Microcomputations as Micropayments in Web-based Services

Ghassan O. Karame, NEC Laboratories Europe, ghassan.karame@neclab.eu
Aurélien Francillon, EURECOM, aurelien.francillon@eurecom.fr
Victor Budilivschi, ETH Zurich, victor@budilivschi.com
Srdjan Čapkun, ETH Zurich, capkuns@inf.ethz.ch
Vedran Čapkun, HEC Paris, capkun@hec.fr

In this paper, we propose a new 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 service provider 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 medical research) or can contribute towards commercial computing projects. We analyze the security and privacy of our proposal and we show that it preserves the privacy of users. 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). In this respect, we implement a prototype of a system based on our model and we deploy our prototype on Amazon Mechanical Turk to evaluate its performance and usability given a large number of users. Our results show that our proposed scheme does not affect the browsing experience of users and is likely to be used by a non-trivial proportion of users. Finally, we empirically show that our scheme incurs comparable bandwidth and CPU consumption to the resource usage incurred by online advertisements featured in popular websites.

Categories and Subject Descriptors: K.4.4 [**Computers and Society**]: Electronic Commerce – payment schemes, security; K.4.1 [**Computers and Society**]: Public Policy Issues – privacy

General Terms: Design, Economics, Experimentation, Human Factors, Security

Additional Key Words and Phrases: Monetization, Distributed Computing, Privacy, Micropayments, Microcomputations

1. INTRODUCTION

In the last couple of years, the literature witnessed [Clemons 2009; Murdoch 2009] a large discussion on possible alternatives that will increase the revenues of online businesses and shareholders. Several websites are set to charge for online content in order to increase their revenues [Murdoch 2009]. Currently, eleven of the largest-selling twenty newspapers in the United States are either charging for access or are set to do so [Guardian 2013]. 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 to 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 [Murdoch 2009; OnlineNews 2010]. 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

Ghassan Karame is affiliated with NEC Laboratories Europe, 69115 Heidelberg, Germany. Aurélien Francillon is affiliated with EURECOM, Campus SophiaTech, 06410 Biot, France. This manuscript was prepared while Victor Budilivschi was affiliated with the Department of Computer Science of ETH Zurich, 8092, Zurich, Switzerland. Srdjan Čapkun is affiliated with the Department of Computer Science of ETH Zurich, 8092, Zurich, Switzerland. Vedran Capkun is affiliated with the Department of Accounting and Management Control, HEC Paris, 78351 Jouy-en-Josas, France, and a member of GREGHEC, Laboratoire CNRS UMR 2959. Vedran Capkun acknowledges financial support from the HEC Foundation.

DOI: http://dx.doi.org/10.1145/0000000.0000000

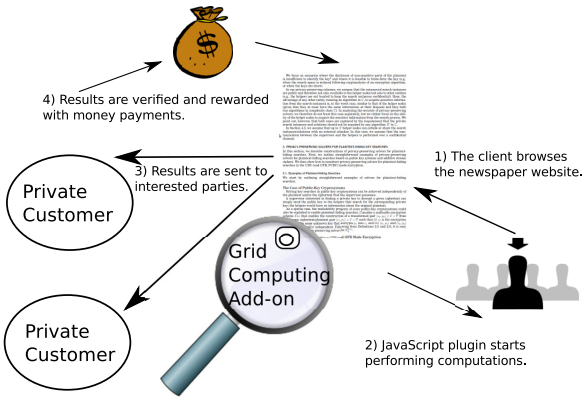


Fig. 1. Our microcomputations scheme: upon accessing the news website, the client's browser performs microcomputations. The results are then sent back to interested third parties in exchange of a payment.

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 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/she 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, the computation results 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 1999; Distributed.Net 1997; SAT 2014], etc.); alternatively, these computations could also be performed on behalf of governmental agencies, research labs and private industries. Note that, 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.

In addition to proposing the use of microcomputations as a micropayment scheme, we make the following additional contributions. We show that our microcomputations scheme naturally supports the anonymity and the privacy of users. Indeed, the microcomputations are independent by design from user profiles, which does not provide incentives for websites to profile users. Nevertheless, we show that the correctness of the results submitted by the user can be verified to a large extent; we analyze the overhead introduced by our scheme with respect to the security of our micropayment model. Finally, we implement a cross-browser prototype that allows websites to perform microcomputations in (unmodified) browsers. In our implementation, the browser fetches JavaScript, Java or Silverlight 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. We deployed our prototype implementation on Amazon Mechanical Turk [MTurk 2005] (MTurk) and we thoroughly evaluate its performance and usability by measuring, and analyzing the observations and experiences of almost 1000 subjects of Amazon MTurk. Our results show that our proposed scheme does not affect the browsing experience of users and is likely to be used by a non-trivial proportion of online users. Finally, we empirically compare the resource usage incurred by our scheme with that of the existing online advertisement model; our findings show that our scheme incurs comparable bandwidth and CPU consumption to the resource usage incurred in most online websites.

The remainder of this paper is organized as follows. In Section 2, we describe the main intuition behind our proposed scheme. In Section 3, we present our framework and we analyze its security and

privacy implications. We describe an implementation of our scheme in Section 4 and we evaluate its performance and usability by means of a large-scale study using Amazon Mechanical Turk in Section 5. In Section 6, we evaluate and compare the resource usage incurred in our scheme with that of the online advertisement model. In Section 7, we overview related work in the area and we conclude the paper in Section 8.

2. MOTIVATION

Several websites are set to charge their users in exchange for online content. However, finding the right business model that does not result in the alienation of online viewership remains a challenge. Previous studies have shown that merchants are reluctant to accept credit or debit card payments due to the cost associated with small transactions (American Banker, January 2010). In 2004, in the UK, cash represented more than 50% of all transactions below a value of 10 British pounds and more than 90% of all transactions below a value of 2 British pounds (Eastwood, 2008 – Business Insight). [Hinds 2004] estimates that a technology for handling payments lower or equal to $1 should have a cost of less or equal to 10 cents; Hinds shows that credit cards' associated cost amounts to 25-35 cents per such transaction. For this reason, many firms providing online micropayment services have been either acquired or gone out of business (e.g., [Cardline 2007]).

On the other hand, a recent survey of online readers of newspapers and magazines in [Continental Research 2009] shows that 63% of readers are not willing to pay in order to read content online, while 21% would be ready to pay a small amount to read each article. Among those, only 11% of potential readers would pay to have access to the entire publication. Moreover, the survey provides additional insights into how much consumers would be sensitive to pricing of newspaper articles. Only 3% (3%) of the participants would definitely (probably) pay 20p (British pence) for an article, with this number increasing to 20% (15%) if the price drops to 2p.

Computing as an Exchange Medium: Our proposed microcomputations scheme shares similarities with "parasitic computing" [Barabasi et al. 2001], 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—and undisguised—micropayment method. Currently, a number of businesses and digital payment schemes are emerging based on the the idea of "computing as an exchange medium" [UnitedDevices 1999; Capcal 2008; CrowdProcess 2013; Nakamoto 2009].

Bitcoin [Nakamoto 2009] is one of the most widespread computing exchange mediums. Bitcoin is a Proof-of-Work (PoW) based currency that allows users to generate digital coins by performing computations. Bitcoin is currently integrated across a number of businesses and has several exchange markets. Bitcoin requires that users are equipped with specific software/hardware in order to be able to "mine" and participate in the system. Our microcomputations scheme on the other hand is fully transparent to the users as the computations can be carried out within their browsers. Finally, we note that browsers are already executing various scripts in order to load and display online advertisements. As such, our proposed scheme emerges as a natural *extension* to the online advertisement model by replacing those scripts whose sole function is to load and render advertisements with "useful" computations. Here, we stress that our model is not intended as a replacement of online advertisements; instead, it can be used as a complement of the existing online advertisement model. Furthermore, our scheme can be used in those contexts where online advertisements are least favorable to be used; indeed, recent studies [Goldstein et al. 2013; Forrester 2008] reveal that there are a number of contexts (e.g., users reading books) in which users are annoyed by the presence of online advertisements; in those contexts, users tend to use tools to block them.

