Pay as you Browse: Microcomputations as Micropayments in Web-based Services

Ghassan O. Karame
Dept. of Computer Science
ETH Zurich, Switzerland
karameg@inf.ethz.ch

Aurélien Francillon
Dept. of Computer Science
ETH Zurich, Switzerland
afrancil@inf.ethz.ch

Srdjan Čapkun
Dept. of Computer Science
ETH Zurich, Switzerland
capkuns@inf.ethz.ch

ABSTRACT

Currently, several online businesses deem that advertising revenues alone are not sufficient to generate profits and are therefore set to charge for online content. In this paper, we explore a complement to the current advertisement model; more specifically, we propose a micropayment model for non-specialized commodity web-services based on microcomputations. In our model, a user that wishes to access online content offered by a website does not need to register or pay to access the website; instead, he will accept to run microcomputations on behalf of the website in exchange for access to the content. These microcomputations can, for example, support ongoing computing projects that have clear social benefits (e.g., projects relating to HIV, dengue, cancer, etc.) or can contribute towards commercial computing projects. We argue that this micropayment model is economically and technically viable and that it can be integrated in existing distributed computing frameworks (e.g., the BOINC platform). We implement a preliminary prototype of a system based on our model through which we evaluate its performance and usability. Finally, we analyze the security and privacy of our proposal and we show that it ensures payment for the content while preserving the privacy of users.

1. INTRODUCTION

In the last couple of years, the drop in Internet advertising revenues [1, 20] has generated a discussion on possible alternatives that will increase the revenues of online businesses and shareholders. Some researchers [20] argue that the loss in Internet revenues was not solely caused by the general economic recession but also finds roots in the online advertisement model itself. Studies have reported that users do not trust online advertisements [2]; some users further use tools to block them.

Given this, several websites are set to charge for online content. For instance, News Corporation has declared that it will start charging for news content by 2011 [1]. Several other online businesses are likely to follow the same move to increase their revenues. This shift is, however, expected to alienate a considerable number of online users.

While users might be willing to pay for low-cost specialized online products such as music and movies, they are not keen on accepting subscription charges to read online news, to sign in Facebook, etc. In fact, studies have shown that only a small fraction of users—almost three percent—are willing to pay to read online news [1, 3]. Users are also not willing to set up and frequently recharge accounts for each online commodity service that they use. These issues make many commodity websites reluctant to charge for content and/or registration. The challenge for most media and online businesses lies, therefore, in extracting revenues from their online content without alienating existing users.

In this paper, we consider this problem and we propose a new framework that enables websites to "charge" for content, thus increasing their revenues, without requiring subscription charges from their users. Our scheme somehow departs from current micropayment methods and offers online businesses an indirect form of remuneration—similar to the current advertisement model. In our scheme (Figure 1), a user wishing to access online content offered by a website does not need to register or pay to access the website; instead, he will accept to run some computations on behalf of the website in exchange for access to the content. After verifying the integrity of the results reported by the user, results of the computations are gathered by the service provider (or by a broker) and sent to a distributed computing partner in exchange for a payment. The computations carried out by the user could correspond to those used in the multitude of available distributed computing platforms (such as SETI@home [4], distributed.net [5], etc.); alternatively, these computations could also be performed on behalf of governmental agencies, research labs and private industries. Note that, similarly to the existing advertisement model, a third party could mediate the exchange between online services and their computing partners. However, unlike the targeted advertisement model where the (privacy-invasive) user profiling increases revenues, our framework does not require the content providers to intrude on the privacy of users. Note that our scheme can be used as a complementary model to the existing advertisement model.

In this respect, our proposed scheme shares similarities with "parasitic computing" [19, 33], where covert computations are executed on users' machines without their consent and knowledge. Our scheme, on the other hand, extends the notion of "parasitic computing" to offer users a transparent—yet undisguised—micropayment method. The

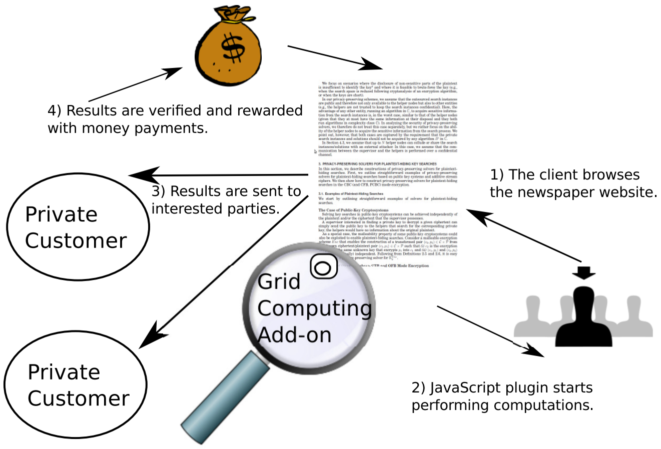

4) Results are verified and rewarded with money payments.
1) The client browses the newspaper website.
3) Results are sent to interested parties.
Private Customer
Grid Computing Add-on
Private Customer
2) JavaScript plugin starts performing computations.

Figure 1: Example of using microcomputations as a payment scheme in online newspapers. Upon accessing the news website, the client's browser performs microcomputations. The results are then sent back to interested third party in exchange of a payment.

main benefits of our scheme are as follows: *(i)* our scheme is likely to boost the revenues of online services; in fact, such a barter is likely to be more accepted by users—when compared to subscription charges—by exploiting their willingness to aid ongoing projects that have clear benefits (e.g., projects relating to HIV, cancer, clean energy, etc.). In addition, our scheme enables users to state which computations/projects they prefer to support. *(ii)* Due to the increasing reliance on distributed computing platforms to solve computationally challenging projects, our scheme will enable existing platforms to harness the idle computing power of users for as long as these users are using online services (e.g., while users are reading online news).

(iii) This entire process can be made transparent to the users as the computations can be carried out within their browsers. *(iv)* Finally, the advantage of malicious users in mounting attacks against our scheme is negligible since the targeted service corresponds to a low-cost commodity content.

Note that a thorough analysis of the economic viability of our scheme is out of the scope of the paper; instead we show in this work the feasibility of using microcomputations as a micropayment method for online content.

In addition to proposing the use of microcomputations as a micropayment scheme, we make the following contributions. We show that our proposed micropayment scheme naturally supports the anonymity and the privacy of users. Nevertheless, we show that the authenticity and the correctness of the results submitted by the user can be verified—irrespective of the purpose and the nature of the computations in question. We further demonstrate that this entire process is transparent to users and does not require any demanding hardware/software to be installed on their machines. For this purpose, we implement a cross-browser framework, using Google Web Toolkit [16], that allows websites to perform computations without any need for additional plugins. In our implementation, the browser fetches computations from a remote server and executes them using the idle CPU of the user's machine for as long as the user is accessing a specific online service. Finally, we performed a preliminary user study to assess the usability of our model.

