Game-theoretic Analysis of An Incentivized Verifiable Computation System

Mahmudun Nabi[1], Sepideh Avizheh[1], Muni Venkateswarlu Kumaramangalam[1], and Reihaneh Safavi-Naini[1]

University of Calgary, AB, Canada
{mahmudun.nabi1,sepideh.avizheh1,munivenkateswarlu.ku,rei}@ucalgary.ca

Abstract. Outsourcing computation allows a weak client to outsource its computation to a powerful server and receive the result of the computation. Verifiable outsourcing enables clients to verify the computation result of untrusted servers. Permissionless distributed outsourcing systems provide an attractive marketplace for users to participate in the system as a problem-giver who needs solution to a problem, or problem-solver who is willing to sell its computational resources. Efficient verification of computation in these systems that use minimum trust assumptions on the computational nodes is a daunting task. In this paper we provide a game-theoretic analysis of an incentivized outsourcing computation system proposed by Harz and Boman [Harz *et. al*, 2018] (HB), at WTSC 2018 (FC Workshop), and show that the system is vulnerable to collusion and Sybil attacks that result in incorrect solutions to be accepted by the system. We also show that malicious computational node can succeed in polluting the blockchain. We also show modifications of the system that incentivizes honest behavior, and improve the system's correctness guarantee. We provide a high-level analysis of the modified system using our game theoretic approach, and show the effectiveness of the proposed modifications.

Keywords: Outsourcing computation · Verifiable distributed computation · Rational adversaries · Incentivized security.

1 Introduction

Outsourcing computation is an intriguing concept that enables clients to expand their computational power at will and when needed. The rise of cloud computing in recent years has been the driving force of outsourcing computation in a variety of settings. Outsourcing to cloud has made computationally expensive applications, such as complex analytics, available on small mobile devices, and has enabled clients to run resource intensive applications by purchasing the required computational resources from cloud providers. In both these cases the user interacts with a single cloud provider, and the service is paid for indirectly (e.g. through purchase of a device), or directly from the cloud. Outsourcing computation to a set of computational nodes has been used in applications such as SETI@Home (Search for Extra-Terrestrial Intelligence)[2] where volunteers download a free program that performs some computation (analyzes radio telescope data), and sends the results back to the outsourcer. The idea of computation as a commodity that can be bought and sold in a market-place has recently gained significant momentum. Systems such as Golem [1] aim to create such a marketplace to allow users to sell the extra cycles on their computers. Outsourcing however has many security and privacy challenges. The outsourcer (O) and the computational nodes (contractors) in general are not mutually trusting, and each party may attempt to subvert the system

for their own goals. The most basic security goal of the outsourcer is the correctness of the computation. Verifiable computation enables the outsourcer to verify the correctness of the result using efficient verification algorithms. Cryptographic schemes for verifiable computation [7,12,15] guarantee high level of security but are inflexible and incur heavy computational cost. A second approach is by assuming rational adversaries and designing mechanisms to incentivize honest behavior, and guarantee correctness of the results [4,9,11,14]. These systems however need complex management processes (e.g. managing rewards, punishments, auditing) to arrive at the correct result, and so may not be appealing to average user.

Two recently proposed systems, TrueBit [16,10], and a system proposed by Harz and Boman (HB) [8], attempt to remove this complexity by introducing an intermediate layer between the outsourcer and the contractors. The intermediate layer (TrueBit or HB system) receives request from the outsourcer, recruits contractors, and runs a public protocol with the guarantee that correct results will be delivered to the outsourcer, and that the contractors are paid for their services. Both systems are implemented in Ethereum platform, and use the trustable computation of Ethereum as the ultimate truth in arriving at the correct result. The goal is to achieve the same correctness guarantee of Ethereum while minimizing the actual request from computation Ethereum. This can be seen as performing computation "off-chain", and use Ethereum to verify the results.

The correctness guarantee in both systems relies on rewards, bounties, jackpots and similar mechanisms and assumes rational entities. TrueBit has a detailed white paper that discusses possible attacks, which are further discussed in blog posts, and other platforms. HB paper provides a high level argument about possible attacks and security of the system. None of the systems have formal game theoretic analysis in part due to the complexity of modelling the range of attack goals and colluding strategies in a highly distributed and non-permissioned systems.

Our work. Our motivation is to provide game theoretic framework for analysis of non-permissioned incentivized verifiable computation systems that use an intermediate layer. The variety of attack goals and possible collusions makes this analysis a daunting task. We start with HB, which compared to TrueBit, uses stronger trust assumptions.

In HB, there is an outsourcer, called *problem-giver*, who outsources a computation. Following [4,11], we assume that the outsourced computation is composed of a finite number of atomic operations. Also, we define the inner state of an algorithm as the concatenation of all the input/outputs of the atomic operations of the algorithm. Computational nodes (contractors) are *problem-solvers* and *verifiers*, the former providing a solution to the outsourced computation, and the latter verifying the solution. The HB system, is implemented as the *Arbiter,* an Ethereum smart contract that enforces the execution of the required steps of the verifiable computation process. In each computation, contractors who want to participate in the computation pay a deposit to the system and register. This deposit will act as their commitment, and together with the reward that is provided by the problem-giver and the distribution rules for the payment to the contractors, is used to incentivize correct computation while keeping the size of computation by Ethereum small.

Our game theoretic analysis of the system, although primarily focuses on the *rational* contractors (solver and verifiers) that have well-defined utilities, will also allow us to evaluate security against *irrational* behaviour where contractors are not driven by maximizing their utilities. This approach was first considered in [4,11] in which each contractor may behave rationally (*diligent* or *lazy*), or be *honest* or *malicious* irrespective of their utilities. Diligent contractors are those who perform the given computation correctly by running the original algorithm (the algorithm that is intended to use for producing the results), while lazy contractors employ some (less computationally expensive) "tricky" algorithm, called *q-algorithm*, for the computation, that produces a correct result with

probability q. In our analysis, we mainly consider the behaviors of rational contractors who strive always to maximize their utility. In Section 6 we also consider goal driven contractors (irrational and malicious) who are not bounded by a utility function. The goal of these contractors is to corrupt the blockchain.