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

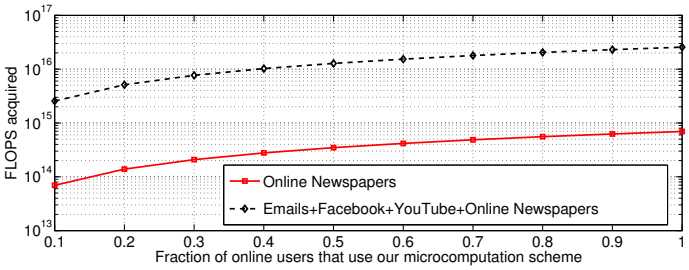


Fig. 2. Total number of FLOPS acquired in our scheme versus the number of newspaper viewers. For comparison purposes, the GIMPS project [GIMPS 1996] aggregates around $44 \cdot 10^{12}$ FLOPS.

on average 50 minutes per month browsing the news sites (these estimates were adapted from the reports in [OnlineViewership 2007] corresponding to the US online newspaper industry).

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[1]. On average, a computer is estimated to use 100 watts per hour [ElectricityUsage 2012] and the cost of a kWh is estimated to be \$0.1 [USCosts 2009]. This suggests that the lower-bound on the cost of computations can be estimated by \$0.01 per hour. In this simple example, online newspapers will be able to generate 1.5 million USD per month based on our proposed model. We point that a thorough economic analysis of our scheme is outside the scope of this paper. Nevertheless, our simple example (and estimates) indicates that our scheme can generate comparable revenues to existing models, such as the online advertisement model. For instance, 1.5 million USD of revenues correspond to each of the 60 million readers of online newspapers clicking on the banners of 1 advertisement per month in the Google AdSense model [Adsense 2010]. Note that, in practice, only a small fraction of users notices online advertisements [ClickRate 2013; Stats 2013].

Furthermore, assuming that the clients have machines that are each capable of performing an average of 10 GFLOPS (estimate acquired from [TOP10 2011]), 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 [SETI 1999] and the GIMPS project [GIMPS 1996] combined[2]. 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 in case our scheme is integrated in FaceBook, YouTube, and major Email providers[3] (shown in the dotted black plot).

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 remuneration. We assume that the outcomes of the outsourced tasks are not confidential; in Section 4.1, we show an example where the instances/solutions of the tasks can be kept secret.

[1]The cost of electricity consumption constitutes one of the main direct costs associated with microcomputations. Given the fluctuations in the market price for computations, this was one of the few workable ways to acquire a lower-bound on the cost of computations.

[2]For simplicity, we do not consider in this analysis the overhead that is incurred by the browsers when executing our microcomputation scheme, nor the corresponding overhead exhibited by the client applications running SETI and GIMPS tasks.

[3]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.

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 $\mathcal{P}$. We point out that this scheme does not require any user registration process. 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. We assume that $\mathcal{I}$ has access to several different tasks, pertaining to different customers; these tasks could be either sequential or parallelizable or even hybrid tasks (i.e., contain both sequential and parallelizable subroutines). We further assume that these tasks require expensive processing power and 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 4.1). 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 channels between $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{I}$, such as delays, congestion, jitter, etc.

Our analysis considers the presence of one or multiple colluding malicious users. These users are motivated to cheat in order to access a service without performing (all of) their assigned computations [Golle and Mironov 2001; Szajda et al. 2003]. For instance, a user might only execute 50% of its assigned computations and defect from running the rest of its tasks. Two or more malicious users might collude to increase their chances of not being detected. Moreover, we also consider the case where $\mathcal{P}$ and $\mathcal{I}$ can be malicious and are motivated to increase their benefit in the system.

3.2. Background: Transforming Distributed Tasks into Verifiable Computations

We start by outlining existing solutions that enable efficient probabilistic verification of the remote execution of *parallel* and *sequential* computations. In Section 3.3, we will leverage these solutions to ensure the security of our scheme.

Parallel computations consist of evaluating a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ for every input value $x \in \mathcal{X}$. If the computed values of $f(.)$ are independent of each other, these computations can be easily parallelized and distributed among the participants. In particular, $\mathcal{X}$ can be partitioned into smaller sub-domains and subtasks are then created by applying $f(.)$ over each sub-domain; in other words, subtask i will evaluate $f(.)$ for every input $x \in \mathcal{X}_i$ ($\mathcal{X}_i$ is a sub-domain of $\mathcal{X}$).

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 pre-computed checks—*ringers* [Golle and Mironov 2001]—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 $\mathcal{X}_i$ and computes the set $S \leftarrow \{f(r_1), f(r_2), .., f(r_n)\}$.[4] The computations performed by $\mathcal{U}$ will be accepted if and only if $\forall r_i \in \mathcal{X}_i$, $f(r_i)$ is correctly computed. Since $\mathcal{U}$ cannot distinguish the ringers from other data values in $\mathcal{X}_i$, $\mathcal{U}$ has to complete all of its assigned work, with high probability, for its computations to be accepted by $\mathcal{I}$. If a malicious $\mathcal{U}$ returns incorrect results in a subset of computations, then it is highly likely (cf. Table I) that at least one ringer task will be incorrectly computed, which would enable $\mathcal{I}$ to immediately detect this misbehavior.

On the other hand, the sequential computations that we consider in this paper consist of evaluating a function $f : \mathcal{D} \rightarrow \mathcal{D}$ for a given input value x in the domain $\mathcal{D}$. Here, f is given by $f = f_1 \circ f_2 \circ \ldots f_M$, $M \in \mathbb{N}^*$, where $\circ$ denotes function composition. Note that a sequential task can contain both sequential and non-sequential sub-functions. We further assume that the supervisor can directly extract the sub-functions $f_i(.), \forall i \in [1, M]$ from the original code (e.g., as subroutines) or, alternatively, the supervisor can use available tools that decompose a code piece into these sub-

[4] In case the computation of $f(.)$ is not small enough, $\mathcal{I}$ can proceed as outlined in [Szajda et al. 2003]; it embeds few ringers initially in a smaller input space and then uses the results reported by various users as ringers in subsequent interactions.

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

Number of ringers/redundant subtasks n	Probability of cheating P_c	Detection probability P
1	1.0	**1.000**
3	0.5	**0.875**
3	0.7	**0.973**
5	0.5	**0.968**
5	0.7	**0.997**
10	0.7	**0.999**

functions according to its control flow structure. In this case, subtasks are created by computing each sub-function $f_i(.), i \in [1, M]$ using its corresponding input $x_i \in \mathcal{D}$. We also assume that $f_i(x_i)$ can be solved in a reasonable amount of CPU time.

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 [Szajda et al. 2003; Karame et al. 2009].

We now describe a scheme—adapted from the findings in [Szajda et al. 2003; Karame et al. 2009]—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 ringers 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. Note that ringers enable the immediate detection of possible collusion among malicious users [Szajda et al. 2003; Karame et al. 2009].

Table I shows the probability of detecting malicious users by using ringers (or selective redundancy) with respect to various parameters. Here, the probability of detecting possible misbehavior by $\mathcal{U}$ is given by $P = 1 - (1 - P_c)^n$, where P_c is the fraction of incorrect results returned by a malicious user.

3.3. Our Scheme: Microcomputations as Micropayments

Our proposed framework (Figure 3) comprises a *customer server*, a *service provider*, and an *intermediary*. 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 [BOINC 2007] that hosts volunteer grid computing projects.

The intermediary $\mathcal{I}$ acquires work units from $\mathcal{C}$, and manages the secure outsourcing of computations to the users using the solutions described in Section 3.2. More specifically, $\mathcal{I}$ aggregates the work units, splits them into microcomputations, embeds indistinguishable ringers or redundancy among the computations, as described in Section 3.2, 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 mi-

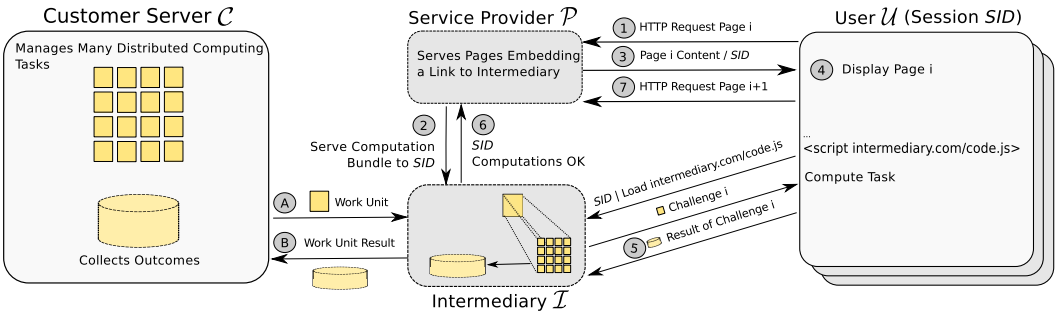


Fig. 3. Our microcomputations scheme: when a client $\mathcal{U}$ requests a service from the service provider $\mathcal{P}$, $\mathcal{P}$ informs the intermediary $\mathcal{I}$ that a new session SID has been initiated. $\mathcal{I}$ then outsources a challenge in the form of microcomputations to $\mathcal{U}$. These computations are chosen from the pool of tasks available at the customer server $\mathcal{C}$ (e.g., BOINC [BOINC 2007]). Further requests by the user are only accepted if the results of the computations are correct.

cropayments, $\mathcal{I}$ also keeps track of the microcomputations that were previously outsourced. Note that this is a common requirement in most existing distributed computing platforms.

