Auditable Metering with Lightweight Security

Matthew K. Franklin and Dahlia Malkhi
AT&T Labs – Research, Florhram Park, New Jersey, USA
{franklin,dalia}@research.att.com

May 4, 1998

Abstract

In this work we suggest a new mechanism for metering the popularity of web-sites: The compact metering scheme. Our approach does not rely on client authentication or on a third party. Instead, we suggest the notion of a *timing function*, a computation that can be performed incrementally, whose output is compact, and whose result can be used to efficiently verify the effort spent with high degree of confidence. We use the difficulty of computing a timing function to leverage the security of a metering method by involving each client in computing the timing function (for some given input) upon visiting a web site, and recording the result of the computation along with the record of the visit. Thus, to forge client visits requires a known investment of computational resources, which grows proportionally to the amount of fraud, and is infeasible for visit counts commonly found in the World Wide Web (WWW). The incremental nature of the timing function is used to create a new measure of client accesses, namely their duration. This paper describes the foundations of the timing scheme and its deployment in the WWW.

1 Introduction

The growing popularity of the Internet and the World Wide Web (WWW) is driving commerce—and with it advertising—on the Internet, into a billion dollar market. As with any other media, the advertising market creates the need for securely and accurately metering distribution circulation in order to accurately price the advertisements and test their effectiveness. On the Internet, metering is natually performed by automatic software mechanisms installed at the web server to collect access information. This creates serious security concerns, as the server has control over the collecting process as well as over stored data. Since the owner of the server can charge higher rates for advertisements by showing a higher number of visits, the owner has a strong economic incentive to inflate the number of visits. The owner could accomplish this by manipulating any unsecured metered data collected and stored on the server. Moreover, any individual could fraudulently increase the number of visits to a web site using a "robot" program, which is configured to generate visits to a web site. The amount of visits is theoretically limitless. Another deficiency of conventional metering schemes is that they fail to solve the proxy problem: This problem results from the failure to accurately meter the number of visits to web pages which have been temporarity stored in a cache or on proxy servers.

In view of the above, it is clear that a need exists for accurately and securely metering visits to a web site. Metering visits to web sites is also necessary for commercial applications other than advertising. For example, metering is required for royalty payments when a web-site displays copyright material.

Initial version of this paper appears in *Financial Cryptography '97*.

In this work we suggest a new mechanism for metering the popularity of web-sites: The compact metering scheme. The scheme offers light-weight security, so that forging a visit record requires a known amount of computational resources, and grows proportionally with the amount of forgery. Light-weight security is a good strategy for metering web populality, as it may be unaffected by fraud on a small-scale, since only a crude and relative measure of site popularity may be needed for purposes such as the pricing of advertisements and payment of royalties. Prior to our work, an application such as web site metering could only be protected through the use of heavyweight security mechanisms (such as digital signatures), or by using on-line trusted authorities. Our lightweight security mechanism can be used on its own, or combined with other security mechanisms to increase resistance to fraud. We describe an implementation of our method that works within the current WWW framework and can be deployed in off-the-shelf web browsers and servers.

1.1 Technical approach

Metering security in the WWW framework could be achieved by several known techniques. Employing standard cryptographic methods to keep self-authenticating records of interactions on the WWW would be secure, and is enabled by some existing extensions to the WWW protocols, such as S-HTTP [10] and SSL [8]. These methods are clearly advantageous in offering a high level of auditability and non-repudiability. However, authenticating all clients has the drawback that all clients must register to obtain authentication keys. Not only is this a heavy administrative burden, but it leads to solutions that threaten the clients' privacy.

A third party census may be used to independently provide measurements on web activity, as offered by the Audit Bureau of Circulations (ABC). Using this scheme, activity can be monitored by an objective authority, which can then certify the measured data and prevent the possibility of manipulating it by a web publisher. The obvious problems with this method are the dependence on a central authority, and the deviation of census data from real activity.

We offer an alternative approach to metering, that does not rely on client authentication or on a third party. We start with a notion of a *metering scheme*, with the following properties:

1. A metering scheme includes a *timing function* that can be computed with increasingly large efforts invested (incremental).

2. The output of a timing scheme need not grow with the amount of effort spent (compact).

3. The effort spent can be efficiently verified from the output with high degree of confidence, where by efficiently we mean with considerably less effort than re-computing the timing function itself (auditable).

