Payments for Outsourced Computations

Bogdan Carbunar and Mahesh V. Tripunitara

Abstract—With the recent advent of cloud computing, the concept of outsourcing computations, initiated by volunteer computing efforts, is being revamped. While the two paradigms differ in several dimensions, they also share challenges, stemming from the lack of trust between outsourcers and workers. In this work, we propose a unifying trust framework, where correct participation is financially rewarded: neither participant is trusted, yet outsourced computations are efficiently verified and validly remunerated. We propose three solutions for this problem, relying on an offline bank to generate and redeem payments; the bank is oblivious to interactions between outsourcers and workers. We propose several attacks that can be launched against our framework and study the effectiveness of our solutions. We implemented our most secure solution and our experiments show that it is efficient: the bank can perform hundreds of payment transactions per second and the overheads imposed on outsourcers and workers are negligible.

Index Terms—Distributed applications, security and privacy protection.

1 Introduction

The ability to execute large, compute intensive jobs, is no longer the privilege of supercomputer owners. With the recent advent of cloud computing and volunteer computing initiatives, users can outsource their computations for execution on computers with spare resources. Cloud computing provides hardware (CPU and storage) capabilities, along with software and electronic services, which clients can elastically rent while abstracting from lower level details. Motivated by the ability of computer owners to donate CPU resources, volunteer computing takes advantage of the parallelizable nature of several large compute problems to distribute jobs to available computers over the internet.

In this work, we consider a general "compute market" framework, encompassing the cloud and volunteer computing paradigms: participating computers can act both as service providers (workers) and as clients (outsourcers). Outsourcers have computing jobs they cannot complete in a timely fashion, whereas workers are willing to spend CPU cycles to run parts of such jobs. While it is natural to motivate participation through the use of financial incentives, the distributed nature of the framework raises trust questions: Outsourcers do not trust the workers to correctly perform computations and workers do not trust outsourcers to pay for completed jobs.

While solutions exist that address the lack of trust of outsourcers on workers (see Section 7), the lack of trust of a worker in the outsourcer is not addressed—W is required to fully trust O. In settings where any cellphone or PC owner can be an outsourcer, this becomes a problem.

We consider, therefore, the following computation model. A *job* takes as inputs a function $f\colon I \rightarrow R$, 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.

In this paper, we propose solutions that address 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. Our solutions employ the ringer concept proposed by Golle and Mironov [9].

Our first solution (see Section 4) requires O to split the key used to obfuscate the payment and hides the subkeys into precomputed, randomly chosen parts of the job. The worker is entitled to a probabilistic verification of the payment received before beginning the computation. However, a malicious outsourcer that generates a single incorrect subkey may pass the verification step but prevent an honest worker from recovering the payment. We address this issue in the second solution (see Section 5), through the use of threshold cryptography. O uses threshold sharing to divide the payment into multiple shares and obfuscates a randomly chosen subset of the shares with solutions to parts of the job. The worker needs to retrieve only a subset of the shares in order to reconstruct the payment. This significantly improves the worker's chance of retrieving the payment even in the presence of a malicious outsourcer generating incorrect shares. However, this solution provides the worker with an unfair advantage in recovering the payment before completing the job: fewer shares need to be discovered.

We address this problem in the third solution (see Section 6). We use exact secret sharing to compute shares of the payment token—all the shares are needed to reconstruct

- B. Carbunar is with the School of Computing and Information Sciences, Florida International University, 11200 SW 8th ST, Mail Stop ECS 354, Miami, FL 33199. E-mail: carbunar@gmail.com.
- M.V. Tripunitara is with the Department of Electrical and Computer Engineering (ECE), The University of Waterloo, 200 University Ave. W, Waterloo, ON N2L 3G1, Canada. E-mail: tripunit@uwaterloo.ca.

Manuscript received 1 Sept. 2010; revised 9 Mar. 2011; accepted 2 May 2011; published online 25 May 2011.
Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-09-0516.
Digital Object Identifier no. 10.1109/TPDS.2011.163.

1045-9219/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

the payment. 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. 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.

We have implemented and tested the third solution on two problems: finding the preimage of a cryptographic (SHA-1) hash, and the abc conjecture [1]. Our experiments show that the solution is efficient. For standard security parameters, on a single off-the-shelf PC, the bank can perform 100 payment generation and payment redemption transactions per second. Moreover, the overheads imposed by our solution on outsourcers and workers are negligible when compared to overheads of jobs. Finally, our solution compares favorably with the solution of Golle and Mironov [9], considering the benefit of providing payment redemption assurances to workers.

This paper makes use of a supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163. The supplement contains theorem proofs and their implications, additional solution explanations, details for several attacks and the defenses provided by our solutions, as well as an empirical evaluation of our third solution. The supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163, is grouped according to the sections in this paper.

