Payments for Outsourced Computations

Bogdan Carbunar
Motorola Labs
USA
carbunar@motorola.com

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

***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. In particular, the bank is not involved in job computation or verification. 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.**

I. 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 highly parallelizable nature of certain computations to distribute sub-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 may not trust the workers to correctly perform computations and workers may not trust outsourcers to provide payments following job completion.

Besides systems for outsourcing computations, open forums that use kudos [2] to rate user posts and establish reputations are also vulnerable to exaggeration attacks. While solutions exist that address the lack of trust that the outsourcer has in a worker (see Section IX), the lack of trust of a worker in the outsourcer 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).

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

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 [12].

Our first solution (see Section IV) requires O to split the key used to obfuscate the payment and hides the subkeys into pre-computed, randomly chosen parts of the job. The worker is entitled to a probabilistic verification of the payment received before beginning the computation. 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 V), 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 entire job: fewer shares need to be discovered.

We address this problem in our final solution (see Section VI). We use exact secret sharing to compute shares of the payment token – all the shares are needed to reconstruct 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.

Our solutions rely on an offline bank: The bank does not have to be online during job outsourcing operations, but only during payment withdrawal and deposit. This ensures that the bank has no involvement in the job outsourcing process. The bank is not required to act as an escrow agent, feature that is essential in ensuring the bank's transparency with regard to the job computation process.

We have tested our third solution on two problems: finding the pre-image of a cryptographic (SHA-1) hash, and the abc conjecture [1]. Our results, described in Section VIII, show that we impose small 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).

II. Model

Our framework is similar to the ones presented in prior work [12], [8]; 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 done in the manner we discuss in Section I, 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 can easily link outsourcers to workers and workers to the jobs they have performed, however, privacy issues are outside the scope of this work. 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. Existing off-the-shelf tools can be used to authenticate participants, encrypt and authenticate messages, thus preventing attacks such as impersonation, eavesdropping, injection and replay attacks.

In the following we adopt the more abstract mechanisms as used in the random oracle model [5]. $G\colon \{0,1\}^* \to \{0,1\}^\infty$ is a random generator and $H\colon \{0,1\}^* \to \{0,1\}^h$ is a random hash function. 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 non-uniform polynomial time [11] adversaries. The function p is public, and p^{-1} is private to B.

III. Ringers - An Overview

The solution from Golle and Mironov [12] (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.

The idea behind ringers is to require the outsourcer to select a small set of random input values from D and to pre-compute 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 pre-images 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 pre-image 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 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.

To express the quality of their solution, Golle and Mironov [12] introduce the notion of a *coverage constant*. The coverage constant (of a set of ringers) denotes the fraction of the job completed by W, given that W is a *rational* cheater. A rational cheater is one that continually assesses the risk-reward trade off between guessing that he has found all the true ringers, and simply completing the entire job. A rational cheater risks stopping his work prematurely if the *payoff* is higher than completing the entire job. We reproduce here Theorem 2 from [12]: The bogus ringers scheme ensures a coverage constant of $1 - \frac{1}{m2^{m+1}} - (\frac{4}{m})^m$.

IV. Payment Splitting Based on Subkeys

We now present our first solution to the simultaneity problem and analyze its properties. Table I summarizes our notations.

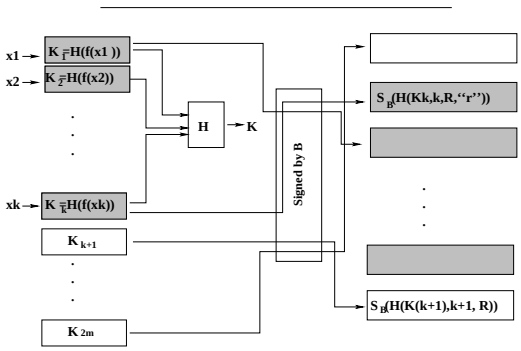


Fig. 1. Payment based on key splitting. The "ringer" subkeys, shown each in a gray rectangle on the left side of the figure, are used to generate the key that obfuscates the payment. The "ringer" subkeys and "bogus" ringers (shown in white rectangles), authenticated by B, are randomly permuted to generate a verification set, shown on the right side, that will later allow W to verify the validity of the payment.

