Co-Utile Peer-to-Peer Decentralized Computing

Josep Domingo-Ferrer, Alberto Blanco-Justicia, David Sánchez, Najeeb Jebreel
Universitat Rovira i Virgili
CYBERCAT-Center for Cybersecurity Research of Catalonia
UNESCO Chair in Data Privacy
Department of Computer Engineering and Mathematics
Av. Països Catalans 26, E-43007 Tarragona, Catalonia
e-mail {josep.domingo, alberto.blanco, david.sanchez, najeebmoharramsalim.jebreel}@urv.cat

***Abstract*—Outsourcing computation allows wielding huge computational power. Even though cloud computing is the most usual type of outsourcing, resorting to idle edge devices for decentralized computation is an increasingly attractive alternative. We tackle the problem of making peer honesty and thus computation correctness self-enforcing in decentralized computing with untrusted peers. To do so, we leverage the co-utility property, which characterizes a situation in which honest co-operation is the best rational option to take even for purely selfish agents; in particular, if a protocol is co-utile, it is self-enforcing. Reputation is a powerful incentive that can make a P2P protocol co-utile. We present a co-utile P2P decentralized computing protocol that builds on a decentralized reputation calculation, which is itself co-utile and therefore self-enforcing. In this protocol, peers are given a computational task including code and data and they are incentivized to compute it correctly. Based also on co-utile reputation, we then present a protocol for federated learning, whereby peers compute on their local private data and have no incentive to randomly attack or poison the model. Our experiments show the viability of our co-utile approach to obtain correct results in both decentralized computation and federated learning.**

***Index Terms*—P2P computing, Reputation, Self-enforcement, Co-utility, Edge computing, Federated learning**

I. Introduction

The possibility of outsourcing computation of heavy tasks to more capable entities or to a network of peers have brought supercomputer-like computational capabilities to end users and low-tier devices. The most prominent example is cloud computing. Even though computation outsourcing is commonly viewed as a centralized architecture, in which computation is delegated to a highly capable server, we can also find examples of decentralized peer-to-peer architectures in which a community of peers collaborate to complete a task. The most recent example is federated learning, promoted by Google and other players for collaborative machine learning based on smartphones [24].

Decentralization has many benefits. It allows a requester to get large computations done without owning large computing facilities or even renting cloud computing power. Computing tasks can be assigned to peers owning edge-computing devices that have free resources, such as idle smartphones. Furthermore, in the case of federated learning, decentralized computing allows the requester to learn a model based on data that are privately owned by the computing peers.

Even though computation outsourcing may be limited to fully trusted networks, in most cases the requester and the edge peers do not trust each other. In such a scenario, privacy and correctness become major concerns [34]. The privacy issue arises when the outsourced task requires analyzing personal or confidential data. This raises the problem of preventing the untrusted peers from learning sensitive information. A variety of methods can be employed to tackle this issue, ranging from fully cryptographic approaches that typically incur large overheads to lightweight data anonymization, which may limit the accuracy of the results [11]. On the other hand, lack of trust also means that the requester cannot be sure whether the results of the computation are correct. Failure to address correctness can bring serious consequences, such as model poisoning in the case of federated learning [2]. How to guarantee that peers correctly compute their assigned tasks is precisely the focus on this work.

A. Previous work

Verifiable computing [13] tackles the problem of dishonest peers that may modify their assigned tasks to return plausible results without performing the actual work [25].

The simplest form of verification is achieved by replication [6], [7]: the requester sends identical tasks to multiple peers and waits for the results. If a minimum quorum of the results agree, the requester assumes those results are correct. If the minimum quorum is not reached (due to off-line peers, communication failure, computation errors or cheating peers), the requester repeats the request to a different set of peers.

Since it is difficult to prevent dishonest peers from submitting false data, verification by replication may require substantial redundancy and thus incur significant overhead.

A solution to control untrusted peers is to use trusted hardware [27], such as secure coprocessors [29] or trusted platform modules [32], but this significantly complicates the network infrastructure and requires a chain of trust along with assumptions that the hardware works correctly.

Mathematical approaches to verifiable computing require the peers to return the results of the task and a proof that the computation was carried out correctly. Initially, verifiable computing proofs typically relied on very expensive constructions like interactive proofs [23] and probabilistically checkable proofs [30]. For many years, these approaches were

purely theoretical: interactive protocols were exponential-time and probabilistic checkable proofs were prohibitively complicated [33]. Modern approaches to verifiable computing take 'only' polynomial cost but are limited to verification of computations expressible as certain kinds of circuits; further, they employ fully homomorphic encryption [13], [8], which is far from being practical. Recently, more efficient (but more *ad-hoc*) solutions have brought verifiable computing closer to practicality [26], [28], [31]. However, none of these words is yet practical enough [33] and different protocols are needed for each task that is to be computed verifiably. Also in recent years, succinct non-interactive arguments (SNARGs) and succinct non-interactive arguments of knowledge (SNARKs) have been introduced to allow verification of NP statements with much reduced complexity [4], [1]. Nonetheless, SNARGs and SNARKs still need to be designed *ad hoc* for the statement to be verified.

An alternative approach aimed at ensuring correctness for the specific case of federated learning is [18]. It uses decentralized reputation to encourage correct computation by peers. A consortium blockchain is leveraged to achieve decentralized peer reputation management. The spirit of this proposal is similar to ours, because we also use decentralized reputation. However, its main drawback is that the computational cost of running a blockchain is likely to offset the computational gains of federated learning.

B. Contribution and plan of this paper

In this work we revisit the initial approach to verifiable computing via replication under the novel perspective of co-utility [12]. Co-utility is a property that characterizes protocols in which the best way for each participant to reach her goal is to help others reach theirs. By applying this notion to computation outsourcing we aim at incenting peers to deliver correct results. In this way, we minimize the required computation redundancy, that is, the most significant overhead of verifiable computing via replication.

To achieve co-utility in computation outsourcing, we rely on reputation as an incentive. In order to preserve the peer-to-peer (P2P) computing paradigm, reputation is also computed in a P2P decentralized way and its correctness is also ensured by co-utility.

In comparison with [18], our approach avoids the costly solution of using a blockchain to maintain decentralized reputations; additionally, our co-utile reputation protocol is self-enforcing and hence rationally sustainable. Compared to the proof-based approaches discussed above, our proposal is more general and flexible, since it allows peers to perform any computational task without requiring *ad hoc* protocols, and more efficient, since it does not entail a heavy cryptographic apparatus.

We first give a protocol for a generic case in which the requester supplies both the code and the data that the peers must use in their computation. We then give a protocol for the case of federated learning, in which peers improve the requester's model based on their own private data sets, that stay unknown to the requester.

