Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

CROWDSOURCING ATOP BLOCKCHAINS

by
Yuan Lu

Traditional crowdsourcing systems, such as Amazon's Mechanical Turk (MTurk), though once acquiring great economic successes, have to fully rely on third-party platforms to serve between the requesters and the workers for basic utilities. These third-parties have to be fully trusted to assist payments, resolve disputes, protect data privacy, manage user authentications, maintain service online, etc. Nevertheless, tremendous real-world incidents indicate how elusive it is to completely trust these platforms in reality, and the reduction of such over-reliance becomes desirable.

In contrast to the arguably vulnerable centralized approaches, a public blockchain is a distributed and transparent global "consensus computer" that is highly robust. The blockchain is usually managed and replicated by a large-scale peer-to-peer network collectively, thus being much more robust to be fully trusted for correctness and availability. It, therefore, becomes enticing to build novel crowdsourcing applications atop blockchains to reduce the over-trust on third-party platforms.

However, this new fascinating technology also brings about new challenges, which were never that severe in the conventional centralized setting. The most serious issue is that the blockchain is usually maintained in the public Internet environment with a broader attack surface open to anyone. This not only causes serious privacy and security issues, but also allows the adversaries to exploit the attack surface to hamper more basic utilities. Worse still, most existing blockchains support only light on-chain computations, and the "smart contract" executed atop the decentralized "consensus computer" must be simple, which incurs serious feasibility problems. In reality, the privacy/security issue and the feasibility problem even restrain each other and create serious tensions to hinder the broader adoption of blockchain.

The dissertation goes through the non-trivial challenges to realize secure yet still practical decentralization (for urgent crowdsourcing use-cases), and lay down the foundation for this line of research. In sum, it makes the next major contributions.

First, it identifies the needed security requirements in decentralized knowledge crowdsourcing (e.g., data privacy), and initiates the research of private decentralized crowdsourcing. In particular, the confidentiality of solicited data is indispensable to prevent free-riders from "pirating" the others' submissions, thus ensuring the quality of solicited knowledge. To this end, a generic private decentralized crowdsourcing framework is dedicatedly designed, analyzed, and implemented.

Furthermore, this dissertation leverages concretely efficient cryptographic design to reduce the cost of the above generic framework. It focuses on decentralizing the special use-case of Amazon MTurk, and conducts multiple specific-purpose optimizations to remove needless generality to squeeze performance. The implementation atop Ethereum demonstrates a handling cost even lower than MTurk.

In addition, it focuses on decentralized crowdsourcing of computing power for specific machine learning tasks. It lets a requester to place deposits in the blockchain to recruit some workers for a designated (randomized) programs. If and only if these workers contribute their resources to compute correctly, they would earn well-deserved payments. For these goals, a simple yet still useful incentive mechanism is developed atop the blockchain to deter rational workers from cheating.

Finally, the research initiates the first systematic study on crowdsourcing blockchains' full nodes to assist superlight clients (e.g., mobile phones and IoT devices) to "read" the blockchain's records. This dissertation presents a novel generic solution through the powerful lens of game-theoretic treatments, which solves the long-standing open problem of designing generic superlight clients for all blockchains.

CROWDSOURCING ATOP BLOCKCHAINS

by
Yuan Lu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2020

APPROVAL PAGE

CROWDSOURCING ATOP BLOCKCHAINS

Yuan Lu

Dr. Guiling Wang, Dissertation Advisor — Date
Professor of Computer Science, NJIT

Dr. Qiang Tang, Dissertation Co-Advisor — Date
Assistant Professor of Computer Science, NJIT

Dr. Baruch M. Schieber, Committee Member — Date
Professor and Chair of Computer Science, NJIT

Dr. Roman Vaculin, Committee Member — Date
Senior Manager and Principal Research Staff Member,
IBM Research, Yorktown Heights, NY

Dr. Jian Yang, Committee Member — Date
Professor of Management Science and Information Systems,
Rutgers Business School – Newark & New Brunswick, Rutgers University, NJ

BIOGRAPHICAL SKETCH

Author: Yuan Lu

Degree: Doctor of Philosophy

Date: August 2020

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, NJ, US, 2020
- Master of Engineering in Electrical Engineering,
  Nankai University, Tianjin, China, 2014
- Bachelor of Science in Information Science and Technology,
  Nankai University, Tianjin, China, 2011
- Bachelor of Business Administration,
  Tianjin University, Tianjin, China, 2011

Major: Computer Science

Presentations and Publications:

Yuan Lu, Qiang Tang, Guiling Wang, "Enhancing the Retailer Gift Card via Blockchain: Trusted Resale and More," in *Journal of Database Management*, 2020.

Yuan Lu, Qiang Tang, Guiling Wang, "Generic Superlight Client for Permissionless Blockchains," in *the 25th European Symposium on Research in Computer Security (ESORICS 2020)*, Virtual Event, UK, September 2020.

Yuan Lu, Qiang Tang, Guiling Wang, "Dragoon: Private Decentralized HITs Made Practical," in *the 40th IEEE International Conference on Distributed Computing Systems (ICDCS 2020)*, Singapore, November 2020.

Yuan Lu, Zhenliang Lu, Qiang Tang, Guiling Wang, "Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited," in *the 39th ACM Symposium on Principles of Distributed Computing (PODC 2020)*, Virtual Event, Italy, August 2020.

Yuan Lu, Qiang Tang, Guiling Wang, "ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain," in *the 38th IEEE International Conference on Distributed Computing Systems (ICDCS 2018)*, Vienna, Austria, July 2018.

Yuan Lu, Qiang Tang, Guiling Wang, "On Enabling Machine Learning Tasks atop Public Blockchains: a Crowdsourcing Approach," in *the 1st Workshop on Blockchain and Sharing Economy Applications (BlockSEA 2018) co-located with IEEE ICDM 2018*, Sentosa, Singapore, November 2018.

Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, Chase Wu, "FairThunder: Fair Peer-to-Peer Content Delivery atop Blockchain," *manuscript (ready to submit)*.

Yuan Lu, Qiang Tang, Guiling Wang, "Decentralized Crowdsourcing of Human Knowledge atop Open Blockchain," *manuscript (ready to submit)*.

Dedicated to my lovely family.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my dissertation advisor, Professor Guiling Wang. Since the first day when she offered me a great opportunity to join NJIT, she believed in me when nobody else has not, and gave me endless helps in the PhD pursuit. On the academic level, she delivered the fundamentals on how to conduct scientific research, and inspired me to focus on the interesting problems in crowdsourcing area. On the personal aspect, her passionate attitude motivated me to continue in the most difficult times. I am also deeply indebted to my dissertation co-advisor Professor Qiang Tang. He has been a tremendous mentor for me. He has taught me, both consciously and unconsciously, what valuable research is and how it to be done. I deeply appreciate his time, ideas and encourages that make my research productive. The enthusiasm that he has for scientific research was motivational for me and helped me go through the most tough times such as when receiving the 5th rejection letter to my game-theoretic light-client paper.

I would like to extend my sincere appreciation to the other members of my committee: Professor Schieber Baruch, Dr. Roman Vaculin, and Professor Jian Yang for their great support, precious time and invaluable advice. Professor Baruch brought me a great example as a computer scientist to balance theoretic studies and real-world applications. Dr. Roman Vaculin is an excellent expert on using the blockchain to resolve real-world problems in machine learning, supply chain, and other business sectors. His novel ideas on this line of research inspired me to think the future of using blockchain in various business sectors more thoroughly. Professor Jian Yang is the one that introduced game theory to me. The game-theory lecture taught by him inspired me a lot, and essentially shed light on a couple of ideas to design secure yet still efficient blockchain-based applications in the game-theoretic setting.

It is worthwhile to note that without the generous support from the Department of Computer Science, it would be impossible for me to complete the PhD degree.

In addition, the Department provides a fantastic environment that encourages me and many enthusiastic peers to exchange our thoughts to simulate novel research ideas. These peers that I met at NJIT are greatly thanked for tremendous fruitful academic discussions. An incomplete list includes Dr. Long Chen, Mr. Songlin He, Mr. Zhenliang Lu, and more. Remarkably, many of them gradually become reliable collaborators, who work extremely hard in contributing our coauthored papers, and deserve my deepest gratitude.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Traditional Crowdsourcing: Over-Reliance on Platforms

Recently, the paradigm of crowdsourcing empowers open collaboration on problem solving and truth finding over the Internet. One remarkable example is the solicitation of annotated data to create machine learning benchmark: the famous ImageNet challenge [41] was created via Amazon's crowdsourcing marketplace, Mechanical Turk (MTurk) [5]. Another notable example is mobile crowdsensing [56] to help citizens' daily life: one (called "requester") can request a group of individuals (called "workers") to use mobile devices to gather information fostering data-driven mobile applications [148, 43]. Recently, the great commercial successes of Uber [145] and Airbnb [2] also indicate the crowdsourcing of physical resources/services as another promising aspect.

The most critical challenge of crowdsourcing is to incentivize workers to contribute knowledge/services on solving the tasks. For the purpose, various monetary incentive mechanisms were introduced [151, 159, 160, 154, 120, 133, 75] to motivate real efforts. To effectively facilitate these mechanisms, the state-of-the-art solution necessarily requires a trusted third-party (e.g., MTurk, Uber, etc.) to host the crowdsourcing tasks through their whole life-cycle, such that these platforms can fulfill the fair exchange between the crowd-shared data/services and the rewards; otherwise, the effectiveness of incentive mechanisms can be hindered by the so-called "free-riding" (i.e., dishonest workers reap rewards without making real efforts) and "false-reporting" (i.e., dishonest requesters try to repudiate the payment).

It is well-known that the reduction of the over-reliance on a trusted third-party is desirable in practice, and the same goes for the case of crowdsourcing. First,

numerous real-world incidents reveal that the party might silently misbehave in-house for self-interests [112]; or, some of its employees [149] or attackers [78] can compromise its functionality. Second, the party often fails to resolve disputes. For instance, requesters have a good chance to collect data without paying at MTurk, because the platform is biased on requesters over workers [103]. Third, a centralized platform inevitably inherits all the vulnerabilities of the single point failure. For example, Waze, a crowdsourcing map app, suffered from 3 unexpected server downs and 11 scheduled service outages during 2010-2013 [148]. Worse still, a central platform hosting all tasks also increases the worry of massive privacy breach. A most fresh lesson to us is the tremendous data leakage of Uber [105].

Noticeably, the trivial idea of directly enhancing the robustness/security of centralized platforms might not work well in practice. One reason is the potential huge cost of in-house robustness/security upgrade. For example, Alibaba has to deploy millions of servers to maintain a robust service during the course of its Singles' Day on-line sale to handle the burst of tremendous transactions; unfortunately, most of these servers will be idling in the daily operations after the shopping day ends, causing significant operational overhead in the centralized system [3]. Let alone, it could be the case that a certain centralized platform would not be trusted due to non-technical reasons.

1.2 Blockchain: High Robustness from Less Assumptions

In contrast to the arguably vulnerable central platforms, the blockchain, in particular, the public/open/permissionless blockchain, is a highly distributed, transparent and immutable global "bulletin board" replicated across the whole Internet as a chain of blocks. The chain is usually maintained by a large-scale peer-to-peer (P2P) network collectively. Each block in the chain will include some messages committed by network peers, and be validated by the whole P2P network according to a

pre-defined consensus protocol. This ensures reliable message deliveries via the untrusted Internet. More interestingly, the messages contained in each block can be program code, the execution of which is enforced and verified by all P2P network peers; hence, a more exotic application called smart contract [138] is enabled on top of the blockchain. All these benefits are achieved without assuming the existence of any fully trusted third-party platform, and only require a very lightweight assumption that a portion of Internet users (e.g., more than half) are honest instead.

Essentially, the smart contract can be abstracted as a decentralized global "consensus computer" that faithfully handles all computations and message deliveries related to a specified task (except the adversary can choose the order of messaging). It becomes enticing to build a *decentralized* crowdsourcing platform atop it for higher robustness to reduce the over-reliance on the fragile assumption of the trusted third-party platform. Such a robust decentralized system might enjoy high availability and correctness, where the availability means that the crowdsourcing service is always readily on-line to serve the users and the correctness indicates that the service always executes correctly for pre-defined functionalities (such as trustfully enforcing the critical incentive mechanisms to motivate the participation of highly-skilled workers).

1.3 Challenges: Non-Triviality due to Limits of Blockchain

Unfortunately, the new fascinating technology of blockchain also brings about new challenges, due to its inherent restrictions.

The first inherent limitation of blockchain is the serious privacy challenge [76, 85, 82], which was never that severe in the centralized setting before, as one notable feature of the blockchain is its *transparency*. The whole chain is replicated by a permissionless network to ensure consistency, thus the data submitted to the blockchain will be visible to the public, which causes the leakage of authentication messages and crowdsourced data immediately. While in the centralized setting, users'

authentication history and the important data are protected by data centers (such as the breached one of Uber's).

Another fundamental issue of the blockchain (more particularly, smart contract) is that the smart contract cannot support complex computations. The reason lies in the fact that: during the blockchain mining procedure (the new block generation), when some output of a smart contract is expected to be recorded (that might also affect the validity of future blocks), honest miners are required to execute the program in order to validate the correctness of the outcome. If such a program is computationally intensive, crafty adversarial nodes may simply skip such verification step (or ignore putting the output at all), and go ahead to propose new blocks. Doing this gives the adversarial nodes substantial advantage of winning the chance for proposing new blocks, as honest nodes would not be able to propose any block until the execution of the smart contract finishes. Such an undesirable feature was known as verifier's dilemma [102].

Besides the previous issues, the smart contract can only validate the internal states of blockchain, but cannot verify the events happening in the "real world" instead of the "blockchain world" [158]. Such a critical issue can be translated as that the blockchain is only trusted to faithfully compute, but is not trusted to rule out fake "real-world" inputs, which unavoidably prevents the applicability of blockchains in various interesting use-cases. For example, the blockchain cannot know whether a particular computer is sending a valid message to another user, if the communication is off the blockchain. That said, the crowdsourcing of physical resources is still non-trivial, even if one has the magic smart contract technology at hand.

After all, one more challenge of implementing decentralized applications is that mobile devices and browser environments cannot afford the cost of executing consensus, which makes them have to rely on a blockchain node to relay the transactions recorded by the blockchain [96, 80]. In many practical use-cases, there

is no such a trusted relay node, and DApp users have to suffer from the risk of being cheated by third-party relays.

That said, the decentralized crowdsourcing, though it is highly appealing, still suffers from the inherent restrictions of blockchain such as privacy leakage, transaction re-ordering, fake real-world input, etc. As a consequence, meaningful decentralization of crowdsourcing is highly non-trivial. In particular, prior to this study, most existing attempts in the sector of decentralized crowdsourcing [21, 139, 92, 111] are arguably problematic, since they do not incorporate proper designs against the vulnerabilities of blockchain, thus rendering the failure of basic utilities when these publicly-known weaknesses are exploited by any malicious nodes from the Internet.

1.4 Main Results and Dissertation Structure

It remains a challenging open problem to attain meaningfully decentralized crowdsourcing systems (which can not only achieve guaranteed utilities but also realize real-world practicality). This dissertation thoroughly studies how to overcome these urgent challenges, and finally achieves provably secure yet still highly feasible decentralized systems for crowdsourcing various resources. The remaining of dissertation is organized as follows.

Chapter 2 introduces relevant preliminaries, including a few standard cryptographic building blocks, such as blockchain's public ledger model, the definitions of encryption, digital signature, zero-knowledge proofs, cryptographic commitments. In addition, it introduces the needed game-theoretic notions that can be used to argue security in cryptographic protocols among rational participants.

Chapter 3 focuses on the crowdsourcing of human knowledge, where serious data leakage and identity breach can be caused by the transparency of blockchain. The basic utilities of the crowdsourcing system for human knowledge will be further harmed, as these crowdsourcing tasks can be private, valuable, and personal

information sensitive. To mitigate this intrinsic issue of blockchain, a private and anonymous crowdsourcing framework called ZebraLancer is presented [98] for generic scenarios of soliciting human knowledge. Advanced cryptographic primitives are employed/invented to resolve data leakage and identity breach problems, more interestingly, fair exchange and user accountability are not sacrificed.

Chapter 4 presents a follow-up study to improve the arguably cumbersome ZebraLancer regarding efficiency. In particular, a concretely efficient system called Dragoon is designed. Dragoon is not only provable secure to protect the privacy of data that are submitted via the transparent blockchain, but also is highly efficiently, because various non-trivial efficiency-driven optimizations are performed to remove the needless generality to squeeze the most performance via concrete implementations.

Chapter 5 discusses the crowdsourcing of computing resources for running machine learning programs. Since the "global computer" instantiated by the blockchain can only support very simple and deterministic computations, and thus cannot perform complex and randomized computations such as machine learning analysis. To solve the issue, a novel and simple incentive game is designed to outsource the computing of machine learning tasks. Through this incentive game, a class of machine learning tasks can be executed off-chain by a few workers, and the correct execution results will be reported by the workers to the blockchain for rationality.