For rational behaviors, we first derive a payoff matrix for the HB system when each contractor behaves independently (chooses to be diligent or lazy on its own), assuming the same q-algorithm is used by all lazy contractors. Table 2 shows the payoff of a solver & verifier, when the set of other verifiers consist of all lazy, all diligent, or a combination of the two. The analysis shows that, depending on the system parameters, for an independent contractor (solver/verfier) being diligent results in the highest utility. To analyze the system, we consider all possible scenario in the system, for instance, one contractor (solver/verifier) being diligent or lazy against the rest of the system that is lazy and, show that being diligent is better off for any independent contractor (see Sec. 4.1).

We then consider collusions where the colluding contractors agree to follow a single behavior (diligent or lazy), and share the reward equally. We assume colluders use the same algorithm when they behave lazily and use a tricky algorithm, or when they are diligent and use the correct algorithm. However, in both the cases they divide the computation into equal parts and each perform only a fraction of the total computation. We analyze the system's payoff matrix when the set of N contractors that are chosen by the Arbiter, consists of a colluding set, and two sets of contractors that are working lazily and diligently, but without any coordination (independently choosing their behaviour). Interestingly our analysis shows that, being part of a collusion maximizes a contractor's utility. We also analyze the system for Sybil attacks where a single contractor creates multiple identities to participate in the system (see Section 4.2). Our analysis shows that existing incentive structure of HB system may not guarantee correct result. (It is easy to see if the solver and all the verifiers collude and use a lazy algorithm, an incorrect solution may be accepted, as none of the verifiers will challenge the proposed solution.)

In Section 5 we propose modifications to the HB system with the goal of improving the correctness guarantee. These consists of modifications that are necessary to ensure the incentives for diligent behavior to guarantee correct result, and make the system robust against the lazy behavior in both collusion and Sybil attacks. In HB, if all the contracted contractors (independently or colluding) return same incorrect results, Arbiter accepts this as a correct result. Modifying the HB system, based on the proposed changes, we show that the modified system alleviates such vulnerabilities and, guarantee high correctness in results (see Section 5.1). We also note that the modifications proposed only helps the HB system to improve its correctness guarantee, but can not completely prevent the collusion and Sybil attacks on the system. For instance, the modifications prevent the colluding contractors being lazy and harm the system (e.g., polluting blockchain sending same incorrect results) but can not stop contractors from colluding (see Section 5.4).

We also propose modifications that would intuitively make the system more socially fair. Since the HB assumes problem-givers are rational, for a computation task, we try to lessen the burden of cost incurred to problem-givers. We suggest HB to utilize the payments that harvested from lazy contractors as penalty to encourage diligent behavior in rational contractors and for social fair. Fixing the HB system from collusion and Sybil attacks for any behaviors (diligent and lazy) is however our future work.

Paper Organization: Next section provides an overview of the HB system. Section 3 presents the game setup and model used for analyzing HB system. In Section 4, detailed analysis of the two attacks (collusion and Sybil) using different attack strategies based on the game setup and parameter setting is discussed. We present our proposed modifications to HB system and its effectiveness in

Section 5. Section 6 evaluates the performance of the modified HB system. Section 7 provides a detailed overview on the existing incentivized verifiable computation systems. Section 8 presents concluding remarks.

2 HB system

Ethereum is a permissionless blockchain that is known as *consensus computer*, meaning that a computation that is sent to Ethereum will be executed by all agents and as long as more than 50% of agents are honest, the computation result will be correct. To prevent attacks such as denial of service however, Ethereum introduces a gas limit to put a bound on the complexity of the requested computation, and so larger computations must be performed off-chain. HB [8] aims to extend correctness guarantee of Ethereum to off-chain computations.

Table 1: HB incentive structure for different behaviors of a solver and verifier

Verifier V	**Solver** S	
	Incorrect	*Correct*
Challenge	S: receives nothing, loses D_s V: receives $v_{fee} + s_{fee}$ + fee shares of all accepting verifiers (accepting verifiers receives nothing, loses D_v)	S: receives s_{fee} + fee shares of all challenging verifiers V: receives nothing, loses D_v (accepting verifiers receives v_{fee})
Accept	S: receives s_{fee} V: receives v_{fee}	S receives s_{fee} V: receives v_{fee}

The entities of the HB system are: *Problem-giver, Problem-solver, Verifier, Arbiter* and the *Judge*. Problem-givers are Ethereum users who outsource their computations for rewards. The solver is a computational contractor who provides computation power to solve computation problems in exchange for receiving a reward, s_{fee}. Verifiers are computational contractors who are contracted to provide computation power to correctly verify the solver's solutions for a reward, v_{fee}, by redoing the computation. Both these types of contractors must pay a deposit to participate in the computation. Arbiter is an Ethereum smart contract that enforces the execution of the required steps of the verifiable computation process. Arbiter acts as an *intermediary* between the problem-giver and the other entities. For each computation request that it receives from problem-giver, Arbiter randomly selects N contractors from the pool of registered contractors, and randomly chooses one as the problem-solver and the remaining $(N-1)$ as verifiers $\mathcal{V} = \{v_1, \ldots, v_{N-1}\}$. The solver publishes a solution together with a hashed trace of computation in the form of a Merkle tree to allow efficient checking of the results. The assumption is that the computational steps run by different contractors will produce the same values and so the same hashes. Verifiers also perform the computation and construct their own local hash tree, and publish the solution. The solution that is provided by the solver will be challenged by the verifiers who find inconsistencies between solver's solution and their own. Arbiter compares the provided solutions and initiate a dispute resolution if the solutions do not match, in which case a step-by-step comparison of the hash tree of the solver and a verifier is performed, and inconsistencies will be resolved by sending a random computational step to the Ethereum (Judge). Judge resolves the dispute and find whether the solver's computation is correct or not. If it is correct, the process continues as long as a verifier challenges the solver's solution. After all challenges are resolved, the problem-giver receives the results of the computation. If the solver is found incorrect and the verifiers solutions are not the same, the computation process is terminated. HB assumes that *Arbiter, judges and problem-givers are trusted and correctly follow*

the protocol. Table 1 shows the distribution of rewards in HB for different behaviors of a solver and verifier.