The rest of this paper is organized as follows. Section II gives background on co-utility. Section III describes our reputation-based co-utile protocol for P2P correct decentralized computing. Section IV justifies the co-utility and the security of the proposed protocol. Section V presents reputation-based co-utile federated learning. Section VI justifies the co-utility, the security and the privacy of reputation-based federated learning. Empirical results are described in Section VII. Conclusions and future research lines are gathered in Section VIII.

II. Background on co-utility

Co-utility models a kind of interaction between rational agents in which the best option for each agent to reach her own goal is to help other agents reach theirs [12]. Co-utility focuses on self-enforcing protocols, that are those from which agents have no rational incentive to deviate.

More specifically, a protocol Π is co-utile if and only if *the three* following conditions hold:

1) Π is self-enforcing;
2) The utility derived by each agent participating in Π is strictly greater than the utility the agent would derive from not participating;
3) There is no alternative protocol Π' giving greater utilities to all agents and strictly greater utility to at least one agent.

The first condition ensures that if participants engage in the protocol, they will not deviate. The second condition is needed to ensure that engaging in the protocol is attractive for everyone. The third condition can be rephrased in game-theoretic terms by saying that the protocol is a Pareto-optimal solution of the underlying game.

III. Reputation-based P2P decentralized computing

In this section we depict the protocol we propose by which a requester R can delegate computation to peers of a P2P network. The protocol has been designed with the following goals in mind:

- *Decentralization*. Delegation of computation and accountability of the results should be controlled by the peers themselves, rather than by a central authority (that may be self-interested or attacked).
- *Anonymity*. Identifiers that reveal the real identity of peers (*e.g.*, IP addresses) should be avoided, to help thwart potential collusions. Instead, pseudonyms should be used.
- *Low overhead*. The overhead of providing accountability of results should not be large for the network and should be small or negligible for the computation requester (otherwise, the benefits of outsourcing computation dilute).
- *Proper management of new peers*. Newcomers should not enjoy advantages. Otherwise, malicious peers may be motivated to create new anonymous identities when identified after abusing the system.

- *Symmetry/asymmetry*. The protocol should support symmetric scenarios (where peers can be both task requesters and task workers) and asymmetric scenarios (where peers are just workers and the requester is an external party).

Under the co-utility framework, executing computation requests from others or performing accountability management are negative utilities for all the peers other than the requester. Therefore, rational agents have in principle no interest to execute these tasks. To enforce co-utility and ensure the sustainability of the network we need an artificial utility that compensates the negative utilities. For this, we rely on reputation, which can be used both as a reward and a penalty [10]. On the one hand, reputation allows peers to build *trust*, which can neutralize negative utilities related to mistrust. On the other hand, reputation also makes the peers accountable for their actions: if a peer misbehaves, her reputation worsens and the other peers mistrust her more and more and are less and less interested in fulfilling her requests. In this manner, free riders and also malicious peers can be identified and penalized (*e.g.*, through denial of their requests).

Several reputation mechanisms have been proposed in the literature for P2P networks; see [16] for a comparison and a discussion. Since in our work reputation is used as a means to turn P2P computing co-utile, the reputation calculation should be decentralized and co-utile itself.

To keep the network decentralized we assume that each peer is associated with a number M of peers that will act as "accountability managers" (AM) of the computation requested by R. The association between peers and their AMs is defined according to a distributed hash table (DHT), which maps a peer with pseudonym ID_i to peers with pseudonyms $h_0(ID_i), ..., h_{M-1}(ID_i)$, where $h_0, h_1, ..., h_{M-1}$ are one-way hash functions. Inversely, we call "pupils" the peers for whom peer $\mathcal{P}_i$ is an accountability manager. Note that the use of pseudonyms provides anonymity, and the use of a DHT prevents anyone from choosing a particular pseudonym as AM for $\mathcal{P}_i$. With this mapping, on average, every peer in the network is the AM of M other peers; hence, the work of peers as accountability managers is balanced. Also, as we discuss later, the M-fold redundancy among AMs ensures that malicious AMs can be identified (and punished).

A. Co-utile decentralized reputation

In [10] we proposed a co-utile extension of the well-known EigenTrust decentralized reputation protocol [17], which fulfills the aforementioned design goals and is robust against rational attacks related to reputation tampering.

The protocol consists in calculating a global reputation for each peer $\mathcal{P}$ based on aggregating the local opinions of the peers that have interacted with $\mathcal{P}$. If we represent the local opinions by a matrix whose component (i, j) contains the opinion of peer $\mathcal{P}_i$ on peer $\mathcal{P}_j$, the distributed calculation mechanism computes global reputation values that approximate the left principal eigenvector of this matrix.

The opinion of peer $\mathcal{P}_i$ on another peer $\mathcal{P}_j$ is the reputation s_{ij} of $\mathcal{P}_j$ local to $\mathcal{P}_i$. This value is defined as the aggregation of ratings (positive or negative) that $\mathcal{P}_i$ has issued as a result of the set Y_{ij} of interactions with $\mathcal{P}_j$:

$$s_{ij} = \sum_{y_{ij} \in Y_{ij}} \mathit{rating}_i(y_{ij}).$$

In order to properly aggregate the local reputation values computed by each peer, a normalized version c_{ij} is first calculated as:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)}.$$

In this manner, the normalized local reputation values lie between 0 and 1 and their sum is 1. This makes all peers equal contributors to the global reputation and avoids dominance by peers with a larger number of experiences. Moreover, this normalized calculation deters peers from colluding by assigning arbitrarily high values to each other. Finally, truncation of negative reputation values to 0 prevents selfish peers from assigning arbitrarily low values to good peers.

A side effect of negative value truncation is that no distinction is made between peers with whom $\mathcal{P}_i$ had a bad experience (negative local reputation) and those with whom $\mathcal{P}_i$ never interacted so far. Hence, newcomers do not enjoy any reputation advantage, because their reputation is indistinguishable from the one of misbehaving agents. As a result, a selfish agent has no incentive to take a new virtual identity in order to "clean" his reputation after misbehaving with other peers. Likewise, newcomers become instantly motivated to positively contribute to the system in order to earn the minimum reputation that other agents require from them.

Local reputation values are then disseminated and aggregated through the network by following the transitive reputation calculation algorithm described in [10], in order to obtain the global reputation value g_i of each peer $\mathcal{P}_i$.