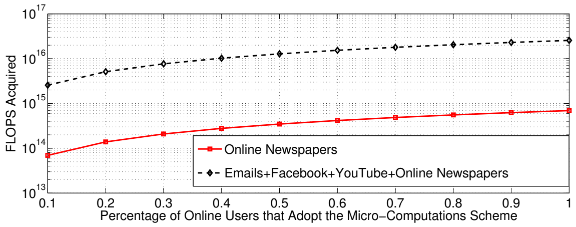


Figure 2: Total Number of FLOPS acquired in our scheme versus the number of Online Newspaper Viewers. For comparison purposes, the GIMPS project [6] aggregates around 44 TFLOPS ($44 \cdot 10^{12}$ FLOPS).

The remainder of this paper is organized as follows. In Section 2, we describe the main intuitions behind our proposed scheme. In Section 3, we present our framework and we analyze its security and privacy implications. We implement our scheme and we assess its performance in Section 4. In Section 5, we discuss additional insights to our proposals. In Section 6, we overview related work in the area and we conclude the paper in Section 7.

2. MOTIVATION

To better illustrate the benefits of our microcomputations scheme, we consider an example where online newspapers would require their clients to perform computations for as long they access their content (i.e., their portal).

For the sake of this example, we assume that there are 60 million unique clients accessing online news per month, and that each client spends on average 50 minutes per month browsing the news sites [7]. Although the pricing for the cost of computations is well understood (ranging between 0.1 and 0.3 $ per hour [8,9]), we rely in this example on a worst case analysis and we estimate the market cost of computations by the cost of electricity consumption of the machines involved in the computations. On average, a computer is estimated to use 100 watts per hour [10] and the cost of a kWh is estimated to be 0.1 $ [11].

This suggests that the cost of computations can be lower-bounded by 0.01 $ per hour.

In that case, online newspapers will be able to generate 1.5 million $ of revenues per month based on our proposed model; when compared to online advertisements, this revenue is equivalent to each of the 60 million users clicking on almost 1 ad per month in the Google AdSense advertisement framework [15].

Furthermore, assuming that the clients have machines that are each capable of performing an average of 10 GFLOPS [18], the total computing power harnessed in this case is equivalent to $\frac{10 \cdot 10^9 \cdot 60 \cdot 10^6 \cdot 50 \cdot 60}{30 \cdot 24 \cdot 3600} \approx 7 \cdot 10^{14}$ FLOPS (0.7 PFLOPS) on average; this exceeds, by orders of magnitude, the collective computing power harnessed by both SETI@home [4] and the GIMPS project [6]. In Figure 2, we show the total number of acquired FLOPS with respect to the fraction of online users that adopt our scheme; for illustrative purposes, we also include the equivalent number of acquired FLOPS

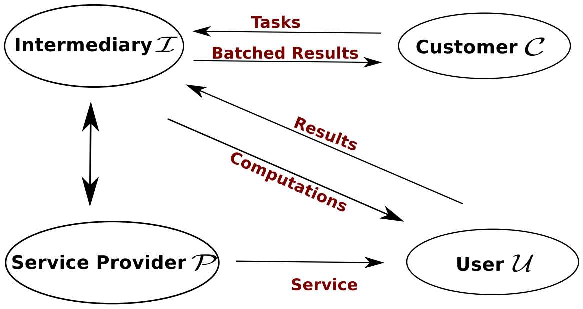


Figure 3: Our Model: when a user $\mathcal{U}$ accesses the service provided by $\mathcal{P}$, the service provider requires that $\mathcal{U}$ runs microcomputations. The results of those computations are communicated by the intermediary $\mathcal{I}$ to the customer $\mathcal{C}$.

in case our scheme is integrated in FaceBook, YouTube and major Email providers [1].

3. MICROPAYMENTS BASED ON MICROCOMPUTATIONS

In this section, we describe and analyze our solution based on the use of microcomputations as a micropayment method.

3.1 System and Attacker Model

We consider the following system. A customer $\mathcal{C}$ (e.g., a research lab or a private client) outsources tasks to a broker $\mathcal{I}$; $\mathcal{I}$ acts as an intermediary between $\mathcal{C}$ and a service provider $\mathcal{P}$ in exchange for financial remuneration. We assume that the outcomes of the outsourced tasks are not confidential; in Section 4, we show an example where the instances of the tasks are kept secret.

When a user $\mathcal{U}$ accesses the service provided by $\mathcal{P}$, $\mathcal{P}$ requires that $\mathcal{U}$ contacts $\mathcal{I}$ and runs a subset of the outsourced tasks—in the form of computations— on behalf of $\mathcal{I}$'s customers. These computations are carried out in the user's browser. In exchange for these computations, the users get access to the service offered by $\mathcal{P}$ and $\mathcal{I}$ financially rewards $\mathcal{P}$. This process is shown in Figure 3. To ensure the security of our system, *(i)* $\mathcal{I}$ (and/or $\mathcal{C}$) needs to be able to efficiently verify the outcomes of the outsourced computations and *(ii)* $\mathcal{I}$ (and $\mathcal{C}$) needs to ensure that any outsourced computation instance can only be used once by users (and by $\mathcal{I}$) in exchange for remuneration.

$\mathcal{I}$ has access to several independent tasks, pertaining to different customers; these tasks could be either fully sequential or parallelizable or even hybrid tasks (i.e., contain both sequential and parallelizable subroutines). Individual tasks are independent of each others.

We further assume that these tasks are *computationally* expensive but require moderate memory access. In general, tasks that require modest memory access can be efficiently outsourced and decomposed into microcomputations as they incur a relatively low communication overhead (as exemplified in Section 4).

Throughout the rest of this paper, we assume a secure channel between $\mathcal{P}$ and $\mathcal{I}$ (e.g., $\mathcal{P}$ and $\mathcal{I}$ can use pre-shared keys) and we abstract away the details of the communication channel between $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{I}$, such as delays, congestion, jitter, etc. We further assume that $\mathcal{P}$ and $\mathcal{I}$ are motivated to increase their benefit in the system and we assume the existence of one or multiple colluding malicious users. These users have knowledge of the measures used by $\mathcal{I}$ to prevent potential tampering with the computations. We assume that malicious users are motivated to cheat in order to access a service without performing all of their assigned computations. For instance, a user might only execute 50 % of its assigned computations and defect from running the rest of its tasks. Here, two or more malicious users might collude to increase their chances of not being detected.

3.2 Transforming Distributed Tasks into Verifiable Computations

In what follows, we outline existing solutions that enable efficient probabilistic verification of the remote execution of *parallel* and *sequential* computations. In Section 3.3, we will leverage on these solutions to ensure the security of our scheme.

3.2.1 Verifying the Remote Execution of Parallel Computations

Parallel computations consist of the evaluation of a function or algorithm $f : \mathcal{D} \to \mathcal{R}$ for every input value $x \in \mathcal{D}$. Subtasks are then created by partitioning $\mathcal{D}$ into subsets $\mathcal{D}_i$; in other words, subtask i will evaluate f (or even a function of f) for every input $x \in \mathcal{D}_i$.