We use the difficulty of computing certain functions to leverage the security of a metering method by involving each client in computing the timing function (for some given input) upon visiting a web site, and recording the result of the computation along with the record of the visit. Thus, to forge client visits requires a known investment of computational resources, which grows proportionally to the amount of fraud, and is infeasible for visit counts commonly found in the WWW.

The incremental nature of the timing function is used to create a new measure of client accesses, namely their *duration*. The term "visit" as used herein refers to the elapsed time a client examines content from a particular web site. A visit duration within the WWW framework is more accurately defined in Section 3. Our metering scheme engages the client in computing the timing function

incrementally throughout the duration of its visit, and thus, the output captures the duration of the visit.

1.2 Organization of paper

The rest of this paper is organized as follows. In Section 2, we review related work. We state our requirements in Section 3. In Section 4, we give our construction of a timing scheme. Our auditable metering protocols are described in Section 5. Implementation issues are discussed in Section 6, and we discuss possible extensions in Section 7. Applications are given in Section 8, and some caveats are raised in Section 9. We conclude in Section 10.

2 Related work

Since the initial publication of the conference paper of this work [6], which initiates the formal study of secure web metering, Naor and Pinkas [9] have studied methods for performing secure web metering using robust secret sharing schemes. Their approach provides computationaly secure metering, but relies on the involvement of a trusted third party (the auditing agency) periodically to compute and distribute the secret shares to both the web server and all potential clients. Their scheme can be used to prove visits by k clients in any time frame, where k is a pre-determined system parameter, and cannot be easily extended to support finer grain visit counting or deal with multiple visits by a single client within a single time frame. Compared with their approach, our scheme can be implemented without changing today's servers or browsers, and provides an additional timing measure for client visits.

Our techniques are more remotely related to various other security mechanisms using similar foundations but in different problem contexts. Dwork and Naor defined a related notion of a *pricing function* [5], whose computation requires a known lower-bound investment of resources. They use pricing functions for preventing the mass utilization of electronic mail. Pricing functions are not incremental, and cannot be utilized (compactly) to measure the duration of visits. Cai et al. define the related notion of *uncheatable benchmark* [3, 1] for efficiently verifying claims of computational power. Some of the methods proposed for uncheatable benchmarking can be used incrementally. The methods used both in pricing functions and uncheatable benchmarks contain a trapdoor that allows efficient computation of the results. If used in the context of auditable metering, a trapdoor could be used to allow efficient and definite verification of the results by an auditor, but would also provide the auditor with the means to forge results en masse. The scheme we offer below has means for efficient probabilistic auditing that does not suffer from this drawback.

Finally, the concept of light weight security has been used in a number of "micro-payment" schemes (beginning with [7]).

3 Requirements

Our goal in this paper is to provide a scheme for automatically metering client accesses to a web site in an auditable way. Our design stems from the following desirable requirements:

1. In order for our scheme to be widely accepted, no change should be required of clients, and the scheme should be deployable in today's commonly used browsers.

2. Many clients may be deterred by registeration procedures, and therefore we require that our solution will require no registration of clients.

3. Likewise, for our scheme to be widely deployed by servers, changes at the web server should be transparent and should not modify any exisiting structure of the web-site contents.

4. Client visits should be recorded whether the contents are obtained from the original web-site or from any intermediary cache or proxy server. In this way, repeated visits by clients behind firewalls are also captured.

5. Both the computation and the storage requirements at the client and server sides should be reasonable.

6. Recorded data should not compromise clients' privacy.

7. The system should provide light-weight security, so that forging visit records requires a known investment of computational resources, which grows proportionally to the scale of forgery. In this way, mass fraud is prevented.

4 A compact metering scheme

A compact metering scheme consists of two components: The first part is a compact timing function, which is an incremental computation performed by a client and whose compact result is sent back to the server and logged there. The second part is an efficient auditing function that verifies the computation time spent producing the logged result. Computation time is expressed in terms of some agreed upon unit of computation, whose complexity is well-analyzed. In this section, we present a timing function and two probabilistic methods for auditing it.

