Secure Distributed *Human Computation*

Craig Gentry
DoCoMo Labs, USA
San Jose, CA 95110
cgentry@docomolabs-usa.com

Zulfikar Ramzan
DoCoMo Labs, USA
San Jose, CA 95110
ramzan@docomolabs-usa.com

Stuart Stubblebine
Stubblebine Research Labs
Madison, NJ 07940
stuart@stubblebine.com

ABSTRACT

This paper is a preliminary exploration of 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; e.g., image analysis, speech recognition, and natural language processing. This paradigm already seems to be employed in several real-world scenarios, but we are unaware of any formal and unified attempt to study it. Nonetheless, this concept spawns interesting research questions in cryptography, algorithm design, human computer interfaces, and programming language / API design, among other fields. There are also interesting implications for Internet commerce and the B24b model. We describe this research area and suggest a basic framework for the design of such systems. We analyze security and reliability against malicious parties using standard probability theory tools. We then derive design principles using standard decision-theory 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 [5].

1. INTRODUCTION

As an example of the advantages of human computation, consider the Cyphermint PayCash system [9], which allows people without bank accounts or credit cards (a sizeable segment of the U.S. population) to automatically and instantly cash checks, pay bills, or make Internet transactions. Since PayCash offers automated financial transactions, and since the system uses publicly-accessible kiosks in (unprotected) public locations, security is obviously critical. For example, in the check-cashing application, the system must decide whether the person at the kiosk cashing the check is indeed the person to whom the check was made out. At first, one might expect that the kiosk employs sophisticated biometric tools, advanced facial recognition algorithms, and the like. This thought is unsettling since such schemes produce false positives, and can often be outwitted by a clever adversary; e.g., someone can try to hold a photograph up to the camera on the kiosk. However, the Cyphermint solution is much simpler – a "human computer" at the back end. Specifically, the kiosk simply takes a digital picture of the person cashing the check and transmits this picture electronically to a central office, where a human worker compares the kiosk's picture to one that was taken when the person registered with Cyphermint. If the two pictures correspond to the same person, then the human worker authorizes the check to be cashed; otherwise, the check is rejected.

In this example, a human assists in solving problems which are easy for humans but still difficult for powerful computers, even when they have state-of-the-art algorithms. Many problems fall into this category; e.g., so called "AI-complete" problems which occur in fields such as image analysis, speech recognition, and natural language processing. Motivated by the above example, this paper undertakes a preliminary study of the concept of secure distributed *human* computation (DHC). At first, DHC 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:** Some anti-spam mechanisms such as SpamNet [23] and Vipul's Razor [26] use human votes to determine if a given email is spam; these ideas extend to the P2P setting [28]. This approach may be effective given that humans can more easily identify junk mail than computers can; e.g., [28] claims 0% false positives in experiments. Each email recipient presses a 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. The system is also robust enough to handle spam messages that are not *exactly* the same, but are highly similar; e.g., some words in the body are different.
- **CAPTCHA Solution Gathering:** Although DHC

EC'05, June 5–8, 2005, Vancouver, British Columbia, Canada.

can help eliminate spam, spammers can hypothetically use DHC to further their goal (see [1], [2]). Consider 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 to prevent spammers from automatically creating thousands of email accounts; spammers, in turn, can farm these CAPTCHAs out to humans in exchange for access to illicit web content. While this example is a malevolent use of DHC, there are more constructive uses.

- **The ESP Game:** In the ESP Game [3], two players are randomly paired over the Internet; they are not permitted to communicate, but both view the same image on their respective web browsers. Each player types in words that describe the image. As soon as both players enter the same word, they get a new image. The goal is to get through fifteen images in $2\frac{1}{2}$ minutes, and the players' scores increase according to various factors. The players derive utility from the entertainment value the game provides. The game organizers derive utility since the game's results are used to label images on the Web, which is valuable for improving image search.

- **Distributed Proofreaders:** The distributed proofreaders project [10] is geared towards eliminating optical character recognition errors in Project Gutenberg [22] electronic books. The idea is to give a (small) portion of the image file and corresponding text (generated by OCR) side-by-side to a human proofreader. The proofreader in turn provides edited text to fix any errors. By giving the same piece of text to several proofreaders, errors can be reliably eliminated.

- **Other examples:** Open source software development loosely falls into the DHC paradigm; here the difficult problem is not something crisp like image 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 that essentially almost anyone can contribute to the Wiki.

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 around 0.7% [11], and outside of specific niche industries, very few people are willing to pay subscription fees for premium Internet content.

However, DHC could yield 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. We note that DHC has several benefits compared to traditional credit card transactions. First, it might be faster for a user to solve a human computation problem as compared to actually getting their credit card out and typing in a 16-digit credit card number, expiration date, and billing address. Second, credit card information can be compromised (e.g., if the merchant web server is compromised); this is not an issue with DHC. Finally, credit card transaction fees are substantial, and make it impossible to offer low-value content. In a sense, then, human computation can form a new type of online currency or bartering system.

As an example, such a mechanism might be useful on the New York Times web site [20] which provides free access to the day's news articles, but charges a fee for archived articles. Such a fee (while necessary from a business perspective) might deter users – especially since it may be possible to (illegally) obtain the article text from some other Internet location; e.g., it was posted to a newsgroup or mailing list. However, instead of charging a fee, the New York Times could give the user a human computation problem – e.g., transcribing an audio feed into text. In exchange for solving the problem, the archived article can be provided. This concept extends to other service offerings; e.g., music downloads or long-distance minutes for solutions. DHC may also enable the Business-to-Four-Billion (B24b) model [21] 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. Although the economic feasibility of B24b is still very much an open question, providing services in exchange for solving DHC problems seems like a useful approach, since it depends on an abundance of human resources, while avoiding cash transactions.