The most efficient solution to verify the remote execution of parallel computations on the user's machine is for the supervisor of the computations $\mathcal{I}$ to rely on selective redundancy or to selectively embed indistinguishable precomputed checks—ringers [24]—within the tasks of the nodes. To verify the integrity of the computations, $\mathcal{I}$ chooses n uniformly distributed random values—the ringers—$r_1, r_2, .., r_n$ from $\mathcal{D}_i$ and computes the set $S \leftarrow \{f(r_1), f(r_2), .., f(r_n)\}$.[2] Note that n is kept secret by $\mathcal{P}$. The computations performed by $\mathcal{U}$ will only be accepted if and only if $\forall r_i \in \mathcal{D}_i$, $f(r_i)$ is correctly computed.

Since $\mathcal{U}$ cannot distinguish the ringers from other data values in $\mathcal{D}_i$, and does not know how many ringers are embedded within the input space, $\mathcal{U}$ has to complete all of its assigned work, with high probability, for its computations to be accepted by $\mathcal{I}$. By using the ringer scheme, the probability of detecting possible misbehavior by $\mathcal{U}$ is given by $P = 1 - (1 - P_c)^n$, where P_c denotes the fraction of incorrect inputs returned by $\mathcal{U}$.

A practical example of using ringers to secure remote cryptographic computations is outlined in Section 4.

3.2.2 Verifying the Remote Execution of Sequential Computations

A sequential task function f is given by $f(x) = (g \circ h \circ j \circ k...)(x)$ where $g(.)$, $h(.)$, $j(.)$, $k(.)$ etc. are the constituent

[1] Here, we used the following estimates (adopted from online reports): there are $5 \cdot 10^8$ Facebook logins per month, 10^9 YouTube video views per month and a total of $6,5 \cdot 10^8$ emails (Gmail, Yahoo, Hotmail and AOL) exchanged per month.

[2] In case the computation of $f(.)$ is not small enough, $\mathcal{I}$ can proceed as outlined in [34]; it embeds few ringers initially in a smaller input space and then uses the results reported by various users as ringers in subsequent interactions.

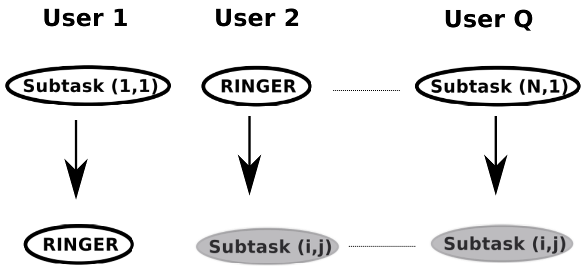


Figure 4: N tasks assigned to Q participants. Each box denotes a subtask. The notation "Subtask (X,Y)" refers to subtask # Y of task # X. The gray boxes refer to redundantly assigned subtasks.

sub-functions. "Hybrid" computations are considered as a special instance of sequential computations since they contain both sequential and non-sequential sub-functions.

Similar to their non-sequential counterpart, ringers could also be used to secure the remote execution of sequential computations. However, unlike parallel computations, ringers can only be efficiently used when several sequential tasks are permuted together and outsourced to users [30, 34].

We now describe a scheme—adopted from the findings in [30, 34]—for securing the remote execution of N distinct and independent tasks on the remote machines of Q ($Q \geq N$) different users. This scheme unfolds as follows:

$\mathcal{I}$ first divides each task into M smaller subtasks. This can be achieved by decomposing the task into its smaller functional components. $\mathcal{I}$ then proceeds to running the N tasks on the machines of Q users in M consecutive rounds.

In round i, $\mathcal{I}$ picks an idle user and according to some probability, it decides to verify its credibility by inserting "security checks" within the computations; alternatively, it can randomly assign to the participant a pending subtask. In this scheme, $\mathcal{P}$ evaluates the credibility of a user by requesting that it runs a subtask whose results are already known to $\mathcal{I}$ (a *ringer*) or by redundantly assigning the same subtask to another user. Note that this process is transparent to users and that they cannot distinguish whether they are running a legitimate subtask or whether their work is being checked by $\mathcal{I}$. Round i ends when all N users are assigned a job. In this way, $\mathcal{I}$ checks the work of several participants in each round.

At the beginning of round $i + 1$, $\mathcal{I}$ collects the results reported by the users and checks the correctness of the ringers and the redundantly assigned subtasks. If these verifications pass, $\mathcal{I}$ re-permutes the next logical subtasks (since each task is sequential) among the users while using the corresponding outputs of the last round as inputs to the subtasks of this round. $\mathcal{P}$ repeats this process until all subtasks are executed (Figure 4). Table 1 shows the probability of detecting malicious users by using ringers (or selective redundancy) with respect to various parameters. Further details on the performance and efficiency of this scheme can be found in [30, 34].

3.3 Our Scheme: Microcomputations as Micropayments

Our proposed framework (Figure 6) comprises of the following modules:

The Customer Server $\mathcal{C}$: The customer server $\mathcal{C}$ aggregates and outsources different task jobs pertaining to various distributed computing projects. For instance, $\mathcal{C}$ could correspond to the existing BOINC server [13] that hosts volunteer grid computing projects.

Number of Ringers or Redundancy (per bundle)	Probability of Cheating	Detection Probability
1	1	**1**
3	0.5	**0.875**
5	0.5	**0.96875**
10	0.5	**0.999**

Table 1: Probability of detecting misbehavior (in parallel and sequential tasks) using ringers and/or selective redundancy with respect to different input parameters.

The Intermediary $\mathcal{I}$: The intermediary acquires work units from $\mathcal{C}$, and manages the outsourcing of computations to the users. Namely, $\mathcal{I}$ splits the work units into microcomputations, embeds indistinguishable ringers or redundancy among the computations, as described in the previous sections, and sends the computations to users. Later on, $\mathcal{I}$ aggregates the individual results reported by users after checking their integrity and dispatches the entire work unit result to $\mathcal{C}$. To prevent users from re-using previous microcomputations as micropayments, $\mathcal{I}$ also keeps track of the microcomputations that were previously outsourced[3].

The Service Provider $\mathcal{P}$: The service provider offers content to various users (e.g. Facebook, online newspapers). In what follows, we describe our scheme in greater detail.

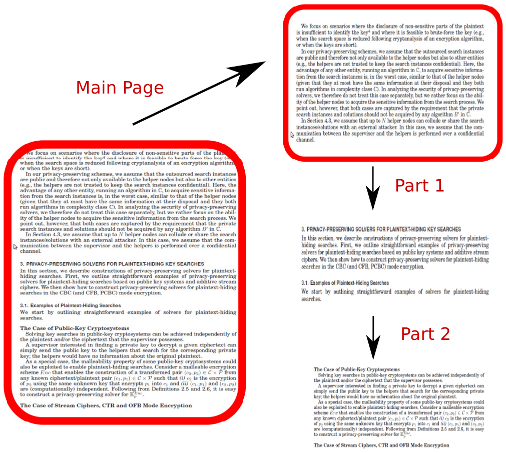
Webpage Viewing.

We start by describing one possible way to display webpages to fit our proposed model.

We assume that the webpage content is split into parts; users can fetch the subsequent parts manually as they browse the content, e.g., through a "Next Page" button, once they correctly perform the required microcomputations. An example of displaying an online newspaper article is shown in Figure 5. This abides by the current model adopted in online newspapers where *(i)* the abstract (first page) of article can be viewed free of charge while a full-article view requires payment or registration and *(ii)* content is split between different pages that can be accessed manually through a "Next" button. However, unlike current solutions, our model allows users to pay as they browse (users only need to perform the microcomputations for the pages that were loaded on their browsers).

