

Secure Distributed *Human* Computation

Abstract. This paper introduces a line of research on secure distributed *human* computation. We consider the general paradigm of using large-scale distributed computation to solve difficult problems, but where humans can act as agents and provide candidate solutions. We are especially motivated by problem classes that appear to be difficult for computers to solve effectively, but are easier for humans to solve; for example, image analysis, speech recognition, and natural language processing. This paradigm already seems to be employed in a number of real-world scenarios, but we are unaware of any attempt to formally study it. Nonetheless, this concept spawns interesting research questions in cryptography, algorithm design, human computer interfaces, and programming language / API design, among other fields. Furthermore, there are interesting implications for internet commerce and the B24b model. In this paper, we introduce this area of research and put forth a basic framework and architecture for the secure design of such systems. We analyze security and reliability against malicious parties using standard tools from probability theory. We then derive design principles using standard game-theoretic concepts. Finally, we list various extensions and open problems.

“Now we have in our hands a method for going beyond the computed, leapfrogging it, passing through it. We will combine the mechanics of computation with human thought; we will have the equivalent of intelligent computers, billions of them. I can’t predict what the consequences will be in detail, but they will be incalculable.”

Congressman Brant, from Isaac Asimov’s *Feeling of Power*; taken from [3].

1 Introduction

LARGE-SCALE DISTRIBUTED COMPUTATION. The Internet enabled the possibility of creating a giant distributed computer system by harnessing idle CPU cycles on machines worldwide. Projects aimed at exploiting such resources have included SETI@HOME’s search of extra-terrestrial life through large scale data mining [7], Entropia.com’s GIMPS project for finding large Mersenne Primes [9], and Distributed.Net’s attempt to break cryptosystems via exhaustive key search. Nonetheless, there are numerous everyday problems that traditional computers struggle with despite the amount of computational power allocated toward solving them; e.g., so called “AI-complete” problems which occur in fields such as image analysis, speech recognition, and natural language processing. Remarkably, many of these everyday problems can be effortlessly solved by humans.

INCORPORATING HUMAN COMPUTATION. Motivated by such problems, this paper embarks on a formal study of secure distributed *human* computation (DHC). Here, *human* end clients are working to solve particular problems that are still difficult for computers executing state-of-the-art algorithms. At first glance, this notion of distributed human computation might sound far-fetched. However, it turns out that there are a number of situations today that exemplify this paradigm.

- **Collaborative Filtering for Spam Prevention:** A number of spam prevention mechanisms such as SpamNet [15] and Vipul’s Razor [18] use human votes to determine if a piece of email is spam; these ideas are extended to the P2P setting in [19]. This approach may be effective given that humans can more easily identify junk mail than computers can; e.g., [19] claims 0% false positives in experiments. Here, each email recipient presses a spam button if it receives what it considers to be spam. If enough people vote that a given email is spam, then it is flagged as such, and either not delivered to other users, or delivered into a special spam folder.

- **CAPTCHA Solution Gathering:** Although DHC can help eliminate spam, many spammers have used forms of it to actually further their goal. Consider, for example, free email providers who have incorporated special puzzles, known as CAPTCHAs, that are easily solved by humans, but challenging for computers, during the account creation phase in order to prevent spammers from automatically creating thousands of email accounts; spammers, in turn, have farmed these CAPTCHAs out to humans in exchange for access to illicit web content [1], [2].
- **Automated Check Cashing Machines:** Cyphermint, Inc. [5] presently builds kiosks that are placed in convenience stores which allow people to automatically cash checks. The kiosk essentially validates a check by taking a picture of person cashing it and sending this picture electronically to a central office whereupon a human worker compares the picture to the one that was previously stored. If the two pictures correspond to the same person, then the check is cashed. Otherwise it is rejected.
- **Medical Transcription Services:** Today many doctors dictate their medical notes verbally via a standard telephone interface to a system which records the notes, and transmits compressed audio offshore using the Internet. At this point, a human listens to the audio, transcribes the note, and sends it back electronically.
- **MOS calculation:** In Internet telephony (as well as other areas of voice research), the mean opinion score (MOS) is a subjective numerical measure of the quality of human speech at the receiver. The scheme involves giving samples to different humans; the quantitative score then becomes a weighted average of the different responses. Here, the “hard problem” is that it is not clear how to use a computer to measure sound quality; i.e., there is no known algorithm for accurately computing it.

One can also view open source software development as falling into the DHC paradigm in a loose sense. Here the difficult problem is not something crisp like image or speech recognition, but instead that computers have a hard time automatically generating source code. Similarly, building Wikis is also related to this area. Recall that Wikis are online encyclopedias that are written by internet users; the writing is distributed in the sense that essentially almost anyone can contribute to the Wiki if they so desire.

Beyond these examples, one can imagine other potential scenarios. Consider, for example, the New York Times web site [13]. It provides free access to the day’s news articles, but a fee is charged if users desire archived articles. Such a fee may deter most users – especially since it may be possible to obtain the article text from some other Internet location; e.g., someone may have (illegally) posted it to a newsgroup or mailing list. However, instead of charging a fee, the New York Times could give the user an AI-complete problem that it might want solved; in exchange for solving the problem, the archived article can be provided. This concept extends to other service offerings; e.g., one free minute of long-distance cell phone service per problem solved.