The Human Factor. Because we are talking about *human* computation, there are numerous social and ethical issues to consider. For example, as with any human service (and especially one that global reach), we should ensure that the market for human computation is not unduly exploitative. Along the same lines, it is important to consider human capabilities in solving problems. For example, image processing tasks might be difficult for those with impaired sight. In a similar vein, human capabilities differ from time-to-time. For example, a person may not be as able to drive a car safely and solve human computation problems at the same time. While such "human" issues seem interesting in their own right, we limit our discussion to more technical points in this paper.

Relevance to other fields. DHC is relevant to a number of computer science research disciplines, including algorithms (how can one design algorithms that leverage human computation to solve specific problems?), programming languages (how can we better specify programming languages or interfaces so that human input can be provided more naturally?), human-computer interaction (what kinds of interface designs are best suited for incorporating human inputs?), distributed systems (if a single problem is parceled out to multiple individuals, how can we reliably agree on a correct answer?), and cryptography/security (how can we protect a human computation system from malicious behavior?).

With respect to distributed systems, there is a relation to the *distributed consensus* problem, which roughly asks how distinct processes can agree on a single (binary) value based on the votes of each process, where at least one of the processes actually voted for the agreed upon value. In distributed consensus, one is concerned with different types of failure models (e.g., in a *Fail-stop* failure, a process simply ceases all activities, but in a *Byzantine* failure, a process

might be intentionally operating in a malicious manner.) as well as different types of communication models (e.g., in an *asynchronous* model, we cannot assume any bounds on communication delay or processor speed, whereas in a *synchronous* model, we can assume bounds). In an asynchronous model, if at least one process misbehaves then there is no guarantee that a consensus can be reached [12]. For more information on the consensus problem, see Lynch's text [18]. In synchronous networks, the general problem of having multiple (untrusted) parties securely compute an arbitrary function (where each party provides its own private input to the function) has been studied in the cryptographic literature under the term *secure multi-party computation* (see, e.g., [27, 14]).

These areas bear relevance since solving a DHC problem may involve multiple human computations; however, the differences appear more striking than the similarities. First, the parties are human beings instead of computers; second, the parties are themselves not providing actual multiparty computation inputs, but are instead providing candidate answers (which themselves can be construed as inputs into a group decision-making process); third, the "function" to be computed may not always have a clear-cut answer; fourth, the computation may be facilitated by a server that is trusted, but computationally weak in the sense that it is not capable of providing answers itself;[1] fifth, 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.

Drawing further relations to cryptography, we also remark that since DHC is a form of currency, we may use cryptographic tools that have been developed in connection with e-cash. Finally, much of the related work to be discussed below ([16], [17], [2], [1]) has appeared in cryptographic literature. Overall, however, we are well-aware that "security" is less of a cut-and-dried issue in the human computation context than in the cryptographic context, but we view this as an interesting research challenge.

Related work. Much work has been done in building a giant distributed computer system by harnessing idle CPU cycles on the Internet. Examples include SETI@HOME's search of extra-terrestrial life through large-scale data mining [13], Entropia.com's GIMPS project for finding large Mersenne Primes [15], and Distributed.Net's attempts to do exhaustive key search on cryptosystems. Golle-Mironov [16] and Golle-Stubblebine [17] have considered how one can add security measures into such systems in cases when some type of payout is provided in exchange for computation time. We refer to this as distributed computing with payout (DCP). Many of the ideas and analysis from [16] [17] have analogues in the DHC setting. Of course, there are fundamental differences. First, 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 a payout to all participating computers (so long as they are not cheating) that have demonstrated some proof of the effort undertaken[2]. In DHC, it does not seem to make sense to pay everyone – instead, it seems that one should only pay clients who provided an answer that is "good" according to some measure. 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. Finally, DCP leverages otherwise idle CPU cycles; in DHC it seems harder to justify a notion of idle human time since one might always be able to find something else to do that time – of course there are situations such as waiting in line or taking the train where "human cycles" might be construed as idle.

Another line of related work is CAPTCHAs ([2], [1]), which also involve 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. First, automatic problem generation may be infeasible. Second, the solution may not be known *a priori*, but must be inferred from answers given.

Our Contributions. The primary contribution of this paper is to suggest 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). Interestingly, our analysis shows that in the presence of *lurking adversaries* standard tools like Bayesian inference are worse than simple approaches like majority vote for combining individual answers. Next, we use some basic decision theory tools (e.g., utility functions) to derive design principles for a secure DHC system. We also identify open issues and areas for future work. We remark that since our primary aim is a preliminary exploration of an untapped research area, we have not attempted to solve all possible problems that arise. 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 other issues we have not addressed.

Paper Organization. The next section gives the framework and an architecture. Sections 3 and 4 do some preliminary analysis based on simple probability and decision theory tools. Section 5 describes a number of open problems. Finally, Section 6 concludes.

2. FRAMEWORK

We describe one framework for a secure distributed human computing (SDHC). While we have tried to be general, one can imagine numerous variations on what we present.

Organizing principles. An SDHC system is a four tuple $(\mathsf{S}, \mathsf{C}, \mathcal{D}, \mathsf{F})$, where:

- S is a set of problem suppliers who wish to have specific problems solved within a budget $\mathcal{B}$.
- C is a set of (human) problem-solving clients.

[1]Server trust can be minimized by augmenting a DHC system with a voter and results-verifiable voting protocol [8].

[2]The concept of proof of work applies especially to exhaustive search type problems (e.g., finding the pre-image of a given output of a one-way function) where only a single machine will return the correct answer, but all participating machines should be compensated according to their effort.

- $\mathcal{D}$ is a distributor or broker who matches elements of S with elements of C. Elements of S might pay $\mathcal{D}$ to have specific problems solved.