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 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 [12]. 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 $K_1 < ... < K_k$ (if they are not, sort and rename). Let $K = H(K_1; ..; K_k)$ (see Figure 1 for an illustration). 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 $V = \{p^{-1}(H(K_1, R, \text{"}r''\text{})),..,p^{-1}(H(K_k, R, \text{"}r''\text{}))\} \cup \{p^{-1}(H(K_{k+1}, R)),..,p^{-1}(H(K_{2m}, R))\}$ (see Figure 1 for an illustration).
- 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 pre-image $x_j \in D$. W computes $K_j = H(f(x_j))$ and verifies the equality $H(K_j, R, \text{"}r''\text{})=p(Ver(c_j))$. 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 $H(H(f(x)), R, \text{"}r''\text{}) \in p(Ver)$, $SK = SK \cup H(f(x))$ and $p(Ver) = p(Ver) - H(H(f(x)), R, \text{"}r''\text{})$.

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.

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

TABLE I
Notation used in the subkey solution.

- 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. Note that O cannot cancel a payment before the expiration time T of its associated job.

A. Analysis

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 $p(H(K_i, R, \text{“}r''))$ 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 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.

Note that O cannot cancel a job before its expiration time. Otherwise, O could easily cheat by canceling valid jobs and preventing workers from redeeming correctly reconstructed payments.

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

Proof: Consider that during the computation step W can perform the following attack. Before finishing the job, when W discovers a new subkey K_i of K and it has already accumulated more than m subkeys of K, it stops the computation, assumes it has all the subkeys of K and performs the redemption step with B. That is, W guesses the value of k, the number of subkeys of K. If it succeeds to cash the payment, W has successfully cheated O by not performing the whole computation. However, W cannot precisely detect the moment when it has retrieved the last subkey of K. Even if it retrieves K and computes $\sigma_{obf}K^{-1} = p^{-1}(E_{K_s}(t))$, W cannot distinguish this value from a random number, since it does not know the key K_s. Then, given the distribution of k, the number of ringers chosen by O, we use the result of Theorem 2 of [12] to complete the proof. ■

We now propose an attack that the outsourcer can launch and show the defenses provided by our solution.

Invalid share attack: 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 of the attack – the number of “bad” ringers computed by O. We can now prove the following property.

Theorem 2. *If an outsourcer launches an invalid share attack with parameter u, a worker that completes the corresponding job is able to retrieve the payment with probability at least $1 - e^{-uq/(2m-q+1)}$.*

Proof: O generates u out of the k subkeys of K at random. That is, O does not generate u subkeys of K as a hash of the output of the function f applied to one pre-image from D, but uses random numbers instead. Remember that B is not verifying their well-formedness when it signs them, but instead leaves this process for W, in the verification step. The verification phase consists of the random revealing of q out of the $2m$ payment token shares. The probability that any of W's challenges hits an invalid subkey is

$$P = 1 - \frac{2m-u}{2m}\frac{2m-u-1}{2m-1}\cdots\frac{2m-u-q+1}{2m-q+1} = 1 - \frac{\binom{2m-u}{q}}{\binom{2m}{q}}$$

$$> 1 - (\frac{2m-u-q+1}{2m-q+1})^q > 1 - e^{-uq/(2m-q+1)}$$

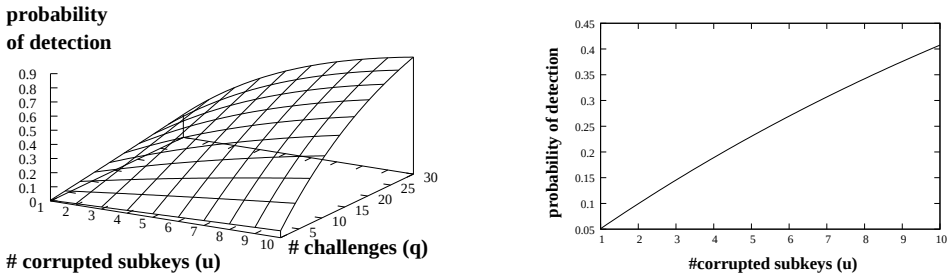


Fig. 2. (a) Probability of detecting a cheating outsourcer as a function of the number of challenges q and the number of "bad" subkeys, u. The value of m is 100, u ranges between 1 and 10 and q between 1 and 30. (b) Detection probability when the number of "bad" subkeys u ranges from 1 to 10 and the number of challenges is 10 (m=100). For u=1, the probability of detection is very small, 5%.