Finally, the service provider $\mathcal{P}$ 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. 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 compute when the pages are 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 established. 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, with the exception of "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 pseudo-random 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. Note that the communication between $\mathcal{U}$ and $\mathcal{I}/\mathcal{P}$ is not performed over a secure channel (cf. Section 3.4).

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, $\mathcal{I}$ assigns different computation loads depending on the content "value" or the period during which the content is being accessed.

As described in Section 3.2, $\mathcal{I}$ verifies the integrity of the results reported by users by inserting ringers (and/or redundant computations) within the outsourced computations. Since $\mathcal{I}$ already knows the solution to the (pre-computed) ringer problems, the integrity verification of the outsourced computations can be performed very efficiently through a table lookup. $\mathcal{I}$ 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 3. We analyze the security properties of this scheme in Section 3.4. We further note that, in our scheme, ringer units (cf. Section 3.2) are also embedded within the work bundles that are outsourced from $\mathcal{C}$ to $\mathcal{I}$.

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[5]. If $\mathcal{U}$ does not explicitly provide any choice, $\mathcal{I}$ assigns subtasks given its default scheduling strategy.

One potential limitation of our scheme lies in the fact that it favors users that are equipped with fast machines. To ensure fairness among heterogeneous users, $\mathcal{I}$ can estimate their computing performance by measuring the time to complete the (default) assigned microcomputations and adjust the difficulty of subsequent bundles accordingly. However, this solution might provide incentives for users to appear slow in order to reduce their computational load [Karame and Capkun 2010]. Another natural solution to this problem—which somehow departs from our current scheme—would be to rely on a marketplace to sell "computational tokens". Here, computations can be carried out by users "offline" (e.g., using their desktop) in exchange for tokens. Users can subsequently access online content from any other devices they possess (e.g., a PDA device) upon presenting these tokens. In this respect, our microcomputations scheme supports mining for Bitcoins [Nakamoto 2009] as a specific type of outsourced computing tasks[6].

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. In fact, these servers already embed the required functionality to divide, allocate, assign, and collect subcomputations from millions of users. As we show in Section 4, our scheme can leverage existing distributed computing platforms, such as the BOINC platform [BOINC 2007], in order to implement the desired functionality of $\mathcal{I}$; this results in a minimal deployment cost for $\mathcal{I}$. Note that our proposed scheme extends the main functionality provided by BOINC by additionally checking the correctness of the computations using the ringer scheme and/or selective redundancy[7]. The additional overhead incurred by inserting these security checks within the computations on $\mathcal{I}$ is negligible; recall that ringers are pre-computed tasks that are already at the disposal of $\mathcal{I}$.

Clearly, ringers and/or selective redundancy incur additional cost on the clients. More specifically, let n denote the number of ringers and/or redundantly assigned subtasks, and let $\bar{N}$ denote the effective number of subtasks to be executed. In this case, the additional computational overhead can be estimated by $\frac{n}{n+\bar{N}}$, assuming that all subtasks share comparable computational loads. Note that in the case of parallel tasks, ringers do not incur considerable communication overhead, since in such tasks, the task code is fetched only once by the browsers, which subsequently execute the same function over different inputs (cf. Section 3.2). This is not the case for sequential tasks, in which the browsers need to constantly fetch new functions to be executed. In Section 6, we evaluate the resource consumption of our proposal and we show that the performance penalty incurred in our scheme is comparable to that of the online advertisement model.

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

[5]Here, we assume that $\mathcal{I}$ ensures that all clients bear comparable computational load—irrespective of the type of tasks that their browsers are performing.

[6]Here, we assume that $\mathcal{I}$ can obfuscate the search problem (e.g., send the Merkle hash of a subset of the transactions without indicating which transactions are included) to ensure that malicious users cannot claim the mined coins for themselves.

[7]As far as we are aware, existing distributed computing platforms do not embed any mechanisms for verifying the correctness of the outsourced tasks.

over a secure channel, a user A can perform a Man-In-The-Middle (MITM) attack and impersonate another user B in order to access content without performing the required microcomputations. 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 expected outcomes since the targeted service corresponds in our case to low-cost, commodity content. 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 $\mathcal{P}$ when they are subject to such attacks. We point out that this analysis applies to any solution that enables payment for online content.

An important requirement is to verify the results reported by users, since users can easily misreport the outcomes (e.g., by directly editing the page source code from the browser). The correctness of the solutions reported by users are ensured, with high probability, through the use of the solutions described in Section 3.2. Furthermore, 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 [Karame et al. 2012]. 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 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 considerable probability, the *integrity* and the *authenticity* of the remote microcomputations in our setting. We conclude that, in our scheme, rational users are unlikely to acquire content without "correctly" performing the required microcomputations.

On the other hand, the use of ringers equally 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 $\mathcal{I}$ and has to commit enough of its time and resources to correctly execute the microcomputations. This also suggests that our scheme enhances the resilience of $\mathcal{P}$ against Denial-of-Service (DoS) attacks; the verifiable microcomputations act as computational puzzles [Karame and Capkun 2010] in which attackers commit a considerable amount of resources before their requests are served by $\mathcal{P}$.

We point out that ringers and/or selective redundancy can only harden the tampering with outsourced computations; these measures cannot guarantee the detection of all instances of misbehavior. Note that any small erroneous computation can cause the entire result of a sequential task to be useless. As shown in Table I, the higher is the number of ringers (and therefore the bigger is the computational overhead), the higher is the assurance in the correctness of the results. In the worst case, one of the few workable mechanisms to deter a powerful adversary (e.g., which only cheats in a small subset of the computations in order to avoid detection) is to rely on full task replication [Goodrich 2008]. This is not the case for parallel tasks; here, the damage caused by powerful adversaries is minimal compared to their sequential counterpart. This is the case since the results returned by different users are independent. Similar to [Karame et al. 2009; Szajda et al. 2003], we assume that $\mathcal{I}$ dynamically adjusts the number of ringers/redundant subtasks depending on the type of tasks (i.e., sequential vs. parallel) and on the number of detected errors.

Privacy: Current advertisement platforms perform extensive user-profiling and tracking [Kamkar 2010] 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 parties to perform user-profiling. This is the case since the outsourced microcomputations are independent by design 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 described in Section 3.3, the communication between $\mathcal{P}$ and $\mathcal{I}$ solely contains a temporary session identifier; this identifier is refreshed 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[8].

4. PROTOTYPE IMPLEMENTATION

To better assess our scheme, we implemented a prototype that acts as a stand-alone server and distributes RC4 key-search tasks to browsers of the users in exchange for accessing content. Our core module was implemented in JavaScript to ensure that all browsers are able to start it without requiring additional client-side software.

4.1. Implementation Details

The core client is initialized when the browser loads/executes an *iframe* tag that is provided through the JavaScript-based module. Once this client is initialized, it checks whether the host browser supports Java or Silverlight. If Java is supported, the computations will be executed by a Java applet. If the browser does not support Java but does support Silverlight, the computations will be executed by a Silverlight applet. Finally, when the browser supports neither Java nor Silverlight, the computations will be performed natively in JavaScript. This is done to ensure that the microcomputations can execute within the browser in the most efficient way[9].