- F is a set of storefronts; $\mathcal{D}$ makes problems available to elements of C via elements of F. We imagine that elements of C visit elements of F to gain some utility; e.g., to obtain a specific product or service, or simply for intangible value like entertainment, voluntarism, etc. $\mathcal{D}$ may pay members of F or C for help in solving problems (using perhaps payment received from S).

See fig. 1 for a high-level depiction. Note that S (resp. C, F) represents a set. In the sequel we will use the notation $\mathcal{S}$ (resp. $\mathcal{C}, \mathcal{F}$) to represent a single element of this set. The elements of this framework have analogues to what happens presently in Internet advertising. Here $\mathcal{F}$ represents a typical web site (e.g., `NewYorkTimes.com`) and $\mathcal{D}$ is analogous to a service that manages banner advertisements (e.g., DoubleClick or 24/7 Real Media). This way $\mathcal{F}$ need not know anything about the problems being solved. It is simply a channel for allowing problems to be seen by as many clients as possible. Of course, we can easily allow $\mathcal{F} = \mathcal{D}$ (i.e., $\mathcal{C}$ directly talks to $\mathcal{D}$), but we believe that $\mathcal{F} \neq \mathcal{D}$ is the more general case, and in practice will result in more clients looking at more problems.

Between C and $\mathcal{D}$, there is an interactive protocol that aims to obtain the correct answer to various problem instances with high probability. We envision that the protocol involves two phases: a registration phase and an operation phase. The latter phase involves several processes: problem management, scheduling, monitoring, payout, and account management (by $\mathcal{C}$ or $\mathcal{D}$). Details follow.

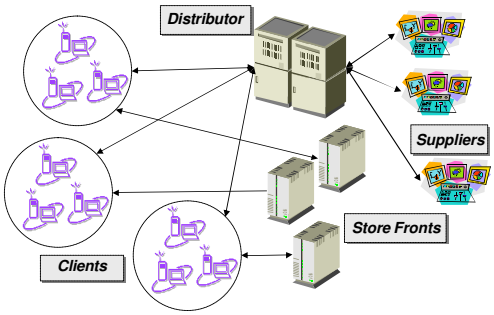

Figure 1: A high-level depiction of an example secure distributed human computing system.

Registration phase. The registration phase is an interactive protocol that occurs between $\mathcal{C}$ and $\mathcal{D}$, when $\mathcal{C}$ first decides he would like to participate in the SDHC system. $\mathcal{C}$ might provide basic information (name, etc.) to $\mathcal{D}$. Thereafter, $\mathcal{C}$ might go through training (i.e., explaining what task needs to be performed). Next, $\mathcal{C}$ might be tested on actual problem data; the answers to these problems could be known in advance or could be unknown (e.g., we can use live problem instances). The latter prevents $\mathcal{C}$ from using a dictionary of known answers, and provides some "free" work to $\mathcal{D}$. The training phase may also incorporate CAPTCHAs [1] [2] to ensure that $\mathcal{C}$ is not running a software agent for automatic account creation. The output of the registration phase might be a provisional rating for $\mathcal{C}$ (corresponding to an estimated likelihood that it answers problems correctly) – this rating may later be updated during the operation phase. In addition to training $\mathcal{C}$ and estimating his accuracy, the training phase also serves as an opportunity cost to discourage $\mathcal{C}$ from establishing a new account after abusing his current account.

Operation Phase. We envision this phase comprising several processes: problem management, scheduling, monitoring, payout, account management and termination.

- Problem Management: The problem management process involves a multi-phase (interactive) probabilistic polynomial time algorithm (PPTA) that takes as input a problem instance Π, an accuracy parameter ϵ and $\mathcal{S}$'s problem budget $\mathcal{B}$. From $\mathcal{B}$ it derives its own budget β containing time bounds t, t' and another integer bound n; i.e., $\mathcal{B}$ specifies a cap on the dollar amount to spend, which translates into how long clients can take, how many clients to ask, etc. The output is a set of n three tuples $(\Pi_1, \epsilon_1, t_1), \ldots (\Pi_n, \epsilon_n, t_n)$, as well as a description for a (probabilistic) reconstruction algorithm $\mathcal{R}$ with the following properties. First, client $\mathcal{C}_i \in \mathsf{C}$ can provide a candidate answer α_i to "sub-problem" Π_i which is correct with probability ϵ_i in time t_i, where we require $t_i \leq t$. Second, the reconstruction algorithm $\mathcal{R}$ on input $\alpha_1, \ldots, \alpha_n$ outputs an answer α to Π which is correct with probability at least ϵ, using at most t' steps[3]. Finally, the partitioning scheme must operate within the budget $\mathcal{B}$.[4] If the algorithm cannot produce a solution meeting the budget constraints, then it outputs $\perp$. With respect to performance, we would like ϵ_i, t_i $(1 \leq i \leq n)$ to be as small as possible. Similarly, we would like the algorithm to not produce the answer $\perp$ with only a small budget (e.g., t, t', n are also small). The algorithm might involve several (recursive) phases – e.g., in which elements of C assist with the "meta-problem" of decomposing a problem into sub-problems. The problem management algorithm interacts with the scheduling algorithm we now describe.

- Scheduling: The scheduler is a multi-phase (interactive) PPTA that takes as input the problems $\Pi_1, \ldots, \Pi_n$ provided by the problem management algorithm as well as information about the subset of C who are currently available (possibly together with information about their past performance and skill sets) and outputs a subset of clients $\mathcal{C}_1, \ldots, \mathcal{C}_n \in \mathsf{C}$ to whom to give these problems. The scheduler then waits to receive responses from the clients, after which it runs $\mathcal{R}$ to obtain the final answer. The scheduler should enforce the requirement that answers be provided within some time bound to be valid; otherwise, it may not be able to arrive at a final answer.

