Fair Payments for Outsourced Computations

Bogdan Carbunar
ARTC, Motorola
USA
carbunar@motorola.com

Mahesh Tripunitara
ECE, Univ. of Waterloo
Canada
tripunit@uwaterloo.ca

***Abstract*—Initiated by volunteer computing efforts, the computation outsourcing problem can become a compelling application for networked set-top-boxes and mobile devices. In this paper we extend such environments with the ability to provide secure payments in exchange for outsourced CPU cycles. Previous contributions in wired networks have almost exclusively tackled only one side of the problem – offering incentives for volunteer participation and preventing worker laziness. This makes sense in static environments where reputable outsourcers have little to gain from incorrectly rewarding honest participation. However, this assumption is no longer valid in ad hoc environments, where unique identities are difficult to provide and anyone can outsource computations. In this paper we propose a solution that simultaneously ensures correct remuneration for jobs completed on time and prevents worker laziness. Our solution relies on an offline bank to generate and redeem payments; the bank is oblivious to interactions between outsourcers and workers. In particular, the bank is not involved in job computation or verification. Our experiments show that the solution is efficient: the bank can perform hundreds of payment transactions per second and the overheads imposed on outsourcers and workers are negligible.**

I. Introduction

While extensively studied for their information dissemination and media hosting capabilities, ad hoc networks have received considerably less attention for their computing outsourcing potential. The reasons are obvious: mobile devices are usually less capable than PCs and their reliance on battery power makes them an unlikely choice for outsourcing their CPU. In this work we consider however a special type of networked devices: set-top-boxes (STBs) and mobile phones. IP based STBs are not only always plugged in and networked but also idle for long intervals. Moreover, many cellphones are left on while charging and at night incur minimal networking fees. Most STBs and smartphones have processors in the 300MHz-1GHz range [6], [18]. This, coupled with the pervasiveness of such devices [1] make them an excellent platform for executing outsourced computations.

Specifically, we refer to volunteer computing projects [5], [2], [15], [22], [4], [17]. Informally, such projects assume an outsourcer that has a job to compute and multiple workers that are willing to spend their CPU cycles to run parts of the job. We consider the following computation model. A *job* takes as inputs a function $f : I \to R$, an input domain $D \subset I$ and a value $y \in R$ and requires the evaluation of f for all values in D. An *outsourcer*, O, seeks one or all $x \in D$ values for which $f(x) = y$. That is, O seeks to invert f for a particular y, and the approach he adopts is brute-force. O partitions the domain I and allocates each partition, along with the function f and value y, to a different job. O posts jobs to a predefined location. Any *worker*, W, can access the job postings, pull the next available job, execute it locally and return the results. In our work we model the case of a single partition (job), and one worker, W. W seeks payment for its work. The problem is that O and W do not trust each other. From the standpoint of O, O does not trust that W will indeed fully do the work he undertakes. For example, W may evaluate f only on a portion of D and seek full payment. From the standpoint of W, even if he dutifully does the work, he does not trust that O will pay him after he has expended the effort.

While solutions exist that address the lack of trust that O has in W (see Section VI), the lack of trust of W in O is not addressed – W is required to fully trust O. This is however an important problem, since in our model the outsourcer can be any participant (user with a PC or mobile device).

In this paper we propose a solution that addresses both issues of trust. We rely on a trusted offline third party, a *bank* B, that acts strictly as a financial institution in the transaction between O and W. B issues payment tokens, which O embeds in jobs. W is able to retrieve a payment token if and only if it completes a job. We achieve this by first using secret sharing to compute shares of the payment token. All the shares are needed to redeem the payment. We then employ the ringer concept proposed by Golle and Mironov [13]. However, instead of generating a single ringer set, O generates a ringer set for each payment share and uses a function of the ringer set to "hide" the share. Once W receives the job and "hidden" payment shares, W and O run a verification protocol, where all but one share are revealed and the correctness of the last share is proved in zero knowledge. While W cannot reveal the last payment share without solving the last ringer set, it is unable to distinguish the revealed payment share even after computing the entire job. This effectively prevents W from performing incomplete computations. Only the bank can retrieve the last share and combine it with the other shares to obtain the payment token.

Note that our solution has two additional features. First, the bank does not have to be online during job outsourcing operations, but only during payment withdrawal and deposit operations. Second, the bank is not required to act as an

[1]There are hundreds of millions of cellphones and tens of millions of IP STBs in use today.

escrow agent. The advantages provided by the former feature are obvious in the case of ad hoc networks. The latter feature is essential in ensuring the bank's transparency from the job computation process. To understand why this is the case, consider a simple solution where the bank holds O's payment in escrow until W completes the job. Then, the bank verifies the job's completeness and in case of a successful verification deposits the payment in the W's account. Such a solution would require the bank not only to be aware of job details but also to be involved in computing and verifying jobs. The liability issues and additional overhead implied, make this an unreasonable proposition for a safe financial institution.

We have implemented our solution using Java and the BouncyCastle security provider, for two computing problems: finding the pre-image of a cryptographic (SHA-1) hash, and the abc conjecture [1]. Our results show that we impose reasonable overheads on the bank (100 payment transactions per second) as well as on the outsourcer and worker (ranging from tens of ms to 1s for various job types and system parameters).

The remainder of the paper is organized as follows. In the next section we describe the ringers concept. In Section III, we present our solution and the intuition behind it. In Section IV, we present the security properties of our solution and in Section V we empirically validate our solution. In Section VI we discuss related work and conclude with Section VII.

II. Ringers - An Overview

The solution from Golle and Mironov [13] (see Section 2.3 there) that we extend is called *ringers*. In this section, we discuss ringers and how they are used to solve the problem of the trust in W. O needs to be able to establish that W does indeed perform all the computations that were outsourced to him. A ringer is a sample of the form $\langle x, f(x) \rangle$. There are two kinds of ringers, *true ringers* and *bogus ringers*. A true ringer is such that $x \in D$, and a bogus ringer is such that $x \notin D$. The solution has the following steps.