B. Decentralized P2P computation outsourcing and reputation calculation protocols

Let R be the requester, that is, the party interested in using the P2P community to perform a demanding computation C. R may be a peer himself or an external party. We assume R has access to a list of the *pseudonyms* of the peers, and has a number M of AMs associated with her. We propose the reputation-based P2P decentralized computing described in Protocol 1 that works as follows.

First R splits the computation C into a number of independent tasks C_1, ..., C_m, where each task includes code and input data, plus any necessary randomness seeds.

Then a computation redundancy level g is selected, where g is a (typically small odd) nonnegative integer. Computational redundancy is used to detect malicious peers, in case the results they report do not match those of the majority. If global reputations $t_1, \ldots, t_N$ for the N peers of the network are not yet available (because the network is just starting collaboration), then $g = g_0$ is taken, where g_0 is a parameter. On the contrary, if global reputations are available, then $g = \gamma(t_1, \ldots, t_N)$ is

taken, where γ is a function that decreases as the proportion of high-reputation peers grows; in particular, when there are many high-reputation peers $\gamma(t_1, \ldots, t_N) < g_0$, that is, in a highly reputable P2P network less computation redundancy is required.

Then R selects a subset L of $m' = m \times g$ peers $ID_1, \ldots, ID_{m'}$ that have the highest global reputation among those peers that are available (not currently busy with other tasks). After that, R computes and publishes a random permutation $\pi(L)$ of the m' selected peer pseudonyms. Then R sends task C_1 to the first g peers in $\pi(L)$, task C_2 to the next g peers in $\pi(L)$, and in general C_i to the i-th group of g peers in $\pi(L)$. Each selected peer runs her assigned task. After all tasks are complete, each accountability manager considers each of his pupils $\mathcal{P}_i$ and looks for the $g-1$ peers $\mathcal{P}_j \in \pi(L)$ that had the same task as $\mathcal{P}_i$; then the accountability manager assigns $\mathcal{P}_i$ normalized local reputation $c_{ji} = 1$ if $\mathcal{P}_i$'s result is the majority result, and $c_{ji} = 0$ otherwise.

At this point, normalized local reputations are available for all peers in L. Now Protocol 2, a simplified version of EigenTrust's decentralized global reputation computation, can be run. The main difference is that, in our protocols, local and global reputations are assigned and computed, respectively, by the accountability managers themselves. To make Protocol 2 robust against self-interested attacks by the accountability managers, the reputation value of a peer $\mathcal{P}_i$ is computed by *all* M accountability managers associated with her. The initial global reputation $t_i^{(0)}$ of peer $\mathcal{P}_i$ is taken to be be the global reputation for that peer computed in previous protocol executions (if available) or a default value (if no previous global reputation is available).

After the computation of the global reputation values, if a requester needs the reputation value of another peer $\mathcal{P}_i$ in order to assign tasks to this peer, she can query the accountability managers of $\mathcal{P}_i$ for the peer's reputation. The M values obtained from the M accountability managers should be the same, because the inputs of the managers are the same. However, if some values are different, the requester can take the most common value.

Protocol 1 REPUTATION-BASED CO-UTILE P2P COMPUTING

Input: N, m, g_0, $\gamma()$

1: R splits the computation C into a number of tasks C_1, C_2, …, C_m, each including code and data (plus any needed random seeds);
2: **if** global peer reputations $t_1, \ldots, t_N$ are available **then**
3: Let $g = \gamma(t_1, \ldots, t_N)$;
4: R selects a subset L of $m' := m \times g$ peers with the highest global reputation among those that are available (not busy);
5: **else**
6: Let $g = g_0$;
7: R selects a random subset L of $m' := m \times g$ available peers;
8: **end if**
9: R computes and publishes a random permutation $\pi(L)$ of the m' selected peer pseudonyms;
10: R sends task C_1 to the first g peers in $\pi(L)$, task C_2 to the next g peers in $\pi(L)$, and in general C_i to the i-th group of g peers in $\pi(L)$;
11: **for all** selected peers $\mathcal{P}$ **do**
12: $\mathcal{P}$ completes its assigned task;
13: **end for**
14: **for all** accountability managers AM **do**
15: **for all** pupil $\mathcal{P}_i \in L$ **do**
16: AM looks for the subset $L_i \subseteq L$ of $g-1$ peers that had the same task as $\mathcal{P}_i$;
17: **for all** $\mathcal{P}_j \in L_i$ **do**
18: **if** $\mathcal{P}_j$'s result is the same as $\mathcal{P}_i$'s **then**
19: AM assigns local reputation $c_{ji} = 1$ to $\mathcal{P}_i$;
20: **else**
21: AM assigns $c_{ji} = 0$ to $\mathcal{P}_i$;
22: **end if**
23: **end for**
24: **end for**
25: **end for**
26: **call** Protocol 2 to update the global reputation t_i of each peer $\mathcal{P}_i$.

IV. CO-UTILITY AND SECURITY OF REPUTATION-BASED P2P COMPUTING

In the following we show that the security of Protocol 1, that is, the correctness of the computations by the peers, rests on co-utility. For peers to be willing to collaborate, they must be rewarded for their work in such a way that the reward is higher than the costs they incur to do the work.

In an asymmetric scenario, in which requester R is an external party, she must reward $\mathcal{P}_i$ after a computation if a majority of local reputations c_{ji} is 1. This can be enforced by having accountability managers request R to reward the peers. Specifically, if a majority of $\mathcal{P}_i$'s M accountability managers make such a request, then R rewards $\mathcal{P}_i$ in some way (money, loyalty points, quality of service, etc.).

On the other hand, in a symmetric scenario, in which peers can be also requesters of their own tasks, reputation can be itself the reward. In this setting, a necessary and sufficient condition for peers to run the requester's tasks is that the requester has a high reputation or, in other words, peers can decline requests if the requester has a low reputation. Therefore, peers can only ensure their access to the computing service if they behaved correctly in previous iterations and earned reputation as a result. Newcomers start with zero reputation; to increase it, they need to be selected for the set L, which will occur when the higher-reputation peers are either busy with other computations or otherwise unwilling to do any more computations.