[3]We remark that $\mathcal{R}$ might also be "online" in the sense that it can still be executed, and provide some guarantees, even if only a subset of the answers are available.

[4]Here $\mathcal{B}$ is an overall resource budget specifying how many users a sub-problem can be given to, how much time these users take, and how much time the reconstruction algorithm must take. It is not clear how to sensibly define the notion of time complexity of problem solving for humans, but the purpose of including a time bound here is to design the problem management algorithm so that no sub-problem takes "too long."

- Payout: Payout involves a PPTA that takes as input C, as well as the transcript of the scheduler's interaction, a description of the scheduler (and its inputs), a description of $\mathcal{R}$, and a payout budget $\mathcal{B}$. It outputs the description of a payout function $p : \mathsf{C} \rightarrow \mathbb{R}$ which specifies how much to pay $\mathcal{C}_1, \ldots, \mathcal{C}_n$. The payout function must respect the budget: $\sum_{i=1}^{n} p(\mathcal{C}_i) \leq \mathcal{B}$. We remark that in some cases, there may be no payout (e.g., if a DHC problem is solved for entertainment value). We also remark that the algorithm may be randomized (e.g., payout could be done in the form of a "lottery ticket" scheme). The payout function should have a positive evaluation only when an answer is "correct" as determined by some type of Bayesian inference, weighted majority vote, or some other mechanism[5]. On the other hand, it may be possible that there is no preferred answer; i.e., the votes are evenly divided or there are multiple valid answers. 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 analysis of Section 4, one should design the scheme so that the payout amount is proportional to $\mathcal{C}$'s perceived skill level; i.e., if $\mathcal{C}$ tends to get many correct answers, we pay him more. This incentivizes $\mathcal{C}$ to answer reliably and therefore deters cheating.

- Monitoring Process: The monitoring process is a PPTA that takes as input the transcript of the execution of the other algorithms and protocols and outputs elements of C who might be cheating. For example, the monitoring process might:
  - Monitor $\mathcal{C}$'s payout account to see if there is a minimum balance; as we will see in the decision theory analysis of Section 4, a minimum balance requirement benefits $\mathcal{D}$.
  - Monitor $\mathcal{C}$'s success rate to see if there are any sharp drop offs.
  - Determine if multiple account creation requests stem from the same source.

- Account Management: The account management process involves an (interactive) PPTA which takes as input the database of long-term system variables and updates them:
  - $\mathcal{C}$'s ratings can be updated in accordance with performance.
  - $\mathcal{C}$ may specify the types of problems he prefers 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 users are fluent in several languages, $\mathcal{D}$ may wish to provide them problems involving text or speech translation.
  - $\mathcal{C}$'s account may be terminated; the termination request may either come directly from $\mathcal{C}$ or it may be initiated by $\mathcal{D}$ itself. In the former case, we may want to pay out the balance. In the latter case, we may want to investigate the behavior which resulted in termination; e.g., if $\mathcal{C}$ was caught cheating, then the final payout may be withheld.

[5]We also remark that we could permit negative payout, wherein clients are penalized for wrong answers.

Threat Model. Threats to an SDHC system include those directed to the registration phase, operation phase, or some combination. 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 [25]. 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. Lurking adversaries may be more problematic in the context of SDHC because a few malicious acts might be attributed to human error. An *inconsistent adversary* responds to tasks in an inconsistent manner.[6] Our approach generally detects the inconsistent adversary since we are able to monitor the accuracy of the adversary over various time windows. Although we tend to assume otherwise, it possible that $\mathcal{D}$ might cheat; e.g., by claiming that solutions were wrong and not providing payout. This problem can be mitigated with a voting protocol that provides voter and results verification (by either the individual voter or by anyone); e.g., see [8].

Architecture. An SDHC system architecture depends on the media capabilities of $\mathcal{C}$ (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 that brokers interactive advertisements on the web. Storefronts provide a URL associated with a problem "payment" mechanism associated with human computation. Upon selecting the URL, $\mathcal{C}$ passes a cookie to $\mathcal{D}$ if it is registered and $\mathcal{C}$ receives a human computation problem. (Otherwise, $\mathcal{C}$ is required to go through the registration process.) In response to receiving $\mathcal{C}$'s cookie, a human computation problem is directed to $\mathcal{C}$. $\mathcal{C}$ responds back to $\mathcal{D}$ with a solution. Upon completion, $\mathcal{F}$ receives an authenticated confirmation (possibly passed through $\mathcal{C}$) that $\mathcal{C}$ has made sufficient "payment" and $\mathcal{C}$ receives the product or service. The distribution network consists of computers that primarily host problems and schedule these problems to $\mathcal{C}$. The scheduler identifies the next problem to be distributed and the associated skill level with the problem. Upon request of a problem by $\mathcal{C}$, a process retrieves $\mathcal{C}$'s skill level and validity. $\mathcal{C}$ is passed a URL associated with next available problem as determined by the scheduler.

3. PROBABILISTIC ANALYSIS OF RELIABILITY IN AN SDHC SYSTEM

We now use simple tools from probability theory to analyze the reliability of votes in an SDHC system. We are

[6]As an example, an inconsistent adversary might simply be lazy, truly inconsistent, or may even employ expert programs in an attempt to answer correctly most of the time.

interested in what happens when clients each vote on the same problem. Our analysis also considers the presence of malicious parties. We consider both standard majority vote and Bayesian inference to combine answers from different clients. We uncover that while Bayesian inference is the defacto standard for traditional settings, it actually performs *worse* than majority vote in the presence of malicious parties. We also analyze the impact of *ringers* [16].