Job Generation O chooses an integer $2m$, the total number of ringers. He picks a random integer $t \in [m+1, \ldots, 2m]$ to be the number of true ringers, and $2m - t$ to be the number of bogus ringers. The distribution of t in $[m+1, \ldots, 2m]$ is $d(t) = 2^{2m-t-1}$. O computes $f(x)$ for every true and bogus ringer x. These post-images are included in the screener S that is sent to W. The screener is used by W to decide what he must store for transmission back to O once he is done with the job. O uses this information to infer whether W did indeed do the entire job, and pays W only if he infers that he did. We clarify how S works in the next step.

Computation and Payment The screener S takes as input a pair $\langle x, f(x) \rangle$ and tests whether $f(x) \in \{y, y_1, \ldots, y_{2m}\}$ where y is the post-image whose pre-image O seeks, and each y_j is the post-image of a true or bogus ringer. If $f(x)$ is indeed in the set, then S outputs x; otherwise it outputs the empty string. W computes f for each element in D, processes each through S, collects all the outputs of S and sends them to O to receive its payment. If W honestly does its work, then what it sends O at the end is the set of true ringers, and possibly the special pre-image for which O is looking. The ringers ensure that W does its entire work. The bogus ringers make it more difficult for W to stop prematurely and still make O believe that it did its entire work.

III. Our Solution

We discuss our solution in a framework similar to the one presented in prior work [13]. The three principals in the solution are the outsourcer O, the bank B, and the worker W. O prepares jobs he wants done in the manner we discuss in Section I, B issues and redeems payment tokens and W does the job. An optimal solution should also satisfy the following constraints. First, it is desirable that only B is involved in the payment generation step, as it plays the role of a bank. Second, only O should be involved in binding the payment to the job.

For our initial description, we adopt the more abstract mechanisms as used in the random oracle model [8]. We provide concrete instantiations in Section V. $G\colon \{0,1\}^* \to \{0,1\}^\infty$ is a random generator and $H\colon \{0,1\}^* \to \{0,1\}^h$ is a random hash function.

Setup: The bank, B, has the following.

- A trapdoor permutation, $\langle p, p^{-1}, d \rangle$ that is secure from non-uniform polynomial time [12] adversaries. The function p is public, and p^{-1} is private to B.
- A generator, $g \in \Gamma$ for a finite cyclic group, Γ of order q where q is prime. All of g, Γ and q are public. All exponentiations of g are done modulo q; we omit the "$mod\ q$" qualification in our writing.
- A random keyed hash $H_K\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^h$ based on H with the key K of length k. The key K is secret to B. We assume that K is chosen with care and H_K is constructed securely based on H. In other words, if H is a random hash function, then so is H_K.

Payment generation: O requests B for a payment token of a certain value. B generates $\langle P, \sigma \rangle$ and sends it to O.

- $P = H_K(M)$ is a *payment token*. M contains the value of the payment token (e.g., "\$ 10") and any other information B may choose to put in it.
- $\sigma = p^{-1}\left(H\left(g^P\right)\right)$. σ is B's signature on g^P.

Job generation: O first generates an instance of a job that consists of the function $f : I \to R$, special image y and sub-domain $D \subset I$ to be explored. O then generates r sets of ringers, $J = \{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$. Each $\mathcal{R}_i = \{H(f(t_{i,1})), \ldots, H(f(t_{i,i_t})), H(f(b_{i,1})), \ldots, H(f(b_{i,i_b}))\}$. Each $H(f(t_{i,j}))$ is a true ringer, and each $H(f(b_{i,j}))$ is a bogus ringer. Each $t_{i,j} \in D$ and each $b_{i,j} \in I - D$. O needs to prove those facts to W when challenged in the verification step below.

Binding payment to job: O's objective is that W is able to extract the payment token only if he does the job. O does three things to bind P to J.

- O splits P into r shares $P_1, \ldots, P_r$ such that $P_1 \times \ldots \times P_r = P\ mod\ q-1$. Recall that r is the number of sets of ringers from the Job Generation step above. O also generates $\mathcal{G} = \{g^{P_1}, \ldots, g^{P_r}\}$.

- O obfuscates each P_i with B's trapdoor permutation. That is, O computes $\mathcal{E}_{B,i} = p(P_i)$.
- O binds each $\mathcal{E}_{B,i}$ to the true ringers in $\mathcal{R}_i$ as follows. O computes $\mathcal{K}_i = G(t_{i,1} \mid\mid \ldots \mid\mid t_{i,i_t})$. We assume a globally agreed-upon ordering for the $t_{i,j}$'s, for example, lexicographic. Without loss of generality, we assume that $t_{i,1}, \ldots, t_{i,i_t}$ is that ordering. O then computes $P_{i,\mathcal{K}} = \mathcal{K}_i \oplus \mathcal{E}_{B,i}$. Let $\mathcal{P} = \{P_{i,\mathcal{K}}, \ldots, P_{r,\mathcal{K}}\}$.

Job Transmission: O sends $\langle J, \mathcal{P}, \mathcal{G}, \sigma, M\rangle$ to W. Recall from the Payment Generation step above that σ is B's signature on g^P. W verifies that the cleartext M is acceptable to him.

Verification: W runs a protocol with O to gain confidence that if he completes the job, then he will be able to retrieve the payment token. To achieve this, W chooses $r-1$ indexes out of r as its challenge. Let i be an index chosen by W. O reveals to W all the $f(t_{i,j})$ and $f(b_{i,l})$ from $\mathcal{R}_i$, the corresponding $t_{i,j}$ and $b_{i,l}$, and P_i. W now does the following for each i in its chosen set of indexes.

- Verifies that $g^{P_i} \in \mathcal{G}$. And for i, j chosen by W such that $i \neq j$, verifies that $g^{P_i} \neq g^{P_j}$.
- Verifies that each $t_{i,j} \in D$, each $b_{i,l} \in I - D$, and each $H(f(t_{i,j}))$ and $H(f(b_{i,l}))$ is in $\mathcal{R}_i$.
- Computes $\widehat{\mathcal{K}_i} = G\left(t_{i,1} \mid\mid \ldots \mid\mid t_{i,\widehat{i_t}}\right)$, where $\widehat{i_t}$ is the number of true ringer pre-images revealed for index i by O and $t_{i,1}, \ldots t_{i,\widehat{i_t}}$ are the lexicographically sorted true ringer pre-images.
- Verifies that $p(P_i) = \widehat{\mathcal{K}_i} \oplus \mathcal{P}_{i,\mathcal{K}}$.