2 Model

Our framework is similar to the ones presented in prior work [6], [9]; we present it here for clarity. The three principals in a solution are the outsourcer O, the bank B, and the worker W. O prepares jobs he wants to be done in the manner we discuss in Section 1, B issues and redeems payment tokens, and W computes the job.

In the ideal case, B should be offline: O and W independently transact with it outside of any exchanges they have as part of the outsourcing. The role of B is to act as a financial "holding company." B has no interest or participation in the nature of the outsourcing between O and W. That is, B is trusted to act as an honest bank and follow the protocol correctly. Outsourcers and workers are assumed to be malicious. Dishonest outsourcers will attempt to have their jobs computed while paying less than agreed. Dishonest workers will attempt to redeem payments while minimizing the work they perform.

We do not consider confidentiality, integrity, and authentication issues, which can be application specific and we believe are outside the scope of this work.

In the following, we adopt the more abstract mechanisms as used in the random oracle model [3]. $G\colon \{0,1\}^* \rightarrow \{0,1\}^\infty$ is a random generator and $H\colon \{0,1\}^* \rightarrow \{0,1\}^h$ is a random hash function. We use the notation $x \hookrightarrow_R D$ to denote the fact that the value x is randomly chosen from the domain D. We also use $x;y$ to denote the concatenation of strings x and y. $E_K(M)$ denotes the symmetric encryption of message M with key K. For a given symmetric key algorithm, let s denote the key's bit size. We also assume the bank has a trapdoor permutation, $\langle p, p^{-1}, d\rangle$ that is secure from nonuniform polynomial time [8] adversaries. The function p is public, and p^{-1} is private to B.

3 Ringers—An Overview

The solution from Golle and Mironov [9] (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 perform all the computations that were outsourced to him.

The idea behind ringers is to require the outsourcer to select a small set of random input values from D and to precompute the image of the function f on those values (true ringers). Besides the image of interest, the outsourcer sends to the worker also the true ringers. The worker needs to retrieve the preimages of *all* the received images. In order to prevent the worker from stopping the work after inverting all but one image, the outsourcer uses bogus ringers, which are values from the image of f that do not have a preimage in D. If the worker is able to invert at least the true ringers, the outsourcer is convinced that the worker has completed a large percent of the job. The solution has the following steps.

Job generation. O chooses an integer $2m$, the total number of ringers. He picks an integer $t \in [m+1, \ldots, 2m]$ which conforms to the probability distribution $d(t) = 2^{m-t-1}$. Let t be the number of true ringers, and $2m - t$ be the number of bogus ringers. O computes $f(x)$ for every true and bogus ringer x. These postimages 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 postimage whose preimage O seeks, and each y_j is the postimage 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 preimage 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.

4 Payment Splitting Based on Subkeys

We now present our first solution, whose notations are summarized in Table 1.

Outsourcer setup. O generates token $t = \langle Id(O), Id(W), SN, v, T\rangle$, containing O's identity, W's identity, a fresh serial

TABLE 1
Notation Used in the Subkey Solution

O	The outsourcer	K_j	Subkeys of K
f	The function of interest for O	K	Obfuscation key
B	The bank	t	Payment token
W	The worker	$obf(t)$	Obfuscated payment token
D	Domain of f outsourced to W	$\sigma_{obf}(t)$	Signed obfuscated token
H	Random hash function	σ	Valid payment
h	Output length of H	V	Validation set
$\langle p, p^{-1}, d\rangle$	Trapdoor permutation of B	Ver	Permuted V
$2m$	Total number of ringers	S	Bank signature
k	Number of payment shares	$\mathcal{P}$	Outsourced payment

number, the currency value v, and the deadline for completing the job. O picks an integer k from the interval $\{m+1, .., 2m\}$ which conforms to the probability distribution $d(k) = 2^{m-k-1}$, similar to [9]. O keeps k secret. O also picks a symmetric key $K_s \hookrightarrow_R \{0,1\}^s$.

Payment splitting. O picks k points $x_1, .., x_k \hookrightarrow_R D$ and generates k values $K_j = H(f(x_j))$, $j = 1..k$ (ringers) and random values $K_{k+1}, .., K_{2m} \hookrightarrow_R \{0,1\}^h$ (bogus ringers). h is the bit size of the output of the one-way function H. Without loss of generality, let $K_1 < \cdots < K_k$ (if they are not, sort and rename). Let $K = H(K_1; ..; K_k)$. The ringers $K_1, .., K_k$ are also called "subkeys" of K. Generate $obf(t) = E_{K_s}(t)p(K)$. Send to B the values $obf(t), t, k, K, K_1, .., K_{2m}, K_s$.