PRELIMINARIES. Suppose that problem Π has t possible answers, denoted by the numbers 1 through t. Furthermore let P be a discrete random variable whose value is equal to the actual answer. Let $\pi_i \stackrel{\text{def}}{=} \Pr[P = i]$ and assume that we know π_i for $1 \leq i \leq t$. Suppose Π is looked at by s different clients $\mathcal{C}_1, \ldots, \mathcal{C}_s$, and and let $C_i \in [1, t]$ denote a discrete random variable representing the answer provided by $\mathcal{C}_i$ (for $1 \leq i \leq s$). Let $\vec{C} = (C_1, \ldots, C_s)$. Let $p_i = \Pr[\mathcal{C}_i \text{ is correct}], 1 \leq i \leq s$. Now, for a problem Π and a client $\mathcal{C}_i$, we define X_i to be 0 if $\mathcal{C}_i$ gave the incorrect answer on Π and 1 otherwise. Moreover, suppose that $\mathcal{C}_i$ actually gave answer α_i. Let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_s)$. We further assume that these responses are independent of each other.[7] We can actually model independence by conservatively assuming that adversaries give the wrong answer with probability 1.

MAJORITY VOTE (WITH ADVERSARIES). We now analyze the reliability of a majority vote strategy for combining answers. We use the simplified Chernoff-Hoeffding bound presented in [19].

THEOREM 1. *Let* $\overline{p} \stackrel{def}{=} \frac{1}{n} \cdot \sum_{i=1}^{n} p_i$. *Then, the majority vote strategy fails with probability at most* $e^{-n\overline{p}(1-1/2\overline{p})^2/2}$.

Proof sketch: This follows from a direct application of the Chernoff-Hoeffding bound: if $X_1, \ldots, X_n$ are outcomes of independent Bernoulli trials where $\Pr[X_i = 1] = p_i$ $(1 \leq i \leq n)$, then for $0 < \delta \leq 1$, $\Pr[X < (1-\delta) \cdot \mu] < e^{-\mu\delta^2/2}$, where $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$ $(= \sum_{i=1}^{n} p_i)$. Now,

$$\Pr[X_i = 1] = \Pr[c_i \text{ is correct}] = p_i.$$

Also, $\mu - E[X] = n\overline{p}$. The result follows from setting $\delta = 1/2$. □

Now, observe that for $\overline{p} > 1/2$, the probability of an incorrect answer decreases exponentially in n. Suppose there are h honest parties and 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. The honest majority requirement is not unreasonable, and is typical in the secure general multiparty computation literature [14].[8] 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 are honest, then the probability with which these clients must supply correct answers only has to be above 3/4. Now, if we solve for δ as function of ϵ in the Chernoff-Bound equation, we find that having a $\overline{p} \cdot \left(1 - \sqrt{(2 \ln 1/\epsilon)/(n\overline{p})}\right)$ portion of the votes guarantees that the probability of a majority vote failing is at most ϵ.

BAYESIAN INFERENCE (WITH ADVERSARIES). 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 $\text{argmax}_{j \in [1,t]} \Pr[P = j | \vec{C} = \vec{\alpha}]$. Set ρ_{ij} equal to p_i, if $C_i = j$, and $1 - p_i$ otherwise. Then, the Bayesian likelihood probability is computed as follows:

$$\begin{aligned}\Pr[P = j \mid \vec{C} = \vec{\alpha}] &= \frac{\Pr[\vec{C} = \vec{\alpha} \mid P = j] \cdot \pi_j}{\sum_{k=1}^{t} (\Pr[\vec{C} = \vec{\alpha} \mid P = k] \cdot \pi_k)} \\ &= \frac{\pi_j}{\sum_{k=1}^{t} ((\prod_{i=1}^{s} \rho_{ik}) \cdot \pi_k)}\end{aligned}$$

This follows since $\Pr[\vec{C} = \vec{\alpha} | P = j] = \prod_{i=1}^{s} \rho_{ij}$. Now consider when classifiers are malicious; i.e., clients deliberately give wrong 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. We assume that the number of possible answers t is 2. Consider the standard Bayesian formula for determining if the answer to a problem Π is 1:

$$\Pr[P = 1 \mid \vec{\alpha}] = \frac{\Pr[\vec{\alpha} \mid P = 1] \cdot \pi_1}{\Pr[\vec{\alpha} \mid P = 1] \cdot \pi_1 + \Pr[\vec{\alpha} \mid P = 2] \cdot \pi_2} \quad (1)$$

Suppose that P is indeed 1, 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 eqn. 1 is greater than 1/2 (i.e., $\mathcal{D}$ can determine the correct answer *despite* the presence of malicious parties):

$$\begin{aligned}&\frac{p_h^{h_r}(1-p_h)^{h_w}(1-p_m)^m \cdot \pi_1}{p_h^{h_r}(1-p_h)^{h_w}(1-p_m)^m\pi_1 + (1-p_h)^{h_r}p_h^{h_w}p_m^m\pi_2} > 1/2 \\ &\iff \left(\frac{p_h}{1-p_h}\right)^{\Delta_h} > \left(\frac{p_m}{1-p_m}\right)^m \frac{\pi_2}{\pi_1},\end{aligned}$$

where $\Delta_h \stackrel{\text{def}}{=} h_r - h_w$. If we assume $\pi_2 \geq \pi_1$,[9] then a sufficient condition for security is:

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

If $p_m = p_h$ (in which case for $\pi_1 = \pi_2$, Bayesian classification of a binary value is equivalent to standard majority vote),

[7] 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.

[8] Of course secure general multiparty computation protocols only require that *active adversaries* be in the minority; these protocols can usually tolerate even more passive (i.e., honest but curious) adversaries.