In addition, let $i_1, \ldots, i_{r-1}$ be the indexes W chose, and i_r the remaining index for which the ringer pre-images and P_{i_r} have not been disclosed to W by O. W verifies that:

$$H\left(\left(g^{P_{i_r}}\right)^{\left(P_{i_1} \times \ldots \times P_{i_{r-1}}\right)}\right) = p(\sigma)$$

Computation: At the end of the Verification step, W is left with one set of ringers. Without loss of generality, we assume that this is $\mathcal{R}_r$. An honest W does the following:

- Computes f on each value, $v_i \in D$.
- Checks whether $H(f(v_i)) \in \mathcal{R}_r$. If yes, it adds v_i to a set $\mathcal{V}$.

Payment extraction: To extract what it believes to be $\mathcal{E}_{B,r} = p(P_r)$, W does the following. (Recall that we assume that r is the index that was not chosen by W during the verification step.)

- Computes $\widehat{\mathcal{K}_r} = G(v_1 \mid\mid \ldots \mid\mid v_{i_v})$, where $v_1, \ldots, v_{i_v} \in \mathcal{V}$ are sorted lexicographically.
- Computes $\widehat{\mathcal{E}_{B,r}} = \widehat{\mathcal{K}_r} \oplus P_{r,\mathcal{K}}$.
- Submits $\left\langle P_1, \ldots, P_{r-1}, \widehat{\mathcal{E}_{B,r}}\right\rangle$ and M to B for reimbursement.

Payment redemption: For successful redemption, B checks that M is valid, and $P_1 \times \ldots \times P_{r-1} \times p^{-1}\left(\widehat{\mathcal{E}_{B,r}}\right) = H_K(M)$. If p is homomorphic under multiplication, then W can instead submit M and what it thinks is $p(P)$. If the check verifies, B credits W with the corresponding amount. Otherwise, it rejects the payment.

A. Intuition

We present proofs of security properties we desire in Section IV. Here, we discuss the intuition behind our construction in the previous section. The intent behind splitting the payment token P into r shares is to be able to embed each in a set of ringers. The intent behind having r ringers is to run a "cut-and-choose" type protocol in the Verification step – W chooses exactly 1 out of the r sets of ringers on which to base his computation; the remaining ones are revealed to him by O. The intent behind obfuscating a payment share P_i as $\mathcal{E}_{B,i} = p(P_i)$ is so that when W recovers a payment share, it is unrecognizable to him. Therefore, unless he completes the entire computation (or all the ringers in the set are true ringers and he discovers all of them), he cannot be sure that there are no more true ringers to be discovered. B, however, can easily recover P_i from $\mathcal{E}_{B,i}$.

The intent behind encrypting the obfuscated payment share as $K_i \oplus \mathcal{E}_{B,i}$ is to make the recovery of $\mathcal{E}_{B,i}$ directly dependent on discovering all the true ringers. The generator g and its associated operations are used so W can be confident that O is not cheating. That is, the g^{P_i} values enable W to verify that all the shares are indeed linked to a value σ signed by B. W trusts B's signature σ, and bases its trust in O on whether it is able to verify that signature before starting the computation step.

B. Issues and Resolutions

We now discuss some issues with our solution and resolutions for them.

***O*'s special values.** Recall that one of the reasons O may outsource the computation is that he has special values $Y = \{y_1, \ldots, y_s\} \subset R$ for which he seeks pre-images in D. In our solution, the values in Y do not appear. Our resolution to this relies on the "lazy but honest" assumption about W. The tuple sent to W by O in the Job Transmission step can include Y. W is then trusted to return any pre-images he finds for values in Y to O at the end.

Double spending. We investigate the possibility that the payment token P is "double spent." There are various versions of this problem: (i) O may redeem P with B himself before an honest W has had the opportunity to complete the job. (ii) O may embed the same P in jobs to two different workers, W_1 and W_2. (iii) W may attempt to get reimbursed for the same P more than once. Our proposed resolution is for B to generate an additional tuple, $T = \langle \mu, O, W, t_o, t_e, s\rangle$ as part of the Payment Generation step. O also must communicate this T to W during the Job Transmission step. T contains a unique serial number, μ, the identities of W and O, the time that P is issued, t_o, the time that P expires, t_e, and a signature s of B over all these fields. Only W may redeem P during the time interval $[t_o, t_e]$. O is allowed to redeem P after time t_e if it has not been redeemed already. W can check that he has a valid and acceptable T before commencing the Computation step. B retains μ forever to prevent double-spending of P. The bank is still offline, as the worker can redeem a recovered payment anytime before t_e.

B as an oracle. W may use B as an oracle to guess the key $\mathcal{K}_r$ without completing the Computation step. A simple approach W may adopt is to guess that he has discovered all the true ringers at some point in the Computation step, construct $\widehat{\mathcal{K}_r}$ as his guess for the key based on the true ringers he has discovered so far, and check whether B honors his request for redemption based on $\widehat{\mathcal{K}_r}$. A straightforward resolution to this is to adopt the approach of Golle and Mironov [13] – B allows W only one attempt at reimbursement.

Collisions of H. It is possible that a collision of H results in an incorrect inference on the part of W about a true ringer. Specifically, during the Computation step, it is possible that W discovers a double $u = \langle v, f(v)\rangle$, where $v \in D$, such that $H(f(v)) \in \mathcal{R}_r$ and $f(v)$ was never intended by O to be part of $\mathcal{R}_r$. The $f(v)$ may correspond to either a true or a false ringer in $\mathcal{R}_r$. Either way, W will incorporate v into his list of true ringer pre-images in computing the key $\mathcal{K}_r$, which will yield the incorrect key. Note that the probability of this event can be decreased if H is applied on u instead of only $f(v)$. The probability of collision becomes then about $2^{-h/2}$ where h is the number of bits in the output of H. We do not propose any resolution to this issue, other than to suggest that W must be aware of the risk of this happening, even if O is honest.

Pre-images of bogus ringers It is possible that a bogus ringer, $f(b_{i,j})$, has a pre-image, $d \in D$. This would cause W to incorporate d into his construction of the key $\mathcal{K}_r$, which would yield an incorrect $\mathcal{E}_{B,r}$ and cause his request for redemption of the payment token to be rejected by B. It may be the case that both O and W are honest, and W is denied his payment. Certainly, the probability of this event can be reduced using the idea mentioned in the previous paragraph, that is, applying H over both the pre-image and the image, $u = \langle v, f(v)\rangle$, instead of only the image, $f(v)$.