Chapter 6 initiates the first systematic study on the long-existing open problem of generic superlight client of permissionless blockchains. The proposed protocol allows the light client to recruit some full nodes in the blockchain network, and instantiates an incentive game between the low-capable client and the blockchains' full nodes via dedicatedly designed smart contract. The desired Nash equilibrium of the incentive game would ensure the full nodes to correctly forward blockchain's records to the resource-starved end-user.

Chapter 7 briefly summarizes a few relevant preliminary results that are worth to be further explored to extend the scope of the thesis.

Finally, in Chapter 8, the dissertation ends up with a few detailed discussions to summarize the earlier mentioned results, check out the fundamentally methodological inspirations, and point out a few promising follow-up directions.

CHAPTER 2

PRELIMINARIES

2.1 Notations and Abbreviations

Here are some notations and abbreviations that we use through the dissertation:

- *Negligible function.* By negligible function $negl(\cdot)$, it denotes a function (in some security parameter λ) for any positive polynomial function $poly(\cdot)$, there exists an integer N, such that for all $\lambda > N$, $|negl(\lambda)| < \frac{1}{poly(\lambda)}$.

- *Negligible probability and overwhelming probability.* If the probability of an event is a negligible function (in λ), it is said that the event happens with negligible probability; If the probability of an event is 1 except with negligible probability, the event is said with overwhelming probability.

- *String concatenation.* By $x||y$ it denotes a string concatenating strings x and y.

- *Uniformly sampling.* By $x \xleftarrow{\$} \{0,1\}^\lambda$, it denotes to uniformly sample a string x from the set of all λ-bit strings.

- *Abbreviations.* Through the paper, TM means Turing machine, ITM represents interactive Turing machine, and P.P.T. is short for probabilistic polynomial-time. For rigorous definitions about TM, ITM, P.P.T. TM, and P.P.T. ITM, see [134] for details.

2.2 Cryptographic Abstraction of Blockchain

2.2.1 Ideal Global Ledger Model

The (permissionless) blockchain instantiates a so-called public ledger or global ledger, which is essentially an ever-growing linearized transaction log collectively maintained by a network of untrusted Internet nodes. An honest node in such a network, also known as an honest full node, keeps a blockchain replica consistent with that of all other honest ones, through following a set of pre-defined rules called consensus protocol.

In general, we can view a blockchain network as an ideal public ledger that possesses two critical properties: persistence and liveness:

- Persistence can enforce the convergence of the local replicas of all honest nodes, i.e., if a transaction appears in the local replica of an honest node, it will eventually be at the same position in the replica of every other honest node, when the underlying network delay is finite.

- Liveness enables that anyone can announce valid transactions to the blockchain network. More importantly, one can expect the transactions to be eventually written into the local replicas of all the honest nodes, if the underlying network can eventually delivers the transactions.

These two properties ensure that: (i) A full node, who joins and executes the consensus protocol, can "read" the blockchain ledger from its local replica and also "write" any valid transaction into the chain by broadcasting. (ii) Any node, who may not participate in the consensus protocol, is able to "write" valid transactions into the blockchain. In practice, the writing can be realized by gossiping with some honest full nodes to "broadcast" the transactions.

2.2.2 Smart Contracts as Ideal Functionalities

Shortly after the emergence of blockchain, it was realized this fantastic technology can achieve much more beyond a bookkeeping ledger, as the transactions can also contain versatile program codes (e.g., Turing-complete scripts) in addition to monetary transfer records. In particular, the blockchain could be treated as a "bulletin board", where a piece of script along with all needed inputs can be posted. Furthermore, when the execution environment of the script is shared across the whole network through the underlying consensus rules, the same executing results of these codes can be enforced and verified by the blockchain network collectively. More interestingly, the piece of code posted on the blockchain (a.k.a., smart contract) can even hold some deposits (in the form of cryptocurrency) and finally pay out the deposits according to certain execution results, making self-executing conditional payment possible.

The blockchain, therefore, instantiates a transparent global "consensus computer" that can be employed by any node to faithfully handle with conditional money transfers, Turing-complete computations and message deliveries related to a pre-specified task. The only exception is that the blockchain cannot "send" messages to a node who does not join the consensus, e.g., a lightweight node. More specifically, the smart contract can be abstracted as an ideal functionality acting with Internet users to assist the following tasks [85, 82]:

- *Global clock.* There is an explicit global clock that would proceed in rounds. The rationale behind the abstraction is that the blockchain's liveness ensures it to grow block by block.

- *Reliable delivery of messages.* All messages sent to smart contract will be delivered to all parties by the start of next clock. This is because a message sent to the blockchain is in the form of a validly signed transaction broadcasted to the whole blockchain network, and then it will be solicited by a block and written into the blockchain [57] in polynomial time (under synchrony assumption).

- *Correct computation.* Smart contract can be seen as a state machine driven by messages sent to it [23]. By the increment of each block, the states of smart contract are changed according to the messages sent in the past clock period. The transitions of states (i.e., computations) are always correct, as being collectively validated by the while blockchain network.

- *Transparency.* All internal states and message deliveries will be visible to everyone in the whole blockchain network (intuitively, anyone).

- *Pseudonym (blockchain address).* By default, the sender of a message in the blockchain is referred to a pseudonym, a.k.a., blockchain address. In practice, a blockchain address is usually bounded to the hash of a public key of a digital signature scheme. Also, any smart contract deployed in the blockchain can also be referred by a unique address, such that one can call the contract to be executed, by committing a transaction pointing to this unique address.

2.3 Other Relevant Cryptographic Primitives

2.3.1 Public Key Encryption

A public key encryption scheme has a tuple of three algorithms $(\mathsf{PKE.KGen}, \mathsf{Enc}, \mathsf{Dec})$. $\mathsf{PKE.KGen}(\lambda) \rightarrow (sk, pk)$ is a probabilistic algorithm that takes a security parameter

λ as input and returns a public key pk and a secret key sk as output. $\mathsf{Enc}_{pk}(m) \rightarrow c$ is a probabilistic algorithm that takes a message m and the public key pk as input and returns a ciphertext c as output. $\mathsf{Dec}_{sk}(c) \rightarrow m$ is a deterministic algorithm that takes the ciphertext c and the secret key sk as input and returns the message m as output. Through the dissertation, the public key encryption scheme is required to satisfy the following correctness and security properties:

- *Correctness.* For any message m, the probability $\Pr[\mathsf{Enc}_{sk}(c) \equiv m \mid (sk, pk) \leftarrow \mathsf{PKE.KGen}(\lambda) \ \wedge \ c \leftarrow \mathsf{Enc}_{pk}(m)] = 1$, which is taken over the random coins of all probabilistic algorithms.

- *IND-CPA security.* The encryption scheme must achieve the security of indistinguishability under chosen plaintext attack (IND-CPA), which is equivalent to semantic security and essentially captures that only negligible information about the message can be feasibly extracted from the ciphertext [77].

2.3.2 Cryptographic Hash Function

A cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a function from the domain of arbitrary bit string to the domain of λ-bit strings, where λ is the security parameter. Usually, for a cryptographic hash function with security parameter λ, any polynomial time algorithms: (i) cannot find two strings m and m' ($m \neq m'$) such that $\mathcal{H}(m) = \mathcal{H}(m')$ except with negligible probability in λ, which is known as the property of collision resistance; (ii) given y, cannot find x, such that $\mathcal{H}(x) = y$, with all but negligible probability in λ, which is known as the property of preimage resistance.

In addition to the preimage resistance and collision resistance, this dissertation sometimes considers a stronger model of cryptographic hash function, namely, the random oracle model which is also global and programmable (by the adversary) due to standard cryptographic practices [25].

2.3.3 Cryptographic Commitment

The cryptographic notion of a commitment scheme realizes a digital "locked box", so the committed content can be hidden, and later be uniquely opened. A commitment scheme is a tuple of two algorithms, one of which is the commit algorithm and the other one of which is the open algorithm.

The commit and open algorithms can be denoted by $c \leftarrow \mathsf{Commit}(m, K)$ and $0/1 \leftarrow \mathsf{Open}(c, m', K)$, respectively. Note that be K it denotes a blinding key.

The security requirements for the commitment scheme can be defined as:

- *Hiding*, that means the receiver seeing the commitment c learns nothing about the committed message m.
- *Binding*, that means the sender cannot cheat the receiver to accept (i.e., output 1) if revealing m' different from the previously committed message m.

Both of the above two properties shall hold will all but except negligible probably (in the security parameter λ). In addition, the commitment notion always satisfies the basic correctness requirement, which means if the Open algorithm would always output 1 to accept, if it takes as input the commitment c, and the correct blinding key K and the correct message m.

2.3.4 Zero-Knowledge Proof

A zero-knowledge proof protocol (zk-proof) is a two-party protocol between a prover and a verifier. It allows the prover to generate a cryptographic proof convincing the verifier that the truthness of a certain statement $(x, w) \in \mathcal{L}$, where x is some public input, w is some private input of the prover (i.e., witness), and $\mathcal{L}$ is a specific language. Moreover, the protocol scripts (including the proof) shall not reveal any information except that the statement is true (i.e., $(x, w) \in \mathcal{L}$) to the verifier.

In greater detail, zk-proof shall satisfy the following security guarantees with an overwhelming probability:

- *Soundness*, that no prover can produce a proof to convince a verifier that $\exists w$ s.t. $(x, w) \in \mathcal{L}$, if $\nexists w$ s.t. $(x, w) \in \mathcal{L}$; sometimes, we require a stronger soundness that for any prover, there exists an extractor algorithm which interacts with the prover and can actually output the witness w (a.k.a., *proof-of-knowledge*).

- *Zero-knowledge*, that the proof distribution can be simulated by a P.P.T. simulator without seeing any secret state, i.e., it leaks only negligible information about the witness w.

The zero-knowledge succinct non-interactive argument of knowledge (namely, zk-SNARK) is a generic framework of zk-proof for any NP language $\mathcal{L}$, which further allows such a proof to be generated non-interactively in common-reference string model. More importantly, the proof is *succinct*, i.e., the proof size is independent on the complexity of the statement to be proved, and is always a small constant. More precisely, zk-SNARK is a tuple of three algorithms. A setup algorithm can output the public parameters (i.e., common-reference string) to establish a SNARK for a NP-complete language $\mathcal{L} = \{x \mid \exists w, s.t., C(x, w) = 1\}$, where $C(\cdot, \cdot) = 1$ can be any NP relationship. The Prover algorithm can leverage the established zk-SNARK to generate a constant-size proof attesting the trueness of a statement $x \in \mathcal{L}$ with witness w. The $\mathsf{Verifier}$ algorithm can efficiently check the validity of the proof.

2.3.5 Verifiable Decryption

Through the dissertation, it considers a specific verifiable public key encryption (VPKE) consisting of a tuple of algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Verify})$ with concrete verifiability to allow the decryptor to produce the plaintext along with a proof attesting the correct decryption [28].

In short, KGen can set up a pair of encryption-decryption algorithms $(\mathsf{Enc}_h, \mathsf{Dec}_k)$, where h and k are public and private keys respectively. We let any $(\mathsf{Enc}_h, \mathsf{Dec}_k) \leftarrow \mathsf{KGen}(1^\lambda)$ to be a public key encryption scheme satisfying semantic security. For presentation simplicity, we also let $(\mathsf{Enc}_h, \mathsf{Dec}_k)$ denote the public-secret key pair (h, k). Moreover, for any $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$, the Prove_k algorithm explicitly inputs

the private key k and the ciphertext c, and outputs a message m with a proof π; the Verify_h algorithm explicitly inputs the public key h and (m, c, π), and outputs 1/0 to accept/reject the statement that $m = \mathsf{Dec}_k(c)$.

Beside, we let VPKE to satisfy the following extra properties (i.e., a specifically verifiable decryption):

- *Completeness.* $\Pr[\mathsf{Verify}_h(m, c, \pi) = 1 \mid (m, \pi) \leftarrow \mathsf{Prove}_k(c)] = 1$, for $\forall$ c and $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$;

- *Soundness.* For any $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$ and c, any P.P.T. $\mathcal{A}$ cannot produce π fooling Verify_h to accept that c is decrypted to m' if $m' \neq \mathsf{Dec}_k(c)$, with except negligible probability;

- *Zero-knowledge.* The proof π can be simulated by a P.P.T. simulator S_{VPKE}, on input only public knowledge m, h and c that indeed satisfy $(m, c, h) \in \mathcal{L}_{VPKE} := \{\vec{x} := (m, c, h) \mid m = \mathsf{Dec}_k(c) \wedge (h, k) \leftarrow \mathsf{KGen}(1^\lambda)\}$, which ensures the protocol leaks nothing more than the truthness of the statement $m = \mathsf{Dec}_k(c)$.

2.3.6 Simulation-Based Paradigm

To formalize and prove security, a real world and an ideal world can be defined and compared: (i) in the real world, there is an actual protocol Π among the parties, some of which can be corrupted by an adversary $\mathcal{A}$; (ii) in the ideal world, an "imaginary" trusted ideal functionality F replaces the protocol and interacts with honest parties and a simulator S. We say that Π *securely realizes* F, if for $\forall$ P.P.T. adversary $\mathcal{A}$ in the real-world, $\exists$ a P.P.T. simulator S in the ideal-world, s.t. the two worlds cannot be distinguished, which means: no P.P.T. distinguisher D can attain non-negligible advantage to distinguish "the joint distribution over the outputs of honest parties and the adversary $\mathcal{A}$ in the real world" from "the joint distribution over the outputs of honest parties and the simulator S in the ideal world". Intuitively, all admissible ideal-world simulators cannot break any security guarantees, as the ideal-world is secure by default; then, the indistinguishability of real-world adversaries and ideal-

world simulators will immediately imply the computational security of real-world protocol.

Moreover, we consider the static adversary through the dissertation, which means the adversary can corrupt some parties only before the protocol starts. Protocols proven secure in the real/ideal paradigm can be composed sequentially, due to the transitivity of security reductions [65], which is known as *sequential composition theorem* [65, 29]. The advantage of simulation-based paradigm is that all desired behaviors of the protocol can be precisely described by the ideal functionality. Remarkably, this approach has been widely adopted to analyze decentralized protocols [85, 82, 17] to capture the subtle adversary in the blockchain.

2.4 Game-Theoretic Security

The security model in cryptographic settings assume that the participating parties are either completely honest or arbitrarily malicious. In game-theoretic settings, a party is neither honest nor malicious, but rational, and thus seeks for its best utility. In this scenario, the "security" can then be defined as the realization of desired Nash equilibrium or its refinements.

2.4.1 Security in Normal-Form Game

Particularly, a game Γ joined by n players, if described in normal-form, can consist of: (i) a set of players denoted by $\mathbf{W} = \{W_1, \ldots, W_n\}$; (ii) a space of players' pure strategies, denoted by $\mathbf{S} = \mathbf{S_1} \times \cdots \times \mathbf{S_n}$, in which $\mathbf{S_i}$ denotes the pure strategies of player W_i, while a strategy σ_i of player W_i is chosen from his strategy space (possibly by a randomized way), and we call $\vec{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$ as a joint strategy of the players; (iii) a utility function that defines the utility of each player under different joint strategies. 5 A Nash equilibrium is a particular joint strategy where no player can realize better utility by unilaterally changing his strategy. Standard Nash

equilibrium assumes each player makes decision independently. More generally, we may also consider collusion among part of the players. Throughout this dissertation, a particular refinement of Nash equilibrium that can be determined by iterated elimination of weakly-dominated strategies will be applied, as one always can expect a player would not play a strategy, if there is a better alternative for that strategy. In case there exists such a refined equilibrium $\vec{\sigma}$ in Γ, we call $(\Gamma, \vec{\sigma})$ a *practical mechanism*.

2.4.2 Security in Extensive-Form Game

Here are the deferred formal definitions of the finite incomplete-information extensive-form game and the sequential equilibrium.