We further assume that the service provider always responds with the first part of the content whenever a new session is started. Before serving subsequent parts, the service provider ensures that users have correctly performed the microcomputations. This prevents users from reading the first part of the content without performing the computations, and then restarting a new session with $\mathcal{P}$ to read the subsequent part and so on. Note that, except for the

[3] This is a common requirement in most existing distributed computing platforms.



We focus on scenarios where the disclosure of non-sensitive parts of the plaintext is insufficient to identify the key[4] and where it is feasible to brute-force the key (e.g., when the search space is reduced following cryptanalysis of an encryption algorithm, or when the keys are short).

In our privacy-preserving schemes, we assume that the outsourced search instances are public and therefore not only available to the helper nodes but also to other entities (e.g., the helpers are not trusted to keep the search instances confidential). Here, the advantage of any other entity, running an algorithm in $\mathbb{C}$, to acquire sensitive information from the search instances is, in the worst case, similar to that of the helper nodes (given that they at most have the same information at their disposal and they both run algorithms in complexity class $\mathbb{C}$). In analyzing the security of privacy-preserving solvers, we therefore do not treat this case separately, but we rather focus on the ability of the helper nodes to acquire the sensitive information from the search process. We point out, however, that both cases are captured by the requirement that the private search instances and solutions should not be acquired by any algorithm B' in $\mathbb{C}$.

In Section 4.3, we assume that up to N helper nodes can collude or share the search instances/solutions with an external attacker. In this case, we assume that the communication between the supervisor and the helpers is performed over a confidential channel.

3. PRIVACY-PRESERVING SOLVERS FOR PLAINTEXT-HIDING KEY SEARCHES

In this section, we describe constructions of privacy-preserving solvers for plaintext-hiding searches. First, we outline straightforward examples of privacy-preserving solvers for plaintext-hiding searches based on public key systems and additive stream ciphers. We then show how to construct privacy-preserving solvers for plaintext-hiding searches in the CBC (and CFB, PCBC) mode encryption.

3.1. Examples of Plaintext-Hiding Searches

We start by outlining straightforward examples of solvers for plaintext-hiding searches.

The Case of Public-Key Cryptosystems

Solving key searches in public-key cryptosystems can be achieved independently of the plaintext and/or the ciphertext that the supervisor possesses.

A supervisor interested in finding a private key to decrypt a given ciphertext can simply send the public key to the helpers that search for the corresponding private key; the helpers would have no information about the original plaintext.

As a special case, the malleability property of some public-key cryptosystems could also be exploited to enable plaintext-hiding searches. Consider a malleable encryption scheme Enc that enables the construction of a transformed pair $(c_2, p_2) \in \mathcal{C} \times \mathcal{P}$ from any known ciphertext/plaintext pair $(c_1, p_1) \in \mathcal{C} \times \mathcal{P}$ such that *(i)* c_2 is the encryption of p_2 using the same unknown key that encrypts p_1 into c_1 and *(ii)* (c_1, p_1) and (c_2, p_2) are (computationally) independent. Following from Definitions 2.5 and 2.6, it is easy to construct a privacy-preserving solver for $\mathbb{K}_p^{Enc}$.

The Case of Stream Ciphers, CTR and OFB Mode Encryption

We focus on scenarios where the disclosure of non-sensitive parts of the plaintext is insufficient to identify the key[4] and where it is feasible to brute-force the key (e.g., when the search space is reduced following cryptanalysis of an encryption algorithm, or when the keys are short).

In our privacy-preserving schemes, we assume that the outsourced search instances are public and therefore not only available to the helper nodes but also to other entities (e.g., the helpers are not trusted to keep the search instances confidential). Here, the advantage of any other entity, running an algorithm in $\mathbb{C}$, to acquire sensitive information from the search instances is, in the worst case, similar to that of the helper nodes (given that they at most have the same information at their disposal and they both run algorithms in complexity class $\mathbb{C}$). In analyzing the security of privacy-preserving solvers, we therefore do not treat this case separately, but we rather focus on the ability of the helper nodes to acquire the sensitive information from the search process. We point out, however, that both cases are captured by the requirement that the private search instances and solutions should not be acquired by any algorithm B' in $\mathbb{C}$.

In Section 4.3, we assume that up to N helper nodes can collude or share the search instances/solutions with an external attacker. In this case, we assume that the communication between the supervisor and the helpers is performed over a confidential channel.

3. PRIVACY-PRESERVING SOLVERS FOR PLAINTEXT-HIDING KEY SEARCHES

In this section, we describe constructions of privacy-preserving solvers for plaintext-hiding searches. First, we outline straightforward examples of privacy-preserving solvers for plaintext-hiding searches based on public key systems and additive stream ciphers. We then show how to construct privacy-preserving solvers for plaintext-hiding searches in the CBC (and CFB, PCBC) mode encryption.

3.1. Examples of Plaintext-Hiding Searches

We start by outlining straightforward examples of solvers for plaintext-hiding searches.

The Case of Public-Key Cryptosystems

Solving key searches in public-key cryptosystems can be achieved independently of the plaintext and/or the ciphertext that the supervisor possesses.

A supervisor interested in finding a private key to decrypt a given ciphertext can simply send the public key to the helpers that search for the corresponding private key; the helpers would have no information about the original plaintext.

As a special case, the malleability property of some public-key cryptosystems could also be exploited to enable plaintext-hiding searches. Consider a malleable encryption scheme Enc that enables the construction of a transformed pair $(c_2, p_2) \in \mathcal{C} \times \mathcal{P}$ from any known ciphertext/plaintext pair $(c_1, p_1) \in \mathcal{C} \times \mathcal{P}$ such that *(i)* c_2 is the encryption of p_2 using the same unknown key that encrypts p_1 into c_1 and *(ii)* (c_1, p_1) and (c_2, p_2) are (computationally) independent. Following from Definitions 2.5 and 2.6, it is easy to construct a privacy-preserving solver for $\mathbb{K}_p^{Enc}$.

The Case of Stream Ciphers, CTR and OFB Mode Encryption

Figure 5: An example of displaying an online newspaper article: the content is split into parts that are fetched manually through a "Next Page" button. The framed pages can be accessible via a public URL without the need to perform microcomputations. Arrows model the various clicks performed by users.

"free" pages, content is not loaded on the browser of the user if there is no support for the tools that are required to perform the computations (e.g., if JavaScript is disabled).

Scheme Description.

When a client $\mathcal{U}$ initially requests a service from $\mathcal{P}$, $\mathcal{P}$ responds with the first part of the content, a link to a script hosted by $\mathcal{I}$, along with a session ID, SID. SID is a pseudorandom identifier to identify the current session in subsequent interactions. $\mathcal{P}$ also informs $\mathcal{I}$ that a new session SID has been initiated. Recall that the communication between $\mathcal{P}$ and $\mathcal{I}$ is performed over a secure channel.