[9] Note that this is the more conservative estimate; if $\pi_1 > \pi_2$, then the 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.

then the security condition is reduced to $h_r > m + h_w$, which is consistent with our intuition that if everyone performs identically up to now, then we need an honest 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 eqn. 2 to better reason about it. Assume that p_m is 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:

$$p_m = 1 - \delta \qquad \text{and} \qquad p_h = 1 - k\delta \approx (1-\delta)^k = p_m^k.$$

Then, the right hand side of eqn. 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., $1 - (1-0.9)^2 = 0.99$. Interestingly, from this one can see that *the majority vote strategy outperforms Bayesian inference.* In particular a majority vote strategy requires a two-thirds honest majority to be correct with probability greater than 0.75 rather than 0.9 (and the likelihood of an error goes down exponentially in the number of users asked). In hindsight, it is not surprising that majority vote can outperform Bayesian inference in the presence of malicious parties. After all, if malicious parties have performed well up to a certain point, then their vote may be more heavily weighted in the Bayesian case, whereas majority vote makes no such assumptions.

RINGERS. One way to mitigate the risk of cheaters is to use *ringers* as suggested by Golle-Mironov [16], who considered distributed computing with payout for inverting a one-way function. Ringers are questions to which the answer is already known. Including them allows us to determine if a (previously cooperating) client decided to defect (e.g., he suddenly starts using a computer program to reap a payout with little work). Ringers are a more natural fit for traditional distributed computing tasks since answers are well defined – if a computer is wrong, then it is cheating. In DHC, answers are not always clear cut, and an honest user may simply be wrong on occasion. Moreover, it's not clear *how* one can cheaply design DHC ringers (as we discuss in Section 5). Nonetheless, 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

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

for $\epsilon > 0$. Setting $\delta = \epsilon / p$, and massaging the equation allows us to bound this expression by $e^{-R\epsilon^2/2p}$ via a straightforward application of the Chernoff-Hoeffding bound. 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, even a small deviation is unlikely and suggests cheating.

4. DERIVING DESIGN PRINCIPLES FROM BASIC DECISION THEORY

We use basic decision theory tools to derive design principles for an SDHC system. The analysis assumes that the human clients are rational and either risk averse or risk neutral, but not risk seeking.[10] We can think of a DHC process as a multi-party game.[11] 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 $\mathcal{C}_i$ makes an earnest attempt to answer the problem correctly, we say that he cooperates; otherwise, we say that he defects. Let $\mathcal{T}_i(\Pi_t)$ (respectively $\mathcal{D}_i(\Pi)$) denote the reward that $\mathcal{C}_i$ ($i \in [1, n]$) receives for cooperating (respectively defecting) on problem Π_t issued at time t. Then, we are interested in ensuring that $\mathbf{E}[\mathcal{T}_i(\Pi)] - \mathbf{E}[\mathcal{D}_i(\Pi)] > 0$. We let $\mathcal{A}_i$ denote a function that computes $\mathcal{C}_i$'s ability (where ability is defined in some sensible way, based on past performance); 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 $\mathcal{C}_i$, that computes the amount paid to $\mathcal{C}_i$ for a given answer. We only consider the payment function in situations where $\mathcal{C}_i$ provided what is believed to be the correct answer. When it is clear from context, we will drop the arguments and subscripts associated with $\mathcal{T}, \mathcal{D}, \mathcal{P}$. Now, a simple first approximation of the expected gain from cooperation versus defection $\mathbf{E}[\mathcal{T}_i(\Pi)] - \mathbf{E}[\mathcal{D}_i(\Pi)]$ is:

$$\begin{aligned} &\mathcal{P}(\mathcal{A}_i, \Pi, \mathcal{B}) \cdot p_{i,\Pi} \\ &+ \sum_{\tau > t} \mathbf{E}[\mathcal{T}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{T}_i(\Pi_\tau) \mid \mathcal{H} = H'] \\ &+ p \cdot R - P, \end{aligned} \tag{3}$$

where H' is an alternate history in which $\mathcal{C}_i$ defects on problem Π, R denotes the balance in $\mathcal{C}_i$'s account, p denotes the

[10] Of course, it's conceivable that clients may be irrational or risk-seeking. This is immaterial for our purposes since our goal is merely to derive design principles to deter cheating/defection for at least the rational users.

[11] 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.

probability that $\mathcal{C}_i$ gets caught, and P denotes what $\mathcal{C}_i$ might gain from cheating. 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 $\sum_{\tau>t} \mathbf{E}[\mathcal{T}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{T}_i(\Pi_\tau) | \mathcal{H} = H']$ associated with future payments since these are a function of $\mathcal{A}(\mathcal{H}_i)$ which would be modified if $\mathcal{C}_i$ decided to defect.

Of course, there may be some ancillary benefit for answering correctly; for simplicity, we ignore that in the analysis. Note that we implicitly assume that if $\mathcal{C}_i$ cheats, then it has provided an *incorrect* answer, in which case $\mathcal{D}$ pays nothing to $\mathcal{C}_i$. Moreover, if $\mathcal{C}_i$ is caught cheating, then it is kicked out and it loses whatever balance has been accumulated. Now, eqn. 3 is a simple approximation. Yet, it provides insight into SDHC system design. We would like eqn. 3 to be greater than 0, and there are mechanisms to make this more likely. First, we can control the incremental benefit $\sum_{\tau>t} \mathbf{E}[\mathcal{T}_i(\Pi_\tau)] - \mathbf{E}[\mathcal{T}_i(\Pi_\tau) \,|\, \mathcal{H} = H']$ in eqn. 3 through two measures (which are valid regardless of which mechanism – majority vote or Bayesian inference – one uses to combine answers):

1. Payout should be an increasing function of a client's ability. This increases the incremental benefit, which increases $\mathbf{E}[\mathcal{T}_i(\Pi)]$, 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 $\mathbf{E}[\mathcal{T}_i(\Pi)]$ and discourages defection.