Proposition 1 (Co-utility)**.** *Assume most peers are honest and let $reward_i$ be the total reward earned by peer $\mathcal{P}_i \in L$ for correctly performing a computation whose cost is $cost_i$. Then, if $reward_i > cost_i$, Protocol 1 is co-utile.*

Protocol 2 CO-UTILE P2P GLOBAL REPUTATION COMPUTATION

1: **for all** $\mathcal{P}_i$ **do**
2: **for all** pupils $\mathcal{P}_d$ of $\mathcal{P}_i$ **do**
3: **for all** $g-1$ peers $\mathcal{P}_j$ that had the same task as $\mathcal{P}_d$ **do**
4: Query all the accountability managers of $\mathcal{P}_j$ for $c_{jd}t_j^{(0)}$;
5: **end for**
6: $k := -1$;
7: **repeat**
8: $k := k+1$;
9: Compute $t_d^{(k+1)} = c_{1d}t_1^{(k)} + c_{2d}t_2^{(k)} + \ldots + c_{nd}t_n^{(k)}$;
10: **for all** $g-1$ peers $\mathcal{P}_j$ that had the same task as $\mathcal{P}_d$ **do**
11: Send $c_{dj}t_d^{(k+1)}$ to all the accountability managers of $\mathcal{P}_j$;
12: Wait for all accountability managers of $\mathcal{P}_j$ to return $c_{jd}t_j^{(k+1)}$;
13: **end for**
14: **until** $|t_d^{(k+1)} - t_d^{(k)}| < \epsilon$; // Parameter $\epsilon > 0$ is a small value
15: **end for**
16: **end for**

Proof. We must examine co-utility for peers in L and for accountability managers. A peer $\mathcal{P}_i \in L$ has two possible strategies:

1) *Compute her task correctly.* Since most peers are honest and those in L have high reputation, a majority of peers in L can be assumed honest and $\mathcal{P}_i$'s correct result earns her a majority of local reputations $c_{ji} = 1$. In turn, this earns $\mathcal{P}_i$ a reward $reward_i > cost_i$ from R. Thus, her utility is
$$u_i = reward_i - cost_i > 0.$$
Additionally, $\mathcal{P}_i$'s global reputation increases, which increases the probability that $\mathcal{P}_i$ is again selected for L in the next protocol execution or that $\mathcal{P}_i$'s tasks are accepted is she becomes a requester (note that only peers in L may earn rewards).
2) *Compute her task incorrectly or do nothing.* In this case, $\mathcal{P}_i$ receives 0 reward and may have had some cost if computing an incorrect result. Hence, $u_i \leq 0$. Furthermore, $\mathcal{P}_i$'s global reputation decreases, which reduces the probability that $\mathcal{P}_i$ is selected for L in the next protocol execution and increases the probability that if $\mathcal{P}_i$ becomes a requester her submitted tasks are declined by other peers.

Hence, for a peer in L the best option is to perform her computation correctly.

Now let us turn to accountability managers:

- An accountability manager might deny service by not computing the global reputation of her pupils. In this case, the querying peer in line 4 of Protocol 2 will receive no answer. The querying peer will consider that accountability manager as not active and will remove it from the list of available peers, which will decrease that accountability manager's reputation in the next protocol execution. Thus, a rational peer acting as accountability manager is not interested to deny service.
- Even if not denying service, an accountability manager might award a wrong local reputation c_{ji} in Protocol 1 (lines 19 and 21). However, accountability managers are chosen by means of a hash function, that is, randomly. By assumption, most peers are honest so that if M is sufficiently large, the number of malicious managers can safely be considered less than $M/2$. In this case, the querying peer in line 4 of Protocol 2 can reconstruct the true reputation by a majority decision. Further, the querying peer can lower the local reputation she awards to the corresponding cheating accountability managers.

Hence, the safest strategy for an accountability manager is to perform her role correctly. □

Security understood as a guarantee of correct computation follows from Proposition 1. Using pseudonyms for peers contributes to thwarting collusion, because peers are not sure of each other's identities.

V. REPUTATION-BASED FEDERATED LEARNING

In the case of federated learning, there are several peers each of whom updates the model based on her local data.

A. Public data

In the simplest case, the local data of peers are provided by the requester. Thus, at each round of task assignments, the requester R also assigns local data sets to the different peers, in such a way that each local data set is shared by g peers. The assignment of local data sets to peers can be completely different for each round.

Those peers sharing a certain data set also share the code for computing a model update, including any random seeds. In this way, if the g peers sharing a local data set are honest, they should obtain exactly the same result, which allows using Protocol 1 for each task assignment round.

B. Private local data

Protocol 1 is not applicable if the requester R cannot ensure that groups of g peers have exactly the same local data sets. In particular, this occurs if peers compute model updates based on private local data sets that they do not want to share, for example fitness or health data recorded by their smartphones. Protocol 3 deals with the case of federated learning with private local data and works as follows.

First, the level of computation redundancy g is set in the same way explained in Section III-B, depending on whether global reputations exist or not; in particular, the greater the proportion of highly reputable peers, the smaller is g. Then if global reputations exist, R selects a subset L of g peers that have the highest global reputations among those peers that

are available (not currently busy or marked as unavailable). If no global reputations exist yet, R selects a random subset L of g peers among those available. Subsequently, R sends the current model with parameters θ to all peers in L. Next, every peer $\mathcal{P}_i \in L$ computes a model update with parameters θ_i based on her local data and sends θ_i to *all* accountability managers (not only hers).

At this point, accountability managers start their work. Each manager computes the centroid c_L of the model update parameters of peers in L. Then they compute the distances $dist_i$ between each θ_i and c_L. After that, they compute the first quartile $Q1$ and the third quartile $Q3$ of the set of distances, and the interquartile range $IQR = Q3 - Q1$. This allows them to find the set

$$\lambda = \{\mathcal{P}_i \in L | Q1 - \tau \times IQR \leq dist_i \leq Q3 + \tau \times IQR\}.$$

In a centralized setting, the previous computations yielding λ would be performed by a single party. Yet, in a P2P network, it is more robust and reliable that each accountability manager replicates them.

The next step is for each accountability manager to check which of her pupil peers $\mathcal{P}_i$ are in λ. If $\mathcal{P}_i \in \lambda$, the manager assigns $\mathcal{P}_i$ local reputation $c_{ji} = 1$, for all j s.t. $\mathcal{P}_j \in \lambda$. If $\mathcal{P}_i \notin \lambda$, then $\mathcal{P}_i$'s update is an outlier and the accountability manager assigns local reputation $c_{ji} = 0$, for all j s.t. $\mathcal{P}_j \in \lambda$. An outlying update θ_i may be due to $\mathcal{P}_i$ cheating to poison the model or to $\mathcal{P}_i$ being honest but having genuinely outlying local data. Our distance-based approach to detect poisoning is consistent with [5], [3]; Figure 4 in the latter paper shows that the more malicious poisoning is, the more distant updates it yields.

Finally, Protocol 2 is called to update the global reputations of peers, and the global model parameters θ are updated by aggregating those update parameters θ_i such that $\mathcal{P}_i \in \lambda$ according to the majority of accountability managers.