4.1 A timing function

A timing function is a computation that requires a known amount of time to compute certain outputs. This function can be computed with a reasonable investment of time by any client, but this investment of time is the only known way of producing the result. The basis of such a computation is some grain of computation h whose computational complexity is understood. In practice we can take h to be a well-known hash function such as MD-5 or SHA (producing 128 bit hash values). The timing function combines multiple applications of h to produce an output, such that the number of times that h computes represents the known required effort. Let $f_k(x)$ denote a construction requiring k applications of h. The timing function f computes $f_k(x)$ for increasing values of k. We do not need to a-priori set k: The timing function f is *incremental*, i.e., there is a way to move from $f_k(x)$ to $f_{k+1}(x)$ by applying h once. An important property of the function is that the output of the computation is of constant size, regardless of the number of h applications, and thus not all h results can be sent. We call this property *compact*.

Our timing function is based on a simple min construction f_k as follows. Let h be a hash function whose output is uniformly distributed over the domain $[0 \,..\, 2^{128} - 1]$ (e.g., take h to be MD-5). Define a sequence $x_0, \ldots, x_k$, where $x_i = h(x_{i-1})$, $x_0 = x.r$ (concatenation) for an input value x and a random seed r. Then $f_k(x)$ has the value x_j, $0 \leq j < k$, such that $x_{j+1} \equiv h(x_j) = \min_i\{x_i\}$. The compact timing function f we propose computes $f_k(x)$ for some $k > 0$ and returns a tuple $<k, x_0, f_k(x)>$. Note that this function is indeed incremental, as extending the evaluation of f_k to f_{k+1} requires one additional evaluation of h. The output of f is compact since one triplet of values is returned no matter how large k is[1].

[1] We ignore the size of k itself, which can be represented by 128 bits for all practical values of k.

4.2 A statistical auditing function

The second part of a timing scheme is an efficient auditing function, that verifies the number k of claimed h applications. We first propose a statistical approach, in which the auditor performs a statistical test of the validity of the result of a timing function and when needed, verifies the validity of the result by re-computing it. In section 4.3, we present another approach, in which an auditor estimates the effort in producing the result of a timing function.

The first approach we present is an auditing procedure that statistically tests the output of a timing function in order to detect fraud. By construction of the simple min function f_k, the series $x_1, \ldots, x_k$ behaves as a random sample of k points in the range $[0 \,..\, 2^{128} - 1]$. Let $N = 2^{128} - 1$, and let $X_k^{(1)}$ denote the first order statistic of $x_1, \ldots, x_k$, i.e., $X_k^{(1)} = \min_i\{x_i\}$. Then $f_k(x)$ behaves as the statistic $X_k^{(1)}$ with the following distribution:

$$P(h(f_k) > y) = P(X_k^{(1)} > y) = \left(1 - \frac{y}{N}\right)^k \ .$$

When an auditor is presented with a tuple $<k, x, y>$, as the output from our timing function f, a good way to detect fraud is to evaluate the probability of obtaining y as a result of $f_k(x)$, and mark as suspected low-probability values. Such suspected tuples can be subjected to a foolproof, costly verification, simply by repeating the computation of $x_1, \ldots, x_k$. Due to its cost, the foolproof method can be applied selectively, e.g., at random.

More precisely, let $\hat{k}$ denote the (unkown) size of the random sample used in the timing function. We establish a test hypothesis $H_E = \hat{k} < k/c$, which indicates that the auditor should suspect the tuple as fraudulent. To test this hypothesis, we use a null hypothesis $H_0 = \hat{k} \geq k$ denoting a condition for accepting a tuple. The null hypothesis does not complement the test hypothesis; the factor c indicates the "tolerance" of the auditor to cheating, where c is a system parameter which can be determined by the system administrator. The auditor sets a rejection threshold α for the null hypothesis H_0, and accordingly determines a threshold value x_{min} for which $P(X_k^{(1)} > x_{min}) = (1 - \frac{x_{min}}{N})^k = \alpha$ holds. H_0 is rejected if $x > x_{min}$, which guarantees that the auditor will falsely suspect a tuple produced by a correct application of f_k with probability at most α. Typically, α is chosen to be small, e.g., $\alpha = 0.05$.

To measure the quality of this test, we need to evaluate the probability of detecting fraud, i.e., the probability that a cheater spending (significantly) less effort will produce a tuple within the rejection range. For any $\hat{k} = k/c$, this probability is at least $P(X_{k/c}^{(1)} > x_{min}) = (1 - \frac{x_{min}}{N})^{k/c} = \alpha^{1/c}$, which quickly grows to 1 with c. This means that the probability of detecting fraud grows propotionally to the amount of fraud.