APPLICATIONS TO INTERNET COMMERCE AND B24B. Web sites today typically rely on three revenue sources: advertisements, subscription fees, and e-commerce. Outside of e-commerce, it is challenging to earn sustainable revenue from the first two sources. Indeed, click-through rates on advertisements are abysmally low (around 0.7%) [6], and outside of specific niche industries, very few people are willing to pay subscription fees for premium internet content. We believe that the notion of distributed human computation yields another revenue source; namely, companies who want specific problems solved can farm them out to the hundreds of millions of internet users. In exchange for solving the problem, some service or good is provided. In a sense, then, human cycles rather than CPU cycles, essentially form a new type of online currency.

DHC may also enable the Business-to-Four-Billion (B24b) model [14] which aims to provide digital services (wireless communication, internet, etc.) to the world’s four-billion poorest people. Individually these people have annual incomes less than \$1500 – yet their collective buying power is quite large. Presently, it remains to be seen if such an endeavor is economically feasible. However, instead of cash transactions, one may provide services in exchange for solving DHC problems.

RELEVANCE TO CRYPTOGRAPHY AND OTHER FIELDS. The subject studied in this paper is relevant to a number of research disciplines. With respect to cryptography, one can view this area as a type of secure multi-party computation – but with a few differences from the traditional use of this term in the literature. First, the parties are human beings instead of computers; second, the parties are themselves not providing actual inputs, but are instead providing candidate answers (which themselves can be construed as inputs into a group decision-making process); third, the computation is facilitated by a server that is trusted,¹ but computationally weak in the sense that it is not capable of providing answers itself; fourth, we may not in general be restricted by privacy concerns, although they are important in a number of motivating applications. To analyze security, we may consider the case where the adversaries are rational, and use game-theoretic tools. Similarly, since DHC is a form of currency, there are relations to e-cash, which is studied in cryptography. Finally, we remark that much of the related work to be discussed below ([10], [11], [2], [1]) has appeared in cryptographic literature. In addition, there are interesting implications for other fields:

- **Algorithms:** How would one go about designing algorithms to solve problems knowing that the algorithm could receive assistance or hints from human beings?
- **Human-Computer-Interaction:** What kinds of interfaces should one design for enabling humans to solve problems most effectively? In the simplest case, we may imagine that a human is sitting at a desktop. However, we may also want to consider a number of other settings such as solving problems using a mobile phone, or solving problems while driving to work. The interface may not necessarily be a computer screen, but instead the process could be voice driven.
- **Programming Language / API design:** How would one design programming languages or APIs if part of the computation is being performed by a human? One may even desire a meta-language for incorporating such computation.

RELATED WORK. One close line of related work is in the field of secure wide-scale distributed computing with payout (DCP); see Golle-Mironov [10], and Golle-Stubblebine [11]. Both papers consider how one can make large-scale distributed computing (when payouts are issued in exchange for work) more secure. Many of the ideas and analysis have analogues in the DHC setting. Of course, there are fundamental differences. First off, in DCP computers provide all answers, so the assumption is that they are either correct with probability 1, or else are cheating. In DHC, it is possible for there to be an honest mistake, or for there to be no clear cut “correct” answer to a given problem. Second, DCP gives payouts to all computers that participate (so long as they are not cheating). In the DHC, it does not seem to make sense to pay everyone – instead, it seems that one should only pay clients who clearly provided a correct answer. Third, in DCP, all clients can typically compute the same functions; this does not apply to DHC where human agents can have differing skill sets; e.g., to solve a text translation problem, one must have abilities in two foreign languages. Another line of related work is in CAPTCHA’s [2], [1]. Here, one is interested specifically in problems that humans can solve easily, but which computers cannot. In essence, the problems we consider for our human distributed computation setting are like CAPTCHAs, but with two exceptions:

¹ Server trust can be minimized by augmenting a DHC system with a voter and results verifiable voting protocol.

1. It may not be possible to automatically generate problems.
2. The solution may not be known beforehand, but it must be inferred from the answers given.

OUR CONTRIBUTIONS. The primary contribution of this paper is to introduce what appears to the best of our knowledge to be a new line of research. We further suggest a basic framework and corresponding architecture for an example DHC system. We use basic probability tools to analyze how many malicious parties such a system can tolerate (as a function of how well the honest parties perform). Next, we use some basic game theory tools (such as the notion of a utility function) to derive design principles for a secure distributed human computing system. We also identify open issues and areas for future work. We remark that since the primary aim of the present paper is to introduce a new area of research, we have not attempted (nor do we believe we should attempt) to solve every possible problem that arises. Instead, we hope that our paper will inspire further research – either by improving upon or extending the observations we made, by addressing the open issues we have suggested, or by identifying scenarios that fit within the general human computing paradigm, but which have not been mentioned by the present paper.

PAPER ORGANIZATION. The next section gives the framework and an architecture. Section 3 gives analysis. Section 4 makes concluding remarks and suggests open problems.

2 Framework

ORGANIZING PRINCIPLES. The principals in a secure distributed human computing system (SD-HCS) are the problem broker or *distributor*, *clients*, problem suppliers or *suppliers*, and online sites or *storefronts*. The client desires a product or service provided by the storefront at some cost. The storefront allows clients to pay for the goods by solving an appropriately selected human computation problem. The distributor operates and manages a system for taking problems from problem suppliers and making them available to clients. See figure 1 for a high-level depiction. Between clients and the distributor, there is an interactive protocol whose aim is to obtain the correct answer with high probability. The protocol may be in one of two high-level phases: a registration phase and an operation phase. The latter phase is more involved and incorporates scheduling, monitoring, payout, account management by the user, and account management by the system. We describe each of the underlying components in turn.