3 Game-theoretic analysis of HB

In HB, the problem-giver outsources their computation to a subset of contractors who have been randomly selected by Arbiter from the set of all registered contractors. The registration fee of the contractors will be returned if they unsubscribe (i.e., exit) from an Arbiter without violating the verifiable computation algorithm. The deposits of the selected contractors will be used as part of rewards and fines that will be applied to the contractors, in addition to the fee which will be paid by the problem-giver to the contractors whose work have not been challenged. We assume a contractor will participate in a computation when the received reward is higher than the cost of performing the task. We use the same behaviour for the contractors as those in [4,11]. A contractor, solver or verifier, may be rational and behave diligently or lazily, or be honest or malicious and not bound by utitlity. Rational contractors perform the computation honestly as long as the utility of performing the task correctly is greater than doing it otherwise. A *diligent* rational contractor performs the task honestly, and returns the correct result. A *lazy* contractor *uses a tricky algorithm* that can return the correct result with probability q (see Definition 1).

Definition 1. (q-algorithm). *An algorithm, composed of a finite number of atomic operations, is called a* q-algorithm *if it generates the correct solution with probability* q*, and such that* $0 \leq cost(q) < cost(1)$*. Here,* $cost(q)$ *denotes the cost of employing a* q-algorithm *and* $cost(1)$ *is the cost of the honest computation.*

We assume that the chance of a guessed solution being correct is nearly zero. In HB a solution will be accompanied with the hash tree of the computation steps. Thus even if the q-algorithm can find the solution, the chance of generating the same inner state hashes as the original algorithm, will be *negligible* (i.e., $q \approx 0$). The inner state of an algorithm can be captures at different granularity and will include the computational module input and output. *We also assume all lazy nodes use the same (best) q-algorithm.* This assumption effectively increases the chance of lazy contractors to remain undetected.

Our analysis starts with rational contractors, and later extend it to malicious contractors (in Section 5.6) whose goal is to *pollute the blockchain by making the Arbiter accept incorrect solutions.*

3.1 Payoffs when contractors behave independently

We consider a multi-player game with one solver s and $N-1$ verifiers $\mathcal{V} = \{v_1, \ldots, v_{N-1}\}$, $N > 1$, and assume each player chooses its behaviour independently. Contractors can behave diligently or lazily. The fee for a solver's correct solution is s_{fee} and so a diligent solver will always receive this fee. The fee for a verifier whose result is accepted by the Arbiter is v_{fee}. If a lazy contractor is caught (with probability $1-q$), their reward will be distributed among contractors who have challenged it (i.e., whose solution have been accepted as correct). Note that if a single lazy contractor is caught, all of them will be caught, and so the reward will be received by the diligent contractors.

We first analyze the system *assuming contractors work independently.* Let $u_z^{x,y}(q)$ denotes the *utility function of a contractor of type x, whose behavior is of type y, when there are total z lazy contractors (among the selected contractors) for the given computation.* We use $x \in \{s, v, c\}$, where s denotes the solver, v denotes the verifier and c denotes the colluder, $y \in \{d, \ell\}$ where d and ℓ

denote the diligent and lazy behavior, respectively, and $z \in \{0, 1, \ldots, N\}$. For diligent contractors, $q = 1$, otherwise $q < 1$. Also, the utility of any contractor $u_z^{x,y}(q)$ is linear in the reward and losses. Table 2 shows the utility of players in the above setting.

If a contractor of type x behaves diligently, and challenges a lazy contractor's incorrect solution, their utility will be, $u_z^{x,d}(1) = r + b(1-q) - cost(1)$, where r denotes the reward of acceptable work given by the problem-giver, $r = s_{fee}$ if $x = s$ or $r = v_{fee}$ if $x = v$, and b will be the additional reward (reward share of the lazy contractor) that a diligent contractor receive by challenging the lazy contractor. A diligent contractor will receive additional reward every time it challenges an incorrect solution. A lazy contractor cannot win a challenge against a diligent contractor because Ethereum will be used as the ultimate truth. (A lazy contractor will not challenge another lazy contractor because they both have the same solution.) In HB, a solver can challenge the verifiers, and the verifiers can challenge the solver. But, a verifier cannot challenge the other verifier. If a contractor x is lazy its utility is, $u_z^{x,l}(q) = rq - f(1-q) - cost(q)$, where f is the fine that a lazy contractor, who is caught, pays for providing an incorrect solution. As long as there is one diligent contractor, the system's final result will be correct and all the lazy contractor will be caught. Table 2 gives the payoff matrix of the game. Using the Table 2, in Section 4 we analyze the HB system for collusion and Sybil attacks.

Table 2: The payoff table of a solver against all verifiers, when each contractor (s and v) choose their behaviour (diligent or lazy) independently, assuming all lazy contractors use the same q-algorithm.

Verifiers	**Solver**	
	Diligent	*Lazy*
k_2 *lazy* ($k_3 = N - 1 - k$ *diligent*)	$u_k^{s,d}(1) = s_{fee} + k_2 v_{fee}(1-q) - cost(1)$ $u_k^{v,d}(1) = v_{fee} - cost(1)$ $u_k^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$	$u_k^{s,l}(q) = s_{fee}q - D_s(1-q) - cost(q)$ $u_k^{v,d}(1) = v_{fee} + \frac{s_{fee}+k_2.v_{fee}}{k_3}(1-q) - cost(1)$ $u_k^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$
All diligent ($k_2 = 0$, $k_3 = N - 1$)	$u_0^{s,d}(1) = s_{fee} - cost(1)$ $u_0^{v,d}(1) = v_{fee} - cost(1)$	$u_0^{s,l}(q) = s_{fee}q - D_s(1-q) - cost(q)$ $u_0^{v,d}(1) = v_{fee} + \frac{s_{fee}}{k_3}(1-q) - cost(1)$
All lazy ($k_2 = N - 1$, $k_3 = 0$)	$u_{N-1}^{s,d}(1) = s_{fee} + k_2 v_{fee}(1-q) - cost(1)$ $u_{N-1}^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$	$u_{N-1}^{s,l}(q) = s_{fee} - cost(q)$ $u_{N-1}^{v,l}(q) = v_{fee} - cost(q)$

In Table 2, k_2 and k_3 represents the number of independent lazy and independent diligent verifiers, respectively. Here to observe that, row 2 and row 3 in the Table 2 can be obtained by substituting the respective k_2 and k_3 values in row 1. We presented them separately here to show the utilities of all the involved parties in all the possible scenarios. Therefore, using only row 1 of Table 2, in Section 4 we analyze the HB system for collusion and Sybil attacks.