4.3 An approximate auditing function

In this section, we present an alternative auditing function for the timing function f defined in Section 4.1. Given a tuple $<k, x, y>$ output by a timing function, the approach taken here is to estimate $\hat{k}$—the amount of effort spent in h computations—from the result y. We therefore employ an estimator function μ such that $\mu(y)$ provides a real-valued estimator of $\hat{k}$. Let $H(N)$ be the harmonic function $H(N) = \sum_{i=1}^{N} 1/i \leq \ln(N) + 1$. The auditing function we propose here for $y = f_k(x)$ is $\mu(y) = \frac{N}{yH(N)}$.

In order to analyze the utility of this auditing function, we need to introduce additional definitions. The main properties of interest are captured in the following two definitions:

Definition 1 *Let f_k be our min construction. We say that an estimator μ is* α-accurate *if for all x, $k \geq E[\mu(f_k(x))] \geq \alpha k$, where the expectation is taken over all random selections of f_k.*

Definition 2 *We say that an estimator μ is* β-uncheatable *if for any "cheater" function g that performs k computations of h, the expected value of $\mu(g)$ does not exceed βk .*

The main purpose of this section is to prove the accuracy and uncheatability of our auditing function, as stated in the following theorem:

Theorem 1 *Let f_k be our min construction, and let $\mu(y) = \frac{N}{yH(N)}$. Then for all $k < N^{\epsilon}$, μ is α-accurate and β-uncheatable, where $\alpha = 1 - \epsilon - \frac{1}{lnN}$ and $\beta = 1$.*

The proof of the theorem follows from the following two lemmas:

Lemma 1 $E[\mu(y) : y \leftarrow f_k(x)] = k + \frac{1-kH(k)}{H(N)} + O(\frac{k}{NH(N)})$.

Proof : We already know that $P(f_k(x) > y) = \left(\frac{N-y}{N}\right)^k$. Therefore, we have $P(f_k(x) = j) = P(f_k(x) > j-1) - P(f_k(x) > j) = \left(\frac{N-(j-1)}{N}\right)^k - \left(\frac{N-j}{N}\right)^k$. Then

$$
\begin{aligned}
E[\mu(y) : y \leftarrow f_k(k)] &= \frac{N}{H(N)}\frac{1}{N^k}\sum_{j=1}^{N}((N-j+1)^k - (N-j)^k)\frac{1}{j} \\
&= \frac{N}{H(N)}\frac{1}{N^k}(N^k - \sum_{j=1}^{N-1}(N-j)^k(\frac{1}{j} - \frac{1}{j+1})) \\
&= \frac{N}{H(N)}(1 - \sum_{j=1}^{N-1}(1-\frac{j}{N})^k\frac{1}{j(j+1)}) \\
&= \frac{N}{H(N)}(1 - \sum_{j=1}^{N-1}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(\frac{j}{N})^i\frac{1}{j(j+1)}) \\
&= \frac{N}{H(N)}(1 - \sum_{i=0}^{k}\sum_{j=1}^{N-1}(-1)^i\binom{k}{i}(\frac{j}{N})^i\frac{1}{j(j+1)}) \\
&= \frac{N}{H(N)}(1 - (\sum_{j=1}^{N-1}\frac{1}{j(j+1)}) + \frac{k}{N}(\sum_{j=1}^{N-1}\frac{1}{j+1}) - (\sum_{i=2}^{k}\sum_{j=1}^{N-1}(-1)^i\binom{k}{i}(\frac{j}{N})^i\frac{1}{j(j+1)})) \\
&= \frac{N}{H(N)}(1 - (1-\frac{1}{N}) + \frac{k}{N}(H(N)-1) - (\sum_{i=2}^{k}\frac{(-1)^i\binom{k}{i}}{N^i}\sum_{j=1}^{N-1}\frac{j^{i-1}}{j+1})) \\
&= k + \frac{1-k}{H(N)} - \frac{N}{H(N)}\sum_{i=2}^{k}\frac{(-1)^i\binom{k}{i}}{N^i}\sum_{j=1}^{N-1}(j^{i-2} - j^{i-3} + \ldots + (-1)^{i \bmod 2} + \frac{(-1)^{i+1 \bmod 2}}{j+1}) \\
&= k + \frac{1-k}{H(N)} - \frac{N}{H(N)}(\sum_{i=2}^{k}\frac{(-1)^i\binom{k}{i}}{N^i}(\frac{N^{i-1}}{i-1} + O(N^{i-2}))) \\
&= k + \frac{1-k}{H(N)} - \frac{1}{H(N)}(\sum_{i=2}^{k}\frac{(-1)^i\binom{k}{i}}{i-1}) + \frac{1}{O(NH(N))}\sum_{i=2}^{k}(-1)^i\binom{k}{i} \\
&= k + \frac{1-k-S(k)}{H(N)} + O(\frac{k}{NH(N)}) \ ,
\end{aligned}
$$