The computations that need to be executed by the browser are stored in a computing FIFO queue. When the queue contains a number of microcomputation bundles that falls below a given threshold, our client performs an XHTTP request to $\mathcal{I}$ in order to fetch more work units. Note that the XHTTP request contains the type of computations that the browser can support (i.e., Java, Silverlight or JavaScript) so that $\mathcal{I}$ is capable of responding with the supported type of computations. When microcomputations are received from $\mathcal{I}$, they are stored in the FIFO queue. This queue is continuously polled to extract the new microcomputations to be performed and start a new thread of computations.

In our implementation, the global CPU consumption is throttled to a maximum value of 50% to ensure that the browsing experience of users is unaffected even if they open multiple tabs and execute a number of microcomputations in parallel. Here, for each batch of microcomputations, the core client measures the time t_c they require to execute. The core client then interpolates the required sleep time t_s (zero CPU consumption due to microcomputations) from t_c as follows: $t_s = \frac{t_c(100-\max)}{50}$, assuming that the computations result in 100% CPU utilization within t_c. When the thread finishes executing its assigned microcomputations, the client bundles the computation results in XML format and dispatches the results back to the $\mathcal{I}$. Note that, at all times, our client checks the status of the connection with $\mathcal{I}$ by sending periodic XHTTP-based heartbeats.

The intermediary $\mathcal{I}$ was implemented using Java servlets; in our prototype, $\mathcal{I}$ interfaces with both $\mathcal{P}$ and the users' browsers and supports the functionality described in Section 3.3. Note that our implementation was based on a variant of the framework in Figure 3; here, we assume that $\mathcal{I}$ and $\mathcal{C}$ are co-located on the same server and thus our prototype implementation abstracts away the communication delays between these entities.

In our implementation, the outsourced microcomputations consisted of N sets of $\{Kr, C \oplus P\}$ where P is a given plaintext, C is its corresponding ciphertext encrypted with the RC4 stream cipher using a key k, and Kr is the assigned key-search space. Users have to check, for each of their

[8]This information leakage is not particular to our scheme and applies whenever users browse the web.

[9]In typical cases, Java code can execute at a superior speed when compared to Silverlight code; in turn Silverlight code executes faster than the relatively slow JavaScript code.

assigned search spaces, $\{\forall \widehat{k} \in K_r \,|\, C \oplus P \stackrel{?}{=} RC4(\widehat{k})\}$. If a solution is found, the key is returned to the server. As described in [Karame et al. 2011], this scheme enables a privacy-preserving search for the key, since it does not reveal the plaintext P to any entity that is involved in the search. Here, n ringers are constructed by encrypting a (pseudorandom) plaintext P_r with a key k_r to obtain the ringer's ciphertext C_r. In our implementation, each JavaScript task consists of outsourcing a key interval of size 10^5 keys while Java/Silverlight-based tasks consist of the brute-force search for the key in an interval of width 10^6 keys. Such small microcomputation bundles ensure that the browsing experience of users is not hindered when users wish to briefly explore webpages; in Section 6, we show that this solution does not incur any sensible deterioration when loading webpages.

5. EVALUATION USING AMAZON MECHANICAL TURK

We now proceed *(i)* to assess the user feedback on the acceptability and usability of our scheme and *(ii)* to evaluate the efficiency, network latency and the load that is incurred by our scheme. For that purpose, we conducted a large-scale evaluation of our prototype; we relied on Amazon Mechanical Turk [MTurk 2005] to recruit almost 1,000 human subjects that are willing to run our prototype.

5.1. Study Methodology

Amazon Mechanical Turk [MTurk 2005] (MTurk) is an online marketplace provided by Amazon that enables requesters to recruit workers in solving Human Intelligence Tasks (HITs). These tasks refer to problems that are typically designed to be solved by humans (i.e., they are difficult for a machine to solve). Examples of such tasks include answering opinion surveys, filling CAPTCHAs, etc. In MTurk, tasks are presented in webpages that are accessible to workers. Workers can choose to solve a task in exchange for a small amount of money (e.g., 0.1 US dollars).

To evaluate the performance of our prototype, we constructed a website that hosts *(i)* an article extracted from Wikipedia, that discusses the geography of a state, followed by *(ii)* a questionnaire that aims at gathering the worker's feedback on the performance of our prototype. For presentation purposes, we include the full questionnaire in Appendix A.

The article was spread across several webpages linked with "Back" and "Next" buttons connecting the various pages. By doing so, we successfully transformed Amazon MTurk service into a content provider $\mathcal{P}$ that interfaces with our prototype $\mathcal{I}$ as described in Section 3.3; when workers pressed either "Back" or the "Next" buttons, a JavaScript code was invoked to run our prototype to fetch microcomputations from $\mathcal{I}$. To ensure that JavaScript could run on the browsers of the workers, we explicitly instructed the various workers that chose to solve our HIT to enable JavaScript in their browsers if they intend to participate in our task. Users were however not explicitly informed that they will be running our microcomputations scheme. While users were browsing the article pages, our module executes our outsourced microcomputations within their browser; if the solution returned by a participant is incorrect, the session with the users is terminated and the "Back" and "Next" buttons are disabled. Among the users that chose to participate in our HIT, 25% of the users were randomly chosen as a test group that does not perform any microcomputations.

5.2. Performance Evaluation

Table II summarizes our results. Here, there were a total of 9,702 work units outsourced to 755 out of the 1,007 participants that took part in our survey (recall that only 25% of the 1,007 participants were chosen not to execute computations). Most of these work units were executed in JavaScript since the browsers of most users (almost 70%) did not support Java or Silverlight computations.

As a result, our scheme enabled the testing of approximately 2^{32} different keys by leveraging the aggregate computing power of the 755 participants. On average, computations were running for almost 2.7 seconds within the browser of each participant; these computations were constructed/verified in almost 43.5 ms by $\mathcal{I}$. This incurred a total of 22.47 MB being downloaded by the browsers of 755 participants.

Table II. Performance evaluation of our scheme. Here, the work unit size is 10^5 keys for JavaScript and 10^6 keys for Java and Silverlight computations.

	Total	JavaScript	Java	Silverlight
Total number of keys searched (in millions)	3,521	687	2,496	338
Number of keys searched per second	135,527	44,900	282,993	181,720
Number of work units returned	9,702	6,870	2,496	338
Aggregate computation time (minutes)	433	255	147	31
Network Load (MB)	22	-	-	-
Average Server CPU time (ms)	43	73	45	49
Average User CPU time (ms)	2,676	2,225	3,540	5,469

5.3. Evaluation Results

Besides logging information about the established sessions with $\mathcal{I}$, we also require users to answer a number of questions that reflect their acceptance of our scheme. Given that we are only interested in one opinion per participant, we filtered out multiple participation of the same users. As a result, a total of 1007 unique participants volunteered to answer a number of questions concerning our scheme in exchange for a total payment of $500 (we include the full questionnaire in Appendix A). As we require that all participants answered all questions in the survey, our sample was reduced to 668 observations.

In the first part of the questionnaire, we inquired about the willingness of users to view online advertisement-banners, execute scripts in their browser, and to perform online payments in exchange for accessing online content. We relied on a seven point Likert scale in order to better assess the willingness of our subjects to use the three aforementioned options. In the second part of the questionnaire, we categorized users according to their willingness to execute specific types of scripts in their browser, as well as according to the type of content that they would agree to pay for by means of computing. In the third part of the questionnaire, we inquired about the main concerns of our subjects with respect to the aforementioned three options that would allow them to access online content. Finally, we dedicated the last part of the questionnaire to categorize our subjects according to their Internet usage, education, and level of competency in the English language.

Figure 4 summarizes the background of the subjects that participated in our Amazon MTurk's survey[10]. Our results show that most of our subjects *(i)* were familiar with the basic functionality of the Internet for more than 5 years, *(ii)* have at least a bachelor degree, *(iii)* equally use smart phones, laptops and desktop machines to access the Internet, and *(iv)* understand English at a basic level. Here, we note out that almost 28% of participants rated their level in the English language as "none"; we believe that this is due to the fact that these participants do not possess good oral English skills. To remedy this, we control the variable corresponding to the users' "aptitude in the English language" in our regression model (see below).

Willingness to execute our microcomputations scheme: We now analyze the willingness of users to use our scheme in order to access online content, when compared to the reliance on online advertisements and traditional payment methods. Our results are illustrated in Table III. While only 16% of participants mostly agree to pay for online content, almost 60% agree to execute scripts on their browser and 83% agree to view advertisement banners in exchange for online content. The differences between the three options as measured at the seven point Likert scale are statistically significant at 1% according to the Wilcoxon signed-rank test.