Binding payment to job. B verifies first that $m+1 \leq k \leq 2m$ and that $K = H(K_1; ..; K_k)$, where $K_1; ..; K_k$ are the first k keys from the set $K_1, .., K_{2m}$. Then, it verifies that $E_{K_s}(t)p(K) = obf(t)$. If any verification fails, B aborts the protocol. Otherwise, it performs the following actions:

- Store the tuple $\langle t, K_s\rangle$ locally.
- Generate random $R \hookrightarrow_R \{0,1\}^*$. Sign obf(t) to obtain

$$\sigma_{obf}(t) = p^{-1}(obf(t)) = p^{-1}(E_{K_s}(t))K.$$

  Let $\sigma = p^{-1}(E_{K_s}(t))$. Thus, $\sigma_{obf}(t) = \sigma K$. σ denotes a valid payment of value v. Whoever can present this value to B, can cash it.
- Generate validation set

$$V = \{p^{-1}(H(K_1, R, “r”)), .., p^{-1}(H(K_k, R, “r”))\} \cup \{p^{-1}(H(K_{k+1}, R)), .., p^{-1}(H(K_{2m}, R))\}.$$

- Generate signature $S = p^{-1}(H(\sigma_{obf}(t), t, R))$. Send to O the values $\sigma_{obf}(t)$, R, S and the set V.

Payment generation. Let π denote a random permutation. O generates the payment $\mathcal{P} = \langle t, \sigma_{obf}(t), Ver, 2m\rangle$, where $Ver = \pi(V)$. Ver is called the *validation set*.

Job transmission. O sends the job to W, along with the values $\mathcal{P}$, R, S.

Verification. W needs to verify $\mathcal{P}$'s correctness before starting the job. First, it verifies B's signature on $\sigma_{obf}(t)$, using R, the payment token $t = \langle Id(O), Id(W), SN, v, T\rangle$ and the signature $S = p^{-1}((H(\sigma_{obf}(t), t, R))$. If it verifies, W initializes the set $SK = \emptyset$. SK is the set of subkeys of K known to W. Then, W selects indexes $c_1, .., c_q \hookrightarrow_R \{1..2m\}$, $q < r$ and sends them as challenges to O. O processes each challenge c_j in the following manner:

- If the c_jth element Ver, denoted by $Ver(c_j)$, is a ringer subkey signed by B, O reveals the preimage $x_j \in D$. W computes $K_j = H(f(x_j))$ and verifies the equality $H(K_j, R, “r”) = p(Ver(c_j))$. If the equality holds, $SK = SK \cup K_j$ and $Ver = Ver - Ver(c_j)$. Otherwise, W aborts the protocol.
- If the value $Ver(c_j)$ is B's signature on a bogus ringer, O reveals $K_j \in \{K_{r+1}, .., K_{2m}\}$. W verifies the equality $H(K_j) = p(Ver(c_j, R))$. If it holds, $Ver = Ver - Ver(c_j)$. Otherwise, W aborts the protocol.

Computation. W removes B's signature from each element in the set Ver. Let p(Ver) denote the resulting set. W evaluates f on each input value $x \in D$. If $H(H(f(x)),\ R,\ “r”) \in p(Ver)$, $SK = SK \cup H(f(x))$ and $p(Ver) = p(Ver) - H(H(f(x)), R, “r”)$.

Payment extraction. At the end of the computation, to extract the payment, W sorts the elements in SK in increasing order. Compute $K = H(SK[1]; ..; SK[k])$, where k is the size of the SK set and $SK[i]$ denotes its ith element (in sorted order). Compute the value $\sigma = \sigma_{obf}K^{-1}$.

Payment redemption. If the current time is less than T, W sends σ to B, along with the tuple $t = \langle Id(O), Id(W), SN, v, T\rangle$. B accepts such a message only once. It performs the following actions:

- Retrieve the tuple $\langle t, K_s\rangle$ from its local storage.
- Verify that the time T from t exceeds or equals the current time. Verify that the identifier of the sender of the message is indeed the second field of t.
- Verify that $D_{K_s}(p(\sigma)) = t$. If all verifications succeed, B credits W's account in the amount v.

Cancellation. If the current time exceeds T, W cannot redeem the payment. However, O can cancel it by sending t and $S = p^{-1}(H(\sigma_{obf}(t), t, R))$ to B. O cannot cancel a payment before the expiration time T of its associated job.