Additionally, we can increase R by requiring that $\mathcal{C}_i$ maintains a minimum reserve. Finally, we can find ways to increase p or limit P. Naturally, $\mathcal{C}_i$'s gain from cheating may not be easily computed, and it could be situation specific. However, in some cases, we may be able to estimate when P could be large; e.g., in a facial recognition problem from an automated check cashing application, if a large check is to be cashed, then $\mathbf{E}[\mathcal{D}_i(\Pi)]$ could be substantial if the client is in cohoots 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. We also note that P itself represents an "expected" value in the sense that $\mathcal{C}_i$ is less likely to benefit from cheating if it was unable to actually sway the final outcome. This suggests making it hard for any single client to have "swaying power" which cements the argument made in the previous section that majority vote mechanisms may be preferable to Bayesian inference in the presence of lurking adversaries. Of course, we may not want to discount weighted schemes, such as Bayesian inference, since they are better able to extract the full skill set of problem solving clients. However, if we use a weighted scheme, then it makes sense to require higher minimum balances for people with more weight (i.e., people with higher ratings) since they are in a better position to sway the outcome.

To recap, our simplified calculations suggest several design principles: directly relate payout to the client's ability; decrease ability substantially for incorrect answers, and increase it slowly for correct ones; required $\mathcal{C}_i$ to maintain a minimum reserve (possibly based on his rating); and finally, take extra measures to mitigate risk for large P.

5. OPEN AREAS

Because DHC seems relatively untapped, our goal was to informally introduce it, provide some motivating examples, and suggest ideas for designing and analyzing such systems. To further convince the reader that this area suggests a rich set of problems, we provide some sample open problems areas which we hope will inspire future work on this topic.

FRAMEWORKS. Outside of what we presented, what other frameworks for an SDHC system are possible? How can we evaluate and compare these frameworks? What tradeoffs exist between various criteria? Can we optimize frameworks for specific applications? For example, rather than having a central problem broker, it may make sense to distribute this functionality. This may make sense if an application needs to run over a (truly) P2P network or over a multi-hop ad hoc network, neither of which are especially amenable to centralized control. These networks may be more appropriate for the spam prevention via collaborative filtering example we described in the Introduction.

PROBLEM SPACE. Our analysis considered the straw-man case of easily expressible problems with one correct answer. But, many problems fall outside this category; e.g., in text translation, there may be several equally acceptable answers. How can we handle such problems? One possibility is via two-phase problem scheduling. The first phase solicits answers from different parties and the second phase votes on these answers. We might have to use plurality vote instead of majority vote for determining an answer; however, we may still want to reward clients for reasonable (albeit not the most popular) answers. So, some questions are: how many "votes" are needed for a reliable answer, how should the votes be aggregated, and how do we structure payouts? For the second item, we encounter Arrow's impossibility result – basically, that for two or more voters with three or more choices, there is no social choice function that aggregates the votes fairly and consistently – but we may still hope for a "good" (non-ideal) voting method [4].

Problems like text translation pose another challenge. In particular, they may be "too big" for one person to tackle in a short time, especially if the piece being translated is long. On the other hand, it is not clear how to break this problem down since translation is context dependent; e.g., it's not sufficient to give each client one sentence to translate since the translation of one sentence may depend on the contents of another one. Perhaps one can attempt a solution in which each user is given a different piece of text, but with some overlap. In general, how can one design algorithms to solve problems knowing that the algorithm can receive human aid (and what other kinds of "AI-complete" problems can we solve using DHC)? Starting with the DHC paradigm in mind might make the task easier; e.g., it seems less complex to construct a DHC system for identifying the parts of speech for various words in a sentence as this is a discrete problem. With such information at its disposal, an algorithm for another natural language processing task (e.g., speech recognition, translation, etc.) might produce answers more reliably. Along the same lines, one can first try to solve

a problem computationally, and use human feedback to correct errors; e.g., the distributed proofreader's approach of first using OCR to convert an image into text, and then having proofreaders edit the resulting text [10].

Improving Reliability. There are a number of open issues whose resolution can yield more reliable DHC systems. For example, can we introduce 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. Along the same lines, can we find a way to determine problem difficulty and use that to schedule the problem appropriately? One possibility 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). Similarly, how should we analyze the received answers given that they may not be independent (e.g., if some type of collusion is involved)? Certainly, Chernoff-Hoeffding bounds no longer apply for correlated random variables. Perhaps one can try to use a variant of the Chernoff-Hoeffding bound that permits limited dependence [24].

All of these questions are part of a more generic question: how can we better determine correctness? While majority vote is beneficial in certain settings, perhaps there are better things one can do. For example, work on self-checking / correcting functions is nice for crisp mathematical problems, such as determining whether a function is linear (given a faulty oracle) [6] [7]. Can a similar tactic work for human computation problems? Along the same lines, there are some problem classes (such as producing a witness for membership of an element in an $\mathcal{NP}$-hard language) where finding the answer is (presumably) difficult, but verifying correctness is very simple. Perhaps one can achieve something similar for specific classes of AI-complete problems.

Improving Security. This paper suggested two mechanisms for catching cheaters. The first is to calculate when the distribution of $\mathcal{C}$'s given answers and the correct answers are statistically "far" apart. Here "far" might be measured relative to $\mathcal{C}$'s overall performance. The second is to incorporate ringers (this can be viewed as a special instance of the first approach). What other mechanisms exist for catching cheaters? The mechanisms we suggested do not distinguish between honest human error and malicious behavior. Along the same lines, how can one build a reputation system for this problem? From a security perspective, this is tricky since one has to be careful of attacks from people who "build" good reputations and then try to exploit this later for malicious gain. With reference to ringers, how can one efficiently design DHC ringers? Simply recycling old problems can result in replay attacks, but generating new ringers from scratch has its own problems; namely, ringer generation by (trusted) humans might be expensive and automated ringer generation by a computer may be challenging since we do not want another computer to be able to recognize a ringer.