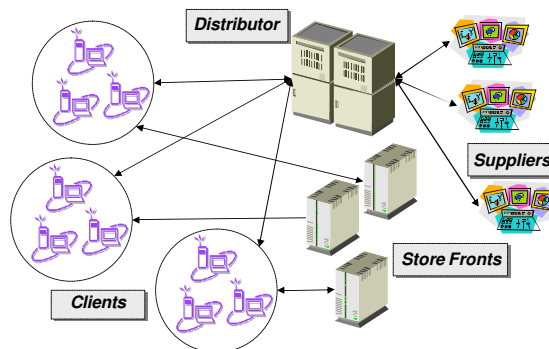


Fig. 1. A high-level depiction of an example secure distributed human computing system.

REGISTRATION PHASE. When a client first decides he would like to participate in the SDHCS, he goes through a registration phase. Here he provides basic registration info (name, etc.). Thereafter, he goes through a training phase. The first part would involve explaining to the client what task needs to be done, and the second phase involves testing the client on actual problem data. A first portion of this data may be generated specifically for testing (i.e., the answers to the associated problems are already known); this portion gives a fairly accurate measure of the client’s ability. A second portion of this data would be generated from live problem instances; this portion makes it difficult for clients to build dictionaries of known problem answers, and allows the system to get some free work from its clients – whether the answer is deemed correct is determined from the answers provided by other already registered clients in the system. The training phase may also incorporate CAPTCHAs [1] [2] to ensure that the client is not running a software agent for automatic account creation. At the end of the registration phase, the client is given a rating (corresponding to an estimated likelihood that it answers problems correctly). In addition to training clients and estimating their accuracy, the training phase also serves as an opportunity cost to discourage clients from establishing new accounts after abusing any current account they have.

OPERATION PHASE. This phase consists of several processes: scheduling, monitoring, payout, account management (by the system or by the user), and termination (either user or system initiated).

- Scheduling Process: A set of problems is provided by the supplier. These problems may be provided at once, or may be streamed. The scheduler determines which clients should receive the problem. Based on the individual answers, the scheduler computes, for each possible answer, the probability that this answer is correct. The scheduler may choose to continue assigning the problem to other users. The scheduler must operate within a problem budget.
- Monitoring Process: This process tries to prevent or detect any undesirable behavior such as cheating. For example:
 - Monitor the user’s payout account to see if there is a minimum balance – we justify the need for a minimum balance in the utility function analysis of section 3.
 - Monitor the user success rate to see if there are any sharp drop offs.
 - Determine if multiple account creation requests stem from the same source.
- Payout: Once a problem is deemed answered, clients are paid. On the one hand, we would like to pay out for “correct” answers – where correctness can be ascertained using Bayesian inference, weighted majority vote, or some other mechanism. On the other hand, it may be possible that there is no preferred answer; i.e., the votes are evenly divided. Payout must be made accordingly. The payout function should be designed to pay only for “reasonable” answers (those that seem consistent with others’ answer) as opposed to unreasonable ones (those that seem to stem from cheating or answering randomly). Additionally, as we will see in the utility function analysis of section 3, one should design the scheme so that the payout amount is proportional to the perceived skill level of the user; i.e., if a user tends to get many correct answers, we pay him more. This gives users incentive to answer reliably and therefore deters cheating.
- Account Management: In general, the account management process addresses any adjustment of the long-term variables:
 - Skill levels of users can be updated in accordance with their performance.
 - Users may specify the types of problems they prefer to solve; e.g., some users may prefer image analysis problems whereas others may prefer natural language processing. Similarly, users may indicate specific skills; e.g., if they have fluency in several languages, the distributor may wish to provide them problems involving text or speech translation.

- A user’s account may be terminated; the termination request may either come directly from the user or it may be initiated by the distributor itself. In the former case, we may want pay out the balance. In the latter case, we may want to investigate the behavior which resulted in termination; for example, if the user was caught cheating, then the final payout may be withheld.

THREATS. Threats to an SDHCS include those directed to the registration phase, operation phase, or some combination of the two phases. The registration phase is subject to automated attacks possibly resulting in new user accounts that might be exploited later. CAPTCHAs have been used to counter online scripted attacks [1, 2]. However, CAPTCHAs fall short from protecting against semi-automated attacks such as those where an unsuspecting person is lured into assisting an attack (i.e., relay attacks). Recent research results give specialized protocols using CAPTCHAs that increase protection against relay attacks [17]. Threats to the operation phase include a *lurking adversary* who is benign until he attacks. By their nature, lurking adversaries are difficult to detect before they strike. The situation of a lurking adversary may be more severe in the context of DHC because a few malicious act might be attributed to human error. An *inconsistent adversary* responds to tasks in an inconsistent manner.² Our approach generally detects the inconsistent adversary since we are able to monitor the accuracy of the adversary over various time windows. We remark that there is also potential for the distributor to cheat; e.g., by claiming that solutions were wrong and not providing payout. This kind of problem can be alleviated by augmenting a SDHCS with a voting protocol that provides voter and results verification (by either the individual voter or by anyone). See, for example, [4]

ARCHITECTURE. The particular architecture for our problem distribution system depends on the media capabilities of the client device (e.g., text only web based, audio only cell phone, or some combination of visual and audio). For a web-based access device, the architecture can be similar to the system of brokering interactive advertisements on the web. Storefronts provide a URL associated with a problem “payment” mechanism associated with human computation. Upon selecting the URL, the client passes a cookie to the problem distribution network if it is registered and the client receives a human computational problem. (Otherwise, the client is required to go through the client registration process.) In response to receiving the client cookie, a human computation problem is directed to the client. The client responds back to the distribution service with a solution. Upon completion, the storefront receives an authenticated confirmation (possibly passed through the client) that the client has made sufficient “payment” and the client receives the product or service.