When given a choice between viewing advertisement banners and executing our scheme, participants expressed their preference to view advertisements banners (508 observations, or 76% of the sample). Furthermore, when given a choice between executing our scheme and paying in exchange for online content, they predominantly choose executing scripts (554 observations, or 83% of the sample). Finally, when asked about the (monetary) amount they would be willing to pay in order to

[10]Similar to all surveys that make use of MTurks, we acknowledge that the investigated user-base is price-sensitive and might not exactly match our population of interest.

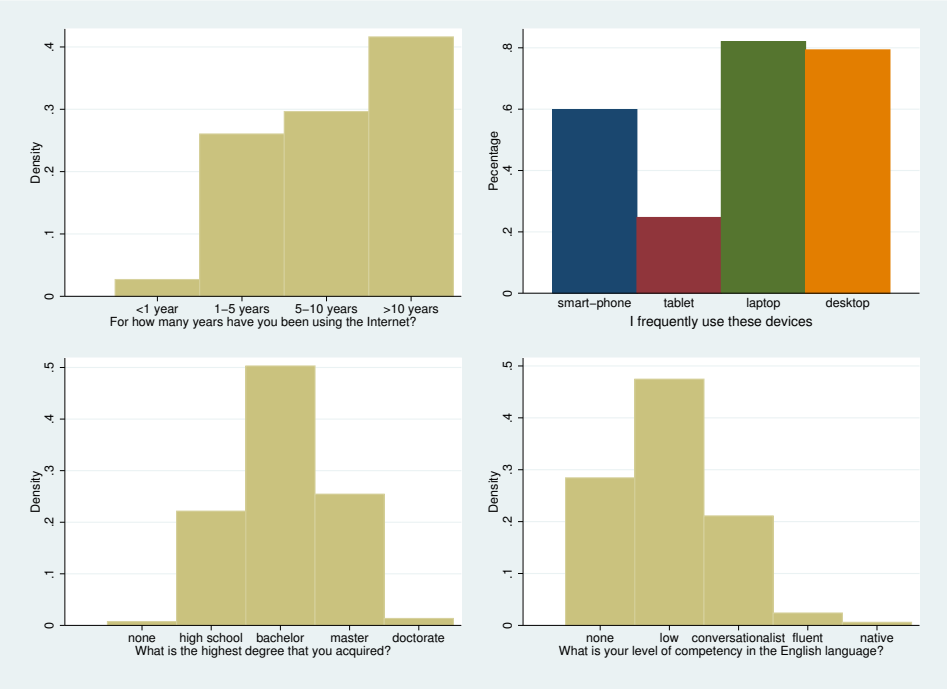


Fig. 4. Background and computing environment of the 667 subjects whose opinions were used throughout the survey. These results correspond to questions 14 through 17 in Appendix A.

Table III. Willingness of users to execute our microcomputations scheme when compared to the online advertisement model and online payments.

	Users that agree to execute		
	Online Advertisements	**Microcomputations Scheme**	**Online Payments**
Totally Agree	200	101	16
Agree	222	135	44
Somewhat Agree	132	163	46
Neutral	45	94	57
Somewhat Disagree	29	81	87
Disagree	20	55	167
Totally Disagree	20	39	251
	668	668	668
Mostly Agree	82.93 %	59.73 %	15.87 %
Mostly Disagree	10.33 %	26.20 %	75.60 %
Neutral	6.74 %	14.07 %	8.53 %

avoid executing our scheme, 392 participants (59% of the sample) were not willing to pay anything to avoid executing microcomputations. These (untabulated) results are consistent with our results in Table III. This indicates that our microcomputations scheme does not alienate users, as most users do not find any need to avoid using it. Furthermore, as shown in Figure 5, our results indicate that the nature of the microcomputations plays an important role with respect to the willingness of users to use our scheme. We also point out that most participants expressed their willingness to pay by means of computations in exchange for music items and articles.

Microcomputations vs. online advertisements: We now analyze the reticence to execute scripts relative to adopting the online advertisement model. First, we estimate an *ordered probit regression model* with the willingness to execute our scheme as the dependent variable (q1.3). The variable

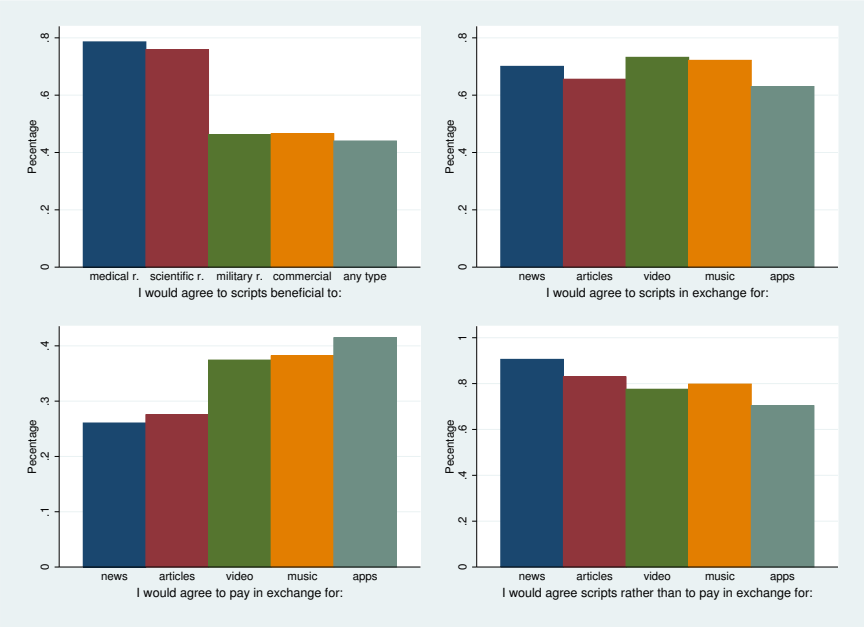


Fig. 5. Preferences of users with respect to adopting our microcomputations scheme.

ranges from one to seven, with one corresponding to the "totally agree" category (i.e., the strongest willingness to execute our scheme), and seven corresponding to the "totally disagree" category (i.e., the weakest willingness to adopt our scheme). Second, we estimate a probit regression model with question 2 (preference between microcomputations and online advertisements) as a dependent variable; here, we denote by one the case where a participant prefers advertisement banners, and two when the participant prefers executing our microcomputations scheme.[11]

As explanatory variables, we use *(i)* question 5 from the questionnaire, that reflects the participants' knowledge whether their browser already executes certain types of scripts (coded on a seven point Likert scale from fully agree to fully disagree), *(ii)* questions 6.1-6.5 from the questionnaire that inquire about the type of microcomputations that the users are willing to execute in their browsers, and *(iii)* questions 11.1-11.4 from the questionnaire, that inquire about the users' concerns to adopt our scheme. Here, we control for question 5 to account for the case where users are not familiar with the basic functionality of online advertisements (i.e., scripts load advertisements). On the other hand, we control for questions 6 and 11 to alleviate concerns that users might exhibit with respect to the type of computations that they are willing to execute (i.e., useful vs. useless computations), or to deal with privacy concerns that might arise when executing our microcomputations scheme. As additional control variables, we use the experience of users with using the Internet (question 14), the usage of devices to access the Internet (questions 15.1-15.4), the educational background of users (question 16), and the level of English of the users (question 17). Table IV shows the descriptive statistics for our independent variables.

Table V depicts the estimated regression models. The coefficient associated with the "q5 – scripts" variable is positive and statistically significant in the first regression model, and negative and statistically significant in the second regression model. This indicates that those participants who are aware that scripts already being run on their browsers (e.g., to render online advertisements) are more likely to agree to executing our scheme in exchange for online content. It further indicates

[11] We also performed tests using the ordered logit regression model; the obtained results were similar to those of the ordered probit model—which is why we omit the presentation of these results in the paper.

Table IV. Descriptive statistics extracted from the independent variables used in our regression models.