Protocol 3 REPUTATION-BASED CO-UTILE P2P FEDERATED LEARNING WITH PRIVATE LOCAL DATA

Input: N, g_0, $\gamma()$, τ

if global peer reputations $t_1, \ldots, t_N$ are available **then**
  Let $g = \gamma(t_1, \ldots, t_N)$;
  R selects a subset L of g peers with the highest global reputations among those that are available (not busy);
else
  Let $g = g_0$;
  R selects a random subset L of g available peers;
end if
R sends the current model parameters θ to all peers in L;
for all $\mathcal{P}_i \in L$ **do**
  $\mathcal{P}_i$ computes an update θ_i of model parameters on her private data;
  $\mathcal{P}_i$ sends θ_i to *all* accountability managers;
end for
for all accountability managers AM **do**
  AM computes the centroid c_L of the model updates of peers in L;
  AM computes the distance $dist_i$ between θ_i and c_L for all $\mathcal{P}_i \in L$;
  AM computes the 25th percentile $Q1$ and the 75th percentile $Q3$ of the set $\{dist_i : \mathcal{P}_i \in L\}$;
  AM computes the interquartile range $IQR = Q3 - Q1$;
  AM computes $\lambda = \{\mathcal{P}_i \in L | Q1 - \tau \times IQR \leq dist_i \leq Q3 + \tau \times IQR\}$;
  Let D_{AM} be the set of AM's pupil peers;
  for all $\mathcal{P}_i \in D_{AM}$ **do**
    if $\mathcal{P}_i \in \lambda$ **then**
      AM assigns local reputation $c_{ji} = 1$ $\forall j$ s.t. $\mathcal{P}_j \in \lambda$;
      AM assigns local reputation $c_{ji} = 0$ $\forall j$ s.t. $\mathcal{P}_j \notin L \setminus \lambda$;
    else
      AM assigns local reputation $c_{ji} = 0$ $\forall j$ s.t. $\mathcal{P}_j \in L$;
    end if
  end for
end for
call Protocol 2 to update the global reputation t_i of each peer $\mathcal{P}_i$;
R updates the global model parameters θ by aggregating those θ_i such that $\mathcal{P}_i \in \lambda$.

VI. CO-UTILITY, SECURITY AND PRIVACY OF REPUTATION-BASED FEDERATED LEARNING

Similarly to what we did for Protocol 1, we will show that the security of Protocol 3, understood as correct computation by the peers, rests on co-utility.

Like we remarked for Protocol 1, in some cases reputation can itself be the reward. This is the situation if peers are interested in being requesters in the future and a high reputation is needed by a requester to have her tasks run by the peers. Yet further, there may be federated learning settings in which not even reputation is needed as a reward. This occurs if peers are genuinely interested in contributing to improving the model parameters θ because a well-adjusted model is beneficial for all of them (*e.g.* to get better recommendations or better pattern recognition). In this case, the entire reputation management could be skipped.

Proposition 2 (Co-utility)**.** *Assume most peers are honest and let $reward_i$ be the total reward earned by peer $\mathcal{P}_i \in L$ for correctly performing a computation whose cost is $cost_i$. Then, if $reward_i > cost_i/(1 - p_\tau)$, where p_τ is the proportion of outliers for parameter τ, Protocol 1 is co-utile.*

Proof. We only need to examine co-utility for peers in L because co-utility for accountability managers follows by an argument similar to that of the proof of Proposition 1. A peer $\mathcal{P}_i \in L$ has two possible strategies:

1) *Honestly follow the protocol.* Since most peers are honest and those in L have high reputation, a majority of peers can be assumed to return a correct model update

computed on their local data. However, it may be the case that $\mathcal{P}_i$'s update θ_i is a genuine outlier, that is, $\mathcal{P}_i \notin \lambda$; in this case an honest behavior would not be rewarded. Yet, by increasing parameter τ the proportion of outliers and therefore the probability p_τ of an honest peer not being rewarded can be made small. Now, an honest $\mathcal{P}_i$'s utility function is $u_i = (1-p_\tau)reward_i - cost_i$. By the assumption of the proposition, $reward_i > cost_i/(1-p_\tau)$ and hence $u_i > 0$. Beyond direct reward, an additional incentive for a peer to be in λ is that her global reputation increases, which in turn increases the probability that $\mathcal{P}_i$ is again selected for L in the next task assignment round (being in L is the only way to earn future rewards).

2) *Deviate from the protocol or do nothing.* If $\mathcal{P}_i$ does nothing and returns no result, her utility is $u_i = 0$ (no cost, no reward). If $\mathcal{P}_i$ deviates and computes a wrong model update θ_i, two subcases can occur:
   - θ_i happens to be similar to the models computed by most peers in L and thus $\mathcal{P}_i \in \lambda$. In this case $\mathcal{P}_i$ is rewarded. This is not unfair, because θ_i does not poison the model significantly. Sensitivity to model poisoning by wrong parameter updates θ_i can be increased by reducing τ (doing so increases the probability that a wrong θ_i excludes $\mathcal{P}_i$ from λ, but it also increases the probability p_τ that an honest peer is not rewarded).
   - θ_i is an outlier and hence $\mathcal{P}_i \notin \lambda$. If the deviation is significant, this is the most likely situation, which brings nonzero cost and zero reward, and hence $u_i < 0$.

   Hence, for a peer in L the best option is to perform her computation correctly. Regarding accountability managers, the co-utility of their correct operation is justified in a way analogous to the proof of Proposition 1.

□

Thus, co-utility makes security in the sense of correct computation the best option for peers. Regarding the privacy of the peers' local data sets, it remains the same as in the underlying federated learning model: any leakage of private data can only come from θ_i.

VII. Empirical results

A. Reputation-based co-utile P2P computing

We implemented Protocol 1 and used it for two types of tasks:

- Tasks with binary result. For such tasks, the correct result is recovered only if there is an absolute majority of correct results (because all incorrect results are equal to each other).
- Tasks with non-binary result. In this case, the correct result is recovered as long as it is the most frequent one (although not necessarily reported by the absolute majority of the g peers). For example, if $g = 5$ and two peers report the correct result r_1 and the other three peers report wrong results r_2, r_3 and r_4 that are different from each other and from r_1, then it can be established that r_1 is the correct result.