Definition 1. Finite incomplete-information extensive-form game Γ *is defined as a tuple of* $\langle \mathbf{N}, \mathbf{H}, P, f_c, (u_i)_{i\in\mathbf{N}}, (\mathbf{I}_i)_{i\in\mathbf{N}} \rangle$ *[117]:*

- $\mathbf{N}$ *is a finite set representing the players.*
- $\mathbf{H}$ *is a set of sequences satisfying: (i)* $\emptyset \in \mathbf{H}$*; (ii) if* $h = \langle a_1, \ldots, a_K \rangle \in \mathbf{H}$*, then any prefix of* h *belongs to* $\mathbf{H}$*. Each member of* $\mathbf{H}$ *is a* history *sequence. The elements of a history are called* actions*. A history sequence* $h = \langle a_1, \ldots, a_K \rangle \in \mathbf{H}$ *is* terminal*, iff* h *is not a prefix of any other histories in* $\mathbf{H}$*. .*
- *Let* $\mathbf{Z}$ *denote the set of terminal histories. For any* non-terminal *history* $h = \langle a_1, \ldots, a_K \rangle \in \mathbf{H} \setminus \mathbf{Z}$*, the set of actions available after* h *can be defined as* $A(h) = \{a | \langle a_1, \ldots, a_K, a \rangle \in \mathbf{H}\}$*.*
- $P : \mathbf{H} \setminus \mathbf{Z} \to \mathbf{N} \cup \{chance\}$ *is the player function to assign a player (or* chance*) to move at a non-terminal history* h*. Particularly when* $P(h) = chance$*, a special "player" called* chance *acts at the history* h*.*
- $(u_i)_{i\in\mathbf{N}} : \mathbf{Z} \to \mathbb{R}^{|\mathbf{N}|}$ *is the utility function that defines the utility of the players at each terminal history (e.g.,* $u_i(h)$ *specifies the utility of player* i *at the terminal history* h*).*
- f_c *is a function associating each history* $h \in \{h | P(h) = chance\}$ *with a probability measure* $f_c(;h)$ *on* $A(h)$*, i.e.,* $f_c(a;h)$ *determines the probability of the occurrence of* $a \in A(h)$ *after the history* h *of the player "*chance*".*
- $(\mathbf{I}_i)_{i\in\mathbf{N}}$ *is a set of partitions. Each* $\mathbf{I}_i$ *is a partition for the set* $\{h | P(h) = i\}$*, and called as the* information partition *of player* i*; a member* $I_{i,j}$ *of the partition* $\mathbf{I}_i$

is a set of histories, and is said to be an information set *of player i. We require $A(h) = A(h')$ if h and h' are in the same information set, and then denote the available actions of player i at an information set $I_{i,j} \in \mathbf{I}_i$ as $A(I_{i,j})$.*

Note in our context, the strategy of a player is to choose a P.P.T. ITM, whose action is to produce a string and feed the string to the other P.P.T. ITMs (i.e., other players) in the cryptographic protocol.

Definition 2. *A* behavioral strategy *(or* strategy *for short) of player i (denoted by s_i) is a collection of independent probability measures denoted by $\{\beta_i(I_{i,j})\}_{I_{i,j} \in \mathbf{I}_i}$, where $\beta_i(I_{i,j})$ is a probability measure over $A(I_{i,j})$ (i.e., the available actions of player i at his information set $I_{i,j}$). We say $\vec{s} = (s)_{i \in \mathbf{N}}$ is a* behavior strategy profile *(or strategy profile for short), if every $s_i \in \vec{s}$ is a behavior strategy of player i. When $\vec{s} = (\{\beta_i(I_{i,j})\}_{I_{i,j} \in \mathbf{I}_i})_{i \in \mathbf{N}}$ assigns positive probability to every action, it is called completely mixed.*

Definition 3. *An* assessment *σ in an extensive game is a pair $(\vec{s}, \mu)$, where $\vec{s}$ is a behavioral strategy profile and μ is a function that assigns to every information set a probability measure on the set of histories in the information set (i.e., every). We say the function μ is a belief system.*

Definition 4. *The* expected utility *of a player i determined by the assessment $\sigma = (\vec{s}, \mu)$ conditioned on $I_{i,j}$ is $\bar{u}_i(\vec{s}, \mu | I_{i,j}) = \sum_{h \in I_{i,j}} \mu(I_{i,j})(h) \sum_{z \in \mathbf{Z}} \rho(\vec{s}|h)(z) u_i(z)$ where $h = \langle a_1, \ldots, a_L \rangle \in \mathbf{H} \setminus \mathbf{Z}$, $z = \langle a_1, \ldots, a_K \rangle \in \mathbf{Z}$, and $\rho(\vec{\sigma}|h)(z)$ denotes the distribution over terminal histories induced by the strategy profile $\vec{s}$ conditioned on the history h being reached (for player $P(h)$ to take an action), i.e.,*

$$\rho(\vec{s}|h)(z) = \begin{cases} 0, \text{ if } h \text{ is not a prefix of } z \\ \prod_{k=L}^{K-1} \beta_{P(a_1,\ldots,a_k)}(a_1, \ldots, a_k)(a_{k+1}), \text{ otherwise} \end{cases}$$

Definition 5. *We say an assessment $\sigma = (\vec{\sigma}, \mu)$ is a ϵ-*sequential equilibrium*, if it is ϵ-sequentially rational and consistent:*

- $(\vec{s}, \mu)$ is *ϵ-sequentially rational* if for every play $i \in \mathbf{N}$ and his every information set $I_{i,j} \in \mathbf{I}_i$, the strategy s_i of player i is a best response to the others' strategies $\vec{s}_{-i}$ given his belief at $I_{i,j}$, i.e., $\bar{u}_i(\vec{s}, \mu | I_{i,j}) + \epsilon \geq \bar{u}_i((s_i^*, \vec{s}_{-i}), \mu | I_{i,j})$ for every strategy s_i^* of every player i at every information set $I_{i,j} \in \mathbf{I}_i$. Note that $\vec{s}_{-i}$ denotes the strategy profile $\vec{s}$ with its i-th element removed, and $(s^*, \vec{s}_{-i})$ denotes $\vec{s}$ with its i-th element replaced by s^*.

- $(\vec{s}, \mu)$ is *consistent*, if $\exists$ a sequence of assessments $((\vec{s}^k, \mu^k))_{k=1}^{\infty}$ converges to $(\vec{s}, \mu)$, where $\vec{s}^k$ is completely mixed and μ^k is derived from $\vec{s}^k$ by Bayes' rules.

Remark. The above definition of extensive game (along with the relevant sequential equilibrium notion) provides a standard way to arguing the security of extensive protocols in the rational model [71, 45, 118]. Generally speaking, the game-theoretic analysis of an interactive protocol starts by defining an extensive game to model the strategies (i.e., probabilistic interactive Turing machines) of each party in the protocol. A utility function would assign every party a certain payoff, for each possible execution induced by the strategies of all parties. Then the security is argued by the properties of the extensive game, for example, its sequential equilibrium [69] or some other refinements of Nash equilibria [70, 71, 45, 118, 84].

CHAPTER 3

ON GENERIC PRIVATE KNOWLEDGE SOLICITATION

3.1 Background

Crowdsourcing empowers open collaboration over the Internet. One remarkable example is the solicitation of human knowledge, e.g., annotated data. The benchmark of famous ImageNet challenge [41] was created via Amazon's crowdsourcing marketplace, Mechanical Turk (MTurk) [5]. One most critical issue in crowdsourcing is that the crowd might lack interests to contribute qualified data. Therefore, various monetary incentive mechanisms were introduced [151, 159, 160, 154, 120, 133, 75] to motivate workers to make real efforts.

To facilitate these mechanisms, the state-of-the-art solution necessarily requires a trusted third-party (e.g., the server of MTurk) to host crowdsourcing tasks to fulfill the fair exchange between the crowd-shared data and the rewards; otherwise, the effectiveness of incentive mechanisms can be hindered by the so-called "free-riding" (i.e., dishonest workers reap rewards without making real efforts) and "false-reporting" (i.e., dishonest requesters try to repudiate the payment).

3.1.1 Motivation

It is well-known that the reduction of the reliance on a trusted third-party is desirable in practice, and the same goes for the case of crowdsourcing.

It becomes enticing to decentralize crowdsourcing atop blockchain, as it is a distributed, transparent and immutable public "bulletin board" organized as a chain of blocks. The blockchain is usually managed and replicated by a peer-to-peer (P2P) network collectively. Each block will include some messages committed by network peers, and be validated by the whole network according to a pre-defined consensus

protocol. This ensures reliable message deliveries via the untrusted Internet. More interestingly, the messages contained in each block can be program code, the execution of which is enforced and verified by all P2P network peers; hence, a more exotic application of smart contract [138] is enabled. Essentially, the smart contract can be viewed as a "decentralized computer" that faithfully handles all computations and message deliveries related to a specified task (except the adversary can choose the order of messaging). It becomes enticing to build a *decentralized* crowdsourcing platform atop.

3.1.2 Challenges

The new blockchain technology also brings about new privacy challenges [76], which were never that severe in the centralized setting before, as one notable feature of the blockchain is its *transparency*. The whole chain is replicated to the whole network to ensure consistency, thus the data submitted to the blockchain will be visible to the public. This causes an immediate problem violating data privacy, considering that many of the crowdsourced data maybe sensitive. For example, even in the intuitively "safe" image annotation tasks, if there are some special ambiguous pictures (e.g., Thematic Apperception Test pictures [113]), the answers to them can be used to infer the personality profiles of workers. Sometimes, the data are simply valuable to the requester who paid to get them. What is worse, since the block confirmation (which corresponds to the time when the submitted answers are actually recorded in a block) normally takes some time after the data is submitted to the network, a malicious worker can simply copy the data committed by others, and submit the same data as his own to run the free-riding attack. Without *data confidentiality*, the incentive mechanisms could be rendered completely ineffective in decentralized settings.

Furthermore, most crowdsourcing systems [5, 148] and incentive mechanisms [159, 160, 154] implicitly require requesters/workers to authenticate to prevent misbehaviors caused by (colluded) counterfeited identities [46]. When crowdsourcing is decentralized, this basic requirement will cause the history of submitting/requesting to be known by everyone via the blockchain (that was previously "protected" in a data center such as the breached one of Uber's). Thus considerable information about workers/requesters [155] will leak to the *public* through their participation history, which seriously impairs their privacy and even demotivates them to join. Notably, if a user frequently joins traffic monitoring tasks, then *anyone* can read the blockchain and figure out his location traces.

To address the above fundamental privacy challenges of decentralizing crowdsourcing, we have to resolve two natural tensions: (i) the tension between the blockchain transparency and the data confidentiality, and (ii) the tension between the anonymity and the accountability. Simple solutions utilizing some standard cryptographic tools (e.g., encryption and/or group signature) to protect the data confidentiality and the anonymity do not work well: the encryption of data immediately prevents smart contracts from enforcing the rewards policy; to allow fully anonymous participation will give a dishonest worker an opportunity of multiple submissions in one crowdsourcing task, and thus he may claim more rewards than what is supposed (similarly for a malicious requester).

3.2 Prior Art

The insufficiencies of the state-of-the-art solutions are thoroughly reviewed.

Centralized crowdsourcing systems. MTurk [5] is the most commercially successful crowdsourcing platform. But it has a well-know vulnerability allowing false-reporters gain short-term advantage [103]. Also, MTurk collects plaintexts of answers, which causes considerable worry of data leakage. Last, the pseudo IDs in

MTurks can be trivially linked by a malicious requester. Dynamo [128] was designed as a privacy wrapper of MTurk. Its pseudo ID can only be linked by the pseudo ID issuer, but still it inherited all other weaknesses of MTurk. SPPEAR [64] considered a couple of privacy issues in data crowdsourcing, and thus introduced a couple more authorities, each of which handled a different functionality. Distributing one authority into multiple reduces the excessive trust, but, unfortunately, it is still not clear how to instantiate all those different authorities in practice.

Decentralized crowdsourcing. We also note there are several attempts [21, 139, 92, 111] using blockchain to decentralize crowdsourcing, but neither of them considers privacy and anonymity which are arguably fundamental for basic utility: the authors of [21] built up a crowd-shared service on top of the blockchain, but the system is not compatible with incentive mechanisms and is not privacy-preserving either; the authors of [139] leveraged the blockchain as a payment channel in their crowdsourcing framework, but it is neither secure against malicious workers and dishonest requesters, nor privacy-preserving; a couple of recent attempts [92, 111] took advantage of the public blockchain to enforce incentives, but these frameworks are neither private nor anonymous, i.e., the collected data and the unique identities (such as certificates) will leak to the whole network of the open blockchain.

Anonymous crowdsourcing. Li and Cao [93] proposed a framework to allow workers generate their own pseudonyms based on their device IDs. But the protocol sacrificed the accountability of workers, because workers can forge pseudonyms without attesting that they are associated to real IDs, which gave a malicious worker chances to forge fake pseudo IDs and cheat for rewards. Rahaman et al. [125] proposed an anonymous-yet-accountable protocol for crowdsourcing based on group signature, and focused on how to revoke the anonymity of misbehaved workers. Misbehaved workers could be identified and further revoked by the group manager. The authors in [64] similarly relied on group signature but introduced a couple of

separate authorities. Our solution can be considered as a proactive version that can prevent worker misbehavior, and without relying on a group manager.

Accountable anonymous authentication. The pioneering works in anonymous e-cash [32, 33] firstly proposed the notion of one-time anonymous authentication. The concept later was studied in the context of one-show anonymous credential [27]. Some works [153, 140] further extended the notion of one-time use to be k-time use, and therefore enabled a more general accountability for anonymous authentications. In [26], the authors considered a special flavor of accountability to periodically allow k-time anonymous authentications. Our new primitive provides a more fine-grained conditional linkage of anonymous authentications, which is needed for preventing multi-submission in each crowdsourcing task.

Linkable group/ring signature. Conceptually similar to the linkability appeared in one-show credential [27], linkable group/ring signatures [115, 95] were proposed to allow a user to sign messages on behalf of his group unlinkably up to twice.

In the work of Man Ho et al. [8], a more general concept of event-oriented linkable group signature was formulated to realize more fine-grained trade-off between accountability and anonymity: a user can sign on behalf of his group unlinkably up to k times per *event*, where an *event* could be a common reference string (e.g., the unique address to call a smart contract deployed in the blockchain). But its main disadvantage is that the group manager can reveal the actual identities of users. Similar event-oriented linkability was discussed in the context of ring signature [143] as well. Even though that construction enjoys unbreakable anonymity, its main drawback becomes the impracticability of "hiding" the real identity behind a large group (mainly because the verification of signatures might require the public keys of all group members).

Our new primitive can be considered as a special cryptographic notion to formalize the subtle balance between event-oriented linkability and irrevocable

anonymity (within a large and dynamic group). Specifically, our scheme ensures that no one can link the actual identity to any authenticated message (which is strictly stronger than [8]), and also it enables a user to "hide" behind a large group of users (i.e., all users registered at RA, which is impossible in practice via [143]).

Privacy-preserving smart contracts. Privacy-preserving smart contract is a recent hot topic in blockchain research. Most of them are for general purpose consideration [82, 161], and thus deploy heavy cryptographic tools including general secure multi-party computation (MPC). Hawk [85] did provide a general framework for privacy-preserving smart contracts using light zk-SNARK, but mainly for reward receiver to prove to the contract. Our work can be considered as a very specially designed MPC protocol, and a lot of dedicated optimizations of zk-SNARK exist which can directly benefit our protocol. Last, cryptocurrencies like Zcash [73] and Ethereum [52] also leverage zk-SNARK to build a public ledger that supports anonymous transactions. We note that they consider more basic blockchain infrastructures, on top of which we may build our application for crowdsourcing.

3.3 Problem Formulation

More precisely, the definitions about the problem and its security requirements will be given as the following.

3.3.1 Modeling Knowledge Crowdsourcing

As illustrated in Figure 3.1, there are four roles in the model of data crowdsourcing, i.e., requesters, workers, a platform and a registration authority. A *requester*, uniquely identified by R, can post a task to collect a certain amount of answers from the crowd. When announcing the task, the requester promises a concrete reward policy to incentivize workers to contribute (see details about the definition of reward policy below). A *worker* with a unique ID W_j, submits his answer A_j and expects to receive

the corresponding reward. The *platform*, a medium assisting the exchange between requesters and workers, is either a trusted party or emulated by a network of peers. The platform considered in this dissertation is jointly maintained by a collection of network peers, and in particular, we will build it atop a open blockchain network. The *registration authority* (RA), can play an important role of verifying and managing unique identities of workers/requesters, by binding each identity to a unique credential (e.g., a digital certificate).

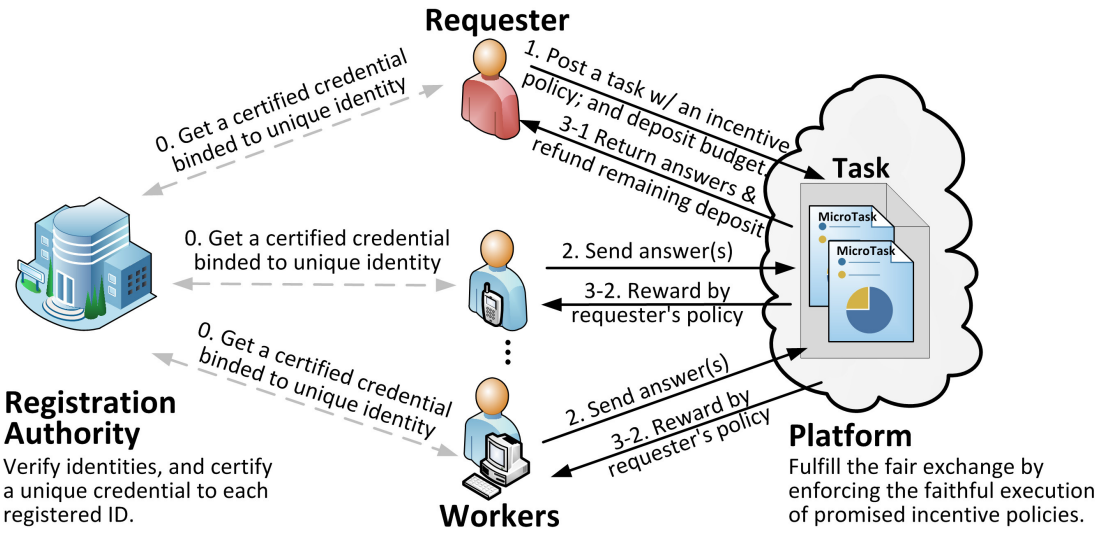

Figure 3.1 The "idealized" model of data crowdsourcing.