4.1 Solution Intuition

The purpose of the random R used during the payment generation step is to bind $\sigma_{obf}(t)$ to V. This proves that these values were signed by B at the same time, preventing O from using $\sigma_{obf}(t)$ and V generated in different protocol instances. While a value of format $p(H(K_i, R))$ from V certifies the fact that B has seen the subkey K_i, a value of format $p(H(K_i, R, “r”))$ also authenticates the fact that O claimed that subkey to be a ringer, subkey of K, where K obfuscates the payment σ. Note that B does not verify the well formedness of the ringers $K_1, .., K_k$. This verification is to be performed by the workers.

Following the job transmission step, W needs to generate and verify challenges. Since B has generated the random R value and has signed both $\sigma_{obf}(t)$ and each key $K_j, j = 1..2m$ with it, the challenge verification procedure allows W to verify that each revealed element in the set Ver is a payment piece and any two revealed pieces belong to the same payment instance (R binds SN, $\sigma_{obf}(t)$ and all K_j values). O cannot pretend that a challenged ringer subkey K_j is not a ringer. This is because B has included the string "r" in its signature of the subkey K_j. During the computation, W needs to retrieve $K_1, .., K_k$, compute K, then recover σ from $\sigma_{obf}(t)$. The bank signed value σ allows W to cash the payment.

TABLE 2
Notation Used in the Threshold Splitting-Based Solution

O	The outsourcer	P	Payment token
f	The function of interest for O	σ	Verification value
B	The bank	s_i	Shares of P
W	The worker	P_i	Payment token shares
D	Domain of f outsourced to W	HS	Hash set of P_i's
H	Random hash function	r_i	Ringers
h	Output length of H	Obf_i	Obfuscated payment shares
$\langle p, p^{-1}, d\rangle$	Trapdoor permutation of B	Clr_i	Cleartext shares
p, q	Security parameters	$\mathcal{P}$	Payment set
M	Payment message	Ver	Verification set

4.2 Analysis

We present a first result, whose proof can be found in the corresponding section of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163.

Theorem 1. *If W retrieves the payment, W has completed the job with probability $1 - \frac{1}{m2^{m+1}} - (\frac{4}{m})^m$.*

We now introduce a property that regards the ability of the outsourcer to prevent honest workers from recovering payments.

Invalid shares property. O attempts to include invalid shares in place of legitimate shares in what is embedded in the job. The objective is to undermine the payment verifiability property and get an honest W to accept the job, but not get paid when he completes it.

Let u be the parameter denoting the number of invalid ringers computed by O. The following property holds.

Theorem 2. *If an outsourcer includes u invalid shares in the job, a worker is able to retrieve the payment with probability at least $1 - e^{-uq/(2m-q+1)}$.*

The proof can be found in the corresponding section of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163, along with a graphical illustration of the worker's probability of detecting an invalid share. Our conclusion is that for $m = 100$ and $q = 10$, W's probability of detecting a single invalid share is 5 percent.

5 Payment Threshold Splitting

Our second solution uses threshold sharing to address the invalid share property. It splits the payment into $2m + p$ shares such that any $2m$ shares reconstruct the payment. The outsourcer obfuscates a subset of the shares with a small subset of the solution of the job to be performed. Table 2 lists the notations we used in the solution. Fig. 1 illustrates our solution.

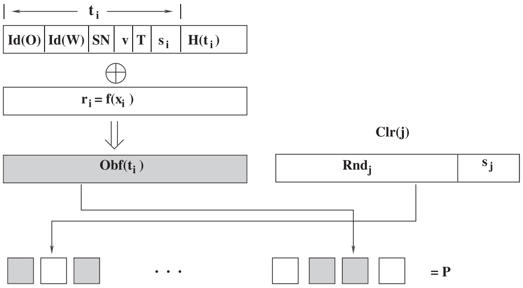

Fig. 1. Generation of Obf and Clr shares. The Obf shares (shown on the left side in the figure) are generated by xor-ing a randomly chosen image of the job to be performed by W with a concatenation between a payment share and the share's hash. The Clr shares (shown on the right side in the figure) are generated by concatenating a payment share with a random number. The Obf and Clr values are randomly permuted (lower side in the figure) to generate the payment structure to be sent to W.

Setup. Let p and q be two security parameters, $c\sqrt{m} < p, q < m$, for a constant c. Pick a symmetric key $K \hookrightarrow_R \{0,1\}^s$. Instantiate a $(2m, 2m + p)$ secret sharing scheme (e.g., Shamir's scheme [13]) such that any but not less than $2m$ shares are required to compute the secret. Let $\mathcal{SS}$ be the reconstruction function, that given any $2m$ or more shares reconstructs the secret.

Payment generation. O generates the message $M = \langle Id(O), Id(W), SN, v, T\rangle$, where SN is a fresh serial number, v is the currency value, and T is the job deadline. Send the tuple along with the key K to B.