V. Payment Threshold Splitting

Figure 2 shows W's probability of discovering a malicious O that generates u bad subkeys, by challenging O to reveal q subkeys. Note that for large q and u values, this probability quickly approaches 1. However, if $u = 1$, that is, O generates only one "bad" ringer, the chance of W of asking O to reveal the single bad subkey is $q/2m$. For $m = 100$ and $q = 10$, this value is 5%.

Our second solution uses threshold sharing to address this problem. 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. As before, the worker can retrieve the shares only if it covers a large percentage of the job. However, the worker does not need all the shares, but only a predefined subset in order to reconstruct the payment. Then, even if the outsourcer generates p bad shares, the worker can still reconstruct the payment. Table II lists our notations.

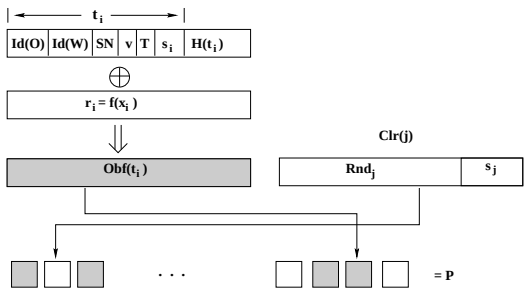


Fig. 3. Generation of Obf and Clr shares. The Obf and Clr values are randomly permuted (lower side in the figure) to generate the payment structure to be sent to W.

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 [18]) 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 $HS = \{H(P_{k+1}), ..., H(P_{2m+p})\}$. 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 $Obf_i = r_i \oplus (P_i; H(P_i))$ (see Figure 3) for an illustration). Let $sz = |Obf_i|$.

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

TABLE II
Notation used in the threshold splitting based solution.

- 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 n 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 in the following manner.

- 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 pre-image 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 non-obfuscated 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 $Shr = Shr \cup s$, $\mathcal{P} = \mathcal{P} - \mathcal{P}(i)$, $Ver = Ver - p^{-1}(H(P))$. That is, 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 $HS = \{H(P_{k+1}), ..., H(P_{2m+p})\}$. 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 non-obfuscated shares that W needs to identify. Initially, $CShr = \emptyset$. W performs the following actions.

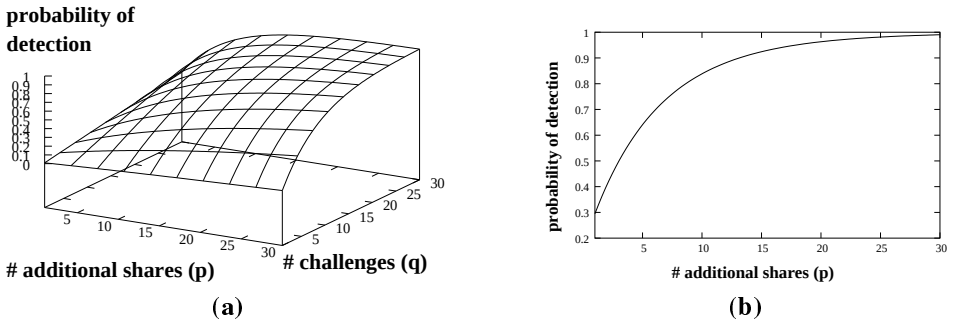
- For each value in $\mathcal{P}$ (there should be $2m+p-k$ elements left), treat the value as if being of format $(Rnd_n; s_n)$, where Rnd_n is a random number and s_n is a payment share. Compute $P_n = \langle Id(O), Id(W), SN, v, T, s_n\rangle$ and look for the hash of this value in the set HS. If 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 then uses all the shares from the set Shr, plus $2m - |Shr|$ shares from $CShr$ to reconstruct the payment P. If it succeeds, it deposits the payment 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.