When $\mathcal{U}$ executes the script and contacts $\mathcal{I}$, the latter then outsources a challenge in the form of microcomputations to $\mathcal{U}$. These microcomputations run in the browser of $\mathcal{U}$. When completed, the results are sent back to $\mathcal{I}$. In our scheme, we assume that all the content offered by $\mathcal{P}$ has similar "value" and as such is reimbursed with a pre-determined amount of microcomputations[4].

$\mathcal{I}$ checks the integrity of the computations as described in the previous paragraphs. Note that the integrity verification of the computations can be performed very efficiently through table-lookup, since $\mathcal{I}$ already knows the solution to the ringer problems. $\mathcal{I}$ then informs $\mathcal{P}$ of the outcome of the verification. If the verification passes, $\mathcal{P}$ accepts further requests for content from $\mathcal{U}$. In that case, the entire process re-iterates as shown in Figure 6. We analyze the security properties of our scheme in Section 3.4.

We point out that the efficiency of outsourcing the microcomputations and the load incurred on $\mathcal{I}$ in this case is comparable to those incurred in existing distributed computing platforms (e.g., the SETI@home server). In fact, these servers already embed enough functionality to allocate, assign and collect sub-computations from millions of users. We further note that, in our scheme, ringer units are also embedded within the work bundles that are outsourced from $\mathcal{C}$ to $\mathcal{I}$. The fraction of ringers units inserted by $\mathcal{C}$ could be as low as 5 %, when compared to the fraction of ringers needed to ensure the correctness of the microcomputations that are run by the users (typically ≥ 30 %). This is the case since users can "whitewash" their history, simply by restarting a new session with $\mathcal{P}$. The intermediary $\mathcal{I}$, on the other hand, has a unique and fixed identity; it suffices for $\mathcal{C}$ to detect misbehavior by $\mathcal{I}$ *once* to stop interacting with it (e.g., $\mathcal{I}$ might face a legal case).

Remark. In our scheme, $\mathcal{U}$ can make a choice with respect to which project/computations it would like to run. For example, a small tab that is loaded with the accessed pages can list all possible computing projects that $\mathcal{U}$ can participate in. If $\mathcal{U}$ does not provide any choice, $\mathcal{I}$ assigns subtasks given its default scheduling strategy.

3.4 Security and Privacy Considerations

In what follows, we analyze the security and privacy offered by our proposed scheme.

Security.

Given our scheme, a user can perform the computations once, save the content locally, and subsequently re-post the content (without the corresponding script) to other users that it colludes with. Furthermore, since the communication between the users and ($\mathcal{P}$, $\mathcal{I}$) is not performed over a secure channel, a user A can perform a Man-In-The-Middle (MITM) attack to impersonate another user B in order to access content without performing the required microcomputations[5]. We argue, however, that the advantage of A in performing all these attacks is negligible; the effort in mounting such attacks exceeds, by far, the outcomes since the targeted service corresponds in our case to low-cost, commodity content that is currently public. In that respect, securing the communication between $\mathcal{U}$ and ($\mathcal{P}$, $\mathcal{I}$) can be seen as an expensive commodity when compared to the advantage of users in performing such attacks. Nevertheless, since impersonation and MITM attacks can be immediately detected by users (new content does not load on their browsers), users can switch to secure communication (e.g., HTTPS) with $\mathcal{I}$ and $\mathcal{P}$ to circumvent such attacks.

One major security requirement here is to verify the results reported by users. Since the microcomputations are bundled in the users' browsers, users can easily misreport the outcomes (e.g., by directly editing the page source code from the browser). In our scheme, ringers (and selective redundancy) efficiently ensure, with high probability, the integrity of remote generic computations —in spite of collusion among users. Since the intermediary $\mathcal{I}$ keeps pointers to the previously outsourced computation, users cannot re-use results pertaining to past computations in exchange for content. This is often referred to in the literature as the "double-spending" problem, where users re-use "expired" tokens as payments. This problem is inherently countered by the use of ringers—even if $\mathcal{I}$ does not keep track of previously solved computations. This is the case since the ringers are

[4] Alternatively, $\mathcal{I}$ can assign different computation loads depending on the content "value" or the period during which the content is being accessed.

[5] One possible alternative would be for user B to send its results along with an HMAC using SID as a key. However, this solution is only effective when user A cannot eavesdrop on the channel between B and $\mathcal{P}$.

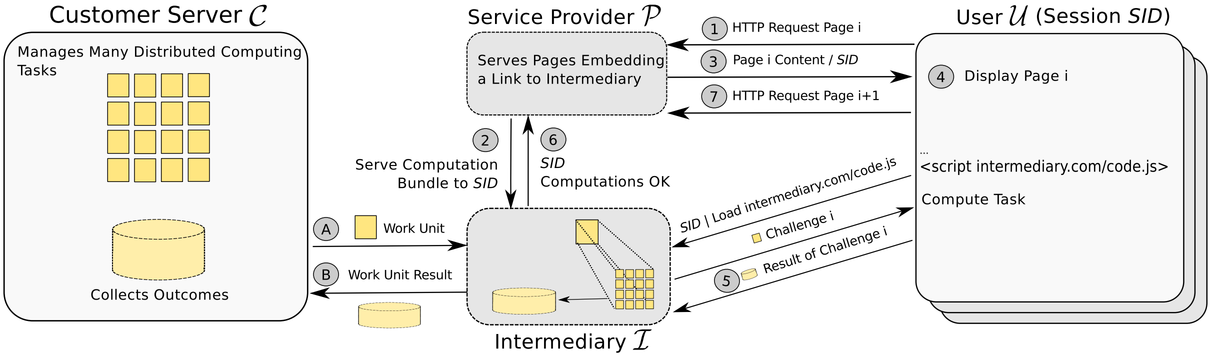

Customer Server $\mathcal{C}$
Manages Many Distributed Computing Tasks
Collects Outcomes
Service Provider $\mathcal{P}$
Serves Pages Embedding a Link to Intermediary
User $\mathcal{U}$ (Session *SID*)
1 HTTP Request Page i
3 Page i Content / *SID*
7 HTTP Request Page i+1
4 Display Page i
...
<script intermediary.com/code.js>
Compute Task
2 Serve Computation Bundle to *SID*
6 *SID* Computations OK
A Work Unit
B Work Unit Result
SID | Load intermediary.com/code.js
Challenge i
5 Result of Challenge i
Intermediary $\mathcal{I}$

Figure 6: Our Microcomputations Scheme: when a client $\mathcal{U}$ requests a service from the service provider $\mathcal{P}$, $\mathcal{P}$ informs the intermediary $\mathcal{I}$ that a new session SID has been initiated. $\mathcal{I}$ then outsources a challenge in the form of microcomputations to $\mathcal{U}$. These computations are chosen from the pool of tasks available at the customer server $\mathcal{C}$ (e.g., BOINC [13]). Further requests by the user are only accepted if the results of the computations are correct.

indistinguishably unique in each outsourced subtask; even if users can predict the algorithm to be executed along with its input instances, they cannot predict which ringer values to report to $\mathcal{I}$. This also prevents users from generating and running fake computations—while claiming that these computations were outsourced by $\mathcal{I}$. In general, it can be easily shown that the use of ringers ensures, with high probability, the *integrity* and the *authenticity* of the remote microcomputations.