The distribution network consists of computers that primarily host problems and schedule these problems to clients. The scheduler identifies the next problem to be distributed and the associated skill level with the problem. Upon request of a problem by a client, a process retrieves the skill level and validity of the client. The client is passed a URL associated with next available problem as determined by the scheduler.

3 Analysis

In this section, we do a simple analysis of the reliability of client voting in an SDHCS. The analysis uses basic probability tools and considers the presence of malicious parties – we consider both the

² As an example, an inconsistent adversary might simply be lazy, truly inconsistent, or may even employ expert programs in an attempt to answer correctly.

use of majority vote and Bayesian inference to combine answers from different clients. We then use game and decision theory tools to help derive design principles.

PRELIMINARIES. Suppose that we have a problem Π with t possible answers. Without loss of generality, let us denote these answers by the numbers 1 through t – for simplicity, we usually consider the case $t = 2$; furthermore let P be a discrete random variable whose value is equal to the actual answer. We assume that the prior distribution on P is known; i.e., for all $i \in [1, t]$, we assume that we already know $\Pr[P = i]$. Denote this value by π_i . Moreover, suppose Π is looked at by s different clients c_1, \dots, c_s , and each gave you an answer. Let $C_i \in [1, t]$ denote a discrete random variable representing the answer provided by c_i on problem P (for $1 \leq i \leq s$). Let $\mathbf{C} = (C_1, \dots, C_s)$. Let $p_i = \Pr[c_i \text{ is correct}]$, $1 \leq i \leq s$. Now, for a problem Π and a client c_i , we define:

$$X_i = \begin{cases} 0, & \text{if } c_i \text{ gave the incorrect answer on } \Pi; \\ 1 & \text{otherwise.} \end{cases}$$

Moreover, suppose that each c_i gave back answer α_i . Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_s)$. We further assume that these responses are independent of each other.³ We can actually model independence by conservatively assuming that adversaries give the wrong answer with probability 1.

MAJORITY VOTE. We now consider the approach of answering using majority vote. We use the simplified Chernoff-Hoeffding bound presented in [12].

Theorem 1 (Chernoff Bound). *Let X_1, \dots, X_n be the outcomes of independent Bernoulli trials where for each $i \in [1, n]$, $\Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^n X_i$, and let $\mu = E[X]$ ($= \sum_{i=1}^n p_i$). Let $0 < \delta \leq 1$. Then: $\Pr[X < (1 - \delta) \cdot \mu] < e^{-\mu\delta^2/2}$.*

Now, it follows that $\Pr[X_i = 1] = \Pr[c_i \text{ is correct}] = p_i$, for $i \in [1, n]$. Let $\bar{p} = 1/n \cdot \sum_{i=1}^n p_i$. Note then that $E[X] = n\bar{p}$. So, the probability that the majority vote strategy fails is:

$$\Pr[X < n/2] = \Pr[X < \frac{1}{2\bar{p}} \cdot n\bar{p}] < e^{-n\bar{p}(1-1/2\bar{p})^2/2}$$

For $\bar{p} > 1/2$, the probability of an incorrect answer decreases exponentially in n . Also, note that $\lim_{p \rightarrow 1} \Pr[X < n/2] < e^{-n/8}$. If on average we have advantage β over guessing (i.e., if $\bar{p} = 1/2 + \beta$, for $0 < \beta \leq 1/2$), then $\Pr[X < n/2] < e^{-n\beta^2/(1+2\beta)}$. Let us consider the case where there are h honest parties. Moreover, suppose that the honest parties have an average probability p_h of providing the correct answers. Let us also conservatively estimate that malicious parties answer incorrectly with probability 1. Then, to maintain the exponentially decreasing property, we require that $h > n/2p_h$. As p_h approaches 1, we require that the majority be honest. We remark that requiring an honest majority is not unreasonable. In fact, the protocols in the literature for traditional secure general multiparty computation require honest majorities [8].⁴ Note that in the case of human computation problems, we expect honest parties to provide answers with probability very close to 1 (since such problems are usually very easy for humans, but much more difficult for machines); therefore not much more than an honest majority is required. Also note that if at least two-thirds of the clients

³ In the presence of a malicious adversary, this may not be true since the adversary might try to control several clients; however, if in the worst case, malicious adversaries get clients under their control to provide only incorrect answers, then the answers are technically independent.

⁴ Technically speaking, secure general multiparty computation protocols require such a bound on the number *active adversaries*; one can usually tolerate even more passive (i.e., honest but curious) adversaries.

are honest, then the probability with which these clients must supply correct answers only has to be above $3/4$. Now, another interesting question to ask is the fraction of votes one needs to ensure that the probability of giving the incorrect answer is at most ϵ . It turns out that the fraction

$$\bar{p} \cdot \left(1 - \sqrt{\frac{2 \ln 1/\epsilon}{n\bar{p}}} \right)$$

is sufficient. This can be derived by solving for δ as a function of ϵ in the Chernoff Bound equation.