A related problem is how we can achieve privacy. In particular, the data provided to a specific user may be confidential (e.g., a problem involving the transcription of medical records). One natural way to "mitigate" this problem is by only distributing sensitive data to a small set of (semi-trusted) individuals. But, this solution is not satisfying since it requires the involvement of additional trusted parties.

Economics and Motivations. The economics of DHC seems to require that clients be compensated far less then what they ordinarily would if they were "in-house" employees. This may primarily be due to the increased cost of assigning a problem to multiple entities when the purpose is to catch dishonest clients. Redundant problem assignment for the purposes of catching dishonest clients is different from adding redundancy to catch inaccurate responses from honest clients. On the other hand, differences in global labor rates may help make the DHC approach more economical than an in-house approach.

Another open question is whether the trustworthiness and reliability of online clients will roughly match that of one's own "in-house" employees over time. Our intuition is that over time the reliability and trustworthiness of an online client will approximate that of an in-house employee. However, an employee who performs a malicious act at the jobsite might be easily caught whereas an online client might operate beyond the reach of prosecution. What techniques can we use to make the threat of getting caught similar to that in a face-to-face employment scenario? Finally, what alternative techniques exist to incentivize clients to do more work without additional compensation? One approach may be to create irrational motivations – e.g., the chance of winning a lottery might incentivize clients even though the expected value of winning may be small compared to the human computation required to obtain a "ticket," or the problems may be packaged in a way that gives clients entertainment value (as in the ESP Game).

Designing Interfaces. There are two types of interfaces in question: human-computer interfaces and programming language interfaces. What kinds of human-computer 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. Also, 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.

6. CONCLUSION

We did an initial exploration of 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.

Our paper put forth a basic framework and architecture for such systems. We then used simple tools from probability theory and game theory to analyze reliability and derive design principles. Our probabilistic analysis showed how many dishonest parties the scheme could tolerate, and described

how you might want to catch them using ringers. We also observed that while a tool like Bayesian inference is the one most commonly used for combining answers, it can perform *worse* than a simple majority vote, especially in the presence of adversaries. Our game-theoretic analysis suggested four design principles for mitigating the risk of defection by malicious parties: directly relate payout to a client's rating; decrease rating substantially for incorrect answers, and increment it slowly for correct ones; require clients to maintain a reserve; and finally, take extra measures when the utility for defection is especially high. We also mentioned numerous open problems in areas such as cryptography, algorithms, human-computer interaction, and programming language / API design. This is just a starting point. We believe that this notion of distributed *human* computation will spawn numerous areas of research interest.

7. ACKNOWLEDGEMENTS

We thank Phil Mackenzie for suggesting the ESP game example. We thank Naresh Apte for pointing us to a paper in which spam prevention is handled using collaborative filtering with human agents. We also thank Frank Bossen, Neil Daswani, Minoru Etoh, Ravi Jain, David Molnar and Khosrow Lashkari for numerous helpful discussions and for feedback on early drafts of this paper.

8. 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. *Eurocrypt 2003.*

[3] L. von Ahn and L. Dabbish. Labeling Images with a Computer Game. *ACM CHI 2004.* See also `http://www.espgame.org/`

[4] K. J. Arrow. Social Choice and Individual Values. *Coles Foundation for Research in Economics Monograph 12, 2nd edition, Yale University Press,* 1963.

[5] R. von Bitter. Mathenauts: Tales of Mathematical Wonder. *Arbor House Pub Co;* June 1987.

[6] M. Blum and S. Kannan. Designing Programs That Check Their Work, *STOC 1989.*

[7] M. Blum, M. Luby, and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *STOC 1990.*

[8] R. Cramer, R. Gennaro and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. *EUROCRYPT 97.*

[9] Cyphermint, Inc. `http://www.cyphermint.com/`

[10] The Distributed Proofreader's project. `http://www.pgdp.net/`

[11] X. Drèze and F. Hussherr. Internet Advertising: Is Anybody Watching? *Journal of Interactive Marketing,* 2003, Vol. 17 (4), 8-23.

[12] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32(2), April 1985, 374-382.

[13] The Search of Extraterrestrial Intelligence Project. University of California, Berkeley. `http://setiathome.berkeley.edu.`

[14] O. Goldreich. Secure Multiparty Computation. *Unpublished Manuscript.* Last revised October 2003.

[15] The Great Internet Mersenne Prime Search. `http://www.mersenne.org.`

[16] P. Golle and I. Mironov. Uncheatable Distributed Computations. *RSA Conference, Cryptographers' Track 2001.*

[17] P. Golle and S. Stubblebine. Distributed computing with payout: task assignment for financial- and strong- security. *Financial Cryptography 2001.*

[18] N. Lynch. Distributed Algorithms. *Morgan Kaufmann Publishers*, 1996.

[19] R. Motwani and P. Raghavan. Randomized algorithms. *Cambridge University Press,* New York, NY, 1995.

[20] The New York Times Web Site. `http://www.nytimes.com.`

[21] C. K. Prahalad and S. Hart. The Fortune at the Bottom of the Pyramid. *Strategy + Business*, Issue 26, Q1 2000.

[22] Project Gutenberg. `http://www.gutenberg.net/`

[23] Spam Net Web Site. `http://www.cloudmark.com.`

[24] 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)

[25] S. Stubblebine and P. van Oorschot. Online Dictionary Attacks with Login Histories and Human-in-the-Loop. *Financial Cryptography 2004.*

[26] Vipul's Razor Web Site. `http://sourceforge.net/projects/razor.`

[27] A. Yao. Protocols for Secure Computations. *IEEE Symposium on Foundations of Computer Science (FOCS) 1982.*

[28] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowicz. Approximate Object Location and Spam Filtering. *ACM Middleware, 2003.*