We conclude that, in our scheme, rational users cannot acquire content without "correctly" performing the required microcomputations.

On the other hand, the use of ringers also prevents $\mathcal{I}$ from reporting incorrect work unit results to the customer $\mathcal{C}$ since such a misbehavior will be detected with high probability. Note that a service provider might try to impersonate another provider in order to increase its revenues. This will be immediately detected since the communication between service providers and $\mathcal{I}$ is performed over a secure channel. Furthermore, unlike the advertisement model, our model allows both $\mathcal{P}$ and $\mathcal{I}$ to keep track of the number of page accesses; $\mathcal{P}$ and $\mathcal{I}$ can then compare the number of page requests to settle disputes. In this respect, since our micropayment scheme is based on "verifiable" microcomputations, $\mathcal{P}$ cannot over-charge $\mathcal{I}$ without committing enough of its time and resources to correctly execute the microcomputations. This also suggests that our scheme is resilient to Denial-of-Service (DoS) attacks; the verifiable microcomputations act as computational puzzles [28] that ensure that an attacker commits a considerable amount of resources, when compared to $\mathcal{I}$, before its request is served.

Privacy.

Current advertisement platforms perform extensive user-profiling and tracking [27] to build a fine grained user profile. In contrast, our micropayment scheme inherently supports the privacy of the users and does not embed any incentive for any party to perform user-profiling. This is the case since the ongoing microcomputations are independent by construction of the users' preferences and profiles. Furthermore, the users do not need to register or create accounts to "pay" in exchange of content in our scheme.

As shown in Section 3.3, the communication between $\mathcal{P}$ and $\mathcal{I}$ solely contains a temporary session identifier; this identifier is changed in every established session and therefore cannot be used for tracking purposes. As such, the only knowledge that is leaked to $\mathcal{P}$ and $\mathcal{I}$ corresponds to the content that is being accessed, the IP of the user and some information about the users' browsers and/or the computational loads on the users' machines[6]. In order to further increase their privacy, users could access content and perform the computations using anonymizing networks.

4. PROTOTYPE DESIGN & EVALUATION

4.1 Prototype Description

To evaluate our scheme, we implemented a prototype that acts as a stand-alone server that distributes RC4 brute-force key search to browsers of the users in exchange of accessing content. Our implementation is based on the Google Web Toolkit (GWT) [16]. GWT allows to develop both the server-side and the client-side code in JAVA. The client-side code is then compiled to JavaScript code to be interpreted directly by the web-browser, without the need of further client-side support (e.g., this enables our framework to directly link with the BOINC platform). GWT also handles automatically the differences in JavaScript support from different browser's version and providers

Figure 7 depicts the architecture of our test prototype. Note that our implementation was based on a variant of the framework in Figure 6; here, we assume that $\mathcal{P}$, $\mathcal{I}$ and $\mathcal{C}$ are co-located on the same server and thus our prototype implementation abstracts away the communication between these entities. Since users are generally equipped with various machines and network connections (e.g., university network, broadband access, etc.), the main aim of our prototype implementation is to analyze and evaluate the performance and usability of our proposals from a user's perspective. Namely, the objectives of our preliminary evaluation were *(i)* to get an initial user feedback on the acceptability and usability of the scheme and *(ii)* to evaluate network latency and the load that is incurred by our scheme.

[6] This information leakage is not particular to our scheme and applies generically when users browse the web [21].

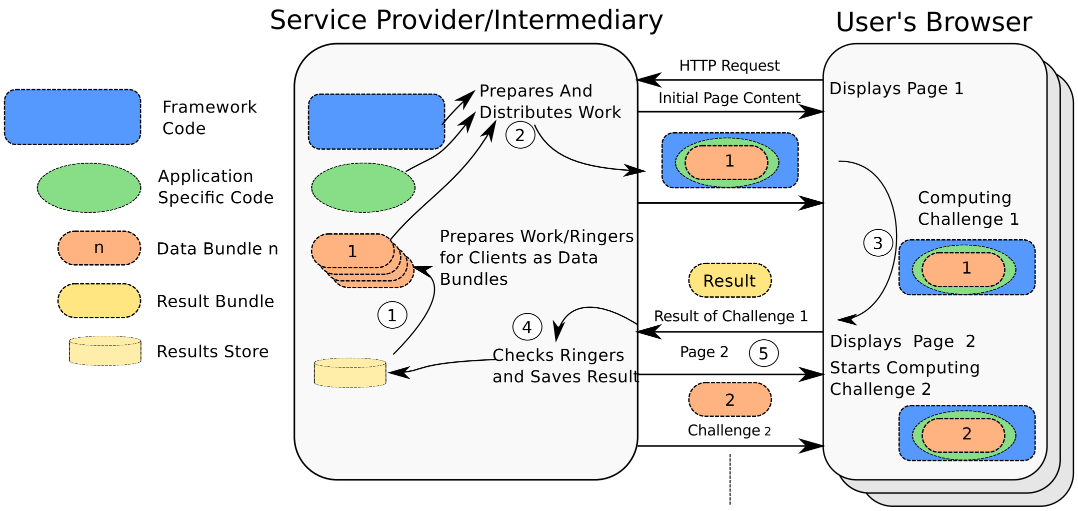


Figure 7: Prototype Design: Our prototype was implemented using the Google Web Toolkit (GWT) [16]. Our implementation is based on a variant of the framework presented in Figure 6; here, we assume that $\mathcal{P}$, $\mathcal{I}$ and $\mathcal{C}$ are co-located on the same server.

Construction of the Computation Bundles.

In our prototype implementation, the outsourced microcomputations consist of N sets of $\{Kr, C \oplus P\}$ where P is a given plaintext, C is its corresponding ciphertext encrypted with RC4 using a key k, and Kr is the assigned key-search space. Users have to check, for each of their assigned search spaces, $\{\forall\widehat{k} \in K_r | \, C \oplus P \overset{?}{=} RC4(\widehat{k})\}$. If a solution is found, the key is returned to the server. As described in [29], this scheme enables a privacy-preserving cryptographic search for the key, since it does not reveal the plaintext P to any entity that is involved in the cryptographic search.

Here, n ringers are constructed by encrypting a random plaintext P_r with a random key k_r to obtain the ringer's ciphertext C_r.

Evaluation Setup.

To evaluate our prototype, we constructed a webpage that hosts a computer science book chapter. The chapter is broken into pages that are displayed as JPEG images and that can be navigated through a “Next Page” button. When users click that button, JavaScript computations automatically run for approximately 15 seconds on their browsers; the button is only enabled when the computations are correctly performed and returned to the server. After the last page of the chapter is loaded, the “Next Button” points to a page that hosts the questions shown in Table 3.

Our preliminary evaluation consisted of three experiments. In the first experiment, we sent the link of our test webpage to our research group comprised of 10 computer science PhD students and post-docs. For the purpose of this experiment, 50 % of the users were randomly chosen not to perform any computations; in that case, the “Next Button” was simply disabled for 15 seconds.