BAYESIAN INFERENCE. Bayesian inference is a standard method for combining classifiers – in this case, the individual clients are “classifiers” or experts. We can use it to estimate $\operatorname{argmax}_{j \in [1,t]} \Pr[P = j | \mathbf{C} = \boldsymbol{\alpha}]$. The underlying probability can be computed as follows:

$$\Pr[P = j | \mathbf{C} = \boldsymbol{\alpha}] = \Pr[\mathbf{C} = \boldsymbol{\alpha} | P = j] \cdot \frac{\pi_j}{\Pr[\mathbf{C} = \boldsymbol{\alpha}]} = \frac{\Pr[\mathbf{C} = \boldsymbol{\alpha} | P = j] \cdot \pi_j}{\sum_{k=1}^t (\Pr[\mathbf{C} = \boldsymbol{\alpha} | P = k] \cdot \pi_k)}$$

Let’s see why this can be computed. Let ρ_{ij} be defined as follows:

$$\rho_{ij} = \begin{cases} p_i, & \text{if } C_i = j; \\ 1 - p_i, & \text{otherwise.} \end{cases}$$

Note that ρ_{ij} can be computed given \mathbf{C} . Now, $\Pr[\mathbf{C} = \boldsymbol{\alpha} | P = j] = \prod_{i=1}^s \rho_{ij}$. Therefore,

$$\Pr[P = j | \mathbf{C} = \boldsymbol{\alpha}] = \frac{\Pr[\mathbf{C} = \boldsymbol{\alpha} | P = j] \cdot \pi_j}{\sum_{k=1}^t (\Pr[\mathbf{C} = \boldsymbol{\alpha} | P = k] \cdot \pi_k)} = \prod_{i=1}^s \rho_{ij} \cdot \frac{\pi_j}{\sum_{k=1}^t ((\prod_{i=1}^s \rho_{ik}) \cdot \pi_k)}$$

BAYESIAN INFERENCE WITH ADVERSARIES. We now consider what happens in the presence of malicious classifiers; i.e., where clients deliberately give erroneous answers. We are unaware of any previous work in this setting. This may be especially devastating if the malicious clients have hitherto performed well, and thus given us reason to believe in them. Consider the standard Bayesian formula for determining if the answer to a problem Π is 0 (when the only answers are 0 or 1):

$$\Pr[P = 0 | \boldsymbol{\alpha}] = \frac{\Pr[\boldsymbol{\alpha} | P = 0] \cdot \pi_0}{\Pr[\boldsymbol{\alpha} | P = 0] \pi_0 + \Pr[\boldsymbol{\alpha} | P = 1] \pi_1} \quad (1)$$

Suppose that 0 is indeed the correct answer, and that we have m malicious parties and h honest parties. For simplicity, suppose that the malicious parties trick us into thinking that they give a correct answer with probability p_m (e.g., they might perform well initially, then deviate; alternately, star performers might be sought out and bribed to encourage cheating). Let p_h denote the probability with which the honest parties are correct. Finally, suppose that among the h honest parties, h_r give the right answer, and h_w mistakenly give the wrong answer. We are interested in the situation when equation 1 is greater than $1/2$ (this is the case when the distributor can determine the correct answer *despite* the presence of malicious parties):

$$\begin{aligned} & \frac{p_h^{h_r} (1 - p_h)^{h_w} (1 - p_m)^m \cdot \pi_0}{p_h^{h_r} (1 - p_h)^{h_w} (1 - p_m)^m \pi_0 + (1 - p_h)^{h_r} p_h^{h_w} p_m^m \pi_1} > 1/2 \\ \iff & p_h^{h_r} (1 - p_h)^{h_w} (1 - p_m)^m \pi_0 > (1 - p_h)^{h_r} p_h^{h_w} p_m^m \pi_1 \\ \iff & \left(\frac{p_h}{1 - p_h} \right)^{\Delta_h} > \left(\frac{p_m}{1 - p_m} \right)^m \frac{\pi_1}{\pi_0} \end{aligned}$$

Where $\Delta_h \stackrel{\text{def}}{=} h_r - h_w$. If we assume $\pi_1 \geq \pi_0$,⁵ then a sufficient condition for security is:

$$\frac{\Delta_h}{m} > \frac{\log(p_m) + \log(1 - p_m)}{\log(p_h) + \log(1 - p_h)} \quad (2)$$

In the case that $p_m = p_h$ (in which case for $\pi_0 = \pi_1$, Bayesian classification of a binary value is equivalent to standard majority vote), then the security condition is reduced to $h_r > m + h_w$, which is consistent with our intuition that if everyone performs identical, then the number of honest and correct people have to be in the majority. The expected value of h_w is $(1 - p)h$, so if p_h is high, then with high probability, h_w is small, in which case a little more than an honest majority is sufficient for security. We now analyze equation 2 to better reason about it. Let us assume that p_m is very high (i.e., close to 1). We wish to consider the less desirable case that $p_m > p_h$. For a small real number $\delta \lll 1$, and a real number $k > 1$, we can denote:

$$\begin{aligned} p_m &= 1 - \delta \\ p_h &= 1 - k\delta \approx (1 - \delta)^k = p_m^k \end{aligned}$$

Then, the right hand side of equation 2 reduces to:

$$\frac{\log(p_m) + \log(1 - p_m)}{\log(p_h) + \log(1 - p_h)} \approx \frac{\log(p_m) + \log(\delta)}{k \log(p_m) + \log(k\delta)} \approx \frac{\log \delta}{\log k + \log \delta}.$$