Variable	N	Mean	Std.	Min	1st Quartile	Median	3rd Quartile	Max
q5 - Scripts	668	2.35	1.15	1	2	2	3	7
q6.1 - Medical research	668	2.56	1.44	1	2	2	3	7
q6.2 - Scientific research	668	2.68	1.47	1	2	2	3	7
q6.3 - Military research	668	3.83	1.86	1	2	4	5	7
q6.4 - Commercial projects	668	3.79	1.87	1	2	4	5	7
q6.5 - Any type of computations	668	3.81	1.79	1	2	4	5	7
q11.1 - Browsing experience	668	3.04	1.51	1	2	3	4	7
q11.2 - Computations	668	2.94	1.45	1	2	3	4	7
q11.3 - Computer viruses	668	2.31	1.46	1	1	2	3	7
q11.4 - Privacy	668	2.18	1.34	1	1	2	3	7
q14 - Years of Internet	668	3.10	0.88	1	2	3	4	4
q15.1 - Smart-phone	668	3.34	2.49	1	1	2	6	7
q15.2 - Tablet	668	5.25	2.15	1	4	6	7	7
q15.3 - Laptop computer	668	2.12	1.76	1	1	1	3	7
q15.4 - Desktop computer	668	2.12	1.82	1	1	1	3	7
q16 - Degree	668	3.04	0.75	1	3	3	4	5
q17 - English level	668	1.99	0.80	1	1	2	2	5

Table V. Determinants of willingness to execute microcomputations. "***" refer to observations that are statistically significant at 1% according the Wilcoxon signed-rank test. Similarly, "**" refer to observations that are statistically significant at 5%. We depict the corresponding Zstat in between parentheses.

	q1.3	q2
q5 - Scripts	0.153*** (3.747)	-0.161*** (2.859)
q6.1 - Medical research	0.059 (1.071)	-0.089 (1.225)
q6.2 - Scientific research	0.135** (2.402)	0.007 (0.099)
q6.3 - Military research	0.047 (1.414)	-0.045 (1.156)
q6.4 - Commercial projects	0.088** (2.454)	0.020 (0.449)
q6.5 - Any type of computations	0.104*** (2.933)	-0.082*** (1.895)
q11.1 - Browsing experience	0.013 (0.343)	0.070 (1.581)
q11.2 - Computations	-0.102** (2.389)	-0.008 (0.156)
q11.3 - Computer viruses	-0.094** (2.222)	-0.043 (0.838)
q11.4 - Privacy	-0.020 (0.442)	0.070 (1.427)
q14 - Years of Internet	0.002** (0.037)	0.003 (0.034)
q15.1 - Smart-phone	0.007 (0.360)	0.024 (1.043)
q15.2 - Tablet	0.011 (0.492)	0.045 (1.570)
q15.3 - Laptop computer	0.028 (1.089)	-0.013 (0.371)
q15.4 - Desktop computer	-0.017 (0.698)	0.015 (0.461)
q16 - Degree	-0.025 (0.451)	-0.043 (0.564)
q17 - English level	0.075 (1.314)	-0.095 (1.149)
Constant		-0.012 (0.024)
Number of observations	668	668
Log-pseudolikelihood	-1114.0262	-345.96014
Wald chi2	245.8	39.69
Prob>chi2	0.000	0.001
Pseudo R2	0.101	0.059

that the same participants will be more likely to adopt our scheme over viewing online advertisement banners. Given our results, we also find no discernible difference in preferences for the type of microcomputations to be executed in browsers based on privacy concerns. Note also that the coefficients for q6.1-q6.5 are sometimes statistically significant in the first regression model; this suggests that users are more likely to execute microcomputations if they are used for scientific research and commercial projects; however, we find no evidence that the type of the computations is important for the choice between microcomputations and online advertisements.

6. EVALUATING RESOURCE CONSUMPTION

We now evaluate the resources incurred on the client side by our microcomputations scheme. To do so, we measure: *(i)* the page load time (PLT), *(ii)* the memory consumption, *(iii)* the CPU usage and

Table VI. Websites used throughout our experiments in Section 6.

Category	Websites
News Sites	news.yahoo.com, cnn, huffingtonpost, bbc, weather.com, nytimes, news.google, my.yahoo.com, reddit.com, foxnews
Computer Sites	facebook.com, youtube.com, yahoo.com, wikipedia.org, google.com, twitter, linkedin, search.yahoo.com, alexa.com, msn.com
Games Sites	battle, pch, ign, miniclip, pogo, 888.com, gamespot, steampowered, gamefaqs, games.yahoo
Shopping Sites	amazon.com, ebay, netflix, amazon.co.uk, walmart, ikea, target, groupon, bestbuy, multiply

(iv) the network transfer delays incurred in our microcomputations scheme and we compare them with those incurred when users browse high-rated websites that rely on online advertisements. *Since the browsing experience in the current online advertisement model is accepted by most users, our study provides empirical evidence on the browsing experience of the users that adopt our scheme.*

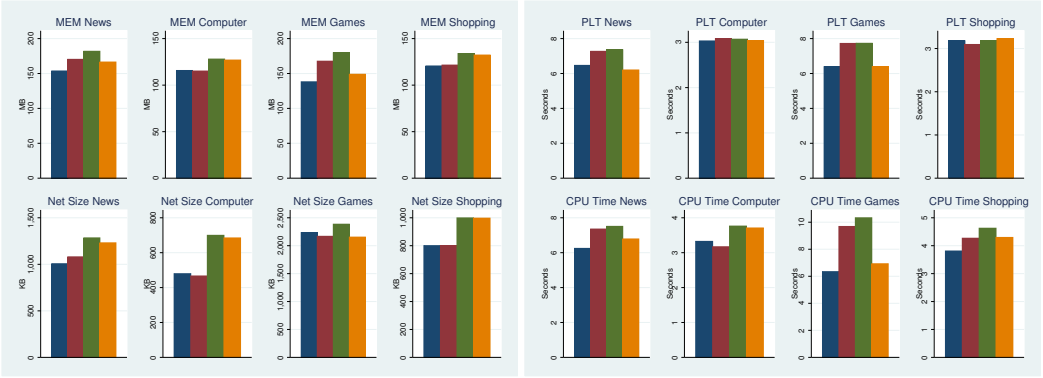
6.1. Evaluation Setup

To conduct our evaluation, we relied on a testbed for measuring the performance of page loading in both our scheme, and in the online advertisement model. This testbed (derived from [Cui and Biersack 2013]) consists of a computer running Firefox 12 on a Fedora Release 15. The execution of Firefox is instrumented with a custom plug-in that monitors resource consumption while connecting to a number of websites. We measure resource consumption while browsing online pages in four different setups: *(i)* when the page is not modified from the source, *(ii)* by removing advertisements from the page, and *(iii)* by replacing the advertisements within the webpage with our microcomputations scheme, and *(iv)* by combining the use of online advertisements with our scheme. To disable the effect of advertisements, we used an advertisement filtering proxy (*Squid* proxy [Squid 1996] together with the AdZapper plug-in [Simpson] and the *squid_redirect* script). Note that the reliance on a distinct proxy server, instead of using a browser plug-in like AdBlock plug-in [AdBlock 2004], makes the measurements independent of the resources used by the filtering itself. We also configured the Squid server not to perform caching, not to disturb our measurements.

On the other hand, to evaluate the resources required by our microcomputations scheme, we injected a script in the loaded page using user side scripting with Greasemonkey [Greasemonkey 2012]. This allows us to accurately simulate the use of microcomputations by the evaluated site. This script connects to our remote microcomputations server as described in the previous section; more specifically, we relied on the same setup adopted in Section 4.1 (see Table II for further details on the performance of our scheme).

We performed our experiments in a real world setting, on live websites, using an unmodified Firefox browser whose cache is cleared before each measurement. More specifically, we relied throughout our experiments on a number of popular websites, chosen among the top Alexia [Alexia 2012] websites in the following categories: News, Computers, Shopping and Games (cf. Table VI).

To minimize the impact of noise in our measurements, each data point that we collected was the median of 30 independent measurements. More specifically, for each website, we first start Firefox on a new profile, and load the homepage of the website. We then monitor the *page loading time* , *network usage*, CPU consumption, and the browser's *memory usage* (*RES* memory) every 100ms. When the website is fully loaded, we stop the browser and we record the median of the memory usage and its total CPU usage (*user* and *system* time). We repeat this experiment for a total of 30 times for every website. Here, we assume that the browsers have to execute a single microcomputation bundle correctly in order to be able to load the page. Recall that in our case each JavaScript bundle consists of outsourcing a key interval of size 10^5 keys while Java/Silverlight-based tasks consist of the brute-force search for the key in an interval of width 10^6 keys. Note that subsequent microcomputation bundles can only improve the page load performance since the browsers will not download new task code, but only new inputs to the search problem.