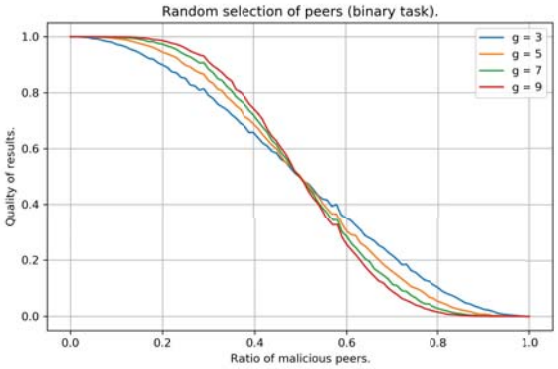

Fig. 1. Protocol 1. Binary task quality with random peer selection

Taking the above into account, we measured the quality of the results as follows. We counted how many of the m tasks assigned in each run of Protocol 1 were such that the correct result was the most frequent one among the g results returned by the peers running the task. Then we defined the quality as the previous count divided by m, which gave a metric between 0 and 1.

Figure 1 displays the quality of the results as a function of the proportion of malicious peers in the network when using Protocol 1 with a random choice of peers (that is, when global reputations are not available) for tasks with *binary* results and several computation redundancy levels g. Figure 2 shows the same but when global reputations are available and peers are selected by reputation. It can be seen that, when reputations cannot be used, one needs to increase the redundancy g to improve the quality; however this redundancy-based improvement only works as long as a majority of peers is honest, otherwise more redundancy yields worse quality. In contrast, when reputations are available, the best results are obtained with the lowest redundancy $g = 3$ as long as a majority of peers is honest; if the majority is malicious, it is better to use more redundancy. Thus, if reputations are available and the majority is honest, choosing peers based on reputations allows minimizing redundancy and hence the overhead.

Figure 3 displays the quality of the results as a function of the proportion of malicious peers when using Protocol 1 with a random choice of peers (that is, when global reputations are not available) for tasks with *non-binary* results and several computation redundancy levels g. Figure 4 shows the same but when global reputations are available and peers are selected by reputation. It can be seen that, when reputations cannot be used, one needs to increase the redundancy g to improve the quality. Unlike in the binary case, this redundancy-based improvement works even if honest peers are as few as 20%. In contrast, when reputations are available, the best results

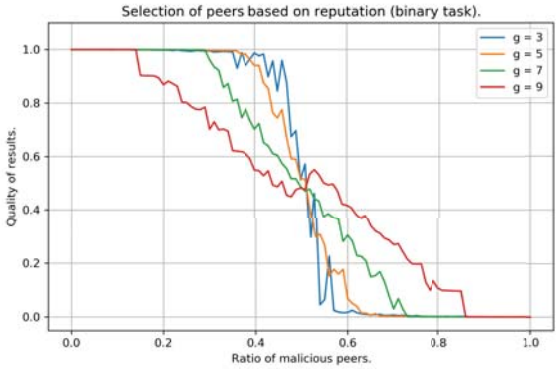

Fig. 2. Protocol 1. Binary task quality with reputation-based peer selection

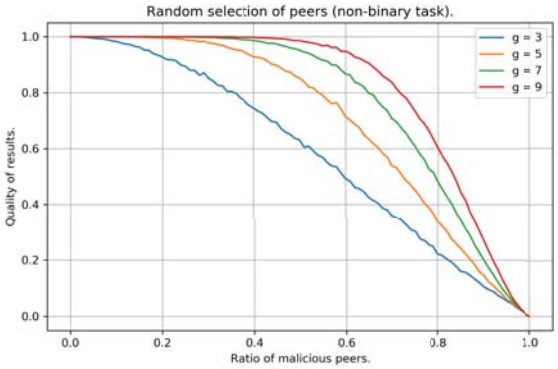

Fig. 3. Protocol 1. Non-binary task quality with random peer selection

are obtained with the lowest redundancy $g = 3$, like for binary tasks. The difference with the binary case is that the lowest redundancy yields the best results *no matter what the proportion of honest peers is*. Thus, for non-binary tasks it is even more clear that the use of reputations allows reducing the redundancy while improving the quality.

Note 1 (Further redundancy reduction for $g = 3$)**.** The lowest redundancy $g = 3$ turns out to be the best choice when reputations are used. In fact, it is possible to obtain majority results with less than 3 replications, as follows. First, replicate the task twice. If the two results agree, take the common result as the good one (as it will be majority even if a third replication is run). If they disagree, run a third task replication and take the majority result of the three replications (if any) as the good one. Let p_h be the proportion of honest peers. For a binary task, the expected number of replications is $E(g) = 2(p_h^2 + (1 - p_h)^2) + 3(1 - (p_h^2 + (1 - p_h)^2))$; it can be seen that when $p_h = 1$ (all honest) or $p_h = 0$ (all dishonest), we have $E(g) = 2$; the highest redundancy occurs when $p_h = 1/2$, in which case $E(g) = 2.5$, which is still less than 3. For a non-binary task, $E(g)$ depends on the number and distribution of incorrect results; anyway, it is a number between 2 and 3, obtained as $E(g) = 2p_a + 3(1 - p_a)$, where p_a is the probability that the first two results agree.

B. Reputation-based federated learning

For the experiments on Protocol 3, we used the MNIST public dataset. The dataset consists of $70,000$ images of handwritten digits (classes), from 0 to 9. Each image is gray-scaled and 28×28 in size. We used 60,000 of them for training and the remaining $10,000$ for testing. We divided the training dataset into 100 equally sized shards which were distributed among the 100 peers in the network. Whenever a training task was posted to the network, the requester R randomly chose 15% of the 100 peers to act as workers, which meant a set L with $g = 15$ peers. Each peer in L, called worker in what follows, had a different combination of $M = 7$ peers who acted as accountability managers.

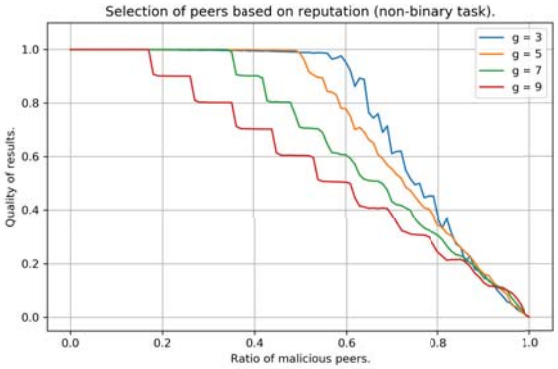

Fig. 4. Protocol 1. Non-binary task quality with reputation-based peer selection

We used a convolutional neural network (CNN) with two convolutional layers and max pooling layers followed by two fully connected layers. We used 10 filters of size 5×5 in the first convolution layer and 20 filters of size 5×5 in the second layer. The convolution layers were followed by two fully connected layers with 50 hidden units. The first fully connected layer took the inputs from the convolutional layers and applied weights to predict the correct label. The fully connected output layer gave the final probabilities for each label. R trained the global model for 30 global training epochs for each task. In each epoch, R sent a copy of the global model to workers in L and asked them to train the model (locally) using a momentum stochastic gradient descent optimizer with 3 local epochs, local batch size 8, learning rate 0.0002 and momentum 0.9.