Payment signature. B computes a payment token $P = E_K(p^{-1}(M))$ and verification value $\sigma = p^{-1}(H(M))$. B stores the tuple $\langle SN, v, T, t, K\rangle$ in local storage, indexed by serial number. B sends P and σ to O. The convention is that whoever knows P can cache the payment.

Payment splitting. O uses the P and σ values received from B to perform the following actions:

- Use the $(2m, 2m + p)$ secret sharing scheme to generate $2m + p$ shares $s_1, .., s_{2m+p}$ of P.
- Pick an integer $k \hookrightarrow_R \{m + p + 1, .., 2m - q\}$ with distribution $d(k) = 2^{m-p-q-k-1}$. k is secret and denotes the number of ringers.
- Use the shares $s_1, .., s_{2m+p}$ to generate $2m + p$ payment tokens $P_i = \langle Id(O), Id(W), SN, v, T, s_i\rangle$, $i = 1..2m + p$. Each payment token is a wrapper for one of the shares s_i. Send the payment tokens P_i to B along with P, k, m, p, and q.

Share signature. When B receives this message, it first verifies that $m + p + 1 < k < 2m - q$. It then compares P against the value previously stored for O and uses the reconstruction function $\mathcal{SS}$ to verify that all the shares s_i contained in the token shares P_i are unique and that any $2m$ of them indeed reconstruct P. This verification step could be probabilistic. If any verification fails, B aborts and penalizes O's account. Otherwise, B performs the following steps:

- Generate the hash set $HS = \{H(P_{k+1}), \ldots, H(P_{2m+p})\}$. Store HS along with the tuple stored under SN, $\langle SN, v, T, t, K, HS\rangle$.
- For each payment token P_i, generate $p^{-1}(H(t_i))$, $i = 1..2m + p$. Send these values to O.

Binding payment to job. O uses the values received from B to embed the payment into a job as follows:

- Choose k values $x_1, .., x_k \hookrightarrow_R D$ and compute their images, $r_i = f(x_i)$, $i = 1..k$. The r_i's are called *ringers*.
- Use each ringer r_i to compute the obfuscated payment share $Obf_i = r_i \oplus (P_i; H(P_i))$. Let $sz = |Obf_i|$.

- For all remaining $2m+p-k$ $(l=k+1..2m+p)$ shares, compute cleartext shares $Clr_l=(Rnd_l;s_l)$, where $Rnd_l \hookrightarrow_R \{0,1\}^{sz-|s_l|}$.
- Let π_1 be a random permutation. Generate the outsourced payment set $\mathcal{P}=\pi_1\{Obf_1,..,Obf_k, Clr_{k+1},..,Clr_{2m+p}\}$, containing both obfuscated and cleartext payment shares.
- Let π_2 be a random permutation. Generate the verification set

$$Ver=\pi_2(\{p^{-1}(H(t_1)),..,p^{-1}(H(t_k))\} \cup \{R_{k+1},..,R_{2m+p}\}),$$

where $R_{k+1},..,R_{2m}$ are random values of the same bit length as the output of p^{-1}. Ver consists both of B's signatures on the k obfuscated payment tokens (from the set $\mathcal{P}$) and $2m+p-k$ indistinguishable random values.

Job transmission. O sends SN, v, 2m + p, T, $\mathcal{P}$, Ver, and σ to the worker W along with the job. As mentioned in the payment generation, $\sigma=p^{-1}(H(M))$.

Verification. After receiving the job, W proceeds to verify the correctness of the payment $\mathcal{P}$. It first verifies the correctness of the job payment, using $\sigma=p^{-1}(H(M))$. That is, W verifies that the payment was generated by O for W, has the serial number SN, is for currency amount v, is valid for redemption before time T, and is authenticated by B. If these checks verify, W initializes Shr, its set of discovered payment token shares, to the empty set. W selects random indexes $c_1,..,c_q \hookrightarrow_R \{1,..,2m+p\}$, $q<m$ and sends them to O. O processes each index c_j separately as follows:

- If the c_jth element of the payment set $\mathcal{P}$, denoted by $\mathcal{P}(c_j)$, corresponds to an obfuscated payment token share, P_o, O sends the preimage x of the ringer used for the obfuscation of this value. W computes $\mathcal{P}(c_j)\oplus f(x)$. If the $\mathcal{P}(c_j)$ value is valid, the result of this operation should have the format $(P_o;H(P_o))$. W verifies that the value P_o has the format $P_o=\langle Id(O),Id(W),SN,v,T,s_o\rangle$. W then verifies that the set Ver contains B's signature on the $H(P_o)$ value. If any of these checks fails, W aborts the protocol. Otherwise, update the sets $Shr=Shr\cup s_o$, $\mathcal{P}=\mathcal{P}-\mathcal{P}(c_j)$ and $Ver=Ver-p^{-1}(H(P_o))$.
- If $\mathcal{P}(c_j)$ is a nonobfuscated payment token of format $(Rnd_n;s_n)$, O sends the signed value $p^{-1}(H(P_n))$ received from B during the share signature step (but not sent to W during job transmission). W checks that $H(Id(O),Id(W),SN,v,T,s_n)=p(p^{-1}(H(P_n)))$. If this verification fails, W aborts the protocol. Otherwise, it updates the set $Shr=Shr\cup s_n$.