We remark that the well established identities are necessary demand of real-world crowdsourcing systems, for example MTurk and Waze, to prevent misbehaviors such as Sybil attack. Moreover, many auction-based incentive mechanisms [160, 154] are built upon the non-collusive game theory that implicitly requires established identities to ensure one bid from one unique ID. We employ RA to establish identities. In practice, a RA can be instantiated by (i) the platform itself, (ii) the certificate authority who provides authentication service, or (iii) the hardware manufacturer who makes trusted devices that can faithfully sign messages [129]. Our solution should be able to inherit these established RAs in the real-world.

In this dissertation, w.l.o.g., we assume that each unique identity is only allowed to submit one answer to a task. Also, we consider settings where the value of crowd-shared answers can be evaluated by a well-defined process such as auctions and quality-aware rewards, and also the corresponding rewards, c.f. [120, 133, 75] about quality-aware rewards and [160, 154] about auction-based incentives. These incentives share the same essence as follows.

Suppose an authenticated requester publishes a task T with a budget τ to collect n answers from n workers. An authenticated worker interested in it will then submit his answers. The *reward* of an answer A_j will follow a well-defined process determined by some auxiliary variables, i.e., the reward of A_j can be defined as $R_j := \mathsf{R}(A_j; a_1, \ldots, a_m, \tau)$, where R is a function parameterized by some auxiliary variables denoted by $a_1, \ldots, a_m, \tau$. Remark that τ is the budget of the requester, and we will use $R_j := \mathsf{R}(A_j; \tau)$ for short.

Particularly, in some simple tasks (e.g., multiple choice problems), the quality of an answer can be straightforwardly evaluated by all answers to the same task, i.e., $R_j = \mathsf{R}(A_j; A_1, \ldots, A_n, \tau)$, with using majority voting or estimation maximization iterations [120, 133, 75]. More generally, [9] proposed a universal method to evaluate quality: (i) some workers are requested to answer a complex task; (ii) other different workers are then requested to grade each answer collected in the previous stage. What's more, our model captures the essence of auction-based incentive mechanism such as [154, 160], when the parameters $a_1, \ldots, a_m$ represent the bids of workers (and other necessary auxiliary inputs such as their reputation scores).

Our work considers the general definition above and will be extendable to the scope of auction-based incentives, even though the protocol design and implementations in this dissertation mainly focus on how to instantiate quality-aware incentives. Also note that the flat-rate incentive, that is each submitted answer will

get a flat-rate payment [5], can be considered as a special case of the above abstraction as well.

3.3.2 Defining Security Goals

Next, we specify the basic security requirements for our (decentralized) crowdsourcing system on top of the existing infrastructure of open blockchain.

Data confidentiality. This property requires that the communication transcripts (including the blocks in the blockchain) do not leak anything to anyone (except the requester) about the input parameters $a_1, \ldots, a_m$ of the incentive policy R. Because these parameters might actually be confidential data or sealed bids. We can adapt the classical semantic security [67] style definition from cryptography for this purpose: the distribution of the public communication can be simulated with only public knowledge.

Anonymity. Private information of worker/requester can be explored by linking tasks they join/publish [155]. Intuitively, we might require two anonymity properties for workers: (i) *unlinkability between a submission and a particular worker* and (ii) *unlinkability among all tasks joined by a particular worker*. However, (i) indeed can be implied by (ii), because the break of first one can obviously lead up to the break of the latter one. Similarly, the anonymity of requester can be understood as the unlinkability among all tasks published by her. The requirement of worker anonymity can be formulated via the following game. An adversary $\mathcal{A}$ corrupts a requester, the registration authority (RA), and the platform (e.g., the blockchain); suppose there are only two honest workers, W_0 and W_1. In the beginning, the adversary announces a number of n tasks. For each task T_i, suppose there are a set of participating workers $\boldsymbol{W}_{T_i}$. After seeing all the communications, for any $T_i \neq T_j$, $\mathcal{A}$ cannot tell whether $\boldsymbol{W}_{T_i} \cap \boldsymbol{W}_{T_j} = \emptyset$ better than guessing. We note that the anonymity should hold even if all entities, including the requester and the platform (except W_0 and W_1), are

corrupted. The requester anonymity can be defined via the above game similarly, and we omit the details.

Security against a malicious requester. A malicious requester may avoid paying rewards (defined by the policy R), e.g., launches the *false-reporting* attack. Security in this case can be formulated via the following security game: an adversary $\mathcal{A}$ corrupts the requester and executes the protocol to publish a task with a promised reward policy R and a budget τ. Let us define a bad event B_1 to be that there exists a worker W_j, who submits answers A_j and receives a payment smaller than $R_j = \mathsf{R}(A_j; \tau)$. We require that for every polynomial time adversary $\mathcal{A}$, the probability $\Pr[B_1]$ is negligibly small.

Security against malicious workers. A dishonest worker may try to harvest more rewards than what he deserves. Security in this case can be formulated as follows: an adversary $\mathcal{A}$ corrupts one worker,[1] and participates in the protocol interacting with a requester (and the platform). $\mathcal{A}$ submits some answers $\boldsymbol{A} := \{A_1, \ldots, A_n\}, n \geq 1$. Let us define the bad event B_2 as that $\mathcal{A}$ receives a payment greater than $\max_{\{A_j \in \boldsymbol{A}\}} R_j := \mathsf{R}(A_j; \tau)$ from the requester. We require that for all polynomial time $\mathcal{A}$, $\Pr[B_2]$ is negligibly small.

We remark that the above securities against a malicious requester and malicious workers have capture the special fairness of the exchange between crowd-shared data and rewards.

3.4 Common-Prefix Linkable Anonymous Authentication

Before the formal description of ZebraLancer's protocol, let us introduce the new primitive for achieving the anonymous-yet-accountable authentication first. As briefly shown in Figure 3.2, our new primitive can be built atop any certification procedure,

[1] We remark that we focus on resolving the *new* challenges introduced by blockchain, and put forth the best possible security, as if there is a fully trusted third-party serving as the crowdsourcing platform. For example, it is not quite clear how to handle the collusion of many identities, even in the centralized setting; thus such a problem is out of the scope of this dissertation.

thus we include a certification generation procedure that can be inherited from any existing one. Also, we insist on *non-interactive* authentication, thus all the steps (including the authentication step) are described as algorithms instead of protocols. Formally, a common-prefix-linkable anonymous authentication scheme is composed of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$. This algorithm outputs the system's master public key mpk, and system's master secret key msk, where λ is the security parameter.
- $\mathsf{CertGen}(msk, pk)$. This algorithm outputs a certificate $cert$ to validate the public key.
- $\mathsf{Auth}(m, sk, pk, cert, mpk)$. This algorithm generates an attestation π on a message m that: the sender of m indeed owns a secret key corresponding to a valid certificate.
- $\mathsf{Verify}(m, mpk, \pi)$. This algorithm outputs 0/1 to decide whether the attestation is valid or not for the attested message, with using system's master public key.
- $\mathsf{Link}(mpk, m_1, \pi_1, m_2, \pi_2)$. This algorithm takes inputs two valid message-attestation pairs. If m_1, m_2 have a common-prefix with length λ, and π_1, π_2 are generated from a same certificate, it outputs 1; otherwise outputs 0.

We also define the properties for a common-prefix-linkable anonymous authentication. *Correctness* is straightforward that an honestly generated authentication can be verified. *Unforgeability* also follows from the standard notion of authentications, that if one does not own any valid certificate, she cannot authenticate any message. We mainly focus on the formalizing the security notions of *common-prefix-linkability* and *anonymity*.

The first one characterizes a special *accountability* requirement in anonymous authentication. It requires that no efficient adversary can authenticate two messages with a common-prefix without being linked, if using a same certificate. More generally, if an attacker corrupts q users, she cannot authenticate $q + 1$ messages sharing a common-prefix, without being noticed. Formally, consider the following cryptographic game between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$:

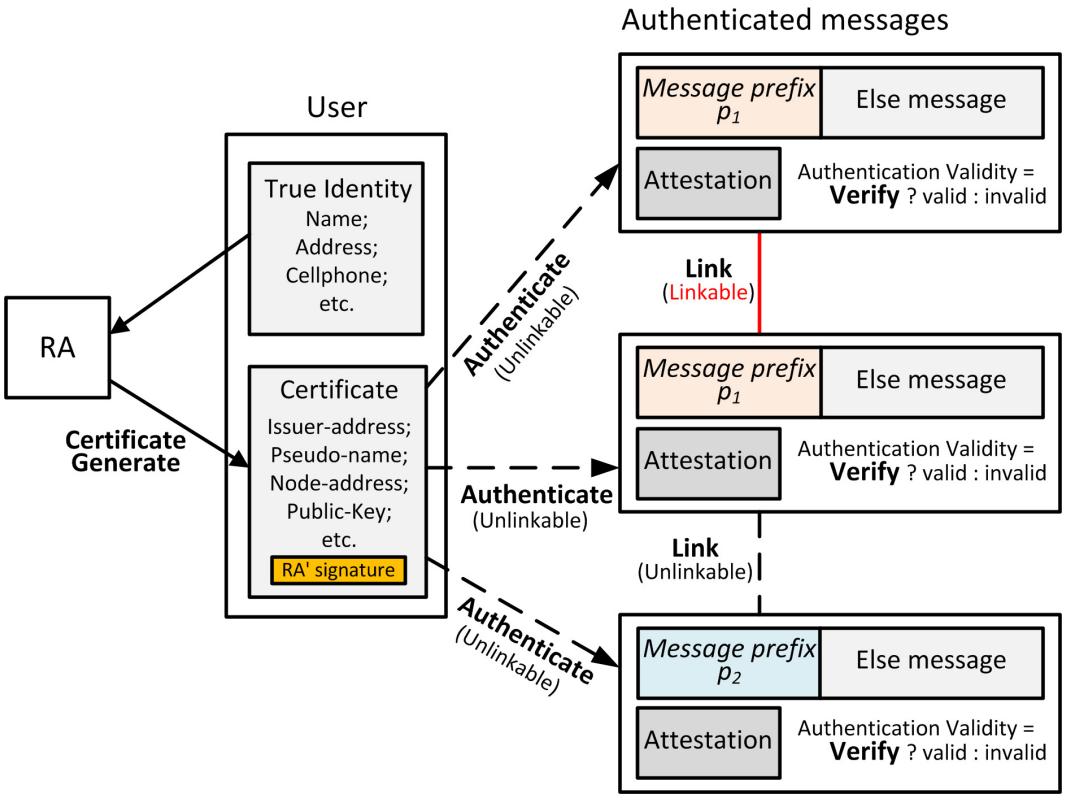

Figure 3.2 Subtle linkability of the common-prefix-linkable anonymous authentication scheme. All involved algorithms except Setup are shown in bold.

1. Setup. The challenger $\mathcal{C}$ runs the Setup algorithm and obtains the master keys.

2. CertGen queries. The adversary $\mathcal{A}$ submits q public keys with different identities and obtains q different certificates: $cert_1, \ldots, cert_q$.

3. Auth. The adversary $\mathcal{A}$ chooses $q+1$ messages $p||m_1, \ldots, p||m_{q+1}$ sharing a common-prefix p (with $|p| = \lambda$) and authenticates to the challenger $\mathcal{C}$ by generating the corresponding attestations $\pi_1, \ldots, \pi_{q+1}$.

Adversary $\mathcal{A}$ wins if all $q+1$ authentications pass the verification, and no pair of those authentications were linked.

Definition 6 (Common-prefix-linkability)**.** *For any probabilistic polynomial time adversary $\mathcal{A}$,* $\Pr[\mathcal{A}$ *wins in the above game]* *is negligible in some security parameter λ (over all random coins flipped by the challenger).*

Next is the *anonymity* guarantee in normal cases. We would like to ensure the anonymity against any party, including the public, the registration authority, and the verifier who can ask for multiple (and potentially correlated) authentication queries. Also, our strong anonymity requires that no one can even tell whether a user is authenticating for different messages, if these messages have different prefixes. The basic requirement for anonymity is that no one can recognize the real identity from the authentication transcript. But our *unlinkability* requirement is strictly stronger, as if one can recognize identity, obviously, she can link two authentications by firstly recovering the actual identities.

To capture the unlinkability (among the authentications of different-prefix messages), we can imagine the most stringent setting, where there are only two honest users in the system, the adversary still cannot properly link any of them from a sequence of authentications. Formally, consider the following game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. Setup. The adversary $\mathcal{A}$ generates the master key pair.

2. CertGen. The adversary $\mathcal{A}$ runs the certificate generation procedure as a registration authority with the challenger. The challenger submits two public keys pk_0, pk_1 and the adversary generates the corresponding certificates for them $cert_0, cert_1$. $\mathcal{A}$ can always generate certificates for public keys generated by herself.

3. Auth-queries. The adversary $\mathcal{A}$ asks the challenger to serially use $(sk_0, pk_0, cert_0)$ and $(sk_1, pk_1, cert_1)$ to do a sequence of authentications on messages chosen by her. Also, the number q of authentication queries is chosen by $\mathcal{A}$. The adversary obtains $2q$ message-attestation pairs.

4. Challenge. The adversary $\mathcal{A}$ chooses a new message m^* which does not have a common prefix with any of the messages asked in the Auth-queries, and asks the challenger to do one more authentication. $\mathcal{C}$ picks a random bit b and authenticates on m^* using $sk_b, pk_b, cert_b$. After receiving the attestation π_b, $\mathcal{A}$ outputs her guess b'.

It is said that the adversary wins the above game, if $b' = b$.

Definition 7 (Anonymity)**.** *An authentication scheme is unlinkable, if* $\forall$ *probabilistic polynomial-time algorithm* $\mathcal{A}$, $|\Pr[\mathcal{A}$ *wins in the above game*$] - \frac{1}{2}|$ *is negligible.*[2]

Construction. Now we proceed to construct such a primitive. Same as many anonymous authentication constructions, we will also use the zero-knowledge proof technique towards anonymity. In particular, we will leverage zk-SNARK to give an efficient construction. For the above concept of common-prefix-linkable anonymous authentication, we need to further support the special accountability requirement. The idea is as follows, since the condition that "breaks" the linkability is common-prefix, thus the authentication will do a special treatment on the prefix. In particular, the authentication shows a tag committing to the prefix together with the user's secret key, and then presents zero-knowledge proof that such a tag is properly formed, i.e., computed by hashing the prefix and a secret key. To ensure other basic security notions, we will also compute the other tag that commits to the whole message. The user will further prove in zero-knowledge that the secret key corresponds to a certified public key.

Note that our main goal is to decentralize crowdsourcing, such a new anonymous authentication primitive could be further studied systematically in future works. Concretely, we present the detailed construction as follows:

1. $\mathsf{Setup}(\lambda)$. This algorithm establishes the public parameters PP that will be needed for the zk-SNARK system. Also, the algorithm generates a key pair (msk, mpk) which is for a digital signature scheme.[3]

2. $\mathsf{CertGen}(msk, pk_i)$: This algorithm runs a signing algorithm on pk_i,[4] and obtains a signature σ_i. It outputs $cert_i := \sigma_i$.

[2]We remark that our definition of anonymity is strictly stronger than that definition of the event-oriented linkable group signature [8], in which the RA can revoke user anonymity under certain conditions.

[3]To be more precise, the public parameter generation could be from another algorithm. For simplicity, we put it here. In the security game for anonymity, the adversary only generates msk, mpk, not the public parameter.

[4]We assume an external identification procedure that can check the actual identity bound to each public key, and the user key pairs are generated using common algorithms, e.g., for a digital signature. We ignore the details here.

3. $\mathsf{Auth}(p||m, sk_i, pk_i, cert_i, PP)$: On inputting a message $p||m$ having a prefix p.

   The algorithm first computes two tags (or interchangeably called headers later), $t_1 = H(p, sk_i)$ and $t_2 = H(p||m, sk_i)$, where H is a secure hash function.