We used the precision metric to evaluate the performance of the model. Precision is the amount of true positives TP divided by the sum of true positives and false positives FP, that is $TP/(TP + FP)$.

The dashed line in Figures 5, 6 and 7 shows the classification precision achieved by the global model at each epoch when all workers act honestly. The global model achieves

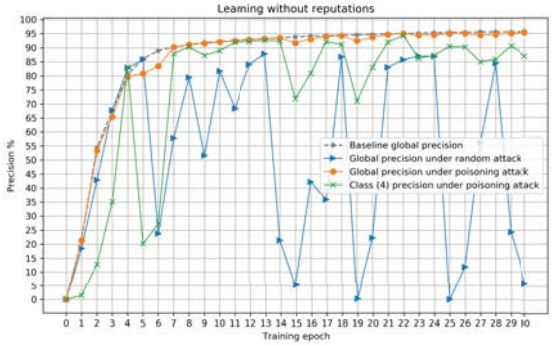

Fig. 5. Protocol 3. Global model precision evolution without reputations

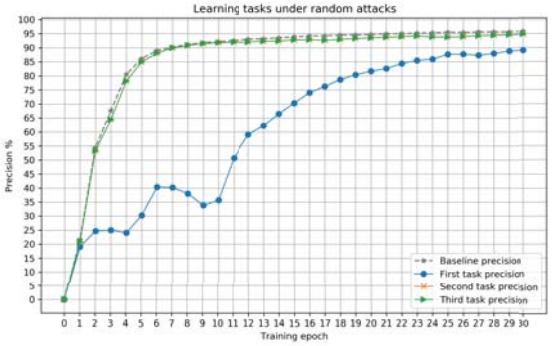

Fig. 6. Protocol 3. Global model precision evolution under random attacks using reputations

a precision around 96% after 30 epochs. In addition to the all-honest case, we considered scenarios where 20% of the workers were malicious. Malicious workers launched either a random or a poisoning attack at each global epoch with 70% probability, otherwise they acted honestly. Attacks were as follows:

- Random attack. The attacker tried to prevent the model from converging by sending random updates.
- Poisoning attack. The attacker tried to cheat the global model into misclassifying some specific inputs. In our experiments, the target class was 4, which the malicious workers tried to misclassify as 7.

Figure 5 shows the evolution of precision when the global model was trained without reputations, which meant that the $g = 15$ workers were selected at random. Clearly, random attacks prevented the model from converging. In contrast, poisoning attacks did not affect much the global precision of the model, which remained very close to baseline precision. This is not surprising, because by design poisoning attacks were aimed at introducing bias for class 4 but not for the other classes.

We next introduced reputations. Since reputations were available, the requester R chose the $g = 15$ peers with highest reputation as workers.

Solid lines in Figure 6 show the evolution of the precision of the global model under random attacks for the first, second and third tasks using reputations. Accountability managers used $\tau = 1.5$ to compute the set λ of good updates. Given that reputations were not yet available when running the first task, the model was unable to converge at the beginning because malicious workers could not be excluded. However, in the last epochs of the first task, most malicious workers were already filtered out. Thus, the opinions of their accountability managers were effective to neutralize their impact on the performance of the global model. For the second and the third tasks, all malicious peers had been detected and excluded from the very beginning (their reputation had been set to 0), which increased the model precision to the same level attained when all workers were honest.

Figure 7 shows the evolution of the model under poisoning attacks. Accountability managers used $\tau = 1$. The accuracy for the targeted class, 4, dropped as the model misclassified it as class 7 after the malicious peers launched their attacks. Detecting stealthy attackers is more difficult than detecting random attackers and hence it took longer to neutralize the former. This is why the model achieved good accuracy only in the fifth learning task, unlike for random attacks in which malicious peers were eliminated already at the end of the first task.

VIII. Conclusions and future research

Co-utility can make correctness in decentralized computing rationally sustainable. Decentralized reputation is a powerful incentive to achieve co-utility. We have described protocols for reputation-based decentralized P2P computing and for reputation-based federated learning. In both applications, reputation allows reducing the computation redundancy needed to withstand malicious peers.

Future work will aim at improving the robustness of reputation-based federated learning against sophisticated poisoning attacks, such as those using generative adversarial networks (*e.g.* [15]).

Acknowledgment and disclaimer

This work was motivated by a research contract from Huawei Finland. We thank Rami Haffar for help with the experiments. We acknowledge support from: European Commission (project H2020-871042 "SoBigData++"), Government of Catalonia (ICREA Acadèmia Prize to J. Domingo-Ferrer and grant 2017 SGR 705) and Spanish Government (projects RTI2018-095094-B-C21 and TIN2016-80250-R). The authors are with the UNESCO Chair in Data Privacy, but their views here are not necessarily shared by UNESCO.

References

[1] H. Ahmad, L. Wang, H. Hong, J. Li, H. Dawood, M. Ahmed, Y. Yang. Primitives towards verifiable computing: a survey. Frontiers in Computer Science 12(3) (2018) 451-478.

[2] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, V. Shmatikov. How to backdoor federated learning. arXiv preprint 1807.00459v3, Aug. 6, 2019.