A. Analysis

Intuition: The set Ver does not contain B's signatures on the cleartext payment tokens, $Clr_{k+1}, .., Clr_{2m+p}$, to pre-



(a) (b)

Fig. 4. (a) Probability of detection of malicious outsourcers as a function of the number of additional shares p and of the number of challenges q, for $m = 100$. (b) Probability of detection as a function of the number of additional shares p, for $q = 30$. For a $p = 30$ and $q = 30$, the probability of detecting an outsourcer that corrupts $p + 1$ shares is larger than 99%. Note that in this case $p = q = 3\sqrt{m}$.

vent 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 as mentioned in the previous solution, O cannot cancel a payment before the expiration time of the associated job and prevent a worker from redeeming the recovered payment.

We first show that this solution is resilient to the invalid share attack.

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

Proof: For the attack to succeed, O must replace at least $p + 1$ legitimate payment shares with bit strings that cannot be used to reconstruct the original payment. We recall that the bank does not verify the well-formedness of the payment shares when it signs them, but instead leaves this process to W, in the verification step. The verification step consists of the random revelation of q out of the $2m$+p payment token shares. The probability that any of W's challenges chooses an invalid payment share is

$$1-\frac{(2m-1)...(2m-q)}{(2m+p)...(2m+p-q+1)} > 1-(\frac{2m-q}{2m+p-q+1})^q =$$

$$1-(1-\frac{p+1}{2m+p-q+1})^q > 1-e^{-(p+1)q/(2m+p-q+1)} >$$

$$1-e^{-c^2/2}$$

Note that the last inequality holds due to the fact that $c\sqrt{m} < p, q$ (see Setup). ■

Figure 4 depicts W's chance of detecting a malicious outsourcer that sends $p + 1$ "bad" shares, when $m = 100$. We note that as the values of p and q increase, the probability quickly becomes close to 1. For instance, even when p=20, for 30 challenges (out of the 220 total shares), the probability of capturing a malicious O that cheats as much as to prevent W from recovering the payment, is larger than 96%. For p=30, the probability becomes larger than 99%. The consequence of Claim 3 is that our scheme is quite robust against the invalid payment shares attack. The worker W is able to detect the attack in the query phase with high probability.

We now propose another attack that the worker can launch.

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 $r \geq \left(\frac{2m-k+x+1}{p}\right)^p$. 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.

VI. Exact Payment Splitting

The first two solutions have two important drawbacks. First, they are either vulnerable to the invalid payment share or the premature payment reconstruction attack. Second, they require heavy bank involvement. We now propose a solution that addresses these problems: it thwarts both attacks while involving B only in the payment generation step. Moreover,

in this solution, only O is involved in binding a payment to a job.

As before, let $G\colon \{0,1\}^* \to \{0,1\}^\infty$ be a random generator and $H : \{0,1\}^* \to \{0,1\}^h$ be a random hash function. We provide concrete instantiations in Section VIII. Table III lists our notational choices.

Setup: The bank, B, has the following.

- A trapdoor permutation, $\langle p, p^{-1}, d\rangle$ that is secure from non-uniform polynomial time [11] 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 $\mathcal{G} = \left\{g^{P_1}, \ldots, g^{P_r}\right\}$.
- 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 $\widehat{\mathcal{K}_i} = G\left(t_{i,1} || \ldots || t_{i,\widehat{i_t}}\right)$, 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:

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

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 $\widehat{\mathcal{K}_r} = G(v_1 || \ldots || v_{i_v})$, 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 VII. 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

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

TABLE III
Notation used in the exact payment splitting solution.

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 [12] – 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 5 in Section VII shows, O must balance this with the risk that W may successfully redeem P without completing the computation.