In the second experiment, we sent the link of our test webpage to 32 students that were attending a computer science course at ETHZ. These were Masters and Bachelors students affiliated with various departmental tracks. The students were informed that they had to read the book chapter as an exercise for the lecture and that they have to give some answers at the end of their reading. They were also informed that the ‘Next Button” will only be enabled after some time, to ensure that they are actually reading each page of the chapter. In this experiment, the subjects had to answer a technical question related to the contents of the chapter along with the question included in second row of Table 3. 25 % of the subjects were randomly chosen not to perform any computations as a test group. To ensure that the purpose of the experiment was not visible to students, the remaining questions were asked in person at the end of the following lecture. In both experiments, the subjects were not hinted that they might be running computations on their browsers. Finally, in a third brief experiment, we accessed our webpage from an Android 2.2 smartphone, an Apple Iphone and from an IPad to evaluate the phones' CPU busy time.

4.2 Evaluation of our Scheme

Table 2 summarizes the statistics that we collected from our preliminary experiments. In total, the computing server served 1548 requests pertaining to 92 different users. Each request was served in 8.34 ms, totaling an aggregate server busy CPU time of approximately 13 seconds. The aggregate time that all users spent in performing the computations was 188 minutes, which enabled them to test for approximately 2^{26} RC4 keys. In this case, the ratio of the CPU time expended by JavaScript clients when compared to the server time is 867 (i.e., 1 day of server computation corresponds to 2.3 years in clients' computations time). While we acknowledge that faster implementations (e.g., using a specialized plugin that implements algorithms in native code) could be built, we note that even the relatively “slow” JavaScript-based environment can efficiently host the outsourcing of microcomputations.

Furthermore, in our experiments, the total network overhead was marginal (8.65 MB). This corresponds to 80 KB of static microcomputation script per session (that is loaded and cached by browsers). However, even bigger script sizes, e.g., of 200 KB (corresponding to more complex microcom-

Total Number of Keys Searched	2^{26} keys
Number of Data Bundles Sent	1548
Number of Individual Sessions	92
Aggregate Computation Time	188 minutes
Total Network Load	8.65 MB
Average Server CPU Time	8.34 ms per request
Average User CPU Time	15 s per request

Table 2: Prototype Deployment Statistics

putations), do not result in any noticeable performance degradation in our scheme when compared to the online advertisement model, provided that the computations require moderate memory usage. In fact, an estimated average of 300 KB of advertisement content is loaded on the browser of users on each page load[7].

In the third experiment, we noticed a maximum CPU usage of 75 % on the tested phones during which all devices remained fully responsive. We therefore observe that our scheme also results in acceptable browsing experience on smart phones.

Preliminary User Study Results.

The results of the preliminary user study that we conducted are shown in Table 3. Our findings *suggest* that *(i)* the ongoing microcomputations did not affect the browsing experience of subjects since no subject noticed them and *(ii)* most subjects are unlikely to be alienated by our micropayment scheme and as such our scheme seems to be well suited for browsing online newspapers or reading ebooks, where users are likely to spend some time on each page.

An interesting observation was that the vast majority of the subjects—even those that were reluctant on using our scheme—showed interest in adopting the microcomputations model when it is used to support socially beneficial projects. Most subjects wanted to be aware of what their browsers were running and preferred to have choice with respect to the purpose of the computations they would be running. As mentioned in Section 3.3, our scheme enables users to choose which computations to support on their browsers, thus enhancing the acceptability of our micropayment model.

Limitations of our study: Although these results are encouraging, we acknowledge that they are only preliminary given the limited number of subjects and given the fact that the subjects were mostly computer science students.

There are also a number of factors that we did not consider in this initial user study. For instance, one pertinent question here is whether users prefer slower browsing, that is somehow unavoidable in the microcomputations scheme, rather than simply paying for online services or viewing advertisements. Another important factor relates to the load incurred by these computations given a number of concurrent web-browsing sessions that are run simultaneously by users (e.g., users opening several tabs). To draw more decisive conclusions on the acceptability and usability of our scheme when compared to other possible alternatives (refer to Section 5), we plan to address these issues in a prospective large-scale study.

Evaluation	**Yes**	**No**
Did users detect that computations were running?	0 (0)%	100 (100)%
Did users experience unusual CPU load?	0 (0)%	100 (100)%
Would users generally agree to run computations as a way of payment?	85 (70)%	15 (30)%
Would users generally agree to run computations to support existing research efforts?	97 (95)%	3 (5)%
Was it unethical to execute code without the consent of the users?	72 (80)%	28 (20)%
Would the users like to be aware of the computations?	96 (100)%	4 (0)%

Table 3: Preliminary user study results. Numbers between parentheses correspond to the results from Experiment 1, numbers without parentheses denote the results from Experiment 2.

5. DISCUSSION

In what follows, we briefly discuss further insights with respect to our proposed scheme.

Marketplace for Computations.

In our scheme, since the microcomputations are carried out as the users browse content, the users would have to wait for the computations to complete before viewing subsequent content. Here, one potential limitation lies in the fact that this payment scheme favors users that are equipped with fast machines; those users are more likely to perform the computations faster and therefore to browse more at ease when compared to users that are equipped with "slower" (and/or energy-constrained) machines. One possible solution to ensure fairness among heterogeneous users is to estimate their computing performance from the time it took them to complete the assigned microcomputations and adjust their difficulty accordingly. Other techniques to securely verify the computing performance of remote devices have been recently reported [28].

Another natural solution to this problem—which somehow departs from our current scheme—would be to rely on a marketplace to sell "computational tokens"; computations can be carried out by users "offline" in exchange of tokens that users can use subsequently to access online content from any other devices they possess (e.g., a PDA device). We are currently investigating the feasibility of efficiently creating verifiable and anonymous tokens for generic computations.

Towards a Larger Computing Force.

We believe that our prototype implementation can also accommodate for many types of computationally intensive tasks, namely those pertaining to existing distributed computing platforms since our scheme incurs comparable functionality and efficiency when compared to current projects.

We do acknowledge that it might not be straightforward

[7] Due to the absence of references, this estimate was obtained empirically by averaging the size of advertisements that were embedded in three major online news sites (NY Times, BBC, Le Monde) and fetched by browsers on each page load.

to embed ringers in all classes of computations; in that case, redundantly assigning subtasks among users could be used as a mean to transform any task into a probabilistically verifiable computation [25]. As a proof of concept, we are currently extending our prototype implementation to accommodate for the use of various distributed projects to generate microcomputations.

Note that the premise of performing microcomputations extends well beyond simply browsing online pages. Users could be encouraged to run computations as they listen to an MP3 song (in that case, the computations are embedded in the MP3 bundle) or when streaming a YouTube video.

Currently, several distributed computing projects are continuously seeking to recruit additional volunteers to help in various areas such as biology, mathematics, medicine, cryptography, physics, science, etc. Our work therefore indirectly motivates for the need to support ongoing scientific projects, namely by offering them higher exposure. If all popular services, such as Facebook, YouTube, etc. adopt a variant (even based on voluntary participation) of our proposed scheme, then serious advances in several scientific or socially-beneficial projects might be reached in few years.

6. RELATED WORK

The emergence of cloud-computing infrastructures motivated the use of distributed computing as a way to generate revenues by managing the distributed computations of private clients and businesses [8, 12].

Security in Distributed Computing.