   Then, let $\vec{w} = (sk_i, pk_i, cert_i)$ represent the private witness, and $\vec{x} = (p||m, mpk)$ be all common knowledge, the algorithm runs zk-SNARK proving algorithm $\mathsf{Prover}(\vec{x}, \vec{w}, PP)$ for the following language $\mathcal{L}_T := \{t_1, t_2, \vec{x} = (p||m, mpk) \mid \exists \vec{w} = (sk_i, pk_i, cert_i)$ s.t. $\mathsf{CertVrfy}(cert_i, pk_i, mpk) = 1 \wedge pair(pk_i, sk_i) = 1 \wedge t_1 = H(p, sk_i) \wedge t_2 = H(p||m, sk_i)\}$, where the $\mathsf{CertVrfy}$ algorithm checks the validity of the certificate using a signature verification, and *pair* algorithm verifies whether two keys are a consistent public-secret key pair.

   This proving algorithm yields a proof η for the statement $\vec{x} \in \mathcal{L}$ (also for the proof-of-knowledge of $\vec{w}$). Finally, the algorithm outputs $\pi := (t_1, t_2, \eta)$.

4. $\mathsf{Verify}(p||m, \pi, mpk, PP)$: this algorithm runs the verifying algorithm of zk-SNARK $\mathsf{Verifier}$ on $\vec{x}, \pi$ and PP, and outputs the decision bit $d \in \{0, 1\}$.

5. $\mathsf{Link}(m_1, \pi_1, m_2, \pi_2)$: On inputting two attestations $\pi_1 := (t_1^1, t_2^1, \eta_1)$ and $\pi_2 := (t_1^2, t_2^2, \eta_2)$, the algorithm simply checks $t_1^1 \stackrel{?}{=} t_1^2$. If yes, output 1; otherwise, output 0. We also use $\mathsf{Link}(\pi_1, \pi_2)$ for short.

Security analysis (sketch). Here we briefly sketch the security analysis for the construction. As the scheme is of independent interests, we defer detailed analyses/reductions to an extended dissertation where the scientific behind will be formally studied. Regarding *correctness*, it is trivial, because of the completeness of underlying SNARK. Regarding *unforgeability*, we require an uncertified attacker cannot authenticate. The only transcripts can be seen by the adversary are headers and the zero-knowledge attestation. Headers include one generated by hashing the concatenation of $p||m, sk$. In order to provide a header, the attacker has to know the corresponding sk, as it can be extracted in the random oracle queries. Thus there are only two different ways for the attacker: (i) the attacker generates forges the certificate, which clearly violates the signature security; (ii) the attacker forges the attestation using an invalid certificate, which clearly violates the proof-of-knowledge of the zk-SNARK.

Regarding the *common-prefix-linkability*, it is also fairly straightforward, as the final authentication transcript contains a header computed by $H(p, sk)$ which is an invariable for a common prefix p using the same secret key sk.

Regarding the *anonymity/unlinkability*, we require that after seeing a bunch of authentication transcripts from one user, the attacker cannot figure out whether a new authentication comes from the same user. This holds even if the attacker can be the registration authority that issues all the certificates. To see this, as the attacker will not be able to figure the value of the sk from all public value, thus the headers/tags can be considered as random values. It follows that $H(p, sk)$ and a random value r cannot be distinguished (similarly for $H(p||m, sk)$). More importantly, due to the zero-knowledge property of zk-SNARK, given r, a simulator can simulate a valid proof η^* by controlling the common reference string of the zk-SNARK. That said, the public transcript t_1, t_2, η can be simulated by r_1, r_2, η^* where r_1, r_2 are uniform values, and η^* is a simulated proof, all of which has nothing to do with the actual witness sk.

Summarizing the above intuitive analyses, we have the following theorem:

Theorem 1. *Conditioned on that the hash function to be modeled as a random oracle and the zk-SNARK is zero-knowledge, the construction of the common-prefix-linkable anonymous authentication satisfies anonymity. Conditioned on the underlying digital signature scheme used is secure, and the zk-SNARK satisfies proof-of-knowledge, our construction of the common-prefix-linkable anonymous authentication will be unforgeable. It is also correct and common-prefix linkable.*

3.5 ZebraLancer: Private and Anonymous Decentralized Crowdsourcing

In this section, we will construct a private and anonymous protocol to address the critical challenges of decentralizing crowdsourcing, without sacrificing security against "free-riders" and "false-reporters". The procedures of crowdsourcing will be

decentralized atop an existing network of blockchain. More specifically, we will tackle the new privacy and anonymity challenges brought by the blockchain.

As we briefly mentioned in previous sections, the system will implicitly has a separate registration service that validates each participant's unique identity before issuing a certificate. Such setup alleviates some basic problems that every worker is allowed to submit no more than a fixed number k of answers. For simplicity, we consider here $k = 1$.

3.5.1 Design Intuition

Our basic strategy is to let the smart contract (which is hosted by the blockchain network) to enforce the fair exchange between the submitted answers and their corresponding rewards, but without revealing any data or any identity to the blockchain. Let us walk through the high level ideas first.

The requester firstly codifies a reward policy parameterized by her budget (i.e., $\mathsf{R}(\cdot;\tau)$) into a smart contract. She broadcasts a transaction containing the contract code and the budget. Once the smart contract is included in the blockchain, it can be referred by a unique blockchain address, and the budget should be deposited to this address (otherwise, no one would participate). After that, any worker who is interested in contributing could simply submit his answer to the blockchain, via a transaction pointing to the contract's address.

As pointed out before, we have to protect the *confidentiality* of the answers, in order to ensure that answers from different workers are independent. The workers encrypt the answers under the requester's public key. Now the contract cannot see the answers so it cannot calculate the corresponding rewards. But the requester can retrieve all the encrypted answers and decrypt them off-chain, and further learn the rewards they deserve. It would be necessary that the requester will *correctly* instruct the smart contract how to proceed forward. Concretely, we will leverage

the practical cryptographic tool of zk-SNARK to enforce the requester to prove: she indeed *followed the pre-specified reward policy* calculating the rewards. Detailedly, the requester should prove her instruction for rewarding is computed as follows: (i) obtain all answers by decrypting all encrypted answers using a secret key corresponding to the public key contained in the smart contract; (ii) use all those answers and the announced $\mathsf{R}(\cdot;\tau)$ to compute the quality of each answer.

A more challenging issue arises regarding *anonymity-yet-accountability*. Also as briefly pointed before, we would like to achieve a balance between anonymity and accountability. Here we put forth a new cryptographic primitive to resolve the natural tension. A user can anonymously authenticate on messages (which are composed of a fixed length prefix and the remaining part). But if the two authenticated messages share a common prefix, anyone can tell whether they are done by a same user or not. Moreover, no one can link any two message-authentication pairs, as long as these messages have different prefixes. Having this new primitive in hand, a simple and intuitive solution to the anonymous-yet-accountable protocol is to let each worker anonymously authenticate on a message $\alpha_{\mathscr{C}}||C_i$, whenever the encrypted answer $C_i = \mathsf{Enc}(epk, A_i)$ is submitted to a task contract $\mathscr{C}$ that can be uniquely addressed by $\alpha_{\mathscr{C}}$. This implies that all submissions from a same worker to one task can be linked and then counted, but any two submissions to two different tasks will be provably unlinkable, even if they are submitted by a same user. Also, the number of maximum allowed submissions in each task can be easily tuned (by counting linked submissions).

Last, we also need to augment the smart contract by building the general algorithm of verifying zero-knowledge proofs in it. In particular, when the smart contract receives an instruction regarding rewards and its proof, the verification algorithm will be executed. All inputs of the verification algorithm are common knowledge stored in the open blockchain, e.g., the budget, the encrypted answers and the public key of encryption. If a dishonest requester reports a false instruction, her

proof cannot be verified and the contract will simply drop the instruction. What's more, if the smart contract does not receive a *correct* instruction within a time limit, it can directly disseminate the budget to all workers evenly as punishment (which can be considered as part of the pre-specified incentive mechanism), since the budget has been deposited. In this way, the requester cannot gain any benefit by deviating from the protocol, and she will be self-enforced to respond properly and timely, resulting in that each worker will receive the expected reward. On the other hand, a dishonest worker can never claim more rewards than that he is supposed to get, as the reward is calculated by the requester herself.

Now we are ready to present a general protocol for a class of crowdsourcing tasks having proper quality-aware incentives mechanisms defined earlier. As zk-SNARK requires a setup phase, we consider that a setup algorithm generated the public parameters PP for this purpose, and published it as common knowledge.[5] Our descriptions focuses on the application atop the open blockchain, and therefore omits details of sending messages through the underlying blockchain infrastructure. For example, "one uses blockchain address α to send a message m to the blockchain" will represent that he broadcasts a blockchain transaction containing the message m, the public key associated to α, and the signature properly generated under the corresponding secret key.

Remark that here we let each worker/requester to generate a different blockchain address for each task (i.e., a *one-task-only* address) as a simple solution to avoid de-anonymization in the underlying blockchain.[6]

[5]This in practice can be done via a secure multiparty computation protocol [20] to eliminate potential backdoors.

[6]Our anonymous protocol mainly focuses on the application layer such as the crowdsourcing functionality that is built on top of the blockchain infrastructure. If the underlying blockchain layer supports anonymous transaction, such as Zcash [73], the worker and the requester can re-use account addresses. We further remark that the anonymity in network layer are out the scope of this dissertation, we may deploy our protocol on existing infrastructure such as Tor.

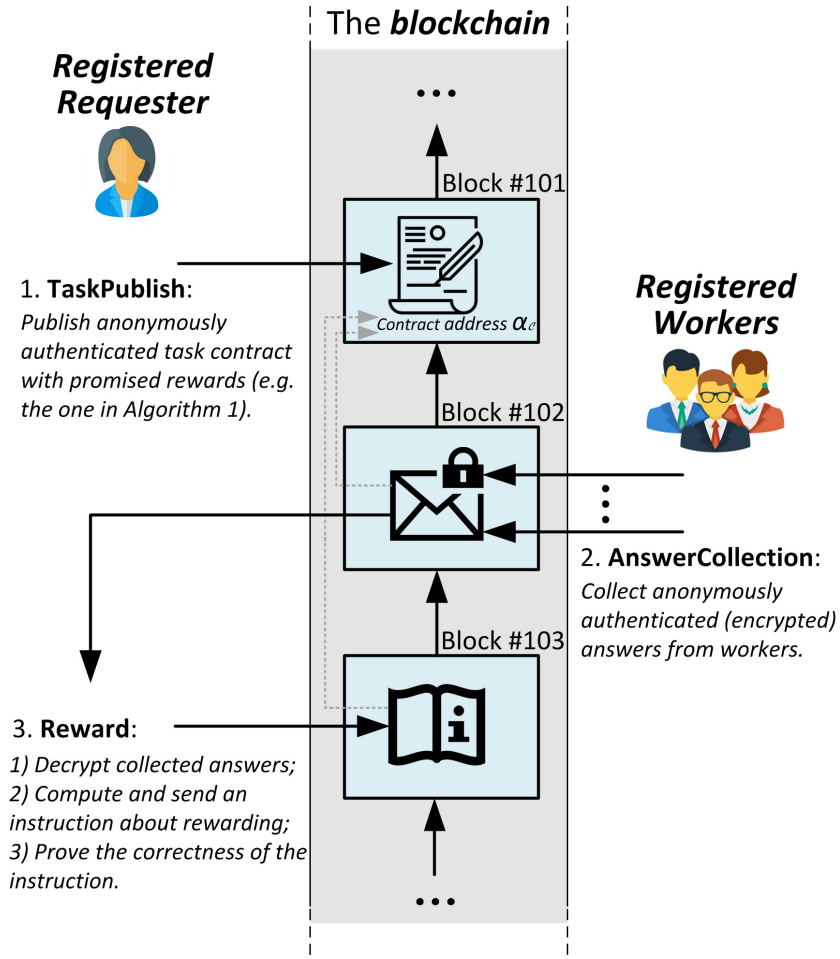

Figure 3.3 The schematic diagram of the ZebraLancer protocol.

3.5.2 Details of ZebraLancer Protocol

As briefly illustrated in Figure 3.3, the protocol of ZebraLancer proceeds as follows.

Register. *Everyone registers at RA to get a certificate bound to his/her unique ID, which is done off-line only once for per each participant.* A requester, having a unique ID denoted by R, creates a public-secret key pair (pk_R, sk_R), and registers at the registration authority (RA) to obtain a certificate $cert_R$ binding pk_R to R. Each worker, having a unique ID denoted by W_i, also generates his public and secret key pair (pk_i, sk_i), and registers his public key at RA to obtain a certificate $cert_i$ binding pk_i and W_i.

TaskPublish. *A requester anonymously authenticates and publishes a task contract with a promised reward policy.* When the requester R has a crowdsourcing task, she generates a *fresh* blockchain account address α_R, and a key pair (epk, esk) (which will be used for workers to encrypt submissions) for this task only. R then prepares parameters Param, including the encryption key epk, the number of answers to collect (denoted by n), the deadline, the budget τ, the reward policy R, SNARK's public parameters PP, RA's public key mpk, and also $\pi_R = \mathsf{Auth}(\alpha_{\mathscr{C}}||\alpha_R, sk_R, pk_R, cert_R, PP)$.[7] The requester then codes a smart contract $\mathscr{C}$ that contains all above information for her task. After compiling $\mathscr{C}$, she puts $\mathscr{C}$'s code and a transfer of the budget into a blockchain transaction, and uses the one-task-only address α_R to send the transaction into the blockchain network. When a block containing $\mathscr{C}$ is appended to the blockchain, $\mathscr{C}$ gets an immutable blockchain address $\alpha_{\mathscr{C}}$ to hold the budget and interact with anyone. [8] See Algorithm 1 below for a concrete example of task contract. (The important component of verifying zk-proofs is done by calling a library *libsnark.*Verifier integrated into the blockchain infrastructure, and implementation details will be explained in Section 4.4.5).

AnswerCollection. *The contract collects anonymously authenticated encrypted answers from workers who didn't submit before.* If a registered worker W_i is interested in contributing, he first validates the contract content (e.g., checking the parameters), then generates a *one-time* blockchain address α_i. He encrypts his

[7]We remark that the requester should authenticate on her blockchain address α_R along with the task contract, and workers will join the task only if the task contract is indeed sent from a blockchain address as same as the authenticated α_R. A malicious requester cannot "authenticate" a task by copying other valid authentications. In addition, each worker has to authenticate on his blockchain address α_i along with his answer submission as well. The task contract will check the submission is indeed sent from a blockchain address same to the authenticated α_i. Otherwise, a malicious worker can launch free-riding through copying and re-sending authenticated submissions that have been broadcasted but not yet confirmed by a block.

[8]We emphasize that $\alpha_{\mathscr{C}}$ will be unique per each contract. In practice, $\alpha_{\mathscr{C}}$ can be computed via $H(\alpha_R||counter)$, where H is a secure hash function, and *counter* is governed by the blockchain to be increased by exact one for each contract created by the blockchain address α_R. It's also clear that the requester R can predicate $\alpha_{\mathscr{C}}$ before $\mathscr{C}$ is on-chain, such that she can compute π_R off-line and let it be a parameter of contract $\mathscr{C}$.

answer A_i under the task's public key *epk* to obtain ciphertext C_i. He then uses common-prefix-linkable anonymous authentication scheme to generate an attestation $\pi_i = \mathsf{Auth}(\alpha_{\mathscr{C}}||\alpha_i||C_i, sk_i, pk_i, cert_i, PP)$.[7] Then he uses his *one-time* address α_i to send C_i, π_i to the blockchain network (with a pointer to $\alpha_{\mathscr{C}}$, i.e., the unique address of the contract $\mathscr{C}$). Then, $\mathscr{C}$ runs $\mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_i||C_i, \pi_i, mpk, PP)$, and also executes $\mathsf{Link}(\pi_i, \pi_*)$ for each valid authentication attestation π_* that was received before (including requester's, namely π_R). Such that, $\mathscr{C}$ can ensure C_i is the first submission of a registered worker. For unauthenticated or double submissions, $\mathscr{C}$ simply drops it.[9] The contract $\mathscr{C}$ will keep on collecting answers, until it receives n answers or the deadline (in unit of block) passes. It also records each address α_i that sends C_i. Remark that Link algorithm will be executed $O(n^2)$ times, but it is a simple equality check over a pair of hashes, such that the cost of running it for several times will be nearly nothing in practice.