If O is honest, he can calculate the number of redemption attempts W must be allowed so he has a minimum probability of successful redemption given, for example, a probability that a bogus ringer has a pre-image in D. O can then communicate this to B so B can incorporate this number in his redemption policy. However, O can be lazy in that he can choose not to communicate anything to B for the maximum number of redemption attempts to allow for W (or simply communicate the 1). Consequently, our only "resolution" to this issue is that W must be aware that even if he does the computation honestly, there is a probability that his redemption attempt will fail. If p is the probability that a bogus ringer has a pre-image in D, then the probability that W's legitimate redemption attempt fails is $1-(1-p)^{i_b}$ where i_b is the number of bogus ringers.

It may appear then, that O has an incentive to maximize the number of bogus ringers in the hope that W's legitimate redemption attempt fails. However, as Theorem 2 in Section IV shows, O must balance this with the risk that W may successfully redeem P without completing the computation.

IV. Security Properties

In this section, we present and prove the security properties that our solution from Section III possesses. We do not consider the extensions we discuss in Section III-B in our proofs, and only consider the original solution from Section III. We conjecture that the extensions do not affect our security properties. We consider two classes of security properties: protection from a dishonest outsourcer, and protection from a dishonest worker.

A. Protection from a dishonest O

The objective of a dishonest O is to get W to complete the job, but not be able to redeem P. We first express our assertion in the following theorem in terms of W's success probability after the Computation step.

Theorem 1. *An honest W successfully redeems the payment token with probability $1 - 1/r$, where r is the number of sets of ringers.*

Proof: (Intuition) Assume that W is honest and completes the computation, and yet is unable to redeem the payment. This means that the verification by B fails. Recall from Section III that W submits to B: the "payment message" M, and $\left\langle P_1, \ldots, P_{r-1}, \widehat{\mathcal{E}_{B,r}}\right\rangle$. B verifies that M is valid, and $P_1 \times \ldots \times P_{r-1} \times p^{-1}\left(\widehat{\mathcal{E}_{B,r}}\right) = H_K(M)$. If B's verification fails, then this means that $\widehat{\mathcal{E}_{B,r}} \neq \mathcal{E}_{B,r}$. (The other components are verified by W during the Job Transmission and Verification steps prior.) Recall that $\widehat{\mathcal{E}_{B,r}} = \widehat{\mathcal{K}_r} \oplus P_{r,\mathcal{K}}$ where $\widehat{\mathcal{K}_r} = G(v_1 || \ldots v_{i_v})$ and $P_{r,\mathcal{K}}$ is the encrypted and obfuscated payment share that corresponds to the r^{th} set of ringers.

One case is that $\widehat{\mathcal{K}_r} \neq \mathcal{K}_r$, then this means that W was unable to reconstruct the key from the true ringers he found. We assume that the bogus ringers have no pre-images in D. This can only be because O cheated with the construction. The other case is that $P_{r,\mathcal{K}}$ is invalid. Recall that $P_{r,\mathcal{K}} = \mathcal{K}_r \oplus p(P_r)$. In this case as well, this can be only because O used either an invalid $\mathcal{K}_r$, or applied p incorrectly, or used an invalid P_r. All of the above attempts by O to cheat would have been detected by W in the Verification step, unless O cheated on exactly one set of ringers, and W happened to not choose that set for examination in the Verification step. Consequently, O succeeds with a probability of only $1/r$. ■

From this, it is clear that any attempts by O to cheat are discovered at the Verification step. Consequently, we can make the following assertion about W's success probability before he invests in the Computation step.

Corollary 1. *Successful completion of the Verification step implies that W has a success probability of $1 - 1/r$ in redemption once he completes the Computation step.*

B. Protection from a dishonest W

In this section, we assume that O is honest. W may attempt to reconstruct a legitimate $\mathcal{E}_{B,r} = p(P_r)$ without completing the job.

Theorem 2. *If W is able to reconstruct $\mathcal{E}_{B,r} = p(P_r)$ without finishing the job with probability p, a Golle-Mironov worker can successfully stop early with probability at least $p - \epsilon$, where ϵ is the probability that W correlates $p(P_r)$ and g^{P_r}.*

Proof: (Intuition) We build the proof by reducing Golle-Mironov's solution to our solution. Let us assume that there exists a PPT algorithm $\mathcal{A}$ which when run by a worker can reconstruct $\mathcal{E}_{B,r} = p(P_r)$ without computing the entire job. We then build a PPT algorithm $\mathcal{B}$ that allows a Golle-Mironov worker to successfully stop early its computation. $\mathcal{B}$ works in the following manner. First, it interacts with the (Golle-Mironov) outsourcer O and receives a job consisting of the function f, domain D and set of ringers $f(x_1), .., f(x_{2m})$. $\mathcal{B}$ then runs the Payment Generation protocol with bank B to obtain a valid payment P which it splits into r shares, $P_1, .., P_r$. $\mathcal{B}$ then starts to compute the job received from the outsourcer.

Let us consider the step (l) of this computation where $\mathcal{B}$ has processed l input values from the domain D and has discovered k ringers, where $m < k < l < 2m$. Let $x_1, .., x_k$ be the (Golle-Mironov style) ringer pre-images discovered. At this step, $\mathcal{B}$ runs the Job Generation and Binding Payment to Job protocols to compute r ringer sets for $\mathcal{A}$ as follows. For $r - 1$ of the ringer sets, it computes each ringer set using inputs from D which it has already processed but which are not Golle-Mironov style ringers. It uses these $r-1$ ringer sets to obfuscate $r-1$ payment shares, $P_1, .., P_{r-1}$. $\mathcal{B}$ computes the last ringer set to be $H(f(x_1)), .., H(f(x_{2m}))$. It also computes key $K_r = G(x_1||...||x_k)$ and uses it to obfuscate the last payment share P_r. $\mathcal{B}$ then runs the Job Transmission protocol with $\mathcal{A}$ in the following manner. At each step it sends the values previously computed to $\mathcal{A}$ and they engage in the Verification protocol. If the $r - 1$ indexes challenged by $\mathcal{A}$ contain r, $\mathcal{B}$, stops and starts over. If the $r - 1$ indexes do not contain r, $\mathcal{B}$ follows the Verification protocol until the end. $\mathcal{B}$ then interacts with $\mathcal{A}$ as if it were the bank B. That is, if $\mathcal{A}$ returns $p(P_r)$, the last obfuscated payment share, $\mathcal{B}$ stops and returns $x_1, .., x_k$ to the outsourcer. Otherwise, $\mathcal{B}$ proceeds with the step $(l + 1)$ of its computation and repeats the above procedure.