(a) Memory usage (in MB) and network load (in KB). (b) Page Load Time (PLT) and CPU usage (in seconds).

Fig. 6. Comparison of performance metrics. Here, we consider the median of 30 independent measurements performed over four categories of websites. We refer the reader to Table VI for details on the websites that we considered in each category. Blue bars refer to the case when the website is loaded without advertisements or microcomputations. Magenta bars refer to the loading performance of the website as is (i.e., with advertisements). Green bars refer to the case where the current website is augmented with the use of our microcomputations scheme (i.e., advertisements and microcomputations). Finally, yellow bars refer to the case when the website is loaded without advertisements but with microcomputations.

6.2. Evaluation Results

Figure 6 summarizes the performance overhead incurred by online advertisements when compared to the overhead of microcomputations. In Section 4.1, we evaluate the overall performance of our scheme when executed by the 1000 Amazon MTurk subjects; in this section, we go one step deeper and we analyze the performance witnessed on the individual machines of clients when fetching and executing a single microcomputation bundle at page load time.

Our results indicate that the impact of our microcomputations scheme on the performance of the page loading time, network load, CPU and memory usage is minimal and matches the overhead incurred by exiting online advertisements. This is especially true for sites that host a large number of online advertisements such as those websites classified within the news and computer categories. In this case, the resources used by our microcomputations scheme are comparable to those needed by online advertisements. However, for the categories that typically feature few advertisements (e.g., shopping, games), the overhead incurred by our microcomputations scheme is more sensible.

Our results further show that even if our microcomputations scheme is used alongside with the current online advertisement model, then the performance penalty measured with respect the page loading time and CPU usage can be, to a large extent, tolerated.

7. RELATED WORK

In this section, we overview related work in the areas of secure distributed computing, and privacy-preserving online advertising.

A comprehensive survey in the area of result-checking and self-correcting programs can be found in [Wasserman and Blum 1997]. Golle *et al.* [Golle and Stubblebine 2001] propose a security framework for commercial distributed computations that relies on selective redundancy. Goodrich *et al.* [Goodrich 2008] discuss mechanisms to duplicate tasks among participants in grid computing applications as a means to efficiently counter collusion among malicious participants.

Centmail [Goel et al. 2009] proposes to introduce certified microdonations as a way to combat spam emails. A distributed DES cracker that runs on browsers has been proposed in [Cracker 2008]. Provos *et al.* [Provos et al. 2007] analyze threats that originate from omnipresent scripts that run on the browsers of users. Horton *et al.* [Horton and Seberry 1998] show that Java applets in web-browsers can be used to perform covert distributed computations without the knowledge of users.

Databank [Lukose and Lillibridge 2006] proposes a model in which a provider pays its clients in exchange of their private information. Adnostic [Toubiana et al. 2010] is an ad-targeting model which maintains user profiles locally. Privad [Guha et al. 2011] enables private targeted advertisement by storing the profiles of users on their computers instead of being hosted by a third party.

8. CONCLUDING REMARKS

In this work, we proposed, analyzed, and evaluated a new micropayment model based on micro-computations that can be transparently executed by users within current web-browsers.

The main benefits of our scheme are as follows. *(i)* Our scheme does not considerably affect the browsing experience of users and is likely to be used by a non-trivial proportion of online users; 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., computing projects relating to HIV, cancer, clean energy). *(ii)* Our scheme is mostly suited to those settings where online advertisements are not well-received by users [Goldstein et al. 2013] (e.g., reading online newspapers, books). 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 or when streaming a YouTube video. Indeed, in such settings, our scheme does not distract users since the computations are carried out in the background. *(iii)* Our scheme can be further used in conjunction with online advertisements in order to increase the revenues of websites; our results show that even in this case, the browsing experience of users is not affected.

Similar to existing distributed computing platforms, our scheme finds direct applicability in the outsourcing of parallel tasks, since such tasks *(i)* can be easily distributed among users with minimal communication overhead, and *(ii)* can be verified with little computational overhead when compared to their sequential counterpart. In this respect, our findings suggest that that the type of the outsourced computations plays little effect in the adoption of our proposed scheme.

Acknowledgments

The authors thank the anonymous reviewers for the helpful comments. The authors also thank Heng Cui for the help in performing the resource consumption evaluation of our scheme.

REFERENCES

AdBlock 2004. The Economics of Online News. AdBlock Plus, Available from http://en.wikipedia.org/wiki/Adblock_Plus.

Adsense 2010. Google AdSense. Available from http://en.wikipedia.org/wiki/AdSense.

Alexia 2012. Alexia Top Sites. Available from: http://www.alexa.com/topsites.

BARABASI, A. L., FREEH, V. W., JEONG, H., AND BROCKMAN, J. B. 2001. Parasitic Computing. In *Nature*. Vol. 412.

BOINC 2007. BOINC. Available from http://boinc.berkeley.edu/.

Capcal 2008. Capcal – How Testing is done on the Cloud, Available from http://www.capcal.com/.

Cardline 2007. Micropayments Sitll Not Profitable Online. Available from: http://www.highbeam.com/doc/1G1-164436911.html.

CLEMONS, E. 2009. Why Advertising is failing on the Internet? Available from: http://techcrunch.com/2009/03/22/why-advertising-is-failing-on-the-internet/.

ClickRate 2013. Display Advertising Click Through Rates, Available from http://www.smartinsights.com/internet-advertising/internet-advertising-analytics/display-advertising-clickthrough-rates/.

Continental Research 2009. Micropayments may be the answer for publishers. Available from: http://www.bdrc-continental.com/EasysiteWeb/getresource.axd?AssetID=2373&type=full&servicetype=Inline.

Cracker 2008. Browser-Based Distributed DES Cracker. Available from http://descrack.justinsamuel.com/.

CrowdProcess 2013. CrowdProcess, Available from http://crowdprocess.com//.

CUI, H. AND BIERSACK, E. 2013. Troubleshooting Slow Webpage Downloads. In *IEEE INFOCOM TMA Workshop*. Torino, Italy, 405–410.

Distributed.Net 1997. Distributed.Net, Available from http://distributed.net/.

ElectricityUsage 2012. How much electricity does my computer use? Available from: http://michaelbluejay.com/electricity/computers.html.

Forrester 2008. Forrester Research. Available from http://forrester.typepad.com/groundswell/2008/12/people-dont-tru.html.

GIMPS 1996. The Great Internet Mersenne Prime Search, Available from http://www.mersenne.org/prime.htm.

GOEL, S., HOFMAN, J., LANGFORD, J., PENNOCK, D. M., AND REEVES, D. M. 2009. Centmail: Rate Limiting via Certified Micro-Donations. In *Proceedings of CEAS*.

GOLDSTEIN, D. G., MCAFEE, R. P., AND SURI, S. 2013. The cost of annoying ads. In *Proceedings of the 22nd international conference on World Wide Web*. WWW '13. 459–470.

GOLLE, P. AND MIRONOV, I. 2001. Uncheatable Distributed Computations. In *Proceedings of RSA*. 425–440.

GOLLE, P. AND STUBBLEBINE, S. 2001. Secure Distributed Computing in a Commercial Environment. In *Proceedings of the International Conference on Financial Cryptography*. 289–304.

GOODRICH, M. T. 2008. Pipelined Algorithms to Detect Cheating in Long-Term Grid Computations. In *Theoretical Computer Science, LNCS, Springer*. Vol. 408. 199–207.

Greasemonkey 2012. Greasemonkey Firefox add-on. Available from: http://www.greasespot.net/.

Guardian 2013. Why US newspaper publishers favour paywalls, Available from http://www.guardian.co.uk/media/greenslade/2013/jan/01/paywalls-us-press-publishing.

GUHA, S., CHENG, B., AND FRANCIS, P. 2011. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

HINDS, D. 2004. Micropayments- a technology with a promising but uncertain future. In *Communications of the ACM Vol. 47*. Number 5. 44.

HORTON, J. AND SEBERRY, J. 1998. Covert Distributed Computing Using Java Through Web Spoofing. In *Proceedings of ACISP*.

KAMKAR, S. 2010. Evercookie – Never Forget. http://samy.pl/evercookie/.

