include a different color, a different mouse cursor,[10] or a pop-up box. Most of these subtle changes would require either restricting sites to linking through embedded plug-ins, or tighter integration between the client module and the browser.[11]

In addition, the `printwallet` utility should be expanded to display more information about the user's purchases and enhanced with a graphical interface.

---

[10] Mini-Pay [8] uses this technique.

[11] Since we cannot rely on the link itself to contain price information, the browser could query the link's target for its payable status and price when the user moves the mouse over the link. Of course, this would involve some communication delay, but it is probably preferable to having the browser query every single link on every single page. Similar techniques are used today with JavaScript to create buttons which change their appearance when the mouse cursor is over them.

4.1.3 Vendor Guidelines

Since most of the user's interaction with vendors is through the client modules, most of the guidelines dealing with vendors have been covered previously. Apart from these guidelines, the most important consideration for implementing vendor modules is to keep the user's interface with web servers consistent – this means ensuring that URLs behave the same for payable content as they do for free content. In addition, it would also make good business sense for vendors to allow clients to view previously purchased payable content repeatedly within a reasonable time from the initial purchase.

4.2 Security Considerations

Micropayment systems represent a departure from normal payment systems, not only in their functionality, but in their attitude towards security. Because the transactions made with micropayment systems are very small, tight security is often sacrificed for performance. Though large-scale fraud must be prevented, if one or two fraudulent payments get made, little is lost.

Specific security properties of the MicroMint system are discussed in [10]. Here, we will examine some implementation-specific security issues, and how they can be prevented.

Currently, MicroMint protocol messages are sent in the clear. This is most probably sufficient for its purpose, since any coins which are transmitted are specific to both users and vendors. Specific cases exist, however, where it may be advantageous to either encrypt the data stream (through SSL or other means) or take additional precautions.

4.2.1 Rogue Brokers

It is conceivable that a third party can create a fraudulent broker, distributing garbage coins to users after they have provided legitimate macro-payment information (such as a credit card number). As a result, precautions must be taken to ensure that clients only send their payment information on to real, legitimate brokers. This can be accomplished by requiring brokers to authenticate themselves with certificates issued by a trusted third party.

In addition, it is also possible that a mischievous broker can refuse to redeem either all or a portion of the coins presented to it by vendors. However, the high-profile nature of a broker would make this type of attack very unattractive, since vendors cannot be prevented from communicating with each other. In addition, brokers may be considered either banks or credit-granting agencies, and thus fall under the watchful eye of governmental regulators.

4.2.2 Rogue Vendors

As stated in [10], as a matter of policy, brokers can choose to redeem each coin only once, by choosing a vendor arbitrarily. Thus, vendors can gain little by attempting to redeem coins twice. Also, as previously mentioned, vendors who refuse to deliver goods after accepting payment can be reported to the broker and dropped from the system.

4.2.3 Rogue Clients

Initial coin purchases from brokers can lead to fraudulent use. If a third party can view the transfer from the broker to client, that party can deduce the coins' user ID (since the hash

function is public) and create vendor specific coins at will. Only after those coins are attempted to be redeemed twice by vendors will the legitimate user and broker know that the coins were stolen. While the broker can issue new coins to the legitimate user, there is little preventing this fraud from occurring again. As a result, it is recommended that a real commercial implementation of MicroMint encrypt the data stream from broker to client.

4.3 Future Work

The purpose of this thesis was to develop a prototype implementation of the MicroMint micropayment system. By the very nature of a prototype, it easily lends itself to being open to future work. Building a real, commercial MicroMint implementation would, of course, require building a real broker, which involves a substantial investment in both time and hardware. There are, however two main areas of research which would be important milestones towards the goal of building a commercially viable system.

4.3.1 Coin Generation

The most significant portion of the MicroMint system which this thesis intentionally overlooked is in the area of coin generation. The current `CoinMint` module is a simple, single-threaded program which iterates a generic hash function. Real implementations of MicroMint would require brokers to be able to hash values several orders of magnitude faster than a single workstation. As mentioned previously, there are several different options for generating large numbers of hashes very quickly, and these would be excellent topics for further investigation.

4.3.2 Other Browsers and Web Servers

Currently, this MicroMint implementation works only with Netscape web browsers for Microsoft Windows 95 or NT and with Netscape web servers for Windows NT and Unix.[12] This represents only a fraction of the browsers and servers which are currently in use. A commercially viable implementation would need to support most hardware and software combinations in order to capture the widest market, so it would be important to develop implementations which run on different browser and server platforms. At a minimum, the client module should also run under Microsoft Internet Explorer, and the broker and vendor modules should run with Apache and Microsoft Internet Information Server web servers.

[12] The broker and vendor modules have been tested on Windows NT and Solaris, but should compile on any Unix which runs Netscape web servers.

References

[1] CyberCash, CyberCoin Web Site, http://www.cybercash.com/cybercash/services/cybercoin.html

[2] Electronic Merchant Systems, Credit Card Rate Sheet, http://www.e-transfer.com/emsrates.htm

[3] Elizondo-Guajardo, R., Liu, C., Mobisson, G., Pearah, D., Ponce de Leon, X., Siegel, C., "Electronic Micropayments," April 23, 1997. http://rpcp.mit.edu/~pearah/micropayments/

[4] Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol – HTTP/1.1," IETF Proposed Standard, RFC 2068. http://www.w3c.org/Protocols/rfc2068/rfc2068

[5] Freed, N. and Borenstein, N. "Multipurpose Internet Mail Extensions," IETF RFC 2045-2049. http://www.cis.ohio-state.edu/rfc/rfc2045.txt, rfc2046.txt, rfc2047.txt, rfc2048.txt, rfc2049.txt

[6] Glassman, S., et al. "The Millicent Protocol for Inexpensive Electronic Commerce." In World Wide Web Journal, Fourth International World Wide Web Conference Proceedings, pages 603-618. O'Reilly, December 1995. http://www.research.digital.com/SRC/personal/steveg/millicent/millicent.html

[7] Herzberg, Amir. "Safeguarding Digital Library Contents." D-Lib Magazine, January 1998. http://www.dlib.org/dlib/january98/ibm/01herzberg.html

[8] Herzberg, A. and Yochai, H. "Mini-Pay: Charging per Click on the Web." http://www.hrl.il.ibm.com/mpay/docs/papers/mpay-long.html

[9] "Plug-in Guide," Netscape Communications Corporation, http://developer.netscape.com/library/documentation/communicator/plugin/index.htm

[10] Rivest, R. L. and Shamir, A. "PayWord and MicroMint: Two simple micropayment schemes," 1996. http://theory.lcs.mit.edu/~rivest/RivestShamir-mpay.ps

[11] 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

[12] Nielsen, Jacob. "The Case for Micropayments," *Jacob Nielsen's Alertbox,* January 25, 1998. http://www.useit.com/alertbox/980125.html

[13] "NSAPI Programmer's Guide," Netscape Communications Corporation, http://developer.netscape.com/library/documentation/enterprise/nsapi/index.htm