Which follows since p_m is close to 1, making $\log(p_m) \approx 0$. Now, let $\rho = \Delta_h/m$ denote the ratio of honest and correct clients to dishonest clients. Then, a sufficient condition for security is: $k^\rho = \delta^{1-\rho}$. For $\rho = 2$ (i.e., a two-thirds majority), we get $k = \sqrt{1/\delta}$. If, for example, $\delta = 0.01$, then $k = 0.1$. This yields $p_m = 0.99$ and $p_h = 0.9$. That is, even if the malicious parties have thus far outperformed the honest parties by 99% to 90%, a two-thirds majority of honest and correct clients is sufficient for obtaining the right answer. This makes intuitive sense since receiving an answer from one person who is right 99% of the time is as useful as receiving two (identical) answers from two independent sources who are right 90% of the time; i.e., $1 - (1 - 0.9)^2 = 0.99$. Note that for the normal majority vote setting, a two-thirds honest majority only needs to be correct with probability greater than 3/4 (and the likelihood of an error goes down exponentially in the number of users asked). In retrospect, it is not surprising that majority vote can outperform Bayesian inference in the presence of malicious parties. After all, Bayesian inference does a weighted combination whereas general majority vote does not – if malicious parties have performed well up to a certain point, then their vote may be more heavily weighted in the Bayesian case.

ANALYSIS USING UTILITY FUNCTIONS. We now use some basic tools from game and decision theory to derive design principles for an SDHCS. We assume that the human clients are rational and either risk averse or risk neutral, but not risk seeking. We can think of a distributed human computing process as a multi-party game.⁶ The *actions* constitute the clients' responses on the given problem, and the *utility* corresponds to the benefit from taking the given action. Below we use utility and expected utility interchangeably. If client c_i makes an earnest attempt to answer the

⁵ Note that this is the more conservative estimate; if $\pi_0 > \pi_1$, then we are in a situation where the more likely prior is correct – in which case fewer honest clients are needed to derive the correct outcome.

⁶ In a technical game-theoretic sense, however, the model in our exposition is *under-specified* since we have not demonstrated an equilibrium – however, this is not a limitation since our present goal is merely to derive design principles rather than a prediction or a preference for a particular strategy.

problem correctly, we say that he cooperates; otherwise, we say that he defects. Let $\mathcal{C}_i(\Pi_t)$ denote the reward that c_i ($i \in [1, n]$) receives for cooperating on problem Π_t issued at time t , and let $\mathcal{D}_i(\Pi)$ denote the reward for defecting. Then, we are interested in ensuring that $\mathbf{E}[\mathcal{C}_i(\Pi)] < \mathbf{E}[\mathcal{D}_i(\Pi)]$ for a sufficiently large number of clients. We let \mathcal{A}_i denote a function that computes c_i 's ability; this function takes as input a history \mathcal{H} corresponding to the answers given by all the clients on the various problems. For a given problem, we have a budget \mathcal{B} corresponding to how much can be paid out (in terms of money or other services) for answers. We have a payment function $\mathcal{P}(\mathcal{A}(\mathcal{H}_i), \Pi, \mathcal{B})$, known to all clients, that computes the amount paid to client i for a given answer. We only consider the payment function in situations where c_i provided what is believed to be the correct answer. When it is clear from context, we will drop the arguments and subscripts associated with $\mathcal{C}, \mathcal{D}, \mathcal{P}$. Now, one simple way to model expected utility from cooperation is:

$$\mathbf{E}[\mathcal{C}_i(\Pi)] = \mathcal{P}(\mathcal{A}_i, \Pi, \mathcal{B}) \cdot p_{i,\Pi} + \sum_{\tau>t} \mathbf{E}[\mathcal{C}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{C}_i(\Pi_\tau) \mid \mathcal{H} = H'], \quad (3)$$

where H' is an alternate history in which c_i defects on problem Π . This formulation captures:

1. the expected reward $\mathcal{P}(\mathcal{A}_i, \Pi, \mathcal{B}, \kappa) \cdot p_{i,\Pi}$ for answering Π correctly (with probability $p_{i,\Pi}$), and
2. the incremental benefit $\sum_{\tau>t} \mathbf{E}[\mathcal{C}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{C}_i(\Pi_\tau) \mid \mathcal{H} = H']$ associated with future payments since these are a function of $\mathcal{A}(\mathcal{H}_i)$ which would be modified if c_i decided to defect (and provided the wrong answer).

Of course, there may be some ancillary benefit for answering correctly; for simplicity, we ignore that in the analysis. Letting R denote the balance in a client's account, letting p denote the probability that it gets caught, and letting P denote what it might gain from cheating, we have that:

$$\begin{aligned} \mathbf{E}[\mathcal{D}_i(\Pi)] &= p \cdot (P - R) + (1 - p) \cdot P \\ &= P + (1 - P) \cdot R \end{aligned} \quad (4)$$