We need to prove first that $\mathcal{B}$ terminates in expected polynomial time. This is true, since each interaction with O, B and $\mathcal{A}$ is expected polynomial time. $\mathcal{B}$ runs only up to $|D|$ computation steps and for each step it runs the Verification protocol an expected r times (before $\mathcal{A}$ chooses the right job to perform).

Then, it is straightforward to see that $\mathcal{A}$ succeeds only if $\mathcal{A}$ recognizes the end of the job before completing it or if $\mathcal{A}$ can correlate $p(P_r)$ and g^{P_r}. By hypothesis, the latter case occurs with probability upper bounded by ϵ. Also, the former case corresponds to the case where $\mathcal{B}$ succeeds. Thus, $Pr[\mathcal{B}\ succeeds] \geq Pr[\mathcal{A}\ succeeds] - \epsilon$. ■

V. Empirical Evaluation

In this section we investigate the costs imposed by our solution on the operation of all system participants. We first consider the bank, which is the system bottleneck, involved both in payment generation and redemption transactions. The bank may be unwilling to implement our solution if the overhead of such transactions is too high. Due to large waiting times and system unavailability, significant transaction costs can negatively impact the number of bank customers. Thus, in the following we place special emphasis on these costs, by evaluating the bank's ability to handle multiple transactions per second.

Second, we are interested in the overhead imposed by our solution on the operation of outsourcers and workers. In particular, we need to compare payment related overheads to the costs of evaluating actual jobs. Outsourcers will be unwilling to use our solution if the associated overheads are similar to the costs of actual jobs. Similarly, workers would expect the payment verification and extraction costs to be much smaller than the job costs.

We have implemented each component of our solution and have tested each on a Linux box with a dual core Intel Pentium 4 that clocks at 3.2GHz and has 2GB of RAM. The code was written in Java and runs on Sun's 1.5.0 Java Runtime Environment (JRE). We used the BouncyCastle security provider [3] to implement the required cryptographic primitives. We have implemented two job types, SHA-1 hash inversion and abc-conjecture jobs. We separately describe the implementation details of each job type.

The SHA-1 inversion job. A job is a triple $\langle SHA - 1, D, y\rangle$. The job consists of applying SHA-1 to each input value from a given domain D, a subset of the space of all input strings of a given length. The result of the job consists of all (if any) input values $x \in D$ for which $SHA - 1(x) = y$. During the job generation step, the outsourcer generates a ringer as $H(SHA - 1(x))$, where $x \in D$ for true ringers and $x \in \ddot{D}$ for bogus ringers. To recover the payment, the worker needs to find all true ringer preimages from the remaining share.

The abc-conjecture job. The abc conjecture is stated as follows. Given three integers a, b and c, where $gcd(a, b) = 1$ and $c = a+b$, define the quality of the triple, $quality(a, b, c) = \log c / \log rad(abc)$, where $rad(x)$ is the product of the distinct prime factors of x. The abc conjecture states then that the number of (a, b, c) triples for which $quality(a, b, c) > 1 + \epsilon$ is finite, for any $\epsilon > 0$. An abc-conjecture job consists of the triple $\langle quality, D_a \times D_b, 1 + \epsilon\rangle$. That is, for each $a \in D_a$ and $b \in D_b$ such that $gcd(a, b) = 1$ compute $quality(a, b, a + b)$. The result of the job consists of all a and b values for which $quality(a, b, a + b) > 1 + \epsilon$. Before outsourcing the job, the outsourcer generates ringers of the form $H(quality(a, b, a + b))$, for randomly chosen $a \in D_a, b \in D_b$ for true ringers and $a \in \ddot{D}_a, b \in \ddot{D}_b$ for bogus ringers. Note that the $quality(a, b, a+b)$ value for ringers does not need to be larger than $1 + \epsilon$.

The focus of our implementation is not on solving the hash inversion or the abc-conjecture problems. Instead, our goal is to study the computation costs imposed by our payment solution on the system participants, in the context of these computations.

Instantiations. We now discuss concrete instantiations for the abstractions used in our solution. We chose SHA-1 to implement the function H and also for implementing the HMAC function H_K. The bank's secret key K was instantiated using a SecretKey object, using a secret key generator provided by BouncyCastle [3]. We used RSA for the bank's trapdoor permutation (p, p^{-1}, d). Let N denote the bit size of the RSA modulus. The generator g and the group order q of group Γ were computed as ElGamal parameters. Let $|q|$ denote the bit size of Γ's order. N and $|q|$ are parameters and their values are specified in our experiments. We used a SecureRandom instance based on a SHA-1 pseudo-random generator to implement the random generator G.

In the following, all results presented are an average over 100 independent experiments.

A. Bank Transaction Costs

In the following we investigate the costs of each procedure involving the bank.

Setup: We start by evaluating the time to perform the initial setup operation. The time to generate 1024 bit RSA parameters is 444ms, the time to generate 256 bit ElGamal parameters is 1943ms and the time to instantiate the HMAC and initialize it with a fresh secret key is 50ms. The total setup time for these parameters is then on average less than 2.5 seconds. Note that this operation needs to be performed only once, at startup.