4 Attacks on HB System

We consider two attacks: (i) *collusion attack* where multiple contractors who have been selected by the Arbiter, form a collusion and work together, and (ii) *Sybil attack* where a single contractor registers under multiple identities in the hope of being selected multiple times in the set chosen by the Arbiter, in both cases to increase their utility.

4.1 Collusion Attack in HB System

A set of contractors might collude to maximize their utility. Similar to [4,11], we assume that the outsourced task (and the q-algorithm) is composed of a finite number of atomic operations. The colluders divide the computation task among themselves, each performing a fraction of the computation and sharing the results to construct a common output. They also share the payment from the system.

Let k_1 be the collusion group size where $1 < k_1 \leq N$. When $k_1 \neq N$, we also assume that the system may have two other sets of contractors, say k_2, $0 \leq k_2 < N$ who are independently lazy and $k_3 = N - (k_1 + k_2)$, $0 \leq k_2 < N$, who are independently diligent. Here to mention that, for a computation task, the values of k_2 and k_3 are unknown to the collusion group k_1. We consider the following two cases to analyze the collusion attack.

The solver is in the collusion. The colluder set consists of the solver and a set of verifiers. The remaining contractors may be independently lazy or diligent. The k_1 colluders can either act diligently (i.e. share the computation of the correct algorithm) or lazily (i.e., by sharing the computation cost of the same q-algorithm). Let δ_c be an indicator value, $\delta_c \in \{0, 1\}$ defined as, $\delta_c = 1$ if the solution of lazy participant is challenged, and $\delta_c = 0$, if it is not. Table 3 shows the utilities in the game between the colluding contractors (solver and verifiers) and the remaining independent players.

Table 3: Utility table for solver-verifier collusion game.

Verifiers	**Colluders**	
	Diligent	*Lazy*
k_2 *lazy verifiers*	$u^{c,d}_{k_2}(1) = r + \frac{k_2.v_{fee}}{k_1}(1-q) - \frac{cost(1)}{k_1}$ $u^{v,l}_{k_2}(q) = rq - D_v(1-q) - cost(q)$	$u^{c,l}_{k_1+k_2}(q) = rq + r(1-q)(1-\delta_c) - D(1-q)\delta_c - \frac{cost(q)}{k_1}$ $u^{v,l}_{k_1+k_2}(q) = rq + r(1-q)(1-\delta_c) - D(1-q)\delta_c - cost(q)$
k_3 *diligent verifiers*	$u^{v,d}_{k_2}(1) = r - cost(1)$	$u^{v,d}_{k_1+k_2}(1) = r + \frac{(k_1+k_2).v_{fee}}{k_3}(1-q) - cost(1)$

We use the table to evaluate *stability of a collusion*. That is, a member of the colluding group will have a higher utility to stay in the collusion. Note that the colluding group also have two behaviour, diligent and lazy. If the colluders behave diligently, the collusion will be stable if the colluders have higher utility than being independent players. That is it should be the case that:

- *Case 1:* $u^{c,d}_{k_2}(1) > u^{v,l}_{k_2}(q)$ (when k_2 verifiers are independently lazy), and
- *Case 2:* $u^{c,d}_{k_2}(1) > u^{v,d}_{k_2}(1)$ (when k_3 verifiers are independently diligent).

For a given set of system parameters, we will have $u^{v,l}_{k_2}(q) > u^{v,d}_{k_2}(1)$, or the reverse. So, we choose $u^{c,d}_{k_2}(1) > max\{u^{v,l}_{k_2}(q), u^{v,d}_{k_2}(1)\}$. This gives the following results (using Table 3):

- if $u^{c,d}_{k_2}(1) > u^{v,l}_{k_2}(q)$, we get $k_1 > \frac{cost(1)}{(r+D_v)(1-q)+cost(q)}$, and
- if $u^{c,d}_{k_2}(1) > u^{v,d}_{k_2}(1)$, we get $k_1 > 1$.

This implies that for a given set of system parameters as long as the collusion size is higher than the corresponding k_1, the diligent colluding strategy will have higher pay-off than working independently (diligent or lazy), and so the collusion will be stable.

Similarly, if the colluders behave lazily, to have higher utility than independent players, the following should be true:

- *Case 3:* $u^{c,l}_{k_1+k_2}(q) > u^{v,l}_{k_1+k_2}(q)$ (when k_2 verifiers are independently lazy), and
- *Case 4:* $u^{c,l}_{k_1+k_2}(q) > u^{v,d}_{k_1+k_2}(1)$ (when k_3 verifiers are independently diligent).

Using a similar argument as above, a collusion remains stable if the following is satisfied: $u^{c,l}_{k_1+k_2}(q) > max\{u^{v,l}_{k_1+k_2}(q), u^{v,d}_{k_1+k_2}(1)\}$. Using the utilities from table 3 we get following results:

- if $u^{c,l}_{k_1+k_2}(q) > u^{v,l}_{k_1+k_2}(q)$, then $k_1 > 1$, and
- if $u^{c,l}_{k_1+k_2}(q) > u^{v,d}_{k_1+k_2}(1)$, $k_1 > \frac{cost(q)}{cost(1)+(1-q)[(N-1)V_{fee}-r-D]}$.

Note that the bounds only depend on the system parameters and so can be computed by a contractor. In other words, a contractor can decide to be independent, or stay in the collusion based on the size of the colluding set that they have been able to form. The above analysis leads to the following theorem.

Theorem 1. *(a) For a given set of system parameters, there are values for the colluding set size k_1, for which being diligent, or being lazy has higher utility than being an independent player.*
(b) For values of k_1 for which both collusion behaviours (diligent and lazy) allow stable collusion, one can find the best strategy (the strategy with the highest utility) of colluders (diligent or lazy) that maximizes their utility.

Note that based on the system parameters of HB, solver-verifier collusion is possible as long as for any strategy (diligent or lazy), the collusion size is higher than the corresponding k_1.