Note that we implicitly assume that if a client cheats, then it has provided an *incorrect* answer, in which case the distributor pays nothing to the client. Moreover, if a client is caught cheating, then it is kicked out and it loses whatever balance has been accumulated. Now, equations 3 and 4 (as we have stated them) are fairly easy to derive and understand. Yet, these equations provide insight into SDHCS design. Since we want $\mathbf{E}[\mathcal{C}_i(\Pi)] < \mathbf{E}[\mathcal{D}_i(\Pi)]$, we would like to design the SDHCS to make $\mathbf{E}[\mathcal{C}_i(\Pi)]$ as large as reasonably possible and make $\mathbf{E}[\mathcal{D}_i(\Pi)]$ as small as reasonably possible. In the former case, we can control the incremental benefit $\sum_{\tau>t} \mathbf{E}[\mathcal{C}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{C}_i(\Pi_\tau) \mid \mathcal{H} = H']$ in equation 3 through two measures:

1. We should make the payout an increasing function of a client's ability. As the ability increases, so should the payout it receives. This increases the incremental benefit, which increases $\mathbf{E}[\mathcal{C}_i(\Pi)]$, and thereby discourages defection.
2. We should penalize the ability heavily for a mistake and offer only a smaller increase in ability for correct answers. Therefore, if a client gives an incorrect answer, it will take longer to get its ability back to where it was before this answer was given. This also increases the incremental benefit, which increases $\mathbf{E}[\mathcal{C}_i(\Pi)]$ and discourages defection.

Similarly, from equation 4, we observe that decreasing $\mathbf{E}[\mathcal{D}_i(\Pi)]$ increases R . This, in turn, can be accomplished through the use of a minimum reserve requirement; that is, we can discourage cheating by requiring that a client maintain a minimum balance at all times. Finally, we note that

equation 4 increases with P . Naturally, this notion of a client’s gain from cheating is unlikely to be easily computed, and it depends in large part on the specific situation. However, for specific situations, we may be able to estimate when P could be large; e.g., in facial recognition problem from an automated check cashing application, if a large check is to be cashed, then the expected utility from defecting could be substantial if the client is in cohorts with the check casher. In such cases, the distributor can take extra measures to mitigate the risks associated with defecting; e.g., the problem could be given to more participants.

To recap, our simplified calculations suggest four design principles: payout should be directly related to a client’s ability; ability should be decreased substantially for incorrect answers, and increased slowly for correct ones; clients should be required to maintain a minimum reserve; and finally, extra measures should be taken when dealing with potentially high utilities for defecting.

RINGERS. One technique for mitigating the risks of cheaters is to use *ringers* as was suggested by Golle and Mironov [10] for the case of distributed computing tasks involving inverting a one-way function. Ringers are questions to which the answer is already known; including them in the computation allows us to determine if a (previously cooperating) client decide to defect. For example, someone may start by providing honest answers to DHC problems; but after a certain point, may suddenly try to use a computer program to reap a payout with little work. In the case of [10] using ringers was a more natural fit since they were dealing with computers that were providing answers to traditional mathematical functions – if the computer was wrong, it was cheating. In our case, answers are not always clear cut, and an honest user may simply be wrong on occasion. Moreover, there is an open issue of how one can design DHC ringers in the first place. On the one hand, ringers can come from previously posed questions; in such a case, one has to be careful that adversaries do not create “dictionaries” of ringers. Alternately, ringers can be created manually and their answers provided by trusted humans; in this case, ringer creation is expensive. Finally, perhaps one can generate ringers using a computer program; it is a lot less clear how to do this – especially since we do not want other computers to be able to recognize ringers.

Suppose R ringers are included in a DHC problem for client C . As before, X_i is an indicator variable for whether C gives the correct answer on problem i . Now, suppose that we estimate $\mathbf{E}[X_i] = p_i$ based on C ’s past performance, and let $p = (\sum_{i=1}^R p_i)/R$. Then, we are interested in

$$\Pr \left[p - \frac{\sum_{i=1}^R X_i}{R} > \epsilon \right]$$

for $\epsilon > 0$. If we assume that the X_i are independently and identically distributed, then we know from the weak law of large numbers that the above probability approaches 0 as R approaches infinity. Since $X = \sum_{i=1}^R X_i$ is a sum of independent Bernoulli trials (which also happen to converge to the normal distribution), we know that $\mathbf{Var}[X] = Rp(1 - p)$ and $\mathbf{E}[X] = Rp$. Setting $\delta = \epsilon/p$, and massaging the equation allows us to apply the Chernoff-Hoeffding bound directly:

$$\begin{aligned} \Pr [p - X/R > \epsilon] &= \Pr [X < \mathbf{E}[X](1 - \delta)] \\ &\leq e^{-\frac{R\epsilon^2}{2p}} \end{aligned}$$

The distributor can observe an ϵ from the actual computation, and compute $e^{-R\epsilon^2/2p}$; if it is below some threshold, then we can flag the client as a potential cheater, and discount his contribution. For a typical user who behaves honestly up until a certain point, we expect p to be very close to 1. Therefore, the likelihood of a deviation by even a small ϵ is unlikely and points to possible cheating.

4 Conclusions and Open Problems

This paper initiated what appears to be an untapped line of research on secure distributed *human* computation. The general paradigm involves using a large-scale distribution mechanism, such as the internet, to farm problems out to humans. Our motivation is the class of “AI-complete” problems which computers have a hard time solving, but which pose little difficulty for humans. We noted several real-world scenarios employing this paradigm and some interesting implications for internet commerce and B24b; yet this paper appears to be the first attempt to formally study this notion. Our paper put forth a framework for the secure design of such systems and provided a basic architecture. We then did a simple security analysis using standard tools from probability theory and game theory. Our probabilistic analysis showed how many dishonest parties the scheme could tolerate, and described how you might want to catch them using ringers. Our game-theoretic analysis suggested design principles for mitigating the risk of defection by malicious parties.