Reward. *The requester computes and prove how to reward properly authenticated anonymous answers.* The requester R keeps listening to the blockchain, and once $\mathscr{C}$ collects n submissions, she retrieves and decrypts all of them to obtain the corresponding answers $A_1, \ldots, A_n$ (if there are not enough submissions when the deadline passes, the requester simply sets the remaining answers to be $\perp$ which has been considered by the incentive mechanism R). Next, the requester computes the reward for each answer $R_i = \mathsf{R}(A_i; A_1, \ldots, A_n, \tau)$ as specified by the policy codified in $\mathscr{C}$. More importantly, she generates a zero knowledge proof π_{reward}, with the secret key *esk* as witness to attest the validity of the instruction. In particular, the proof is for the following NP-language $\mathcal{L} = \{\overline{R}, \overline{P} \mid \exists esk \text{ s.t. } \wedge_{j=1}^{n} A_j = \mathsf{Dec}(esk, C_j) \wedge_{j=1}^{n} R_j = \mathsf{R}(A_j; A_1, \ldots, A_n, \tau) \wedge pair(esk, epk) = 1\}$, where $\overline{P}$ denotes Param together with ciphertexts $C_1, \ldots, C_n$; while $\overline{R} := (R_1, \ldots, R_n)$ is

[9] We remark that our protocol can be extended trivially to allow each worker to submit some k answers in one task by modifying the checking condition programmed in the smart contract of crowdsourcing task.

the instruction about how to reward each answer. After computing $\overline{R}$ and π_{reward}, R puts them into a blockchain transaction, and still use her one-task-only blockchain address α_R to send the transaction to $\mathscr{C}$ (by using a pointer to $\alpha_{\mathscr{C}}$). This finishes the *outsource-then-prove* methodology. Once a newly proposed block contains the reward instruction $\overline{R}$ and its attestation π_{reward}, the contract $\mathscr{C}$ first checks that they are indeed sent from α_R (by verifying the digital signature of the underlying blockchain transaction). Then it leverages SNARK's $\mathsf{Verifier}$ algorithm to verify the proof π_{reward} regarding the correctness of $\overline{R}$. If the verification passes, it transfers each amount R_i to each account α_i, and refunds the remaining balance to α_R. Otherwise, pause. If receiving no valid instruction after a predefined time (in unit of block), the contract simply transfers τ/n to each α_i as part of the policy R.

3.5.3 Security Analysis of ZebraLancer Protocol

Correctness and efficiency. It is clear to see that the requester will obtain data and the workers would receive the right amount of payments. If they all follow the protocol, under the conditions that (i) the blockchain can be modeled as an ideal public ledger, (ii) the underlying zk-SNARK is of completeness, (iii) the public key encryption is correct, and (iv) common-prefix-linkable anonymous authentication satisfies correctness. Regarding *efficiency*, we note the *on-chain* computation (and storage which are two of the major obstacles for applying blockchain in general) is actually very light, as the contract essentially only carries a verification step. Thanks to zk-SNARK, the verification can be efficiently executed by checking only a few pairing equalities; moreover, the special library can be dedicatedly optimized in various ways [15].

Security analysis (sketch). We briefly discuss security here and defer the details to an extended version. The underlying primitives, including our common-prefix-linkable

Algorithm 1: Example using quality-aware incentive

Require : This contract's address $\alpha_{\mathscr{C}}$; requester's one-time blockchain address α_R; requester's authenticating attestation π_R; RA's public key mpk; budget τ; public key epk for encrypting answers; SNARK's public parameters PP; number of requested answers n; deadline of answering in unit of block $T_{\boldsymbol{A}}$; deadline of instructing reward in unit of block $T_{\boldsymbol{I}}$.

List keeping answers' ciphertexts, $\boldsymbol{C} \leftarrow \emptyset$;
Map of anonymous attestations and authenticated one-time blockchain addresses of workers, $\boldsymbol{W} \leftarrow \emptyset$;
if $getBalance(\alpha_{\mathscr{C}}) < \tau \vee \neg\mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_R, \pi_R, mpk, PP)$ **then**
    goto 21 ; ▷ Budget not deposited or requester not identified.
$timer_{\boldsymbol{A}} \leftarrow$ a timer expires after $T_{\boldsymbol{A}}$;
while $||\boldsymbol{C}|| < n \wedge timer_{\boldsymbol{A}}$ *NOT expired* **do**
    if α_i *sends* π_i, C_i **then**
        if $\neg\mathsf{Link}(\pi_i, \pi_R) \wedge \forall \pi_* \in \boldsymbol{W}.keys()\ \neg\mathsf{Link}(\pi_i, \pi_*) \wedge \mathsf{Verify}(\alpha_{\mathscr{C}}||\alpha_i||C_i, \pi_i, mpk, PP)$ **then**
            $\boldsymbol{W}.add(\pi_i \rightarrow \alpha_i)$; $\boldsymbol{C}.add(C_i)$;
$timer_{\boldsymbol{I}} \leftarrow$ a timer expires after $T_{\boldsymbol{I}}$; ▷ Start to wait instruction
while $timer_{\boldsymbol{R}}$ *NOT expired* **do**
    if α_R *sends* $\overline{R} := (R_1, \ldots, R_n)$ *and* π_{reward} **then**
        $\overline{P} \leftarrow (epk, \tau, C_1, \ldots, C_n)$;
        if $libsnark.\mathsf{Verifier}((\overline{P}, \overline{R}), \pi_{reward}, PP)$ **then**
            for *each* $(\pi_i \rightarrow \alpha_i) \in \boldsymbol{W}$ **do**
                $transfer(\alpha_{\mathcal{C}}, \alpha_i, R_i)$;
            goto 21;
$R \leftarrow \tau/||\boldsymbol{W}||$; ▷ Reward all if no correct instruction
for *each* $(\pi_i \rightarrow \alpha_i) \in \boldsymbol{W}$ **do**
    $transfer(\alpha_{\mathcal{C}}, \alpha_i, R)$;
$transfer(\alpha_{\mathcal{C}}, \alpha_R, getBalance(\alpha_{\mathcal{C}}))$; ▷ Refund the remaining
function $getBalance(addr)$
    return the balance of $addr$ in the blockchain ledger;
function $transfer(src, dst, value)$
    if $getBalance(src) < value$ **then**
        return $false$;
    the balance of src subtracts $value$ in the blockchain ledger;
    the balance of dst adds $value$ in the blockchain ledger; **return** $true$;
▷ The correctness and availability of this task contract is governed by the blockchain network; the contract is driven by a global "discrete" clock
▷ $libsnark.\mathsf{Verifier}$ is a library embedded in the runtime environment of smart contract such as EVM

anonymous authentication scheme, are well abstracted, which enable us to argue security in a modular way.

Regarding the *data confidentiality* of answers, all related public transcripts are simply the ciphertexts $C_1, \ldots, C_n$, and the zk-SNARK proof π. The ciphertexts are easily simulatable according to the semantic security of the public key encryption, and the proof π can also be simulated without seeing the secret witness because of the *zero-knowledge* property.

Regarding the *anonymity*, an adversary has two ways to break it: (i) link a worker/requester through his blockchain addresses; (ii) link answers/tasks of a worker/requester through his authenticating attestations. The first case is trivial, simply because every worker/requester will interact with each task by a randomly generated one-task-only blockchain address (and the corresponding public key). The second case is more involved, but the anonymity of workers and requesters can be derived through the anonymity of the common-prefix-linkable anonymous authentication scheme.

Regarding the *security against a malicious requester*, a malicious requester has three chances to gain advantage: (i) deny the policy announced in TaskPublish phase; (ii) cheat in Reward phase; (iii) submit answers to intentionally downgrade others in AnswerCollection phase. The first threat is prevented because the smart contract is public, and the requester cannot deny it once it is posted in the immutable blockchain. The second threat is prohibited by the soundness of the underlying zk-SNARK, since any incorrect instruction passing the verification in the smart contract, directly violates the proof-of-knowledge (i.e., the strong soundness). The last threat is simply handled the unforgeability and common-prefix-linkability of our common-prefix-linkable anonymous authentication scheme.

Security against malicious workers is straightforward, the only ways that malicious workers can cheat are as follows: (i) to submit more than one answers

in AnswerCollection phase; (ii) sending the contract a fake instruction in the name of requester in Reward phase; (iii) altering the policy specified in the contract. The first threat is simply handled by the common-prefix-linkability and unforgeability of common-prefix-linkable anonymous authentication. The second threat can be approached by predicting others' answers, and it is prevented due to the semantical security of public key encryption. The third threat is simply handled by the security of digital signatures. The last issue is trivial, because the blockchain security ensures the announced policy is immutable.

Theorem 2. *The data confidentiality of our protocol holds, if the underlying public key encryption is semantically secure and the used zk-SNARK is of zero-knowledge. The anonymity of our protocol for both workers and requesters will be satisfied, if the underlying common-prefix-linkable anonymous authentication satisfies the anonymity defined in Definition 7, and the zk-SNARK is zero-knowledge. Conditioned on that the blockchain infrastructure we rely on can be modeled as an ideal public ledger, the underlying common-prefix-linkable anonymous authentication satisfies the unforgeability and the common-prefix-linkability, the zk-SNARK satisfies proof-of-knowledge and the digital signature in use is secure, our protocol satisfies: security against a malicious requester and security against malicious workers.*

3.5.4 Implementation of ZebraLancer Protocol

We implement the protocol of ZebraLancer atop Ethereum, and instantiate a series of typical image annotation tasks [133] with using it. Furthermore, we conduct experiments of these tasks in an Ethereum test net to evaluate the applicability.

System in a nutshell. As shown in Figure 3.4, the decentralized application (DApp) of our system is composed of an on-chain part and an off-chain part. The on-chain part consists of crowdsourcing task contracts and an interface contract of the registration authority (RA). The RA's contract simply posits the system's master

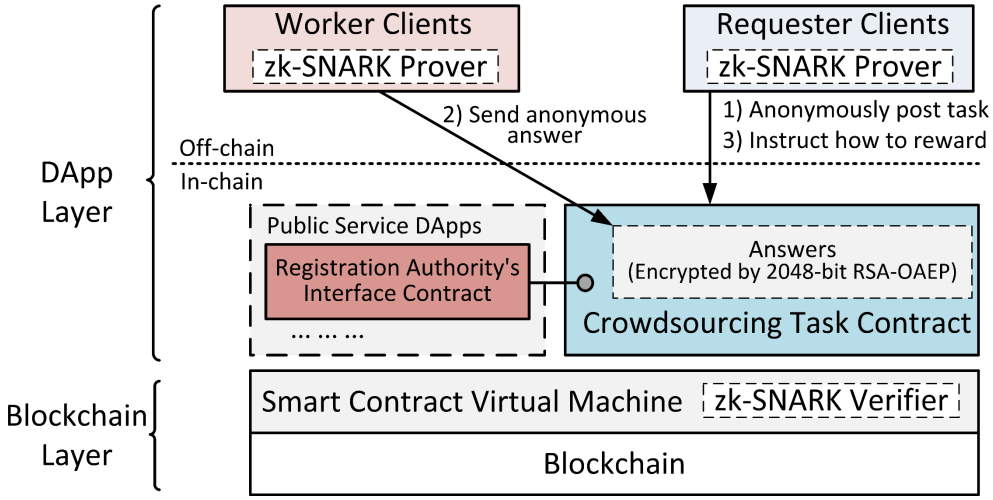

Figure 3.4 The system-level view of ZebraLancer.

public key as a common knowledge stored in the blockchain. The off-chain part consists of requester clients and worker clients. These clients can be blockchain clients wrapped with functionalities required by our system. Specifically, a client of requester should codify a specific task with a given incentive mechanism and announces it as a smart contract. Note that we, as the designers of the DApp, can provide contract templates to requesters for easier instantiation of incentive mechanisms, c.f. [55]. The clients further need an integrated zk-SNARK prover to produce the anonymous authenticating attestations; moreover, a requester client should also leverage SNARK prover to generate proofs attesting the correct execution of incentive policies.

This dissertation also compares ZebraLancer to existing crowdsourcing systems in Table 3.1. The security performance of our system overwhelms others, as it considers the most strict fairness of exchange, user anonymity and data confidentiality, under that condition of minimum trust. For example, our design realizes the fair exchange without leaking data to a third-party information arbiter. And ZebraLancer also guarantees the strongest user anonymity that cannot be broken by any third-party (even the registration authority), while the anonymity of other systems can be broken by a third-party authority (or few colluded third-parties).

Remark that the identities established via a registration service are required by all systems in Table 3.1.

Table 3.1 Comparison between our ZebraLancer and Existing Platforms

	Ours	MTurk [5]	Dynamo [128]	SPPEAR [64]	CrowdBC [92]
Prevention of false-reporting	√	×	×	◯	√
Prevention of free-riding	√	◯	◯	◯	√
Data confidentiality	√	×	×	×	×
User anonymity	√	×	◯	◯	×

Note: √ denotes a realized functionality without using any central trust except the established identities; ◯ denotes a (partially) realized function by relying on a central trust (other than the registration authority); × denotes an unrealized feature. Note that the data confidentiality is marked as ×, if any third-party other than the requesters can access the submitted data.

Implementation challenges. The main challenge of deploying smart contracts in general is that they can only support very light on-chain operations for both computing and storing.[10] Our protocol actually has taken this into consideration. In particular, our on-chain computation only consists SNARK verifications, while the heavy computation of SNARK proofs are all done off the blockchain. Even still, building an efficient privacy-preserving DApp compatible with existing blockchain platform such as Ethereum is not straightforward. For instance, in order to allow smart contracts to call a zk-SNARK verification library, a contract of this library should be thrown into a block, but this library is a general purpose tool that can be too complex to be executed in the smart contract runtime environment, e.g., Ethereum Virtual Machine (EVM). Alternatively, we modify the the runtime environment of smart contracts, so that an optimized zk-SNARK verification library [15] is embedded

[10] We remark the communication overhead is not a serious worry, because: (i) a blockchain network such as Ethereum does not require fully meshed connections, i.e., requesters and workers can only connect a constant number of Ethereum peers; (ii) if necessary, requesters and workers can even run on top of so-called light-weight nodes, which eventually allows them receive and send messages only related to crowdsourcing tasks; (iii) even if there is a trusted arbiter facilitating incentive mechanisms, the only saving in communication is just an instruction about how to reward answers (and its attestation).

in it as a primitive operation. Our modified Ethereum client is written in Java 1.8 with Spring framework, and is available at github.com/maxilbert/ethereumj.

We remark that Ethereum project recently integrated some new cryptographic primitives into EVM to enable SNARK verification as well [52], which ensures our DApp can essentially inherit all Ethereum users to maintain the blockchain infrastructure to govern the faithful execution of the smart contracts in our DApp.

Establishments of zk-SNARKs (off-line). As the feasibility of ZebraLancer highly depends on the tininess of SNARK proofs and the efficiency of SNARK verifications, it becomes critical to establish necessary zk-SNARKs off-line. As formally discussed before, the authentication scheme and nearly all incentive mechanisms can be stated as some well-defined deterministic constraint relationships. We first translate these mathematical statements into their corresponding boolean circuit satisfiability representations. Furthermore, we establish zk-SNARK for each boolean circuit, such that all required public parameters are generated. All the above steps are done off-line, as they are executed for only once when the system is launched. Note that the potential backdoors in these zk-SNARK public parameters could be further eliminated via an off-line protocol based on secure multi-party computation [20]. However, such an off-line setup is beyond the scope of showing our system feasibility.

An image annotation crowdsourcing task. To showcase the usability of our system, we implement a concrete crowdsourcing task of image annotation [133]. The task is to solicit labels for an image which can later be used to train a learning machine. The task requests n answers from n workers, and can be considered as a multi-choice problem. Majority voting is used to estimate the "truth". An answer is seen as "correct", if it equals to the "truth". The reward amount of a worker is τ/n if he answers correctly, otherwise, he receives nothing. In our terminology, the reward $R_i := \mathsf{R}(A_i; A_1, \ldots, A_n, \tau) = \tau/n$, if A_i equals the majority; otherwise,

$R_i = 0$. Following [133], we implement and deploy 5 contracts in the test net to collect 3 answers, 5 answers, 7 answers, 9 answers and 11 answers from anonymous-yet-accountable workers, respectively.

The smart contracts are written in Solidity, a high-level scripting language translatable to smart contracts of Ethereum. We also modify Solidity compiler, such that a programmer can write a contract involving zk-SNARK verifications at high-level. We instantiate the encryption to be RSA-OAEP-2048, the DApp-layer hash function to be SHA-256, and the DApp-layer digital signature to be RSA signature. Moreover, for zk-SNARK, we choose the construction of *libsnark* from [15]. We deploy a test network consisting of four PCs: three PCs are equipped with Intel Xeon E3-1220V2 CPU (PC-As), and the other one is equipped with Intel i7-4790 CPU (PC-B); all PCs have 16 GB main memory and have Ubuntu 14.04 LTS installed. In the test net, a PC-A and a PC-B play the role of miners, and the other two PC-As only validate blocks (i.e., full nodes that do not mine). One full node plays the role of the requester, and anonymously publishes crowdsourcing tasks to the blockchain; and the other full node mimics workers, and sends each anonymously authenticated answer from a different blockchain address. Miners are only responsible to maintain the test net and do not involve in tasks.