VII. Security Properties

We now present and prove the security properties of this solution. We do not consider the extensions we discuss in Section VI-B in our proofs, and only consider the original solution from Section VI. 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 4. *An honest W successfully redeems the payment token with probability $1-1/r$, where r is the number of sets of ringers.*

Proof: 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 VI 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 $1/r$. ■

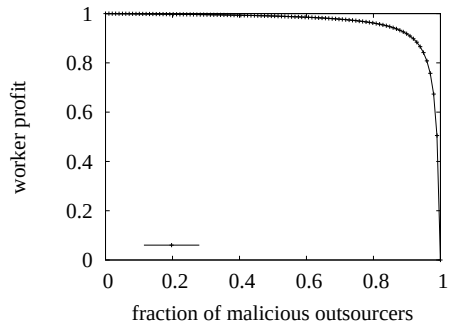


Fig. 5. Effects of Cheating Outsourcers: The profit made by an honest worker when interacting with $f\%$ cheating outsourcers. Even when 80% of outsourcers are cheating, the decrease in profit is around 7%.

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.*

Effects of Cheating: A bad payment that passes the verification step will not be identified by W during the computation. If this were the case, W would be able to also detect when it has revealed the last payment share, stop the computation short and retrieve the payment.

Let n be the number of outsourcers and let f be the fraction of (consistently) dishonest outsourcers. Let m the number of workers. We assume that a worker will interact with a caught dishonest outsourcer only once. Let all payments have the same value, v. Let us assume that a worker receives one job from each outsourcer. Then, fn jobs will contain bad payments and $(1-f)n$ jobs will contain good payments. Then, the worker makes $V=(1-f)nv$ money out of the n jobs.

Let us ignore the cost of verifying a payment. As shown in the evaluation section, this cost is 1.1 seconds even for r=100 and more compute intensive abc-conjecture jobs. Let c be the cost of executing a job. The work the worker does for the n jobs is $C=(1-f)nc+fnc/r$. This is because only fn/r jobs from the fn dishonest outsourcers pass the verification step. Then, the money per job made by the worker is on average

$$Profit=V/C=\frac{(1-f)nv}{(1-f)nc+fnc/r}=\frac{(1-f)v}{(1-f+f/r)c}$$

$$=v/c\times\frac{1-f}{1-f(1-1/r)}$$

The fraction of bad jobs f could be evaluated periodically system-wide: Each worker reports bad payments and proves

them – communication between O and W is over an authenticated channel. Once per epoch (whose length is a system-wide parameter) the value of f is evaluated. The worker's union adjusts the cost to be paid for a job to be $v \times \frac{1+f/r-f}{1-f}$ for the next epoch. Figure 5 shows the decrease in profit made by an honest worker when interacting with $f\%$ cheating outsourcers. The ratio v/c is set to 1 and r is set to 100. The decrease in profit is linear until $f = 80\%$. When $f = 80\%$, the decrease in profit is only 7%.

Note that jobs are likely to have different sizes. The value v for a job should be proportional to the number of CPU cycles required to complete it. Dishonest outsourcers could also prefer to cheat on larger jobs. Note however that the value of r could also be proportional with the job size: for larger jobs the chance of providing a bad payment should be smaller.

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 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}.*

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, $Pr[B\ succeeds] \geq Pr[A\ succeeds] - \epsilon$. ■

VIII. Empirical Evaluation

Our first solution can be used in volunteer computing environments, where outsourcers may be more trusted. Our second solution can be used in cloud computing environments where the cloud providers are more trustworthy. However, given the resilience of our third solution to both cheating outsourcers and workers, as well as its lightweight use of the bank, we believe it should be preferred in most implementations. In this section we investigate the costs imposed by our third 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 our solution and have tested each component on Linux machines with dual core Intel Pentium 4 that clocks at 3.2GHz and 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 $x \in \bar{D}$ 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 $a \in \bar{D_a}, b \in \bar{D_b}$ 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. While periodically changing the system parameters makes sense to enhance security, this issue is beyond the scope of the paper. We note however that changing security parameters needs to be done with care to avoid a situation where the bank rejects valid but outdated payments.

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.

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

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

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 6(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 6(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 [15].