KARAME, G., ANDROULAKI, E., AND CAPKUN, S. 2012. Double-spending Fast Payments in Bitcoin. In *Proceedings of ACM CCS*. 906–917.

KARAME, G. AND CAPKUN, S. 2010. Low-Cost Client Puzzles based on Modular Exponentiation. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 679–697.

KARAME, G., CAPKUN, S., AND MAURER, U. 2011. Privacy-Preserving Outsourcing of Brute-Force Key Searches. In *Proceedings of ACM CCSW*. 101–112.

KARAME, G., STRASSER, M., AND CAPKUN, S. 2009. Secure Remote Execution of Sequential Computations. In *Proceedings of ICICS*. 181–197.

LUKOSE, R. M. AND LILLIBRIDGE, M. 2006. Databank: An Economics Based Privacy Preserving System for Distributed Relevant Advertising and Content. In *Technical Report, HP Laboratories*.

MTurk 2005. Amazon Mechanical Turk. Available from: https://www.mturk.com/mturk/welcome/.

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

NAKAMOTO, S. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System.

OnlineNews 2010. The Economics of Online News. Available from http://www.pewinternet.org/Reports/2010/5--The-economics-of-online-news.aspx.

OnlineViewership 2007. Online Newspaper Viewership Reaches Record in 2007, Available from http://www.naa.org/PressCenter/SearchPressReleases/2008/Online-Newspaper-Viewership.aspx.

PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, N. 2007. The Ghost in the Browser: Analysis of Web-based Malware. In *Proceedings of HotBots*. 4–4.

SAT 2014. Wonderings of a SAT geek. Available from: http://www.msoos.org/2013/09/minisat-in-your-browser/.

SETI 1999. SETI at home. Available from: http://setiathome.ssl.berkeley.edu/.

SIMPSON, C. Ad zapping with squid. Available at http://adzapper.sourceforge.net/.

Squid 1996. Squid Proxy. Available from http://www.squid-cache.org/.

Stats 2013. 10 Horrifying Stats About Display Advertising, Available from http://blog.hubspot.com/horrifying-display-advertising-stats.

SZAJDA, D., LAWSON, B., AND OWEN, J. 2003. Hardening Functions for Large Scale Distributed Computations. In *Proceedings of the IEEE Symposium on Security and Privacy*. 216–.

TOP10 2011. How many GFLOPS does your processor have? Available from http://www.overclock.net/t/947312/how-many-gflops-does-your-processor-have.

TOUBIANA, V., NARAYANAN, A., BONEH, D., NISSENBAUM, H., AND BAROCAS, S. 2010. Adnostic: Privacy Preserving Targeted Advertising. In *Network and Distributed System Security Symposium (NDSS)*.

UnitedDevices 1999. United Devices, Inc, Company Profile, Available from http://biz.yahoo.com/ic/105/105503.html.

USCosts 2009. Electricity Costs in the United States. Available from: http://www.think-energy.net/electricitycosts.htm.

WASSERMAN, H. AND BLUM, M. 1997. Software Reliability via Runtime Result-Checking. In *Journal of the ACM*. Vol. 44. 826–849.

A. QUESTIONNAIRE

Option A: Online Advertisements, **Option B:** Microcomputations scheme, **Option C:** Electronic payment schemes.		
Questions	Sub-Choices	Scale
q1.1	I agree to use Option A	Totally Agree (1) - Totally Disagree (7)
q1.2	I agree at least to consider using Option A	Totally Agree (1) - Totally Disagree (7)
q1.3	I agree to use Option B	Totally Agree (1) - Totally Disagree (7)
q1.4	I agree at least to consider using Option B	Totally Agree (1) - Totally Disagree (7)
q1.5	I agree to use Option C	Totally Agree (1) - Totally Disagree (7)
q1.6	I agree at least to consider using Option C	Totally Agree (1) - Totally Disagree (7)
q2: If I am given a choice of either using Option A or Option B		Option A (1), Option B (2)
q3.1: If I am given a choice of either using Option B or Option C		Option B (1), Option C (2)
q4.1: Which maximum subscription price per month would you be willing to pay for the content to avoid Option B?		Nothing (1), below \$0.1 (2), < \$2 (3), < \$5 (4), below \$10 (5), < \$100 (6)
q5.1: Do you agree that, when you visit most websites, your browser already executes various scripts (e.g., Flash, JavaScript)?		Totally Agree (1) - Totally Disagree (7)
I would agree to choose Option B, if the scripts that are executed in my browser are beneficial to:		
q6.1	Medical research	Very Likely (1) - Very Unlikely (7)
q6.2	Scientific research	Very Likely (1) - Very Unlikely (7)
q6.3	Military research	Very Likely (1) - Very Unlikely (7)
q6.4	Commercial Projects	Very Likely (1) - Very Unlikely (7)
q6.5	Any type of Computations	Very Likely (1) - Very Unlikely (7)
I would generally agree to use Option B in exchange for:		
q7.1	Reading online news/articles	Very Likely (1) - Very Unlikely (7)
q7.2	Downloading online articles	Very Likely (1) - Very Unlikely (7)
q7.3	Watching/downloading online video	Very Likely (1) - Very Unlikely (7)
q7.4	Listening/downloading online music	Very Likely (1) - Very Unlikely (7)
q7.5	Downloading applications for the mobile phone	Very Likely (1) - Very Unlikely (7)
I would generally agree to use Option C in exchange for: (**q8.1-q8.5 are similar to q7.1-q7.5, respectively**)		
If I am given a choice of either using Option B or Option C to acquire/access the following:		
q9.1	Reading online news/articles	Option B (1) - Option C (2)
q9.2	Downloading online articles	Option B (1) - Option C (2)
q9.3	Watching/downloading online video	Option B (1) - Option C (2)
q9.4	Listening/downloading online music	Option B (1) - Option C (2)
q9.5	Downloading applications for the mobile phone	Option B (1) - Option C (2)
In current websites that require monetary online payments:		
q10.1	I dislike the cumbersome registration in current websites	Fully Agree (1) - Fully Disagree (7)
q10.2	I do not trust the website security to submit payments there	Fully Agree (1) - Fully Disagree (7)
q10.3	I do not like to give my credit card credentials	Fully Agree (1) - Fully Disagree (7)
q10.4	I am concerned about my privacy	Fully Agree (1) - Fully Disagree (7)
If I agree to run scripts in exchange of content:		
q11.1	I will be most worried about my browsing experience	Fully Agree (1) - Fully Disagree (7)
q11.2	I will be most worried about what computations I am running	Fully Agree (1) - Fully Disagree (7)
q11.3	I will be most worried about computer viruses	Fully Agree (1) - Fully Disagree (7)
q11.4	I will be worried about my privacy	Fully Agree (1) - Fully Disagree (7)
General Questions		
q12.1	In general, I click on online advertisements	Fully Agree (1) - Fully Disagree (7)
q12.2	In general, I do not like online advertisements	Fully Agree (1) - Fully Disagree (7)
q12.3	In general, I block online advertisements	Fully Agree (1) - Fully Disagree (7)
About the article		
q13.1	I find the content of the article interesting	Fully Agree (1) - Fully Disagree (7)
q13.2	I dislike having "Next" buttons	Fully Agree (1) - Fully Disagree (7)
q13.3	I felt something weird with my browser	Fully Agree (1) - Fully Disagree (7)
q13.4	Scrolling of the page was slower that usual	Fully Agree (1) - Fully Disagree (7)
q13.5	I experienced problems when opening other browser tabs/windows	Fully Agree (1) - Fully Disagree (7)
q13.6	I experienced slowdown of my computer	Fully Agree (1) - Fully Disagree (7)
q13.7	Other	
q14.1: For how many years have you been using the Internet?		less than a year (1), between 1 and 5 years (2), between 6 and 10 years (3), more than 10 years (4)
How often do you use the following?		
q15.1	Smart phone	Very frequently (1) - Never (7)
q15.2	Tablet	Very frequently (1) - Never (7)
q15.3	Laptop computer	Very frequently (1) - Never (7)
q15.4	Desktop computer	Very frequently (1) - Never (7)
q16.1: What is the highest degree that you acquired?		None (1), High school (2), Bachelor (3), Master (4), Doctorate (5)
q17.1: What is your level of competency in the English language?		None (1), Low (2), Conversationalist (3), Fluent (4), Native (5)