We believe that this notion of distributed human computation will spawn numerous areas of research interest. We mentioned open issues related to cryptography, algorithms, human-computer interaction, and programming language / API design in the introduction. In addition, there are two possible areas for directly extending the work in the present paper. First, we focused on the case where answers are multi-valued and much of our analysis was restricted to the case where each problem has a binary answer. But, this fails to capture many situations; e.g., in text translation, there may be several reasonable translations for a piece of text. One way to handle such situations is via a two-phase scheduling – the first phase solicits answers from different parties and the second phase is used to vote on the answers. We would have to use plurality vote instead of majority vote for determining an answer; however, we may still want to reward clients who gave reasonable answers (even though they may not have given the answer most people voted for). We remark also that our analysis assumed the answers to be independent, but this need not be the case. To analyze the case of dependent answers, one may try to use a variant of the Chernoff-Hoeffding bound that permits limited dependence [16].

Another extension would be to find a way to ascertain problem difficulty and use that to schedule the problem appropriately. One way to ascertain difficulty is to give the problem to a few clients, and observe the extent to which the answers deviate from a unanimous vote. The more they do, the more likely the problem is hard (or that there are a number of cheaters).

A third extension would be in to incorporate a notion of confidence. Here clients can not only indicate what they believe the correct answers to be, but also how confident they are in their belief. Over time, one can even correlate this confidence to how a given user performed after making similar judgements.

A fourth extension would be to have different underlying network architectures. For example, a P2P network or a multi-hop ad-hoc network might not permit a centralized problem distributor. Yet, for some applications (e.g., collaborative filtering for anti-spam), it might be more convenient to operate over such a network.

Finally, we remark that there are a number of important privacy ramifications – in particular, how would one handle that the case that the problem issued by a supplier to a distributor violates a privacy agreements the supplier made with its own customer base?

These open issues point to the potentially rich problem space of distributed human computation. We hope that our paper will inspire a line of further work in this area.

References

1. L. VON AHN, M. BLUM AND J. LANGFORD. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):5660, February 2004.
2. L. VON AHN, M. BLUM, N. HOPPER AND J. LANGFORD. CAPTCHA: Using hard AI problems for security. In *Advances in CryptologyEurocrypt '03* (Volume 2656 of Lecture Notes in Computer Science), pages 294311, 2003.
3. R. VON BITTER. Mathenauts: Tales of Mathematical Wonder. *Arbor House Pub Co*; June 1987.
4. R. CRAMER, R. GENNARO AND B. SCHOENMAKERS. A Secure and Optimally Efficient Multi-Authority Election Scheme. *Proc. of EuroCrypt 97*, Springer Verlag LNCS series 1233, pp. 103-118.
5. CYPHERMINT, INC. <http://www.cyphermint.com/>
6. X. DRÈZE AND F. HUSSHERR. Internet Advertising: Is Anybody Watching? *Journal of Interactive Marketing*, 2003, Vol. 17 (4), 8-23.
7. THE SEARCH OF EXTRATERRESTRIAL INTELLIGENCE PROJECT. University of California, Berkeley. <http://setiathome.berkeley.edu>.
8. O. GOLDBREICH. Secure Multiparty Computation. *Unpublished Manuscript*. Last revised October 2003.
9. THE GREAT INTERNET MERSENNE PRIME SEARCH. <http://www.mersenne.org>.
10. P. GOLLE AND I. MIRONOV. Uncheatable Distributed Computations. *Proceedings of the RSA Conference Cryptographers' Track*, 2001.
11. P. GOLLE AND S. STUBBLEBINE. Distributed computing with payout: task assignment for financial- and strong-security. *Financial Cryptography 2001*, LNCS Series, Springer-Verlag, February, 2001.
12. R. MOTWANI AND P. RAGHAVAN. Randomized algorithms. *Cambridge University Press*, New York, NY, 1995.
13. THE NEW YORK TIMES WEB SITE. <http://www.nytimes.com>.
14. C. K. PRAHALAD AND S. HART. The Fortune at the Bottom of the Pyramid. *Strategy + Business*, Issue 26, Q1 2000.
15. SPAM NET WEB SITE. <http://www.cloudmark.com>.
16. J. P. SCHMIDT, A. SIEGEL, AND A. SRINIVASAN. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM J. Discrete Math.* 8(2): 223-250 (1995)
17. S. STUBBLEBINE AND P. VAN OORSCHOT. Online Dictionary Attacks with Login Histories and Human-in-the-Loop. *Financial Cryptography 2004*, LNCS Series, Springer-Verlag, February, 2004.
18. VIPUL'S RAZOR WEB SITE. <http://sourceforge.net/projects/razor>.
19. F. ZHOU, L. ZHUANG, B. ZHAO, L. HUANG, A. D. JOSEPH, AND J. KUBIATOWICZ. Approximate Object Location and Spam Filtering. *Proc. of Middleware, 2003*.

A Acknowledgements

We thank *Anonymous 1* and *Anonymous 2* for suggesting the MOS score example. We thank *Anonymous 3* for suggesting the CAPTCHA calculation example. We thank *Anonymous 4* for pointing us to a paper in which spam prevention is handled using collaborative filtering with human agents. We thank *Anonymous 5* and *Anonymous 6* for feedback on early drafts of this manuscript. Finally, we thank *Anonymous 7* for helpful discussions.