where $S(k) = \sum_{i=2}^{k} \frac{(-1)^i \binom{k}{i}}{i-1}$.

We can find a closed form solution for $S(k)$ as follows[2]: $\frac{1}{i-1} = \int_0^1 x^{i-2} dx$, and so $\sum_{i=2}^{k} \frac{(-1)^i \binom{k}{i}}{i-1} = \sum_{i=2}^{k}((-1)^i \binom{k}{i} \int_0^1 x^{i-2} dx) = \int_0^1 (\sum_{i=2}^{k}(-1)^i \binom{k}{i} x^{i-2}) dx = \int_0^1 \frac{(1-x)^k - 1 + kx}{x^2} dx$. Integrating by parts ($u = (1-x)^k - 1 + kx$, $dv = x^{-2}dx$), we have $S(k) = [\frac{(1-x)^k - 1 + kx}{-x}]_0^1 - \int_0^1 \frac{-k(1-x)^{k-1} + k}{-x} dx = 1 - k + kJ(k)$ where $J(k) = \int_0^1 \frac{1-(1-x)^{k-1}}{x} dx$. Then $J(k) = \int_0^1 \frac{1-y^{k-1}}{1-y} dy = \int_0^1 (1 + y + y^2 + \ldots + y^{k-2}) dy = H(k-1)$, and so $S(k) = 1 - k + kH(k-1) = k(H(k) - 1)$, from which the lemma follows. □

Lemma 2 *Let $\mu(y) = \frac{N}{yH(N)}$. Then μ is 1-uncheatable.*

Proof : Let $(y_1 = h(x_1)), \ldots, (y_k = h(x_k))$ be any sequence of k evaluations of h (on any inputs). The best that the cheater g can do is somehow send all values y_i as forged results. Then

$$\begin{aligned} E[\mu(g)] &= E[\sum_{i=1}^{k} \mu(y_i)] \\ &= E[\sum_{i=1}^{k} \frac{N}{y_i H(N)}] \\ &= \frac{kN}{H(N)} E[\frac{1}{y} : y \in [1..2^{128} - 1]] \\ &= k. \end{aligned}$$

□

Proof :(Theorem 1) That the scheme is 1-uncheatable follows directly from Lemma 2. From Lemma 1, we have that $E[\mu(f_k(x))] = k + \frac{1-kH(k)}{H(N)} + O(\frac{k}{NH(N)}) \geq k(1 - \frac{H(k)}{H(N)}) \geq k(1 - \frac{\ln k + 1}{\ln N}) \geq (1 - \epsilon - (\ln N)^{-1})k$. □

For $k < N^{\epsilon}$, our auditing function therefore gives an estimate that asymptotically approximates the effort spent in a timing function, and cannot be used by a cheater to significantly inflate its metering value.

5 Auditable metering with a timing scheme

The metering protocol involves two parties, a *client* and a *meter*. We assume that clients and meters engage in a external *visiting* protocol by which clients access meters for certain durations. Our assumptions about the (opaque) visiting protocol borrow from the HTTP protocol [2], which motivated our work. However, other protocols may suit our abstract representation as well.

5.1 Assumptions about meters and clients

Let m denote a meter and c a client. m and c communicate via a reliable FIFO communication channel. We assume that m and c engage in a *visiting* protocol (whose details are opaque), that generates two events:

1. A *start-visit* at m signals the beginning of the visiting protocol at m, and precedes any information sent from m to c.

[2]Thanks to Yuval Peres for this derivation

2. An *end-visit* event at c signals the termination of the visiting protocol at c. Following the end-visit event, c does not send any further information to m within the visiting protocol.