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

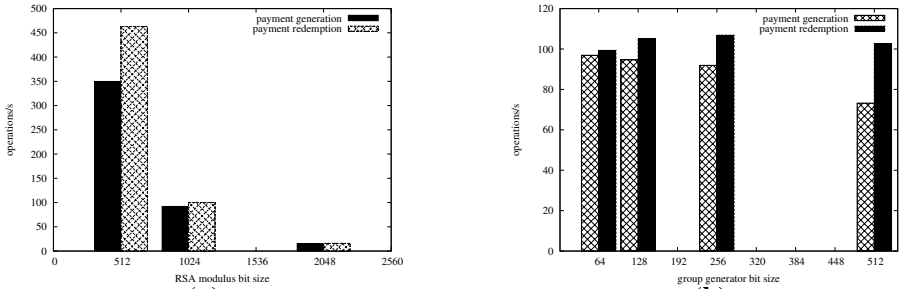


Fig. 6. Bank cost for payment generation and redemption transactions. (a) Performance when the RSA modulus size increases from 512 to 2048. For $N = 1024$ even a simple PC allows the bank to perform 100 of each transaction type per second. (b) Increasing Γ's group order from 64 to 512 bits does not influence 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.

$r = 100$, 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 [12] 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.

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

We implement our solution using up to 100 ringer sets, where the total number of ringers in each set is 10. We limit the job computation time by considering only domains where the largest possible element is 10^6. Figure 7(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.

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

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 7(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.

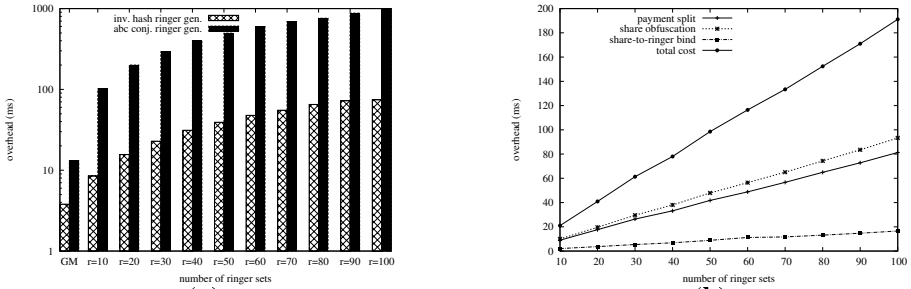


Fig. 7. Outsourcer costs as a function of the number of ringer sets employed. (a) Comparison of costs incurred by the ringer generation step of our solution and Golle-Mironov, for hash inversion and abc-conjecture jobs. Even for 100 ringer sets our solution imposes less then 1s overheads on the outsourcer. (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.

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.

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

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 8(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 [1]. Figure 8(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 8(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 8(c) are actually quite small (the highest value is under 0.3ms) and are due to running the experiments on a real machine.

D. Experimental Conclusions

Our experiments show that our protocol is efficient. First, for standard security parameters, on a single off-the-shelf PC,

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

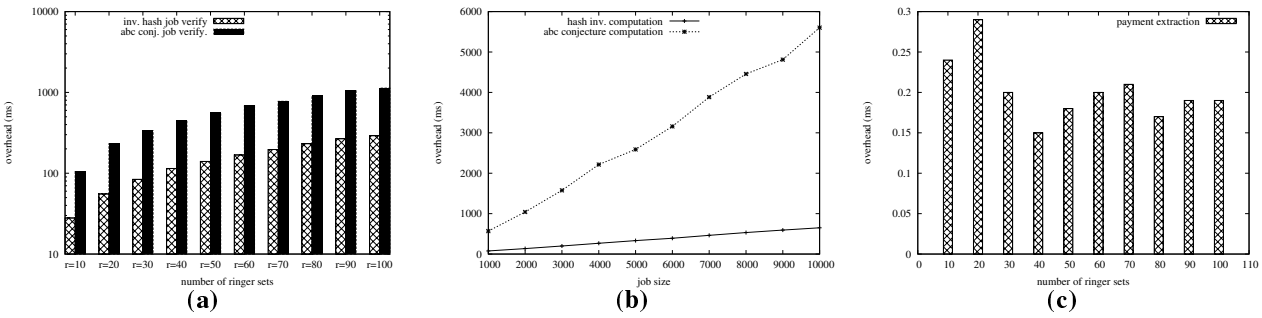


Fig. 8. Worker costs. (a) Job verification cost as a function of the number of ringer sets. Even for 100 ringers sets and the more compute intensive abc-conjecture jobs, our solution takes only 1.1s. (b) Actual computation overhead, function of the input domain D cardinality. Growth is linear and shows 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.

the bank can perform 100 payment generation and payment redemption transactions per second. Both transactions are job independent, making the bank efficient irrespective of job complexities. The traffic created by payment generation and redemption transactions is of 1 respectively 3 packets. Since the bank's overhead is 10 ms for either transaction, the delay incurred by clients during these transactions is likely to be dominated by network latencies (tens of milliseconds).