In HB, if colluders act diligently when $k_1 < N$, we get $k_1 > \frac{cost(1)-cost(q)}{(r+D)(1-q)}$. Since $cost(q) < cost(1) \leq r$ and $q \approx 0$ (e.g. by using inner state hash [4]), $r + D > cost(1) - cost(q)$. So, $\frac{cost(1)-cost(q)}{(r+D)(1-q)} < max\{1, \frac{cost(1)}{(r+D)(1-q)+cost(q)}\}$. So, in this case, the best strategy is to acts diligently. However, when $k_1 = N$, since $\delta_c = 0$ and $q = 0$ (in HB), from Table 3 we get $r - \frac{cost(q)}{k_1} > r - \frac{cost(1)}{k_1}$. So, in this case being lazy is the best strategy.

Collusion of verifiers only. Assume that k_1 verifiers are colluding and doing the computation jointly. The colluders may act diligently, or lazily. The remaining players, consist of k_2 lazy, and $k_3 = N - (k_1 + k_2)$ diligent, players. As the solver is an independent player, it may be in the diligent, or in the lazy, group. Table 4 shows the payoff of a colluding verifier against the payoff of an independent verifier in each case.

Using an argument similar to the above case (solver is in the collusion), we will have: $u^{(c,d)(s,d)}_{k_2}(1) > max(u^{(v,l)(s,d)}_{k_2}(q), u^{(v,d)(s,d)}_{k_2}(1))$ and $u^{(c,d)(s,l)}_{k_2}(1) > max(u^{(v,l)(s,l)}_{k_2}(q), u^{(v,d)(s,l)}_{k_2}(1))$.

For colluders behaving lazily a collusion remains stable if the followings are satisfied: $u^{(c,l)(s,d)}_{k_1+k_2}(1) > max(u^{(v,l)(s,d)}_{k_1+k_2}(q), u^{(v,d)(s,d)}_{k_1+k_2}(1))$ and $u^{(c,l)(s,l)}_{k_1+k_2}(1) > max(u^{(v,l)(s,l)}_{k_1+k_2}(q), u^{(v,d)(s,l)}_{k_1+k_2}(1))$.

These gives lower bounds on collusion sizes for which colluders will have higher utility with their corresponding strategies, compared to playing independently.

The above analysis leads to a theorem similar to Theorem 1. The above two theorems show that HB cannot protect against colluding attacks.

Table 4: Pay-off matrix for verifiers collusion game

Remaining players		**Colluding verifiers (k_1)**	
Solver	**Verifiers**	*Diligent*	*Lazy*
Diligent	k_2 *Lazy* $k_3 - 1$ *Diligent*	$u_{k_2}^{(c,d)(s,d)}(1) = v_{fee} - \frac{cost(1)}{k_1}$ $u_{k_2}^{(v,l)(s,d)}(q) = v_{fee}q - D_v(1-q) - cost(q)$ $u_{k_2}^{(v,d)(s,d)}(1) = v_{fee} - cost(1)$	$u_{k_1+k_2}^{(c,l)(s,d)}(q) = v_{fee}q - D_v(1-q) - \frac{cost(q)}{k_1}$ $u_{k_1+k_2}^{(v,l)(s,d)}(q) = v_{fee}q - D_v(1-q) - cost(q)$ $u_{k_1+k_2}^{(v,d)(s,d)}(1) = v_{fee} - cost(1)$
Lazy	$k_2 - 1$ *Lazy* k_3 *Diligent*	$u_{k_2}^{(c,d)(s,l)}(1) = v_{fee} + \frac{s_{fee}+(k_2-1)v_{fee}}{k_1+k_3}(1-q) - \frac{cost(1)}{k_1}$ $u_{k_2}^{(v,l)(s,l)}(q) = v_{fee}q - D_v(1-q) - cost(q)$ $u_{k_2}^{(v,d)(s,l)}(1) = v_{fee} + \frac{s_{fee}+(k_2-1)v_{fee}}{k_1+k_3}(1-q) - cost(1)$	$u_{k_1+k_2}^{(c,l)(s,l)}(q) = v_{fee}q + v_{fee}(1-q)(1-\delta_c) - D_v(1-q)\delta_c - \frac{cost(q)}{k_1}$ $u_{k_1+k_2}^{(v,l)(s,l)}(q) = v_{fee}q + v_{fee}(1-q)(1-\delta_c) - D_v(1-q)\delta_c - cost(q)$ $u_{k_1+k_2}^{(v,l)(s,l)}(1) = v_{fee} + \frac{s_{fee}+(k_1+k_2-1)v_{fee}}{k_3}(1-q) - cost(1)$

4.2 Sybil Attack in HB System

In a Sybil attack in HB, a single party creates multiple identities with the goal of increasing their chance of being selected by the arbiter. Consider an attacker who creates K Sybil identities, and it is hoping that k_1 of them are selected. Additionally, consider that n identities exist in the system and N of them are selected for a given task. Then K can be estimated as follow: If n and N are large, then the proportion of identities that belongs to the attacker in the system, $\frac{K}{n}$, is equal to the proportion of identities chosen for the given task, $\frac{k_1}{N}$. In another way, $\frac{k_1}{N} = \frac{K}{n}$. So, attacker should create $K = \frac{k_1}{N}n$ identities in order to gets k_1 of them accepted for a given task. For example if $n = 10$ and $N = 5$, and attacker wants to have $k_1 = 2$, then it should register $\frac{2}{5} \times 10 = 4$ identities in the system. Sybil attack can happen when all the selected identities of the Sybil node are verifiers, or it also includes the solver.

We consider two cases: $k_1 < N$, or $k_1 = N$. If $k_1 < N$, we consider k_2 independently lazy verifiers and $k_3 (= N - k_1 - k_2)$ independently diligent verifiers. Otherwise, if $k_1 = N$, $k_2 = k_3 = 0$. Sybil attack can be seen as a realization of collusion attack with the difference that the Sybil attacker must provides deposit for all k_1 identities and will receive all the rewards, and so instead of considering the utility of a member of the collusion, we will consider the utility of the whole colluding. Using an analysis similar to previous section we will have the following analysis.

Sybil identities includes the solver. Let $U(a, b)$ represents the total utility of the Sybil attacker for strategy $a \in \{d, l\}$, when $b \in \{0, 1, \ldots, N\}$ is the number of identities that are selected by the Arbiter.