Computation. W evaluates f on each $x\in D$. Then, it computes $f(x)\oplus\mathcal{P}(i)$, for all $i=1..2m+p$. $\mathcal{P}(i)$ denotes the ith element of the outsourced payment set $\mathcal{P}$. If the result is of the form $(P;H(P))$, with P of format $\langle Id(O),Id(W), SN,v,T,s\rangle$ and $p^{-1}(H(P))\in Ver$, then update the sets $Shr=Shr\cup s$, $\mathcal{P}=\mathcal{P}-\mathcal{P}(i)$, $Ver=Ver-p^{-1}(H(P))$: an obfuscated share has been discovered.

Redemption. If W finishes the job before the deadline T, it sends the share set Shr to B, along with the tuple $\langle SN,v,T\rangle$. B retrieves from its local storage the tuple $\langle SN,v,T,t,K,HS\rangle$ indexed under SN, where $HS=\{H(P_{k+1}),\ldots,H(P_{2m+p})\}$. B verifies that the request comes from the worker W whose id is contained in the token $P=E_K(p^{-1}(\langle Id(O),Id(W),SN,v,T\rangle))$. B only accepts this redemption request once and if the current time is less than T. B sends to W the set HS. Let $CShr$ be the set of nonobfuscated shares that W needs to identify. Initially, $CShr=\emptyset$. W performs the following actions:

- For each value in $\mathcal{P}$ (there should be $2m+p-k$ elements left), treat the value as if being of format $(Rnd_n;s_n)$, where Rnd_n is a random number and s_n is a payment share. Compute $P_n=\langle Id(O),Id(W), SN,v,T,s_n\rangle$ and look for the hash of this value in the set HS. If a match is found, $CShr=CShr\cup s_n$.
- Send the $CShr$ set to B.

B verifies the correctness of the shares in $CShr$, by also looking them up in HS. B uses all the shares from the set Shr, plus $2m-|Shr|$ shares from $CShr$ to reconstruct the payment P. If successful, it deposits P into W's account.

Cancellation. If the current time exceeds T, W cannot redeem the payment. O, however, can cancel the payment, by sending P to B. Then, if W has not redeemed the payment before time T, B reimburses O. O cannot cancel a payment before the expiration time of the associated job.

5.1 Solution Intuition

The set Ver does not contain B's signatures on the cleartext payment tokens, $Clr_{k+1},..,Clr_{2m+p}$, to prevent the worker from immediately distinguishing them from the $Obf_1,..,Obf_k$ shares. During the verification step, O needs to prove either that the challenged share was obfuscated or that it was presented in cleartext to W. The proof consists of showing to W the fact that B has witnessed (signed) the obfuscated and cleartext challenged shares in the format claimed by O. In both cases, the worker receives one payment share. When the worker completes the computation, if it has not retrieved 2m shares, the bank will allow it to search for additional cleartext shares. This process is allowed only once; thus, W has to be certain that it has retrieved all the shares it needs or that it has completed the job. Note that O cannot cancel a payment before the expiration time of the associated job and prevent a worker from redeeming the recovered payment.

5.2 Analysis

The following result shows the worker's resilience to the invalid share property.

Theorem 3. *The probability that an invalid shares property is detected is lower bounded by* $1-e^{-c^2/2}$.

The proof can be found in the corresponding section of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163, along with its graphical illustration and an evaluation of the solution's computation overhead. Our conclusion is that for $m=100$ and $p=30$, a value of $q=30$ increases W's chance of detecting invalid shares to 99 percent. We now describe another property—this time providing an advantage to a dishonest worker.