Note that an end-visit event occurs at the client, hence this formulation can be realized in the (stateless) HTTP protocol. We assume that the environment supports dynamic deployment of programs from m to c, with the following properties: m is capable of sending an executable program p to c, which c may then choose to execute. c may send a termination signal to p as it executes, or can terminate it (ungracefully). Programs deployed from m to c may utilize computational resources and may communicate back and forth with m. We reiterate that these assumptions are practical, and as described in Section 6, are currently achievable in the WWW framework.

5.2 Metering protocol

The purpose of the metering protocol is to maintain at m a record of c's visit, expressing the duration of the visit, *i.e.*, the time between the *start-visit* and the *end-visit* events. For a meter m to record the duration of a client c's visit with a metering scheme, they engage in the protocol given in Figure 1.

---

1. **Initiation:** When m incurs a *start-visit* event by c, m deploys a program p that, when executed, computes f during its runtime and, when signaled to stop, sends back the result to m and terminates. p (equivalently, f) is initiated at m prior to deployment with the following input:

   $$<m, ts>,$$

   where ts is a unique identifier (timestamp) used by m, and with the property that $ts_2 \neq ts_1$ for any two distinct inputs $<m, ts_1>$, $<m, ts_2>$.

   The program p may contain a bounded computation of f or run endlessly; In the former case, if termination is reached then the computed result is sent back to m. In the latter case, the program terminates by a stop signal (see below), or when destroyed.

2. **Execution:** When a client c receives p it starts executing it.

3. **End:** When a client c incurs an *end-visit* event it signals p to stop. If p is still running it responds to this signal by terminating the computation of f and sending the computed result back to m.

Figure 1: The metering protocol

5.3 Auditing procedure

A visit record has the form $\text{rec} = [m, ts, <k, x, y>]$ where $<k, x, y> = f(m.ts)$. An auditor can either perform the statistical test of Section 4.2, which will mark certain records for foolproof verification and accept those records that contain probabilistically reasonable results (including their timing

measure k); or an auditor can use the approximate estimator of Section 4.3, which ignores k altogether and produces an approximate timing measure.

The following technique can reduce the number of visit records that need to be stored at a web site and reviewed by the auditor. Only keep those visit records which satisfy some predicate P. The predicate should have a predictable success rate, but be hard to predict for individual inputs, e.g., $P_{h,q}(\text{rec})$ is true if and only if $h(\text{rec}) \bmod q = 0$. The auditor verifies that all stored timing records satisfy the predicate, and expands the timing interpretation according to the success rate (e.g., multiply by q for $P \equiv P_{h,q}$).

6 Implementation

We have implemented *webmeter*, a prototype metering tool that can monitor a web site in today's WWW framework. Figure 2 depicts the structure of webmeter. The tool consists of two modules:

proxy: The proxy acts as the interface of the web server to the world. All requests for pages from the web server pass through the proxy. The proxy appends a metering applet to every page it returns from the web server. The proxy module may be placed on the same host as the web server or elsewhere on the communication path between clients and the server.

log keeper: A log keeper accepts the results sent by metering applets and keeps them on stable storage.