Second, the overheads imposed by our solution on outsourcers and workers are negligible when compared to overheads of jobs. These overheads consist of job dependent and job independent components. The job independent components are on the order of milliseconds, thus negligible. The costs of job dependent components are determined by the level of assurance needed by both outsourcers and workers: more ringer sets improve the worker's confidence whereas more ringers per set improve the outsourcer's confidence. However, these overheads are very small when compared to the actual job computation costs. In our experiments, the payment associated overheads for either outsourcer or worker are less then 1.5s for abc-conjecture jobs and only a few hundred milliseconds for SHA-1 inversion jobs.

Our solution compares favorably with Golle-Mironov. Even though expected to be slower, our solution introduces very small overheads. This is a small cost to pay for the additional benefit of providing payment redemption assurances to workers.

IX. Related Work

The model we use in this paper for securely distributing computations in a commercial environment is proposed in [16], [13], [12]. Monrose et al. [16] 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 [13] 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 [12] 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. [9] 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. [19] and Sarmenta [17] propose probabilistic verification mechanisms for increasing the chance of detecting cheaters. In the same setting, Szajda et al. [20] 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 [6] propose a solution where workers are rated for the quality of their work by a predefined number of randomly chosen witnesses. This solution addresses not only the selfishness of workers but also the reluctance of outsourcers to provide fair ratings. Belenkiy et al. [4] 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.

Motivated by the need of resource constrained devices, such as RFID tags, to perform (expensive) cryptographic operations, Hohenberger and Lysyanskaya [14] introduce an outsourcing framework where the workers act as cryptographic helpers to dumb devices. This model introduces the additional constraint of making the worker oblivious to the actual computation while still allowing the outsourcer to efficiently verify its correctness.

This paper extends the work of Carbunar and Tripunitara [7] by introducing two new solutions to the simultaneous computation for payment exchange problem. The first solution obfuscates the payment with a key generated from ringers associated with the job. The second solution uses threshold

cryptography to split the payment into multiple shares, where only a subset of the shares is needed to reconstruct the payment. Some shares are then obfuscated with ringers and some are presented in clear to the worker. 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.

On a related note, Gentry et al. [10] introduce the concept of secure distributed human computations. While computers are still employed to solve large, difficult problems, humans can be used to provide candidate solutions for problems that are hard for computers (e.g., image analysis or speech recognition). This work proposes the use of payouts not only as a reward for solving problems, but also in the reverse manner. That is, humans could be asked to solve simple problems (image labeling, CAPTCHA solution gathering, proofreading short texts, etc) as payment for small Internet services.

X. 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.

XI. 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/.

[2] About SophosFree Talk. http://openforum.sophos.com/t5/About-SophosFreeTalk/FAQ-how-does-the-kudos-system-work/m-p/6.

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

[4] 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.

[5] 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.

[6] 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.

[7] B. Carbunar and M. Tripunitara. Fair payments for outsourced computations. In *Proceedings of the 7th IEEE International Conference on Sensor, Ad Hoc and Mesh Communications and Networks (SECON)*, 2010.

[8] B. Carbunar and M. V. Tripunitara. Conditional payments for computing markets. In *Cryptology and Network Security — CANS 2008*, pages 317–331. Springer Verlag, LNCS 5339, December 2008.

[9] 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.

[10] C. Gentry, Z. Ramzan, and S. G. Stubblebine. Secure distributed human computation. In *ACM Conference on Electronic Commerce*, 2005.

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

[12] 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.

[13] 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.

[14] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC*, 2005.

[15] 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.

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

[17] 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.

[18] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, Nov 1979.

[19] 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.

[20] 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.