Table 3.2 Execution Time of zk-SNARK Verifications

Verification for	Operands Length			Time@	Time@
	Proof	Key	Inputs	PC-A	PC-B
Anonymous authentication	729B	1.2KB	1.5KB	10.9ms	6.2ms
Majority (3-Worker)	729B	16.0KB	3.4KB	15.5ms	9.1ms
Majority (5-Worker)	730B	21.6KB	4.7KB	16.3ms	9.8ms
Majority (7-Worker)	731B	27.3KB	6.0KB	17.0ms	10.3ms
Majority (9-Worker)	729B	32.9KB	7.3KB	17.5ms	12.1ms
Majority (11-Worker)	730B	38.6KB	8.6KB	17.9ms	13.1ms

Performance evaluation. As the main bottleneck is the on-chain computation of the smart contract, we first measure the time cost and the spatial cost of miners, regarding the executions of zk-SNARK verifications used in the above annotation tasks. These zk-SNARKs are established for common-prefix-linkable anonymous authentications and incentive mechanisms, respectively. The results of time cost are listed in Table 3.2. It is clear that zk-SNARK verifications in our system can be efficiently executed in respect of verification time. Moreover, our experiment results also reveal that the spatial cost of zk-SNARK verifications is constant and tiny at both types of PCs (exactly $17MB$ main memory). Also, the required on-chain storage for the task contracts is at the acceptable magnitude of kilobyte[11]. Therefore, the on-chain performance of the system can be clearly practical, regarding time and space.

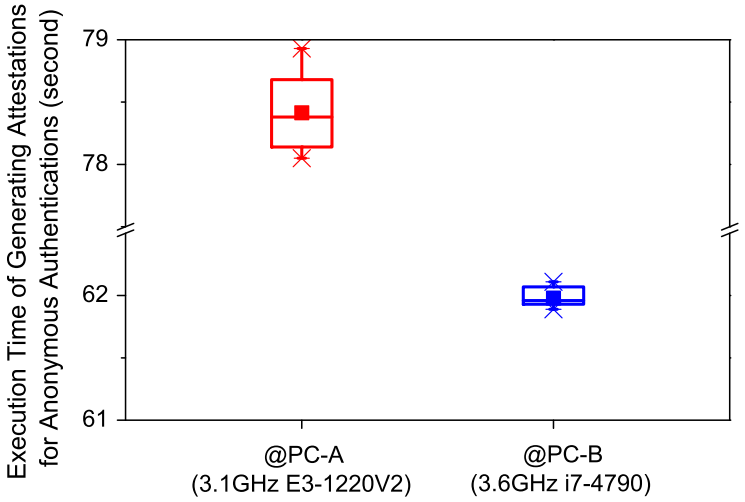

Figure 3.5 The time of generating common-prefix-linkable anonymous authentications in two PCs. The box plot is derived from 12 different experiments.

We also consider the cost of anonymity, if one uses the common-prefix-linkable anonymous authentication. We measure the running time of generating the

[11]Remark that the actual price of exchanging ETH fluctuates and is determined by the market. For instance, the average cost of each kilobyte storage could be as low as 0.05 UDS in Jan 2016, or could be as much as 15 USD in Jan 2018. Thus, to estimate the cost from an economic view is out of scope of the dissertation. It is worthwhile to further note that there are many economic alternatives to minimize the on-chain storage in the later implementations, e.g., to use off-chain storages [137, 16]. These optimizations are beyond the scope of this dissertation, in which we only focus on the technical feasibility instead.

authenticating attestations at PCs. As shown in Figure 3.5, our experiment results clarify that about 78 seconds are spent on generating an anonymous attestation with using PC-A (3.1 GHz CPU). In PC-B (3.6GHz CPU), the running time can be shortened to about 62 seconds. Those are not ideal, but acceptable by the anonymity-sensitive workers. We remark that our protocol can be trivially extended to support non-anonymous mode, in case that one gives up the anonymity privilege: s/he can generate a public-private key pair (for digital signatures), and then registers the public key at RA to receive a certificate bound to the public key; to authenticate, s/he can simply show the certified public key, the certificate, along with a message properly signed under the corresponding secret key, which essentially costs nearly nothing regarding the computational efficiency.

3.6 Summary

In this chapter, a generic blockchain-based framework is constructed, analyzed and implemented to enable the first *private and anonymous* decentralized data crowdsourcing system[12]. Without relying on any third-party information arbiter, the protocol can still guarantee the faithful execution for a class of incentive mechanisms, once they are announced as pre-specified policies in the blockchain. More importantly, confidential data and identities are also protected from the blockchain network, while the underlying blockchain is auditing the correct execution of crowdsourcing tasks. Specifically, this chapter can be summarized as follows.

First, a blockchain based protocol is proposed to realize decentralized crowdsourcing that satisfies: (i) fair exchange between data and rewards, i.e., a worker will be paid the correct amount according to the pre-defined policy of evaluating data, if he submits to the blockchain; (ii) data confidentiality, i.e., the submitted

[12]A couple of recent attempts on decentralized crowdsourcing [21, 139, 92] have been made, however, none of them address the fundamental privacy and anonymity issues. See section for details about their insufficiencies.

data is confidential to anyone other than the requester; and (iii) anonymity and accountability. Intuition behind the fairness and confidentiality is an *outsource-then-prove* methodology that: (i) the requester is enforced to deposit the budget of her incentive policy to a smart contract; (ii) submissions are encrypted under the requester's public key and will be collected by the blockchain; (iii) the evaluation of rewards is *outsourced* to the requester who then needs to send an instruction about how to reward workers. The instruction is ensured to follow the promised incentive policy, because the requester is also required to attach a valid succinct zero-knowledge proof. The worker anonymity of our protocol can ensure: (i) the public, including the requester and the implicit registration authority, is not able to tell whether a data comes from a given worker or not; (ii) if a worker joins multiple tasks announced via the blockchain, no one can link these tasks. More importantly, we also address the threat of multiple-submission exacerbated by anonymity misuse. In particular, if a worker anonymously submits more than the number allowed in *one* task, the scheme allows the blockchain to tell and drop these invalid submissions. Similarly, we achieve the requesters' accountable anonymity.

Second, to achieve the above goal of anonymity while preserving accountability, we propose, define and construct a new cryptographic primitive, called *common-prefix-linkable* anonymous authentication. In most of the time, a user can authenticate messages and attest the validity of identity without being linked. The only exception that anyone can link two authenticated messages is that they share the same prefix and are authenticated by one user. To utilize the new primitive in our protocol, a worker has to submit to a task via an anonymous authentication. The reference of the task will be unique, and should be the common-prefix, such that the special linkability will prevent multi-submission to a task. A requester can also use it to authenticate in each task she publishes, and convince workers that she cannot maliciously submit to intentionally downgrade their rewards. We remark that such a scheme can be

used to anonymously authenticate in a constant time independent to tasks. Such a primitive may be of independent interests for the special flavor of its accountability-yet-anonymity.

Finally, to showcase the feasibility of applying our protocol, we implement the system that we call ZebraLancer for a common image annotation task on top of Ethereum, a real-world blockchain infrastructure. Intensive experiments and performance evaluations are conducted in a Ethereum test net. Since current smart contracts support only primitive operations, tailoring such protocols compatible with existing blockchain platforms is non-trivial.

Since this work is the first attempt of decentralizing crowdsourcing system atop the real-world blockchain in a privacy-preserving way, the area remains largely unexplored. Here we name a few open questions, and we defer solutions to them in our future work. First, there are many incentive mechanisms using reputation systems, can we further extend our implementations to support those incentives? Second, as the current smart contract technology is at an infant stage and can only allow very tiny on-chain storage, can we further optimize our implementations with using off-chain storage [137, 16] or information oracle [158] to assist more large-scale tasks, e.g., to collect annotations for millions of images (i.e., the scale of ImageNet dataset)? Third, our anonymous protocol currently either relies on the underlying blockchain to support anonymous transaction, or requires workers/requesters use one-time blockchain account to submit data and receive reward. Can we design a (DApp-layer) protocol to solve the drawbacks? Last, our protocol relies on a trusted registration authority (RA) to establish identities. Although such a trusted RA could be a reasonable assumption (in view of real-world experiences), it is more tempting to develop an alternative methodology to remove the third-party RA without sacrificing securities. For example, can we adapt the successful invention

of proof-of-work to build up a crowdsourcing framework from literally "zero" trust without any established identity?

CHAPTER 4

ON PRACTICAL PRIVATE KNOWLEDGE SOLICITATION

4.1 Background

4.1.1 Motivation

To overcome blockchain's inherent limits, prior Chapter [98] proposes the general outsource-then-prove framework for *private* decentralized HITs. It enables the requester to prove the quality of answers that are encrypted to her, without revealing the actual answers. Such a proof becomes the crux to ensure privacy and simultaneously deters false-reporting and free-riding. In addition, the feasibility challenge sprouts up as the blockchain needs to verify the proof, which means the proof size and verification cost must be small enough to meet the limited on-chain resources. For above reasons, prior work relies on some *generic* zero-knowledge proof (zk-proof) framework that is succinct in proof size and efficient for verifying, in particular SNARK[1] [60, 15, 119] to reduce the on-chain verification cost. Nonetheless, generic zk-proofs such as SNARK inevitably inherit low performance for the convenience of achieving generality, causing that prior private decentralized HITs suffer from an unbearable off-chain proving cost and a still significant on-chain verifying expense.

Infeasible proving (off-chain). The proving of *generic* zk-proofs (e.g., SNARK) seems inherently complex, due to the burdensome NP-reduction for generality. In particular, prior study [99] reported 56 GB memory and 2 hours are needed to prove whether an encrypted answer coincides with the majority of all encrypted submissions at a very small scale, e.g., at most eleven answers. Such a performance prevents the previous protocol from being usable by any normal requesters using regular PCs.

[1]Remark that though the rise of Intel SGX becomes a seemingly enticing alternative of SNARK to go beyond many limits of blockchain by remote attestations [34], unfortunately, recent Foreshadow attacks [146] allow the adversary to forge "attestations" by stealing the attestation key hardcoded in any SGX Enclave, which seriously challenges the already heavy assumption of "trusted" hardware, and makes it even more illusive to trust SGX in practice.

Costly verification (on-chain). Existing blockchains (e.g., Ethereum) are feasible to verify only few types of *generic* zk-proofs such as SNARK, whose verification need to compute a dozen of expensive pairings over elliptic curve [60, 15, 119]. Even worse, the on-chain verification cost increases with the complexity of the proving statement. As a result, the on-chain verification becomes not only computationally costly, but also financially expensive. Currently in Ethereum, 12 pairings already spend ∼500k gas [53], and verifying a SNARK proof costs even more (about half US dollar).

Given the insufficiencies of the prior art, the next critical problem remains open:

How to design a practical private decentralized HITs protocol for crowdsourcing human knowledge?

4.1.2 Challenges

The major challenge of making *private* decentralized HITs practical is that the blockchain must learn the quality of some encrypted answers, namely, to obtain some properties of what a few ciphertext are encrypting. The state-of-the-art [98] proposed to reduce the problem to generic zk-proofs, by observing the requester can decrypt the answers, and then prove the quality of answers to the blockchain. But this generic approach incurs impractical expenses inherently, because of the underlying heavyweight NP-reduction for generality.

To conquer the above challenge, we follow a different path that deviates from generic zk-proofs to explore a concretely efficient solution. At the core of our *private* decentralized HITs protocol, we design a special-purpose non-interactive proof scheme to efficiently attest the quality of encrypted answers, which removes heavyweight general-purpose zk-proofs and then avoids the inefficiency of generality.

The ideas behind our efficient proving scheme are a variety of special-purpose optimizations to squeeze performance by removing needless generality, such that we reduce the problem of proving encrypted answers' quality from generic-purpose zk-

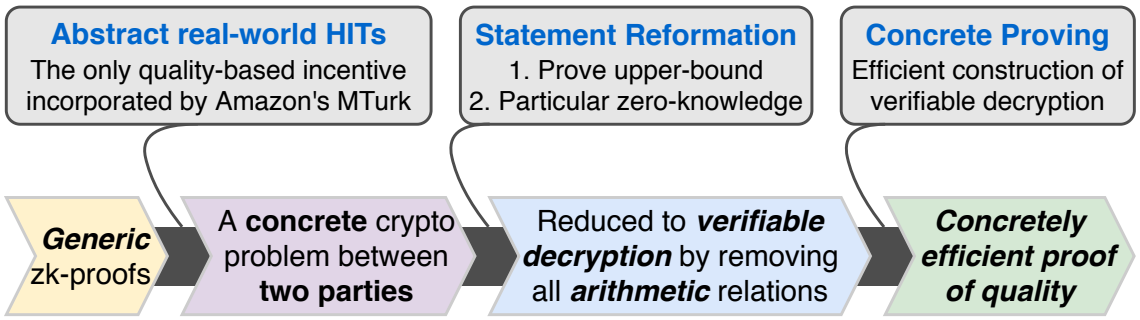

Figure 4.1 The path to realizing efficient proofs for encrypted answers' quality.

proof to particular verifiable decryption. As shown in Figure 4.1, our core ideas are highlighted as following.

Abstracting real-world HITs. The first step is to well formulate an incentive mechanism widely adopted by real-world HITs, namely, the *only* one incorporated by Amazon's MTurk [7]. Some golden standard challenges (i.e., questions with known answers) [61] are mixed with other questions, and the quality of a worker can be determined by her performance on the golden standards.[2] We rigorously define the problem of proving the quality of encrypted answers for the above incentive mechanism. Then, proving the quality of a worker is reducible to a well-defined two-party problem: where the verifier needs to output the performance of the worker on a set of golden standard questions, given only a set of ciphertext answering these golden standards challenges. Nevertheless, solving this two-party problem is still challenging, as it needs to compute the property of what a set of ciphertext are encrypting. The generic version of the issue falls into multi-input functional encryption [66, 19], which is well known for its hardness and has no (nearly) practical solution so far. We thus conduct the following optimizations to further reduce the problem.

[2]This concrete incentive turns to be powerful, say it can capture most HITs in Amazon's MTurk (c.f., the official tutorial [7]) and also adopted by the impactful ImageNet [136] to create large-scale deep learning benchmark.

Statement reformation. The main obstacle of removing the generic-purpose zk-proof framework is the arithmetic relations (i.e., some relationship unrepresentable in the algebraic domain). Observing that, we dedicatedly reform the statement of proving the quality of encrypted answers mainly in two ways to remove all arithmetic relations. First, we let the requester to prove the upper bound of each worker's quality instead of proving the exact number, which is a relaxation to the general cases, but does not abandon any utility, since this property is enough to prevent any corrupted requester from paying less than what a worker deserves in our context where the reward is an increasing function of quality. Second, we realize that given the system's public knowledge, a tiny and constant portion of each worker's answer (i.e., the part answering gold standards) is already leaked, since this little portion is simulatable by the public knowledge; thus we can explicitly reveal these "already-leaked" information. The above reformations allow us to reduce the problem of proving the quality of encrypted answers to standard verifiable encryption without sacrificing securities/utilities.

Concretely efficient proving scheme. Following the above optimizations, the problem eventually is reduced to verifiable encryption, which becomes representable in concrete algebraic relations. Along the way, we present a certain variant of verifiable decryption that is concretely tailored for the scenario of HITs where the plaintexts are short, and thus squeeze most performance out of it and boost private decentralized HITs practically.

4.2 Prior Art

Besides the existing private decentralized HITs [99, 98] discussed earlier, here we briefly review some pertinent generic cryptographic frameworks and discuss their insufficiencies in the concrete context of private decentralized crowdsourcing.

Privacy-preserving blockchain. A variety of studies [85, 161, 31] consider the general framework for privacy-preserving blockchain and smart contract. The approaches are powerful in the sense of their generality, yet are expensive for concrete use-cases in practice. For example, Hawk [85] leverages generic zk-proofs to keep blockchain private, but incurs expensive proving expenses. As such, it is unclear how to leverage these generic frameworks to design concretely efficient protocol for the special-purpose of crowdsourcing [98].

Fair MPC using blockchain. Decentralized crowdsourcing is a special-purpose fair MPC using blockchain. Kiayias, Zhou and Zikas [82] consider the generic version of fair MPC in the presence of blockchain, but it is unclear how to adopt their generic protocol in practice without expensively computational costs. Recently, increasing interests focus on special-purpose variants of fair MPC in aid of blockchain. For example, [17, 86, 40] consider poker games. But these special-purpose solutions are over-tuned for distinct scenarios and are unclear how to be used for private decentralized crowdsourcing.

Multi-input functional encryption. The core problem of private decentralized crowdsourcing is to let the blockchain learn the quality of encrypted answers, which is straightforwardly reducible to multi-input functional encryption (MIFE) [66]. But this generic problem is known for its hardness, and all existing solutions for it are far from being practical. Even worse, MIFE relies on indistinguishability obfuscation [66] or multi-linear maps [19] that are even unclear how to be instantiated from the standard cryptographic assumptions.

4.3 Problem Formalization

This section rigorously defines our security model, by giving the ideal functionality of Human Intelligent Tasks (HITs) that captures the security/utility requirements of the state-of-the-art HITs in reality [42, 135, 147, 126, 4, 61, 47, 120, 7, 133, 136, 144,

74, 103]. Our security modeling sets forth a clear security goal, that is: the HITs in the real world shall be as "secure" as the HITs in an admissible ideal world.