If the solver is in k_1, then-

- when $k_1 < N$ the total utility for the *diligent* strategy is: $U(d, k_1) = k_1.[r + \frac{k_2.v_{fee}}{k_1}(1-q)] - cost(1)$.
- when $k_1 = N$ the total utility for the *lazy* strategy is: $U(l, N) = N.r - cost(q)$

Here, $r = s_{fee}$ for solver (or v_{fee} for verifier), and D is the registration cost (i.e., deposit) by the agent. As these utilities are greater than one diligent identity, considering the assumptions in Theorem 1 hold, Sybil attack of solver-verifiers is possible.

We note that the probability of $k_1 = N$ will be very small as for large values of N, the registration cost of sufficient number of registered identities would become formidable.

Sybil attack of verifiers. Analysis of collusion attack showed that when k_1 satisfies certain conditions, being diligent is the better strategy. For this strategy, since the Sybil attacker does not know how the solver would act, we consider two cases:

- If solver is diligent, the total expected utility for the attacker is: $U(d, k_1) = k_1(v_{fee}) - cost(1)$.
- If solver is lazy, the total expected utility for the attacker is: $U(l, k_1) = k_1[v_{fee} + \frac{s_{fee}+(k_2-1)v_{fee}}{k_1+k_3}(1-q)] - cost(q)$.

These utilities are higher than the utility of a single diligent identity in the system. Using similar analysis we can show that Sybil attack of verifiers is possible.

4.3 Shortcoming of HB

The above analysis of HB showed that it is vulnerable to both collusion and Sybil attacks. Below are additional shortcoming of the system.

- In HB, when a contractor is lazy and get caught, the diligent contractors who catches the lazy contractor gets the reward share (additional reward) of the lazy contractor which has been committed by the problem-giver prior to the computation. We suggest that the additional reward should be harvested from the fine that the lazy contractor pays rather than charging the problem-giver.
- In HB, the deposit (D) amount is set by the Arbiter and the contractors pay this amount before the problem-giver requests for a computation to the arbiter. This deposit works as a commitment to the system and will not be returned to the contractor if it gets caught by cheating. However, as the task difficulty is unknown to the arbiter, the deposit value set might be so little to ensure the honest computation. The deposit value should be fixed based on the task difficulty.

5 Modified HB System

The analysis in of HB showed that it is vulnerable to both collusion and Sybil attacks. In this section, to improve the correctness guarantee of HB system, for the attacks presented in Section 4, we propose the following modifications to the original HB system.

1. Introduce a random auditing mechanism to alleviate the vulnerabilities of collusion and Sybil attacks.
2. Suggest to determine the deposit (D) amount to be paid by the contractor to participate in a given computation based on the complexity of the computation.
3. To encourage diligent behavior in contractors, we propose to penalize lazy contractors (when caught) and use their deposits (instead of using the rewards committed by problem-giver) to reward the honest contractors (more incentives for being diligent). This also helps in reducing computation cost for rational problem-givers and encourages them to use the system more often.

Finally, in Section 5.4, we analyze the effectiveness of these modifications on the HB system.

5.1 Random audit

We propose the following extension to HB system to alleviate the Sybil attack mentioned in Section 4.2. The idea is, when contractors (solver and verifiers) return the results and are *all same*, Arbiter randomly audits (i.e., probabilistic auditing) the results using the Judge to verify whether the contractors used the algorithm that they intend to or not. The random auditing involves of two

parties: the *Arbiter*, who triggers this auditing by sending an intermediary result of a step stored in a Merkle tree and input, if any, to Judge for the given computation. The *Judges*, who are the set of miners in Ethereum, returns a boolean value to Arbiter: If the judge returns *true*, the solver and the verifiers get their reward. Otherwise, the Arbiter penalize the solver and the verifiers by taking away their deposits and reassign the computation to another group of contractors.

5.2 Modification in operational flow of HB System

In HB, each contractor commits a deposit to Arbiter for participating in the system that is independent of problem-giver's computation. It may so happen that the deposit amount is much smaller than the reward (that is calculated based on the complexity of the computation) for computing results. Because of this, the system is vulnerable to attacks such as Sybil. For example, within its financial limits, a rational contractor (Sybil attacker) may create multiple identities to increase its chances of being selected for the computation expecting huge rewards. This causes questioning the correctness guarantee of the system. To avoid such scenarios, we propose that the Arbiter should determine the deposit amount independently for each computation requested based on computation complexity. In more detail, the deposit amount should be fixed after receiving the computation request from the problem-giver. Then, the interested contractors commit this fixed deposit and register to the Arbiter for the computation.

5.3 Modified incentive structure

In HB, when a diligent contractor challenges an incorrect solution published by a lazy contractor, the diligent contractor receive an additional reward (bonus) for helping the system to catch the cheating contractor and the lazy contractor loses his deposit. The bonus given to the diligent contractor is actually the reward that is originally committed by the problem-giver (when the computation is requested) for the contractor who has been caught for his lazy behavior. Since HB assumes that the problem-givers are rational, they always try to optimize (minimize) their cost (payment) for their requested computation. Therefore, to encourage the problem-givers to use the system often and for social good, we propose to use the deposits paid by the cheating contractors to reward diligent contractors instead of using problem-givers payment.

Based on the proposed idea of using lazy contractors deposits to reward diligent contractors and the probabilistic auditing mechanism presented in Section 5.1, we modify the HB incentive structure (given in Table 1) and present a modified incentive structure in Table 5. To be consistent, in Table 5 we use the notations that are defined already (see Section 2) and, with an additional one, β to denote the probability of an Arbiter performing random auditing.

In Table 5, when a solver's incorrect solution is challenged by a verifier, S does not receive reward (s_{fee}) and losses his deposit D_s and (each) challenger (verifier) receives its reward (v_{fee}) with an additional reward, a share of D_s, as a bonus (for example, if there are N_c challengers, then each challenger will get $v_{fee} + D_s/N_c$). If a verifier challenges correct solution of a solver, it losses D_v and does not receive any reward and, S gets his reward and deposits

Table 5: New incentive structure of a solver and verifier with different behaviors in the modified HB system

Verifier V	**Solver** S	
	Incorrect	*Correct*
Challenge	S losses D_s	S receives $s_{fee} + D_v$
	V receives $v_{fee} + D_s$	V losses D_v
Accept	S receives $s_{fee}(1-\beta)$	S receives s_{fee}
	S losses $D_s\beta$	
	V receives $v_{fee}(1-\beta)$	V receives v_{fee}
	V losses $D_v\beta$	

of all the challengers as a bonus. When the
results are same, using the Judge, Arbiter
verifies the results randomly with probability β. For example, in the Table 5, when the solver's solution is incorrect and the verifier accepts it (i.e., the results are same), both S and V either loss their deposits with probability β or receive their rewards with $(1-\beta)$ probability.