Payment Generation and Redemption: The approximate cost of payment generation and redemption transactions is given by Equations 1 and 2. $T_{RSA\_sig}(N)$ and $T_{RSA\_dec}(N)$ are the RSA signature and private key decryption costs for the corresponding RSA modulus N, $T_{exp}(|q|)$ and $T_{mul}(|q|)$ are the costs of modular exponentiation and multiplication in Γ and T_H is the hashing cost. Compared to the other components, the hashing cost is very small and can be safely ignored. In one experiment we recorded the evolution of payment transactions costs as a function of N, which we range from 512 to 2048 bits. We set $|q|$ to be 256 bits and the number of ringer sets, r, to be 2. Each value reported is an average over 100 different experiments.

$$T_{PGen} = T_{RSA\_sig}(N) + T_{exp}(|q|) + 2T_H \qquad (1)$$

$$T_{PRed} = T_{RSA\_dec}(N) + rT_{mul}(|q|) + T_H \qquad (2)$$

In one experiment we recorded the evolution of payment transaction costs as a function of N, ranging from 512 to 2048 bits. We set $|q|$ to be 256 bits and the number of ringer sets, r, to be 2. Figure 1(a) shows our results. As expected from Equations 1 and 2, payment redemption transactions are more efficient than payment generations. For instance, for small N values (512 bits), the bank can redeem almost 500 payments per second and generate 350 payments per second. This is because the time to sign and encrypt are very similar, however, a modular exponentiation is more expensive than 2 multiplications.

For large values of N the costs of the two transactions become almost equal. For instance, for $N = 2048$, both transactions take approximately 66ms. This is because for large N values the RSA signature and private key encryption costs becomes the dominant factor. Note that when $N = 1024$ both transactions take approximately 10ms, allowing a single PC to generate and redeem 100 payments per second. In the following experiments we set N to be 1024 bits.

In a second experiment we study the bank's cost dependency on $|q|$, ranging from 64 to 512 bits (N is set to 1024 bits). Figure 1(b) shows our findings. The payment redemption cost is almost constant, as it depends almost entirely on the RSA modulus size – even for large $|q|$ values the modular multiplication cost is very low. However, the modular exponentiation cost for large $|q|$ values becomes significant. (see Equation 1). This determines a decrease in the number of payment generation transactions performed per second from around 100, for smaller $|q|$ values, to around 70 for $|q| = 512$. In the following experiments we set $|q|$ to be 256, sufficient according to current specifications [16].

Payment size and network delays: The size of a payment token generated by the bank and sent to an outsourcer is $|N| + h$, where h is the hash function output bit size. For the values considered ($|N| = 1024$ bits, h=160 bits for SHA-1), the payment token can fit a single packet (MTU=1500 bytes). The size of the payment structure sent by a worker to the bank during the payment redemption step is $(r - 1)|q| + N$. When $r = 100$, the traffic generated by the payment redemption step is 3 packets.

B. Outsourcer Overhead

We study now the costs incurred in our solution by a participant outsourcing a job. As mentioned before we consider two types of jobs, hash inversions and abc-conjecture jobs. In particular, we are interested in the costs imposed by the generation of ringers as well as the costs to split a payment token, obfuscate the shares and blind each share with a ringer set. Where applicable, we compare these costs against the baseline costs of an outsourcer implementing the Golle-Mironov [13] solution.

Ringer Generation: The cost of generating the ringer sets in our solution is approximately given by equation 3, where cnt is the number of ringers (true and bogus) in a ringer set and T_f is the average cost of computing the function f on one input value from domain D.

$$T_{Ring} = r \times cnt \times (T_H + T_f) \qquad (3)$$

We implement our solution using up to 100 ringer sets, where the total number of ringers in each set is 10. We limit

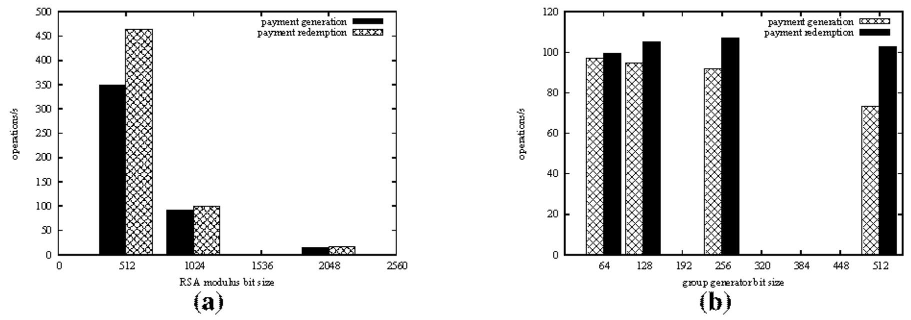

Fig. 1. Bank cost for payment generation and redemption transactions. (a) Performance when the RSA modulus size increases from 512 to 2048. For $N = 1024$ even a simple PC allows the bank to perform 100 of each transaction type per second. (b) Increasing Γ's group order from 64 to 512 bits does not influence the payment redemption cost, however, it decreases the number of payment tokens that can be generated in a second to around 70. We choose $|q|$ then to be 256, which is much larger than the currently recommended values. The bank can then still generate 100 payments per second. The effect of $|q|$ on costs. For $|q| = 256$, the bank can generate and deposit 100 payments per second.

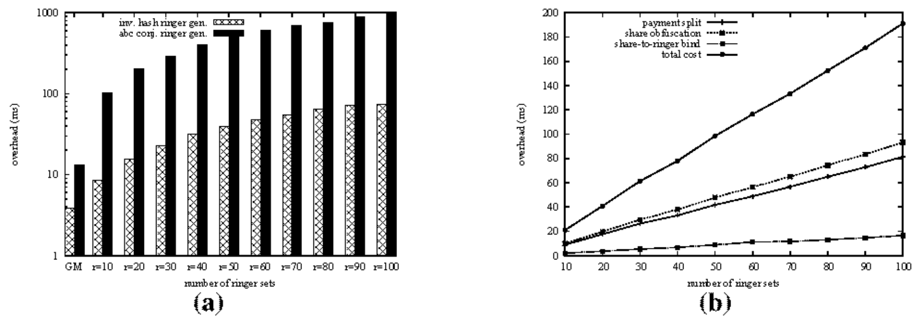

Fig. 2. Outsourcer costs as a function of the number of ringer sets employed. (a) Comparison of costs incurred by the ringer generation step of our solution and Golle-Mironov, for hash inversion and abc-conjecture jobs. Even for 100 ringer sets our solution imposes less then 1s overheads on the outsourcer. (b) Overhead of binding ringer sets to payment shares. For 100 ringer sets the total cost is less than 200ms and is independent of the job type.

the job computation time by considering only domains where the largest possible element is 10^6. Figure 2(a) shows our findings, where each bar is an average over 100 independent experiments (jobs). The first (gray) bar in each pair is the cost (in milliseconds) for hash inversion and the second (black) bar is the cost for abc-conjecture jobs. The first two bars in the graph are the Golle-Mironov costs (r=1). The remaining pairs are for our solution when r ranges from 10 to 100. The y axis is shown in logarithmic scale.