The literature contains several contributions that deal with security of outsourced computations in distributed systems. Golle *et al.* [24] propose to secure a specific class of parallel problems that are run on remote servers: inverse one-way functions, where helper nodes are required to compute the pre-images of several one-way functions. This solution is extended in [30, 34] to secure sequential computations. Goodrich *et al.* [25] discuss mechanisms to duplicate tasks among participants in grid computing applications as a mean to efficiently counter collusion among malicious participants. Gennaro *et al.* [22] introduce the notion of a "verifiable computation scheme" as a mean of enabling privacy-preserving outsourcing of computations to untrusted workers. Karame *et al.* [29] propose a scheme to outsource cryptographic searches to untrusted nodes without these nodes learning information about the inputs nor the outputs of the search functions. A distributed DES cracker that runs on browsers has been proposed in [14]. Provos *et al.* [33] analyze threats that originate from omnipresent scripts that run on the browsers of users.

Hashcash [17] is a proof-of-work system designed to limit email spam and denial of service attacks. Centmail [23] proposes to introduce certified microdonations as a way to combat spam emails. Bitcoin [32] is a peer-to-peer network based on digital currency; it makes use of computations to alleviate double-spending in digital payments. Horton *et al.* [26] show that Java applets in web-browsers can be used to perform covert distributed computations without the knowledge of users. They also briefly describe the notion of "Computing for Sale". Similarly, Barabasi *et al.* [19] show how to achieve parasitic computing—a form of enforced covert computing—simply by manipulating the TCP checksum field of packets.

Online Advertisements.

Recently, several works propose the design of private advertising systems. Databank [31] proposes a business "privacy" model in which the service provider pay the clients in exchange of their private information; their scheme provides additional incentives for the advertisers to protect the privacy of the clients. Adnostic [35] is an ad targeting model that creates and maintains user profiles locally. A proxy orchestrates all potential communication between the users and the service provider; the proxy is, however, assumed not to collude with the service providers.

7. CONCLUSION

In this work, we proposed and analyzed a new micropayment model based on microcomputations that can be transparently integrated within current web-browsers. In our model, a user wishing to access online content offered by a website does not need to register or pay to access the website; instead, he will accept to run some computations on behalf of the website in exchange for access to content. We analyzed the security of our proposal and we showed that—unlike the current advertisement model—our model inherently supports the privacy of users. In addition, we implemented a preliminary prototype and we evaluated the performance and usability of our proposed model. Our initial findings indicate that our scheme did not affect the browsing experience of users and is likely to be adopted by a large fraction of online users.

As a next step, we plan to extend our implementation to accommodate for the use of other computations from the large pool of existing distributed computing projects. We also plan to conduct a large scale user study to better assess the acceptability of our proposed scheme. In that respect, we believe that our proposed model also has clear benefits in supporting those ongoing distributed computing projects that are expected to impact the way we live our lives today. We therefore hope that our findings will motivate further research in this direction.

8. ACKNOWLEDGMENTS

The authors thank Victor Budilivschi for implementing and deploying the prototype. The work presented in this paper was supported (in part) by the Swiss National Science Foundation under Grant 200021-127294.

9. REFERENCES

[1] Murdoch: Web sites to Charge for Content, `http://edition.cnn.com/2009/BUSINESS/05/07/murdoch.web.content/index.html`.

[2] Forrester Research, `http://forrester.typepad.com/groundswell/2008/12/people-dont-tru.html`.

[3] The Economics of Online News, `http://www.pewinternet.org/Reports/2010/5--The-economics-of-online-news.aspx`.

[4] SETI@home, `http://setiathome.ssl.berkeley.edu/`.

[5] Distributed.Net, `http://distributed.net/`.

[6] The Great Internet Mersenne Prime Search, `http://www.mersenne.org/prime.htm`.

[7] Online Newspaper Viewership Reaches Record in 2007, `http://www.naa.org/PressCenter/`

SearchPressReleases/2008/Online-Newspaper-Viewership.aspx.

[8] Capcal – How Testing is done on the Cloud, http://www.capcal.com/.

[9] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/#pricing.

[10] How much electricity does my computer use? , http://michaelbluejay.com/electricity/computers.html.

[11] Electricity Costs in the United States , http://www.think-energy.net/electricitycosts.htm.

[12] United Devices, Inc, Company Profile, http://biz.yahoo.com/ic/105/105503.html.

[13] BOINC. http://boinc.berkeley.edu/.

[14] Browser-Based Distributed DES Cracker. http://descrack.justinsamuel.com/.

[15] Google AdSense. http://en.wikipedia.org/wiki/AdSense.

[16] Google Web Toolkit. http://code.google.com/webtoolkit.

[17] Hashcash. http://www.hashcash.org/.

[18] MaxxPI, TOP10, FLOPS. http://www.maxxpi.net/pages/result-browser/top10---flops.php.

[19] A. L. Barabasi, V. W. Freeh, H. Jeong, and J. B. Brockman. Parasitic Computing. In *Nature*, volume 412, pages 894–897, 2001.

[20] E. Clemons. Why Advertising is failing on the Internet?, 2009. http://techcrunch.com/2009/03/22/why-advertising-is-failing-on-the-internet/.

[21] P. Eckersley. How Unique Is Your Web Browser? In *Proceedings of PETS*, 2010.

[22] R. Gennaro, C. Gentry, and B. Parno. Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *Proceedings of the CRYPTO Conference*, 2010.

[23] S. Goel, J. Hofman, J. Langford, D. M. Pennock, and D. M. Reeves. Centmail: Rate Limiting via Certified Micro-Donations. In *Proceedings of CEAS*, 2009.

[24] P. Golle and I. Mironov. Uncheatable Distributed Computations. In *Proceedings of RSA*, 2001.

[25] M. T. Goodrich. Pipelined Algorithms to Detect Cheating in Long-Term Grid Computations. In *Theoretical Computer Science, LNCS, Springer*, 2008.

[26] J. Horton and J. Seberry. Covert Distributed Computing Using Java Through Web Spoofing.

[27] S. Kamkar. Evercookie – Never Forget. http://samy.pl/evercookie/.

[28] G. Karame and S. Capkun. Low-Cost Client Puzzles based on Modular Exponentiation. In *Proceedings of ESORICS*, 2010.

[29] G. Karame, S. Capkun, and U. Maurer. Privacy-Preserving Outsourcing of Crytographic Searches. In *ETH Zurich, D-INFK, Technical Report No. 662*, 2010.

[30] G. Karame, M. Strasser, and S. Capkun. Secure Remote Execution of Sequential Computations. In *Proceedings of ICICS*, 2009.

[31] R. M. Lukose and M. Lillibridge. Databank: An Economics Based Pirvacy Preserving System for Distributed Relevant Advertising and Content. In *Technical Report, HP Laboratories*, 2006.

[32] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2009.

[33] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost in the Browser: Analysis of Web-based Malware. In *Proceedings of HotBots*, 2007.

[34] D. Szajda, B. Lawson, and J. Owen. Hardening Functions for Large Scale Distributed Computations. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[35] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy Preserving Targeted Advertising. In *Network and Distributed System Security Symposium (NDSS)*, 2010.