5.4 Analysis of the modified HB system

Let $\beta \in [0,1]$ be the probability of auditing by the arbiter when the returned results are same. We assume that arbiter sets the deposit D based on the task difficulty. Although each computational service (solver or verifier) provides the same amount of deposit for participation, we use D_s (solver's deposit) and D_v (verifier's deposit) same as before (in Section 3.1) to distinguish them, where $D = D_s = D_v$. However, in this modified model the usage of the deposit is different than the original HB system. That is instead of burning the deposits of the cheating parties they will be distributed among the honest parties as a bonus. Based on these assumptions and changes, the game pay-off Table 2 of original HB are updated and new utilities are shown in Table 6.

Table 6: Pay-off matrix of the modified HB

Verifiers	**Solver**	
	Diligent	*Lazy*
All diligent	$u_0^{s,d}(1) = s_{fee} - cost(1)$ $u_0^{v,d}(1) = v_{fee} - cost(1)$	$u_0^{s,l}(q) = s_{fee}q - D_s(1-q) - cost(q)$ $u_0^{v,d}(1) = v_{fee} + \frac{D_s}{N-1}(1-q) - cost(1)$
k lazy $N-1-k$ *diligent*	$u_k^{s,d}(1) = s_{fee} + \frac{kD_v}{N-k}(1-q) - cost(1)$ $u_k^{v,d}(1) = v_{fee} + \frac{kD_v}{N-k}(1-q) - cost(1)$ $u_k^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$	$u_k^{s,l}(q) = s_{fee}q - D_s(1-q) - cost(q)$ $u_k^{v,d}(1) = v_{fee} + \frac{D_s+kD_v}{N-1-k}(1-q) - cost(1)$ $u_k^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$
All lazy	$u_{N-1}^{s,d}(1) = s_{fee} + (N-1)D_v(1-q) - cost(1)$ $u_{N-1}^{v,l}(q) = v_{fee}q - D_v(1-q) - cost(q)$	$u_{N-1}^{s,l}(q) = s_{fee}q + s_{fee}(1-q)(1-\beta)$ $-D_s(1-q)\beta - cost(q)$ $u_{N-1}^{v,l}(q) = v_{fee}q + v_{fee}(1-q)(1-\beta)$ $-D_v(1-q)\beta - cost(q)$

Using the above utilities, and the analysis approach in Section 4, we show that with the modified utilities presented in Table 6, collusion attack is still possible and the possibility of Sybil attack is reduced.

5.5 Sybil Attack

The main mechanism that we are using in this modified system to alleviate the Sybil attack is the probabilistic auditing, with probability β, by the judges. β is always greater than zero and so even if all the identities belong to the adversary and all output the same solution, still they will get caught with probability β. However, in the original HB system, if all the identities belong to the adversary, it will not be challenged and so $\beta = 0$. As a result, through our fixes, Sybil attack is prevented while it is possible in the original HB system.

5.6 Malicious contractors

Malicious contractors aim at polluting the blockchain by making the arbiter to accept the incorrect solution. They will collude and share their budgets to maximize the harm to the system. To have

enough, non-negative, balance to participate in the tasks they will act honestly time to time. In this section we find bounds on the expected false rate. The following Theorem shows that malicious contractors need to wait until all are selected as the computation services required for the task and then collude.

Theorem 2. *The best strategy for malicious contractors is to all collude when they are selected for a given task.*

Proof. As we have previously shown in Table 6, whenever all contractors reveal the same incorrect solution and when there is no auditing by judges, $\beta = 0$, arbiter accepts the incorrect solution and the obtained utility for each contractor is maximized:

$$u_z^{x,m}(0) = r(1-\beta) = r \tag{1}$$

Theorem 3 shows the bound for the expected false rate which is the number of incorrect solutions broadcasted to blockchain if malicious contractors follow strategy given in Theorem 2.

Theorem 3. *The expected false rate in the system is $p^N(1-\beta)$, where N is the number of computation services, p, is the prior probability that a computation service is malicious in the system, and β is probability of auditing by the judges.*

Proof. If all the N chosen computation services are malicious they can collude and reveal an incorrect solution. The probability of this case is p^N. However, they may be challenged by judges with probability β if all reveal the same solution. The probability that they do not challenged is $1-\beta$, therefore attackers succeed with probability $p^N(1-\beta)$.

According to Theorem 2, we gain a reduction on the expected false rate proportional to $1-\beta$ compared to HB. In the next section we compare our result with HB assuming some values for p, N and β.

6 Evaluation and comparison

Table 7 shows the expected false solutions broadcasted to blockchain in our scheme versus HB. The prior probability of being lazy is selected to be 0.3, 0.5 and 0.7 to facilitate the comparison to HB. According to HB, if solver reveals a wrong solution and verifiers accept it then false solution goes to blockchain; if the probability that a computation service be lazy and output a wrong solution is p, and the number of verifiers is n, then the probability of false solution is p^{n+1} ($N = n+1$). In our scheme, however, the expected false additionally depends on the probability of auditing (see Theorem 2); if a task is audited by judges with probability β then the probability of false solution will be $p^{n+1}(1-\beta)$. For comparison, we have considered β to be 0.1, 0.5, and 0.9. According to table, the probabilistic auditing reduces the expected false percentage proportional to $1-\beta$. We expect to see more reductions in practice as the auditing time is unknown to computation services.

7 Related work

In blockchain, transactions must be validated and processed at every node in the network. So, any blockchain system experiences limitations in scalability. Outsourcing computation techniques have been proposed in various papers addressing the scalability and security challenges of a blockchain [16,10,8]. Trust models (based on incentive mechanism) [16,10,8] are proposed to off-chain the

Table 7: Comparison between expected false in our scheme and HB.