As indicated by Equation 3, the ringer generation overhead is dependent on the number of ringers (sets). By generating a single ringer set, Golle-Mironov is more efficient, requiring only 4ms for generating hash inversion ringers and 13ms for abc-conjecture ringers. As expected, the cost of our solution grows linearly with the number of ringer sets. The job type is the other factor in the ringer generation cost, since generating a ringer effectively means computing the job on a randomly chosen input point. The ringer generation cost for the abc-conjecture is higher than for hash inversion. The hash inversion ringer generation cost is practically independent of the bit length of the input value (for reasonably sized inputs). This is certainly not the case for abc-conjecture ringers, which require factoring numbers from the input domain. Note however that even for 100 ringer sets of 10 ringers each, the outsourcer's ringer generation cost for the abc-conjecture (of input values a, b and c upper bounded by 1000000) is under 1s. For the same parameters but for the hash inversion problem, this cost is significantly smaller, around 75ms. The outsourcer needs to perform this task only once per job, thus we believe this cost to be very reasonable.

Binding payment to job: Once the ringer sets are computed the outsourcer needs to split a payment token and use the ringer sets to blind each payment share. The cost of this task is independent of the job type and has three components, T_{split} (see Equation 4), $T_{obf} = rT_{RSA\_enc}(N)$ and $T_{bind} = rT_{xor}(N)$, where T_{RSA_enc} is the RSA public key encryption cost, $T_{xor}(N)$ is the time to perform an Xor operation on N bit input values and T_{inv} is the modular inversion cost.

$$T_{split} = T_{inv}(|q|) + rT_{mul}(|q|) + rT_{exp}(|q|) \qquad (4)$$

We measure the time taken by each component when the number of ringer sets r increases from 10 to 100 and the number of ringers (true and bogus) in each set is 10. Figure 2(b) shows our results, averaged over 100 independent experiments. Since the last step, of binding the ringer sets to payment shares, only consists of Xor operations, it imposes the smallest overhead, less than 17ms even for $r = 100$. The split and obfuscation steps impose similar costs, with the obfuscation step being slightly more expensive. This is because these steps are dominated by the cost of r RSA encryptions and modular exponentiations in Γ. However, even for $r = 100$, the total cost of binding a payment to a job is less then 200ms.

In conclusion, the total cost incurred by the outsourcer is under 1.2s for abc-conjecture jobs and under 0.3s for hash inversion jobs even when 100 ringer sets are used. The ringer generation step is job dependent but the ringer to payment binding is independent of job details. As the size of the job increases, the ringer generation overhead becomes dominant (see Equation 5) however it is only a fraction of the total job computation cost.

$$\begin{aligned} Overhead(O) &= (T_{Ring} + T_{split} + T_{obf} + T_{bind})/T_{job} \\ &\approx (r \times cnt \times T_f)/(|D| \times T_f) \\ &= r \times cnt/|D| \end{aligned} \quad (5)$$

C. Worker Costs

Finally, we study the worker overheads. Specifically, we are interested in the three main components, verification, the actual job computation and the extraction of the last payment share.

Payment and Job Verification: The worker needs to verify that if it completes the job, it is able w.h.p. to extract the payment. The verification cost is approximately given in Equation 6, where $T_{RSA\_enc}(N)$ and $T_{RSA\_ver}(N)$ denote the RSA signature verification and public key encryption costs. For all practical purposes these two costs are equivalent.

$$\begin{aligned} T_{Ver} &= (r-1) \times (cnt \times (T_H + T_f) + 2T_{exp}(|q|) + \\ &+ T_{RSA\_enc}(N) + T_{xor}(N)) + T_{RSA\_ver}(N) \end{aligned} \quad (6)$$

We measure the worker's verification cost as a function of the number of ringer sets employed by the outsourcer. That is, we increase r from 10 to 100, each ringer set containing 10 ringers. Figure 3(a) shows the verification cost both for hash inversion and abc-conjecture jobs, each data point being averaged over 100 independent experiments. It is interesting to note that the verification cost is quite similar to the outsourcer's ringer generation cost. The worker's cost is slightly larger, consisting of roughly $r - 1$ additional RSA public key encryptions and $2(r - 1)$ modular exponentiations in the group Γ. However, even for abc-conjecture jobs with 100 ringer sets, each consisting of 10 ringers, the worker's cost is approximately 1.1s. For hash inversion jobs this cost is under 300ms.

Computation Costs: We also briefly investigate the worker's job computation cost as a function of the job size (cardinality of input domain D). For both hash inversion and abc-conjecture job types, we experiment with input domain sizes ranging from 100000 to half a million. Each input domain consists of contiguous ranges of integers up to 10^6 [2]. Figure 3(b) shows the results of this experiment. Note that Golle-Mironov's computation overhead is identical to that of our solution: Besides performing the actual job, both solutions require the worker to lookup each computed value in the set of input ringers (the unrevealed set of ringers in our solution).

As expected, the computation cost increases linearly with the input domain size. The increase is steeper for the abc-conjecture job, reaching almost 300s for 500000 input values. This cost will certainly be higher for larger input domain values. Outsourcing jobs makes sense only if the computation cost is on the order of hours. Note however that even when compared to the jobs considered here, the overheads of our solution, both for outsourcers and workers are negligible.

Payment Extraction: After completing the computation, the worker needs to remove the ringer based blinding factor from the last payment share. The overhead of this operation is roughly an Xor operation, r string concatenations and one random string generation. Figure 3(c) shows the cost of this operation when the number of ringer sets r increases from 10 to 100. Each bar is an average over 100 independent experiments. It is interesting to see that even though theoretically this cost should be linear in r (the number of string concatenations) in practice it is not. This is because the string concatenation cost is negligible. The variations seen in Figure 3(c) are actually quite small (the highest value is under 0.3ms) and are due to running the experiments on a real machine.

VI. Related Work

The model we use in this paper for securely distributing computations in a commercial environment is proposed in [19], [14], [13]. Monrose et al. [19] propose the use of computation proofs to ensure correct worker behavior. A proof consists of the computation state at various points in its execution. In essence then, the proof is a trace where each value in the trace is the result of the computation based on the previous trace value. The worker simultaneously performs the computation and populates the proof trace. The outsourcer probabilistically verifies the computation correctness given the proof, by repeatedly picking a random trace value, executing the computation given that value and comparing the output with the next trace value.