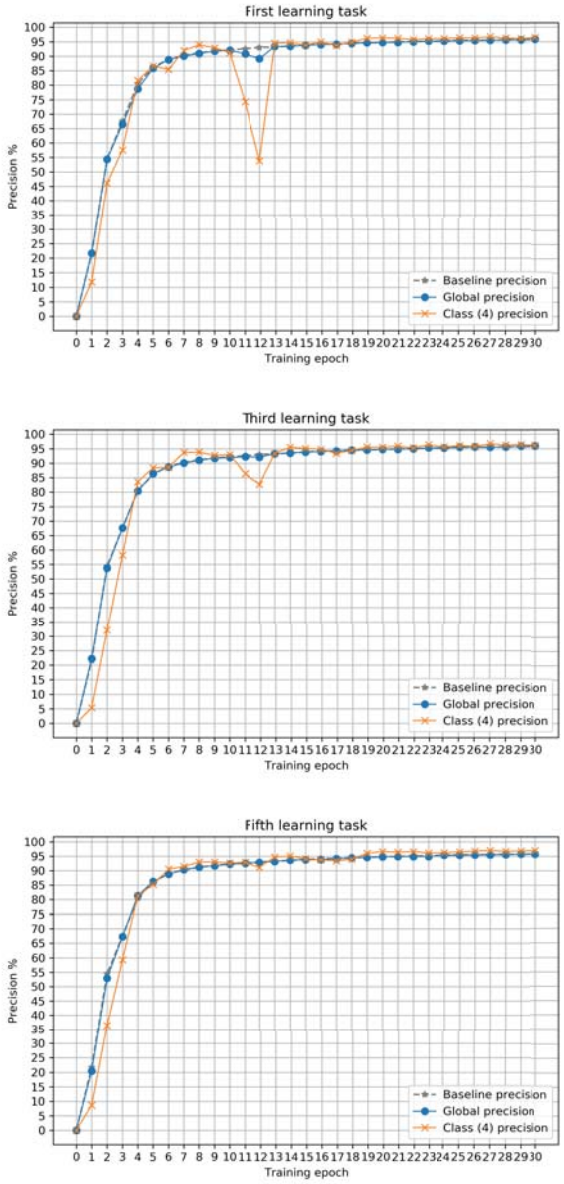

Fig. 7. Protocol 3. Global model precision evolution under poisoning attacks using reputations. Top to down, plots for first, third and fifth tasks.

[3] A. N. Bhagoji, S. Chakraborty, P. Mittal, S. Calo. Analyzing federated learning through an adversarial lens. arXiv preprint 1811.12470v3, Mar. 2, 2019.

[4] N. Bitansky, R. Canetti, A. Chiesa, E. Tromer. From extractable collision resistance to succint non-interactive arguments of knowledge, and back again. In: 3rd Innovations in Theoretical Computer Science Conference-ITCS'12, pp. 326-349. ACM, 2012.

[5] P. Blanchard, E. M. ElMhamdi, R. Guerraoui, J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In 31st Conf. on Neural Information Processing Systems-NIPS 2017, pp. 119-129. NIPS Proceedings, 2017.

[6] R. Canetti, B. Riva, G.N. Rothblum. Practical delegation of computation using multiple servers. In: 18th ACM Conference on Computer and Communications Security-CCS'11, pp. 445-454. ACM, 2011.

[7] M. Castro, B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems 20(4) (2002) 398-461.

[8] K.M. Chung, Y. Kalai, S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In: Advances in Cryptology-CRYPTO 2010, pp. 483-501. Springer, 2010.

[9] W. Diffie, M. E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory IT-22(6) (1976) 644-654.

[10] J. Domingo-Ferrer, O. Farràs, S. Martínez, D. Sánchez, J. Soria-Comas. Self-enforcing protocols via co-utile reputation management. Information Sciences 367-368 (2016) 159-175.

[11] J. Domingo-Ferrer, O. Farràs, J. Ribes-González, D. Sánchez. Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges. Computer Communications 140-151 (2019) 38-60.

[12] J. Domingo-Ferrer, S. Martínez, D. Sánchez, J. Soria-Comas. Co-utility: self-enforcing protocols for the mutual benefit of participants. Engineering Applications of Artificial Intelligence 59 (2017) 148-158.

[13] R. Gennaro, C. Gentry, B. Parno. Non-interactive verifiable computation: outsourcing computation to untrusted workers. In: Advances in Cryptology-CRYPTO 2010, pp. 465-482. Springer, 2010.

[14] R. W. Hamming. Error detecting and error correcting codes. Bell System Technical Journal 29(2) (1950) 147-160.

[15] B. Hitja, G. Ateniese, F. Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In: Proc. of the 2017 ACM SIGSAC Conf. on Computers and Communications Security - CCS'17, pp. 603-618. ACM, 2017.

[16] K. Hoffman, D. Zage, C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. ACM Computing Surveys 42(1) (2009) art. no. 1.

[17] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In: 12th International Conference on World Wide Web, pp. 640-651. ACM, 2003.

[18] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, M. Guizani. Reliable federated learning for mobile networks. arxiv:1910.06837v1, Oct. 14, 2019.

[19] V. Kolesnikov, R. Kumaresan. Improved OT extension for transferring short secrets. In: CRYPTO 2013, Part II, pp. 54-70. Springer, 2013.

[20] V. Kolesnikov, R. Kumaresan, M. Rosulek, N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In: 24th ACM Conference on Computer and Communications Security-CCS'16, pp. 818-829. ACM, 2016.

[21] Y. LeCun, C. Cortes, C.J.C. Burges. The MNIST Database of Handwritten Digits. http://yann.lecun.com/exdb/mnist/ Retrieved March 2, 2020.

[22] K. Leyton-Brown, Y. Shoham. Essentials of Game Theory: A Concise, Multidisciplinary Introduction. Morgan & Claypool, 2008.

[23] C. Lund, L. Fortnow, H.J. Karloff, N. Nisan. Algebraic methods for interactive proof systems. Journal of the ACM 39(4) (1992) 859-868.

[24] B. McMahan, D. Ramage. Federated learning: collaborative machine learning without centralized training data. Google AI Blog, Apr. 6, 2017.

[25] D. Molnar. The SETI@Home problem. ACM Crossroads 7(1) (2000).

[26] B. Parno, C. Gentry, J. Howell, M. Raykova. Pinocchio: Nearly practical verifiable computation. Communications of the ACM 59(2) (2016) 103-112.

[27] A.R. Sadeghi, T. Schneider, M. Winandy. Token-based cloud computing: secure outsourcing of data and arbitrary computations with lower latency. In: Trust and Trustworthy Computing - Trust 2010, pp. 417-429. Springer, 2010.

[28] S. Setty, R. McPherson, A.J. Blumberg M. Walfish. Making argument systems for outsourced computation practical(sometimes). In: 19th Annual Network and Distributed System Security Symposium - NDSS'12. 2012.

[29] S. Smith, S. Weingart. Building a high-performance, programmable secure coprocessor. Computer Networks 31(8) (1999) 831-960.

[30] M. Sudan. Probabilistically checkable proofs. Communications of the ACM 52(3) (2009) 76-84.

[31] J. Thaler, M. Roberts. M. Mitzenmacher, H. Pfister. Verifiable computation with massively parallel interactive proofs. In: 4th USENIX conference on Hot Topics in Cloud Computing - HotCloud'12. USENIX, 2012.

[32] Trusted Computing Group. Trusted platform module main specification. 1.2, Revision 103 (2007).

[33] M. Walfish, A. J. Blumberg. Verifying computations without re-executing them. Communications of the ACM 58(2) (2015) 74-84.

[34] X. Yu, Z. Yan, A.V. Vasilakos. A survey of verifiable computation. Mobile Networks and Applications 22 (2017) 438-453.