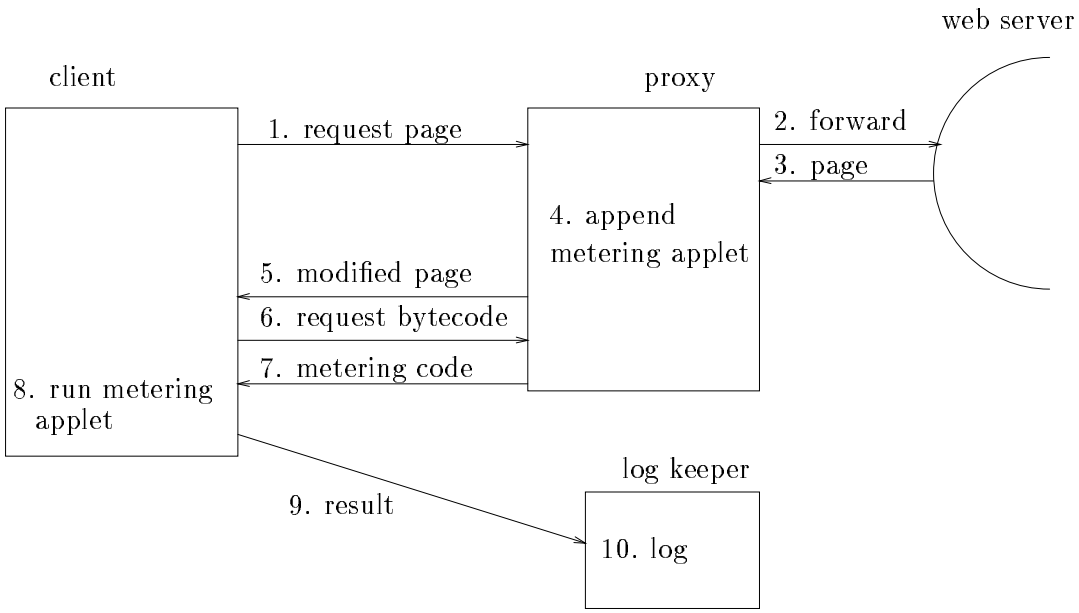

Figure 2: *webmeter* architecture

To understand how our system works, it is first necessary to understand how client browsers retrieve pages including Java applets from a web server. When a browser retrieves a web page written in Hypertext Markup Language (HTML), it takes actions based on the HTML *tags* in that page. One such tag is the `<applet>` tag, which names a file containing Java bytecode. This tag instructs the browser to retrieve the named applet from the web server and run it.

In our system, when a client's browser requests a page, the request is sent to the proxy (Figure 2, step 1). The proxy forwards the request to the end server (step 2) and receives the requested page (step 3). The proxy appends an `<applet>` tag naming the metering applet to the end of the page (step 4) and sends the modified page back to the browser (step 5). The browser, upon identifying the `applet` tag, requests the applet bytecode (step 6) which the proxy provides initialized with the appropriate input (step 7). Consequently, the browser loads the metering applet and executes it (step 8) and sends the timing funntion's result to the log keeper (step 9) which logs it onto stable storage (step 10).

An advantage of our implementation is that repeated visits of a client or a group of clients onto a cached copy of a page will re-run the metering applet each time: Repeated executions will simply use the same input provided by the proxy, but will be seeded with a different random seed every time. In this way, webmeter provides a solution for the proxy problem in metering web accesses.

7 Extensions

As discussed in the introduction, our metering scheme provides light weight security: The metering protocol uses the difficulty to compute visit records to leverage auditability. Our timing scheme requires an investment of computation power in each visit, that can be estimated to within some known accuracy. Thus, the amount of possible fraud is proportional to the amount of computational resources invested in it.

In practice, it is often fruitful to combine light weight mechanisms with selectively-deployed heavy weight mechanisms. For example, in the physical world, one's valuables might be protected by a cheap lock on the front door, and a burglar alarm, and recorded serial numbers, and the threat of arrest. A similar blend of security mechanisms can discourage attacks in the virtual realm. In this section, we suggest several mechanisms for strengthening the auditing of our metering scheme.

Checkpoints: An auditor can use the timestamped inputs (used for initializing the timing function f) to limit the *rate* of possible fraud by preventing the mass-generation of fraudulent visit records off-line. This can be done in the following way: An auditor designates epochs at which it requests a checkpoint of the meter log, and re-initializes the timestamp value for the next epoch. Any input values provided during an epoch must be utilized within this epoch, and later become unusable. Thus, for example, computations performed during off-peak hours cannot produce fraudulent visit records for peak hours.

Census: An independent third party may perform a census of web activity concerning any particular web site, given any means at its disposal. The records held by the meter should agree with the census approximations.

Third party: On-line monitoring by a third party can be required selectively for some predetermined portion of the visits. This could be done by transparently deferring designated requests to the third party site. Selection of visits should be done according to a deterministic predicate with a known success rate, applied on the timing function input. In this way, the expected number of visits in which the third party is involved would be known, approximately, as a portion of the total number of logged visits at the web server.

8 Applications

Many applications may benefit from auditability of web metering. We list some possibilities here.

The primary application motivating our work is the metering of a web site to measure its popularity. Our metering scheme has the advantages of being auditable, as well as accounting every visit to pages containing contents from the web site, even in the case of repeated visits to cache proxy servers.