TABLE 3
Notation Used in the Exact Payment Splitting Solution

O	The outsourcer	P	Payment token
f	The function of interest for O	n	The number of payment shares
B	The bank	P_i	A share of P
W	The worker	r	Number of payment shares
D	Domain of f outsourced to W	σ	B's signature on g^P
$\mathcal{R}_i$	A set of ringers	J	A set of sets of ringers
G	Random generator	$t_{i,j} \in D$	Pre-image for a true ringer
H	Random hash function	$b_{i,j} \notin D$	Pre-image for a bogus ringer
h	Output length of H	$\mathcal{E}_{B,i} = p(P_i)$	An obfuscated payment share
H_K	Keyed hash based	$\widehat{\mathcal{E}_{B,i}}$	W's computed value for $\mathcal{E}_{B,i}$
$\langle p, p^{-1}, d\rangle$	Trapdoor permutation of B	$\mathcal{K}_i$	Symmetric key based on true ringers
Γ	A finite cyclic group	$\widehat{\mathcal{K}_i}$	W's computed value for $\mathcal{K}_i$
g	A generator in Γ	$\mathcal{P}_{i,\mathcal{K}} = \mathcal{K}_i \oplus \mathcal{E}_{B,i}$	Encrypted, obfuscated payment share
q	The prime order of Γ	$\mathcal{P}$	Set of $\mathcal{P}_{i,\mathcal{K}}$'s

Premature payment reconstruction. W attempts to reconstruct a legitimate payment-token based on his knowledge of the redundancy that is built into the payment-splitting scheme. The objective is to allow a cheating worker to stop the job computation step early, recover, and then successfully redeem the payment. After recovering a certain number of payment shares that are embedded in the true ringers, W attempts to verify that the remaining ringers are bogus while simultaneously trying to extract the payment. Assume that he has $k - x$ payment pieces that he has extracted legitimately from true ringers (there are a total of k true ringers).

W premises that the remainder are bogus ringers and chooses sets of $2m - k + x$ from which he extracts what he believes are payment pieces. He then reconstructs each set of $2m$ pieces and checks for duplicates among the reconstructions. If there are any duplicates, then that is the reconstructed payment he seeks. We observe that there are at most $r = \binom{2m+p-k+x}{2m-k+x}$ reconstructions he needs to perform, and $r \geq (\frac{2m-k+x+1}{p})^p$. Thus, the redundancy in payment shares gives the worker an unfair advantage in terminating the computation before completing the job while also being able to recover the payment.

6 Exact Payment Splitting

The first two solutions are either vulnerable to the invalid payment share or the premature payment reconstruction properties *and* require heavy bank involvement. We now propose a solution that addresses these concerns while involving B only in the payment generation. Table 3 lists our notations.

Setup. The bank, B, has the following:

- A trapdoor permutation, $\langle p, p^{-1}, d\rangle$ that is secure from nonuniform polynomial time [8] 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 "$modq$" 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. We provide concrete instantiations in the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163.

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.
- $\sigma = p^{-1}(H(g^P))$. σ is B's signature on g^P.

Job generation. O first generates an instance of a job that consists of the function $f\colon I \to R$, special image y and subdomain $D \subset I$ 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 need 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 $P_1 \times \cdots \times P_r = P \; mod \; q - 1$. 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 $\mathcal{K}_i = G(t_{i,1}\| \cdots \|t_{i,i_t})$. 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 $\widehat{\mathcal{K}_i} = G(t_{i,1} \| \cdots \| t_{i,\widehat{i_t}})$, where $\widehat{i_t}$ is the number of true ringer preimages revealed for index i by O and $t_{i,1}, \ldots t_{i,\widehat{i_t}}$ are the lexicographically sorted true ringer preimages.
- 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 preimages and P_{i_r} have not been disclosed to W by O. W verifies that

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

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 $\widehat{\mathcal{K}_r} = G(v_1 \| \cdots \| v_{i_v})$, 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 $\langle P_1, \ldots, P_{r-1}, \widehat{\mathcal{E}_{B,r}} \rangle$ and M to B for reimbursement.

Payment redemption. For successful redemption, B checks that M is valid, and $P_1 \times \cdots \times P_{r-1} \times p^{-1}(\widehat{\mathcal{E}_{B,r}}) = H_K(M)$. 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.

6.1 Solution Intuition

We present proofs of security properties we desire in Section 6.2. We present a discussion of several small issues and our resolutions, in the corresponding section of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163. 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.

6.2 Security Properties

We now present the security properties of this solution. We consider two classes of security properties: protection from a dishonest outsourcer, and protection from a dishonest worker. We include the proofs in the corresponding section of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163.

Protection from a dishonest O. The objective of a dishonest O is to have W complete the job, without being later able to redeem P. We express our assertion in the following theorem in terms of W's success probability following the Computation step.

Theorem 4. *An honest W successfully redeems the payment token with probability $1 - 1/r$, where r is the number of sets of ringers.*

Consequently, we 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.*

In the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.163, we include a detailed analysis of the impact on honest workers of interacting with multiple dishonest outsourcers. Our conclusion is that even when 80 percent of outsourcers are cheating, a worker's decrease in profit is around 7 percent.