Prior p	Verifiers n	Expected false [%]			Expected false HB [%]
		$\beta = 0.1$	$\beta = 0.5$	$\beta = 0.9$	
0.3	1	8.1	4.5	0.9	9.0
	2	2.43	1.35	0.27	2.7
	3	0.729	0.405	0.081	0.81
	4	0.2187	0.1215	0.0243	0.243
	5	0.06561	0.03645	0.00729	0.0729
	6	0.019683	0.010935	0.002187	0.02187
0.5	1	22.5	12.5	2.5	25.0
	2	11.25	6.25	1.25	12.5
	3	5.625	3.125	0.625	6.25
	4	2.8125	1.5625	0.3125	3.125
	5	1.40625	0.78125	0.15625	1.5625
	6	0.703125	0.390625	0.078125	0.78125
0.7	1	44.1	24.5	4.9	49.0
	2	30.87	17.15	3.43	34.3
	3	21.609	12.005	2.401	24.01
	4	15.1263	8.4035	1.6807	16.807
	5	10.58841	5.88245	1.17649	11.7649
	6	7.411887	4.117715	0.823543	8.23543

computations to third parties to inherit transparency and particular trust for implementing smart contracts in permission-less blockchains. ZoKrates [6], shares the same spirit of off-chaining computations to third parties by supplying a non-interactive zero-knowledge proof with the results. Moreover, several works have been proposed to incorporate multiparty computation onto blockchains [13] [3].

In addition, Dong *et. al* [5] provide a game theoretical analysis for verifiable outsourced computation in cloud computing that involves blockchain technology. The authors assume only two rational clouds who can collude and later one can secretly deviate from the collusive agreement to maximize their own profit. The authors design smart contracts to sabotage collusion. In case of our analysis, we assume that once collusion is formed, colluders do not deviate from their agreement and the goal is to prevent collusion from happening. Moreover, we have considered both rational and malicious adversaries for our analysis. Finally, their system needs a trusted third party to resolve conflicts where the HB system resolves the dispute through the distributed consensus algorithms (i.e., by using Ethereum as Judge) and removes the need of a trusted third party.

In [14] the authors researched the outsourced verification computation from an economics aspect. They provided a general approach based on game theory for optimal contract design for outsourcing computation. They considered the cheating or lazy behavior of the service providers and analyzed it with one and multiple server settings. In the follow up work [9], they identified the optimal settings for the multi-server case when collusion is allowed.

8 Conclusion

Security of incentivized verifiable computation system is an intriguing challenge because of the range of possible attacks and possible collusions. Our goal was to examine the security claims of such an incentivised outsourcing system proposed by Harz and Boman [Harz *et. al*, 2018] (HB), at FC WTSC 2018. We analyse the system, considering rational contractors, using a game-theoretic approach and discuss its correctness guarantee considering two attacks, collusion and Sybil. Our "simple" collusion considers colluders work together to maximize their utility, agree to follow a single behavior (diligent or lazy), and share computation and the rewards equally. Further, we analyze the system for Sybil

attack where a single contractor creates multiple identities and register as multiple contractors to maximize the chance of being selected by the Arbiter, maximizes its utility. Interestingly, our analysis shows that the incentive structure of HB can not protect the system against these attacks. We propose modifications for HB that helps the system to improve its correctness guarantee by motivating diligent behavior in contractors by incentivizing contractors with bonus rewards. We analyse the modified HB system and show that the modifications alleviates vulnerabilities of the attacks and, guarantee high correctness in results. We note that the modifications improves HB's correctness guarantee, but can not prevent the attacks. We also propose to utilize the deposits of lazy contractors to encourage diligent behavior in rational contractors for social fair. Fixing the HB system from collusion and Sybil attacks for any behaviors (diligent and lazy) is however our future work. Finally, we extend our analysis to malicious contractors who are goal driven and are not bounded by a utility function. The goal of these malicious contractors is corrupting the blockchain. We show that the HB system implementing the proposed modifications alleviates the expected false rate compared to the original HB system.

References

1. Golem project. url: https://golem.network/ (2016), accessed on Dec 2018
2. Seti@home. url: https://setiathome.berkeley.edu/ (2018), accessed on Dec 2018
3. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: Security and Privacy (SP), 2014 IEEE Symposium on. pp. 443–458. IEEE (2014)
4. Belenkiy, M., Chase, M., Erway, C.C., Jannotti, J., Küpçü, A., Lysyanskaya, A.: Incentivizing outsourced computation. In: Proceedings of the 3rd international workshop on Economics of networked systems. pp. 85–90. ACM (2008)
5. Dong, C., Wang, Y., Aldweesh, A., McCorry, P., van Moorsel, A.: Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 211–227. ACM (2017)
6. Eberhardt, J., Tai, S.: Zokrates-scalable privacy-preserving off-chain computations. In: Blockchain, 2018 IEEE International Conference on (2018)
7. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Annual Cryptology Conference. pp. 465–482. Springer (2010)
8. Harz, D., Boman, M.: The scalability of trustless trust. In: 2nd Workshop on Trusted Smart Contracts (WTSC'18), International Conference on Financial Cryptography and Data Security (2018)
9. Khouzani, M., Pham, V., Cid, C.: Incentive engineering for outsourced computation in the face of collusion. In: Proceedings of WEIS (2014)
10. Koch, J., Reitwiessner, C.: A predictable incentive mechanism for truebit. arXiv preprint arXiv:1806.11476 (2018)
11. Küpçü, A.: Incentivized outsourced computation resistant to malicious contractors. IEEE Transactions on Dependable and Secure Computing **14**(6), 633–649 (2017)
12. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: Theory of Cryptography Conference. pp. 422–439. Springer (2012)
13. Paul, S., Shrivastava, A.: Robust multiparty computation with faster verification time. In: Information Security and Privacy. pp. 114–131. Springer International Publishing, Cham (2018)
14. Pham, V., Khouzani, M., Cid, C.: Optimal contracts for outsourced computation. In: International Conference on Decision and Game Theory for Security. pp. 79–98. Springer (2014)
15. Setty, S.T., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: NDSS. vol. 1, p. 17 (2012)
16. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains. url: https://people. cs. uchicago. edu/teutsch/papers/truebit pdf (2017)