The incremental timing scheme is novel in providing a measure of the duration of client accesses to a server. A profile of the times spent with a particular content can be of enormous value to content service providers, as well as to advertisers seeking maximum exposure. Information on the time spent viewing the contents of a document from a web site is valuable for pricing advertisements accompanying the document.

We also propose a novel application called **1-800-HTTP**: An Internet Service Provider (ISP) used for connecting private clients to the WWW charges clients for the duration of usage (typically, after some initial quota of flat-rate usage). This charge may prevent potential customers from browsing sites freely. An ISP may enhance its service to clients by partnering with certain commercial sites and *reverse the charges* of connection time spent by clients visiting the partner sites. The motivation to such a paradigm is similar to that of companies offering '800' service phone numbers in the U.S., that reverse the phone charges of customer calls. The scheme we suggest allows metering such visits and reversing the charges for them.

In all of the examples above, the auditable metering scheme was used to prevent (mass) forgery by the meter. Forging mass (and diverse) visits is just as hard with our scheme for the client. This property is useful when metering is employed for royalty payments on copyright material, since it inhibits owners of the copyrights from inflating their own royalties.

9 Caveats

The compact metering scheme uses the difficulty of computing the timing function to leverage auditability. This difficulty (likewise, the computation time invested) may differ between different platforms: At one extreme, clients may be smart Internet terminals with very limited computational power, and at the other extreme, clients may be top of the line workstations. Moreover, a forger using optimized native code has an advantage over an honest client importing mobile code—such as a Java applet—to execute a timing function. Thus, the timing scheme should be tuned carefully to the environment. When metering is deployed by an ISP (e.g., for reverse charging), this problem may be alleviated since an ISP has direct link into the client's environment.

Finally, the success of our method depends on the cooperation of most clients to import and execute mobile code. We note though, that if a client fails to participate in the metering protocol, this will be immediately detected by the meter.

10 Conclusion

The advertising potential of the WWW opens new challenges for securing metering and pricing advertisements. In this work, we considered the problem of automatically metering client accesses to a WWW site in an auditable way. We presented a novel metering scheme that offers light-weight security suitable for preventing mass fraud. The scheme requires no third party involvement, nor does any party record data that can be used to trace the identity of clients. Using Java applets to deploy programs from a server to its client, the scheme involves clients in a large distributed computation that incurs a reasonable computational cost by each client. We gave a detailed description of the implementation of webmeter, a metering tool that implements our metering scheme in the

current WWW framework, and explained how it records all client visits, including ones served by cached copies.

Acknowledgements

We thank Yuval Peres for his help with the derivation of a closed form formula for the expected value of our approximate auditing function.

References

[1] S. Ar, J. Cai, Reliable benchmarks using numerical instability. In *ACM Symposium on Discrete Algorithms (SODA)*, pages 34–43, 1993.

[2] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. RFC-1866, SRI Network Information Center, 1995.

[3] J. Cai, R. Lipton, R. Sedgewick and A. Yao, Towards uncheatable benchmarks. *IEEE Structures*, pages 2–11, 1993.

[4] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security, repelling the wily hacker.* Addison-Wesley, 1994.

[5] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology—CRYPTO '92, volume 740 of Lecture Notes in Computer Science*, pages 139–147, 1993.

[6] M. Franklin and D. Malkhi. Auditable Metering with Lightweight Security. In *Financial Cryptography '97* (FC '97), (LNCS, 1318), R. Hirschfeld (Ed.), Anguilla, February 1997, pp. 151-160.

[7] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. In *Proc. 4th International World Wide Web Conference*, pages 603–618, December 1995. http://www.research.digital.com/SRC/millicent.

[8] K. E. B. Hickman. The SSL Protocol. Technical report, Netscape Communications Corp, 1995.

[9] M. Naor and B. Pinkas. Secure and Efficient Metering. Proceedings of *Eurocrypt '98*, to be published.

[10] A. Schiffman E. Rescorla. The Secure HyperText Transfer Protocol. Technical report, Web Transaction Security Working Group, Enterprise Integration Technologies, July 1995.

[11] E. I. Schwartz. Advertising Webonomics 101. *Wired*, 4(02):74–82, 1996.

[12] Gary Welz. The Internet World Guide to Multimedia on the Internet. http://found.cs.nyu.edu/found.a/CAT/misc/welz/internetmm/index.html; to be published.