Protection from a dishonest W. We now assume that O is honest. W attempts to reconstruct a legitimate $\mathcal{E}_{B,r} = p(P_r)$ without completing the job.

Theorem 5. *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}.*

7 Related Work

The model we use in this paper for securely distributing computations in a commercial environment is proposed in [11], [10], and [9]. Monrose et al. [11] propose the use of computation proofs to ensure correct worker behavior. A proof consists of the computation state at various points in its execution. Golle and Stubblebine [10] 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 [9] 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. [7] 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. [14] and Sarmenta [12] propose probabilistic verification mechanisms for increasing the chance of detecting cheaters. In the same setting, Szajda et al. [15] 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 [4] propose a solution where workers are rated for the quality of their work by a predefined number of randomly chosen witnesses. Belenkiy et al. [2] 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.

This paper extends the work of Carbunar and Tripunitara [5] by introducing two new solutions to the simultaneous computation for payment exchange problem. The two solutions provide various degrees of trust to the worker and outsourcer. As such, each solution is suitable for environments where one of the participants is less trusted than the other. For instance, cloud providers are more trusted than clients and volunteer project outsourcers are more trusted than workers.

8 Conclusions

In this paper, we study an instance of the secure computation outsourcing problem in cloud and volunteer computing scenarios, 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 solutions rely on the existence of a bank that is oblivious to job details. We prove the security of our constructions and show that the overheads imposed by our final solution on the bank, outsourcers, and workers are small.

Acknowledgments

We would like to thank Matthew Piretti for his suggestions on early versions of this work. We thank the reviewers for their comments and suggestions for improving this work.

References

[1] ABC@Home, http://abcathome.com/, 2011.

[2] M. Belenkiy, M. Chase, C.C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing Outsourced Computation," *NetEcon '08: Proc. Third Int'l Workshop Economics of Networked Systems*, 2008.

[3] M. Bellare and P. Rogaway, "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols," *Proc. First ACM Conf. Computer and Comm. (CCS '93) Security*, pp. 62-73, 1993.

[4] B. Carbunar and R. Sion, "Uncheatable Reputation for Distributed Computation Markets," *Proc. Int'l Conf. Financial Cryptography (FC) and Data Security*, 2006.

[5] B. Carbunar and M. Tripunitara, "Fair Payments for Outsourced Computations," *Proc. Seventh IEEE Int'l Conf. Sensor, Ad Hoc and Mesh Comm. and Networks (SECON)*, 2010.

[6] B. Carbunar and M.V. Tripunitara, "Conditional Payments for Computing Markets," *CANS '08: Proc. Int'l Conf. Cryptology and Network Security*, pp. 317-331, Dec. 2008.

[7] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable Grid Computing," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS)*, 2004.

[8] O. Goldreich, *The Foundations of Cryptography*, vol. 1. Cambridge Univ. Press, 2001.

[9] P. Golle and I. Mironov, "Uncheatable Distributed Computations," *Proc. Conf. Topics in Cryptology: Cryptographer's Track at RSA*, pp. 425-440, 2001.

[10] P. Golle and S.G. Stubblebine, "Secure Distributed Computing in a Commercial Environment," *FC '01: Proc. Fifth Int'l Conf. Financial Cryptography*, pp. 289-304, 2002.

[11] F. Monrose, P. Wyckoff, and A. Rubin, "Distributed Execution with Remote Audit," *Proc. Network and Distributed System Security Symp.*, 1999.

[12] L.F.G. Sarmenta, "Sabotage-Tolerance Mechanisms for Volunteer Computing Systems," *Future Generation Computer Systems: Special Issue on Cluster Computing and the Grid*, vol. 18, pp. 561-572, Mar. 2002.

[13] A. Shamir, "How to Share a Secret," *Comm. ACM*, vol. 22, no. 11, pp. 612-613, Nov. 1979.

[14] D. Szajda, B. Lawson, and J. Owen, "Hardening Functions for Large-Scale Distributed Computations," *Proc. IEEE Symp. Security and Privacy*, pp. 216-224, 2003.

[15] D. Szajda, B. Lawson, and J. Owen, "Toward an Optimal Redundancy Strategy for Distributed Computations," *Proc. IEEE Int'l Conf. Cluster Computing (Cluster)*, 2005.

Bogdan Carbunar received the PhD degree in computer science from Purdue University. He is a principal staff researcher in the pervasive platforms and architectures lab of the Applied Research Center at Motorola. His research interests include distributed systems, security, and applied cryptography.

Mahesh V. Tripunitara received the PhD degree in computer science from Purdue University. He is an assistant professor in electrical and computer engineering at the University of Waterloo, Canada. His research is mostly on information security, with a focus on authorization and access control.