Golle and Stubblebine [14] verify the correctness of computation results by duplicating computations: a job is assigned to multiple workers and the results are compared at the outsourcer. Golle and Mironov [13] introduce the ringer concept to elegantly solve the problem of verifying computation completion for the "inversion of one-way function" class of computations. Du et al. [11] address this problem by requiring workers to commit to the computed values using Merkle trees. The outsourcer verifies job completeness by querying the values computed for several sample inputs.

Szajda et al. [22] and Sarmenta [20] propose probabilistic verification mechanisms for increasing the chance of detecting

[2]For abc-conjecture jobs the input consists of two domains, for a and b values.

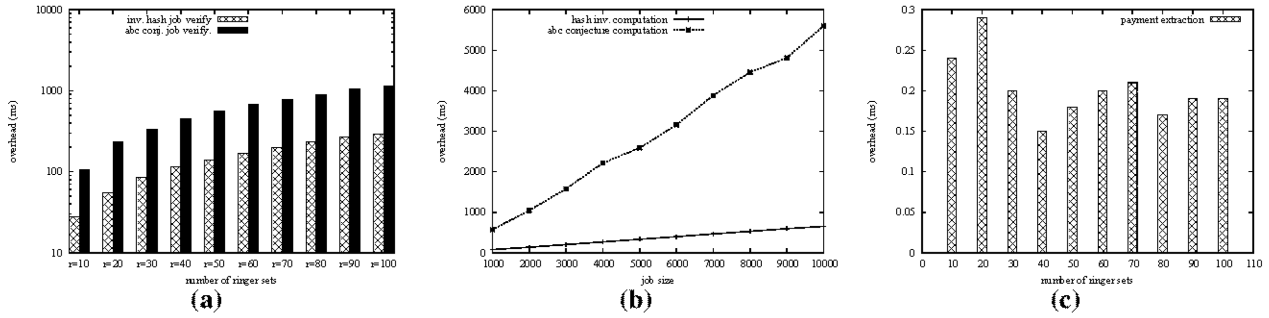

Fig. 3. Worker costs. (a) Job verification cost as a function of the number of ringer sets. Even for 100 ringers sets and the more compute intensive abc-conjecture jobs, our solution takes only 1.1s. (b) Actual computation overhead, function of the input domain D cardinality. Growth is linear and shows that verification costs become negligible for reasonable sized jobs. (c) Payment extraction cost as a function of the number of ringer sets. The number of ringer sets influences only the string concatenation cost. As such, the cost is less than 0.3ms.

cheaters. In the same setting, Szajda et al. [23] propose a strategy for distributing redundant computations, that increases resistance to collusion and decreases associated computation costs. Instead of redundantly distributing computations, Carbunar and Sion [10] propose a solution where workers are rated for the quality of their work by a predefined number of randomly chosen witnesses. Belenkiy et al. [7] propose the use of incentives, by setting rewards and fines, to encourage proper worker behavior. They define a game theoretic approach for setting the fine-to-reward ratio, deciding how often to double-check worker results.

Our work can be viewed as using an instance of conditional e-payments [21], [9]. Conditional e-payments are tools for generating verifiable correct payments that become valid only for future discrete event outcomes. The payment mechanisms proposed in our work can be viewed as conditional, becoming valid only on the event of the worker performing the computation.

VII. Conclusions

In this paper we study an instance of the secure computation outsourcing problem in mesh networks, where the job outsourcer and the workers are mutually distrusting. We employ ringers coupled with secret sharing techniques to provide verifiable and conditional e-payments. Our solution relies on the existence of a bank, but it is oblivious to job details. We prove the security of our constructions and show that the overheads imposed by our solution on the bank, outsourcers and workers are small.

References

[1] ABC@Home. http://abcathome.com/.

[2] Great Internet Mersenne Prime Search (GIMPS). http://www.mersenne.org/.

[3] The legion of the bouncy castle. http://www.bouncycastle.org/documentation.html.

[4] Mersenne primes: History, theorems and lists. http://primes.utm.edu/mersenne/.

[5] SETI@home. http://setiathome.ssl.berkeley.edu/.

[6] Trade korea. http://www.tradekorea.com/products/IP_Set_Top_Box.html.

[7] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *NetEcon '08: Proceedings of the 3rd international workshop on Economics of networked systems*, 2008.

[8] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and communications Security (CCS'93)*, pages 62–73, 1993.

[9] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 101–115, 2007.

[10] B. Carbunar and R. Sion. Uncheatable reputation for distributed computation markets. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, 2006.

[11] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[12] O. Goldreich. *The Foundations of Cryptography - Volume 1*. Cambridge University Press, 2001.

[13] P. Golle and I. Mironov. Uncheatable distributed computations. In *Lecture Notes in Computer Science - Proceedings of RSA Conference 2001, Cryptographer's track*, pages 425–440, 2001.

[14] P. Golle and S. G. Stubblebine. Secure distributed computing in a commercial environment. In *FC '01: Proceedings of the 5th International Conference on Financial Cryptography*, pages 289–304, 2002.

[15] R. Laboratories. Cryptographic challenges. http://www.rsa.com/rsalabs/node.asp?id=2091.

[16] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. In *PKC '00: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, 2000.

[17] F. Margot. Introduction to integer linear programming. http://wpweb2.tepper.cmu.edu/fmargot/introILP.html.

[18] J. Messina. First smartphone with 1ghz processor (w/ video). http://www.physorg.com/news165064171.html.

[19] F. Monrose, P. Wyckoff, and A. Rubin. Distributed execution with remote audit. In *Proceedings of Network and Distributed System Security Symposium*, 1999.

[20] L. F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems: Special Issue on Cluster Computing and the Grid*, 18, March 2002.

[21] L. Shi, B. Carbunar, and R. Sion. Conditional e-cash. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, 2007.

[22] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large-scale distributed computations. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 216–224, 2003.

[23] D. Szajda, B. Lawson, and J. Owen. Toward an optimal redundancy strategy for distributed computations. In *Proceedings of the 2005 IEEE International Conference on Cluster Computing (Cluster)*, 2005.