4.3.1 Reviewing Knowledge Crowdsourcing via HITs in Reality

Let us briefly review the HITs adopted in reality by Amazon's MTurk [42, 135, 147, 126, 4, 61, 47, 120, 7, 133, 136, 144, 74, 103], before presenting the abstraction of their ideal functionality.

There are two explicit roles in a HIT, i.e., the requester and some workers.[3] The *requester*, uniquely identified by $\mathcal{R}$, can post a task $\mathcal{T}$ to collect a certain amount of answers. In the task, $\mathcal{R}$ also promises a concrete reward policy. The *worker* with a unique identifier $\mathcal{W}_j$, submits his answer $\boldsymbol{a}_j$ to expect receive the reward.

A HIT consists of a sequence of questions denoted by $\mathcal{T} = (q_1, \cdots, q_N)$, where each q_i is a multiple choice question and N is the number of questions in the task. The answer of each question must lay in a particular $\mathsf{range} \subset \mathbb{N} \cup 0$ pre-specified when $\mathcal{T}$ is published. W.o.l.g., we can let all questions are binary (i.e., let $\mathsf{Range} = \{0, 1\}$). The above HIT design is based on batched choice questions, which follows real-world practices [42, 135, 147, 126, 4, 61, 47, 120, 7, 133, 136, 144, 74, 103] to remove ambiguity, thus letting workers precisely understand the task. For example, Fei-fei Li *et al.* [42, 127, 136] used the technique to create the deep learning benchmark ImageNet, and Andrew Ng *et al.* [135] suggested it for language annotations.

The quality of an answer is induced by a function $\mathsf{Quality}(\boldsymbol{a}_j; sp)$, where $\boldsymbol{a}_j = (a_{(1,j)}, \cdots, a_{(N,j)})$ is the answer submitted by worker $\mathcal{W}_j$, and sp is some secret

[3] There is an implicit registration authority (RA), who is required by real-world crowdsourcing platforms e.g., MTurk to prevent adversary forging a large number of identities (a.k.a. Sybil attackers). In practice, RAs can be instantiated by (i) the platform itself (e.g., MTurk), and (ii) the certificate authority who provides authentication service. Our solution can inherit these established RAs, and we therefore omits such the implicit RAs, with assuming all identities are granted. If the participants are interested in anonymity, anonymous-yet-accountable authentication scheme [98, 89] can be used; however, those are orthogonal techniques out scope of this paper.

parameters of requester. The output of $\mathsf{Quality}(\cdot)$ is denoted by χ_j, which is said to be the quality of worker $\mathcal{W}_j$.

The abstraction captures the quality-based incentive mechanism adopted by real-world HITs in Amazon's MTurk [133, 136, 7, 144]. For example, a task $\mathcal{T}$ consists of N questions, out of which M questions are golden-standard questions that are "secretly" mixed. The *quality* of a worker can be computed, due to her accuracy in the M golden-standard questions. Formally, in the qualify function $\mathsf{Quality}(\boldsymbol{a}_j; sp)$, the parameter $sp = (G, G_S)$, where $G \subsetneq [1, N]$ represents the randomly chosen indexes of the golden-standard questions, and $G_S = \{s_i | s_i \in \mathsf{range}\}_{i \in G}$ represents the known answers of the golden-standard questions.

Following the real-world practices [133, 136, 7, 144], the quality of an answer $\boldsymbol{a}_j = (a_{(1,j)}, \cdots, a_{(N,j)})$ is: $\mathsf{Quality}(\boldsymbol{a}_j, (G, G_S)) = \sum_{i \in G} [a_{(i,j)} \equiv s_i]$, where $[\cdot]$ is Iverson bracket to convert any logic proposition to 1 if the proposition is true and 0 otherwise. In short, we focus on the particularly useful and interesting mechanisms representable in this form.

4.3.2 Defining Security Goals: Decentralized HITs' Functionality

Now it is ready to present the security notion of HITs in the presence of cryptocurrency. We formalize the ideal functionality of HITs (denoted by $\mathcal{F}_{hit}$) in the $\mathcal{L}$-hybrid model as shown in Figure 4.2, where the blue text shows $\mathcal{F}_{hit}^{\mathcal{L}}$ is proceeding synchronously as the adversary can delay message deliveries up to next clock period [85, 82]; the brown text means that $\mathcal{F}_{hit}^{\mathcal{L}}$ has to proceed asynchronously as if the adversary can arbitrarily delay messages.. Intuitively, $\mathcal{F}_{hit}^{\mathcal{L}}$ abstracts a special-purpose multi-party secure computation, in which: (i) a requester recruits K workers to crowdsource some knowledge, and (ii) each worker gets a payment of $\ddot{\mathrm{B}}/K$ from the requester, if submitting an answer meeting the minimal quality standard Θ.

In greater detail, the ideal functionality $\mathcal{F}_{hit}$ of HITs immediately implies the following security properties:

- *Fairness*. Our ideal functionality captures a strong notion of fairness, that means: the worker get paid, if and only if s/he puts forth a qualified answer (instead of copying and pasting somewhere else). In greater detail, the requester specifies a sequence of N multi-choice questions, which are multi-choice questions having some options in range and contain $|G|$ gold-standard challenges.[4] For each worker, s/he has to (i) meet a pre-specified quality standard Θ and (ii) submit answers in the range of options, in order to receive the pre-defined payment $\ddot{\mathrm{B}}/K$.

- *Audibility of gold-standards*. The choice of golden standards is up to the requester, thus making a realistic worry that a malicious requester uses some bogus as the golden standard solutions. The ideal functionality aims to abstract the best prior art [103, 74] regarding this issue so far, that means the golden standards become public auditable once the HIT is done. This abstraction "simulates" the ad-hoc reputation systems maintained by the MTurk workers to grade the reputations of the MTurk requesters in reality [103, 74].

- *Confidentiality*. It means any worker cannot learn the advantage information during the course of protocol execution. Without the property, workers can copy and paste to free ride, indicating that privacy would be a minimal requirement to ensure the basic usefulness of decentralized HITs. Our ideal functionality naturally captures the property.

Adversary Model. We consider probabilistic polynomial-time adversary in the real world. It can corrupt the requester and/or some workers statically, before the real-world protocol begins. The uncorrupted parties are said to be honest. Following the standard blockchain model [85, 82], we also abstract the ability of the real-world adversary to control the communication (between the blockchain and honest parties) as: (i) it follows the synchrony assumption [57, 85], namely, we let there is a global clock [57, 85], and the adversary can delay any messages sent to the blockchain up to a-priori known time (w.l.o.g., up to the next clock); (ii) the adversary can manipulate

[4] We explicitly consider that $|G|$ and range are small constant in the HITs ideal functionality. Such modeling follows real-world practices [42, 135, 147, 126, 4, 61, 47, 120, 7, 133, 136, 144, 74, 103]. In particular, $|\mathsf{range}|$ is a small constant in practice, because it represents few options of each multi-choice question in HIT; and $|G|$ is also a small constant, as it represents few gold-standard challenges in a HIT task.

The ideal functionality of HIT $\mathcal{F}_{hit}^{\mathcal{L}}$

Given accesses to oracle $\mathcal{L}$, the functionality $\mathcal{F}_{hit}^{\mathcal{L}}$ interacts with a requester $\mathcal{R}$, a set of workers $\{\mathcal{W}_j\}$ and adversary $\mathcal{S}$.

Phase 1: Publish Task

- Upon receiving $(\mathsf{publish}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, G, G_S)$ from $\mathcal{R}$, leak $(\mathsf{publishing}, \mathcal{R}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, |G|, |G_S|)$ to $\mathcal{S}$, until the beginning of next clock period, proceed with the following delayed executions:
  - send $(\mathsf{freeze}, \mathcal{P}_i, \ddot{\mathbb{B}})$ to $\mathcal{L}$, if return $(\mathsf{frozen}, \mathcal{F}_{hit}^{\mathcal{L}}, \mathcal{P}_i, \ddot{\mathbb{B}})$:
    * store N, $\ddot{\mathbb{B}}$, K, Range, $\bar{\chi}$ and sp as internal states;
    * initialize $\mathsf{answers} \leftarrow \emptyset$, and goto next phase;

Phase 2: Collect Answers

- Upon receiving $(\mathsf{answer}, \boldsymbol{a}_j)$ from $\mathcal{W}_j$, leak the message $(\mathsf{answering}, \mathcal{W}_j, |\boldsymbol{a}_j|)$ to $\mathcal{S}$, till receiving $(\mathsf{approved})$ from $\mathcal{S}$, continue with the delayed executions as follows:
  - if $(\mathcal{W}_j, \cdot) \in \mathsf{answers}$, do nothing;
  - else, $\mathsf{answers} \leftarrow \mathsf{answers} \cup (\mathcal{W}_j, \boldsymbol{a}_j)$, send $\mathsf{answers}$ to $\mathcal{R}$, leak $(\mathcal{W}_j, |\boldsymbol{a}_j|)$ to $\mathcal{S}$, go to phase 3 if $|\mathsf{answers}| = K$.

Phase 3: Evaluate Answers

- Upon entering this phase, leak all received messages to $\mathcal{S}$, until the beginning of next clock period, proceed to run the following delayed executions for each $\mathcal{W}_j \in \{\mathcal{W}_j \mid (\mathcal{W}_j, \cdot) \in \mathsf{answers}\}$:
  - if receiving $(\mathsf{evaluate}, \mathcal{W}_j)$ from $\mathcal{R}$, proceed as:
    * check whether $\mathsf{Quality}(\boldsymbol{a}_j, (G, G_S)) \geq \Theta$, if that is the case, send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$, and leak $(\mathsf{evaluated}, \mathcal{W}_j, G, G_S)$ to all entities including $\mathcal{S}$;
  - if receiving $(\mathsf{outrange}, \mathcal{W}_j, i)$ from $\mathcal{R}$, proceed as:
    * if $a_{(i,j)} \notin \mathsf{range}$, leak $(\mathsf{outranged}, \mathcal{W}_j, a_{(i,j)})$ to all entities, otherwise send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$.
  - else, no message from $\mathcal{R}$ was received, proceed as:
    * if $\boldsymbol{a}_j \neq \perp$, send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$.

Figure 4.2 The (stateful) ideal functionality of coin-aided HIT $\mathcal{F}_{hit}^{\mathcal{L}}$.

the order of so-far-undelivered messages sent to the blockchain, which is known as the "rushing" adversary.

Expressivity of HITs' ideal functionality. The ideal functionality $\mathcal{F}_{hit}$ not only captures the elegant state-of-the-art of collecting image/language/video annotations [42, 127, 136, 135, 144, 147, 133] but also reflects the common scenario of crowdsourcing human knowledge. Consider the next example: Alice is running a small startup, and aims to provide a service to visualize the availabilities of street parkings. Unfortunately, at each moment, Alice only knows the availabilities of street parkings at quite few spots, since she cannot afford the cost of monitoring every corner around the city. The little a-priori knowledge of Alice is her "golden standards", and such information is too little to boost a useful service. She can crowdsource more street parking information from a few workers, with using her few golden standards to control the quality of solicited data.

4.4 Dragoon: Highly Efficient Private Decentralized Crowdsourcing

This section elaborates our practical protocol for decentralized HITs. We begin with an important building block for proving the quality of encrypted answers. Then we showcase the smart contract functionality $\mathscr{C}_{hit}$ that interacts with the workers and the requester. Later, the detailed protocol is given in the presence of $\mathscr{C}_{hit}$. We finally prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{hit}$ of HITs.

4.4.1 Proof of Quality of Encrypted Answer

The core building block of our novel decentralized protocol is to allow the requester *efficiently prove the quality of encrypted answers*. We formally define this concrete purpose to set forth the notion called proof of quality of encrypted answers (PoQoEA), and then present an efficient reduction from it to verifiable decryption (VPKE).

Defining PoQoEA. The problem we are addressing here is to prove that: an encrypted answer $\boldsymbol{c}_j$ can be decrypted to obtain some $\boldsymbol{a}_j$ s.t. the quality of $\boldsymbol{a}_j$ is χ, without leaking anything other than $\boldsymbol{c}_j$, χ and the parameters of quality function.

To capture the problem, the state-of-the-art [98, 85] adopts the standard notion of zk-proof in order to support generic quality measurements. Different from existing solutions, we particularly tailor the notion of zk-proof to obtain a fine-tuned notion of PoQoEA for the widely adopted quality function defined in §4.3. Namely, we consider $\mathsf{Quality}(\cdot\ ; G, G_s)$ where G is the index of gold-standards and $G_s = \{s_i\}_{i \in G}$ is the ground truth of golden standards, and aim to remove the unnecessary generality in the concrete setting. Precisely, given the quality function $\mathsf{Quality}(\cdot\ ; G, G_s)$ and any established public key encryption scheme $(\mathsf{Enc}_h, \mathsf{Dec}_k) \leftarrow \mathsf{KeyGen}(1^\lambda)$, we can define PoQoEA as a tuple of hereunder algorithms $(\mathsf{Prover}_k, \mathsf{Verifier}_h)$:

1. $\mathsf{Prover}_k(\mathbf{c}_j, \chi, G, G_s) \rightarrow \pi$. Given the encrypted answer $\mathbf{c}_j = (c_{1,j}, \ldots, c_{N,j})$, the quality χ, and the golden standards (G, G_s), it outputs a proof π attesting χ is the quality of $\mathbf{c}_j$; it also explicitly takes the secret decryption key k as input;

2. $\mathsf{Verifier}_h(\mathbf{c}_j, \chi, \pi, G, G_s) \rightarrow 0/1$. It outputs 0 (reject) or 1 (accept), according to whether π is a valid proof attesting χ is the actual quality of $\mathbf{c}_j$; the algorithm explicitly takes the public encryption key h as input;

   Moreover, PoQoEA shall satisfy the following properties:

- *Completeness*. PoQoEA is complete, if for any G, G_s, $\mathbf{c}_j$, χ and $(\mathsf{Enc}_h, \mathsf{Dec}_k)$ s.t. $\chi = \mathsf{Quality}(\mathsf{Dec}_k(\mathbf{c}_j); G, G_s)$, there is $\Pr[\mathsf{Verifier}_h(\mathbf{c}_j, \chi, \pi, G, G_s) = 1 \mid \pi \leftarrow \mathsf{Prover}_k(\mathbf{c}_j, \chi, G, G_s)] = 1$;

- *"Upper-bound" soundness*. PoQoEA is upper-bound sound, if for any G, G_s, $\mathbf{c}_j$, χ and $(\mathsf{Enc}_h, \mathsf{Dec}_k)$, for $\forall$ P.P.T. $\mathcal{A}$, there is $\Pr[\mathsf{Verifier}_h(\mathbf{c}_j, \chi, \pi', G, G_s) = 1 \wedge \chi < \mathsf{Quality}(\boldsymbol{a}_j; G, G_s) \wedge \mathbf{a}_j = \mathsf{Dec}_k(\mathbf{c}_j) \mid \pi' \leftarrow \mathcal{A}(G, G_s, \chi, \mathbf{c}_j, \lambda, \mathsf{Enc}_h, \mathsf{Dec}_k)] \leq negl(\lambda)$, where $negl(\lambda)$ is a negligible function in λ, which means it is computationally infeasible to produce a valid proof, if χ is not the upper bound of the quality of what $\mathbf{c}_j$ is encrypting;

- *"Special" zero-knowledge*. Conditioned on $|G|$ and the range of elements in G_s are small constants, for any G, G_s, $\mathbf{c}_j$, χ and $(\mathsf{Dec}_k, \mathsf{Enc}_h)$, $\exists$ a P.P.T. simulator $\mathcal{S}$ that can simulate the communication scripts of PoQoEA protocol on input only h, G, G_s, $\mathbf{c}_j$, and χ.

Rationale behind the finely-tuned abstraction. The notion of PoQoEA is defined to remove needless generality in the special case of HITs. Compared to the state-of-the-art notion [98], PoQoEA is more promising to be efficiently constructed, as it brings the following definitional advantages as follows.

First, we adopt *"upper-bound" soundness* to ensure that any probably corrupted requester cannot forge the upper bound of quality of each worker. Such the tuning stems from a basic fact that: the reward of a worker is an increasing function in quality, so the upper bound of the worker's quality at least reflects the well-deserved reward of the worker. As a result, any cheating requester has to pay at least as much as the honest requester.

Second, another major difference is the relaxed *special zero-knowledge*: PoQoEA is zero-knowledge, only if $|G|$ and range are small constants, so anything simulatable by the gold standards can be leaked. Nevertheless, the conditions are prevalent in the special context of HITs [42, 135, 147, 126, 4, 61, 47, 120, 7, 133, 136, 144, 74, 103]. Recall that G represents the few golden standard questions, and range means the few options of each question in HITs, indicating that both are small constants in reality.

In sum, even though PoQoEA is seemingly over-tuned, it essentially coincides with the generic zk-proof of the quality of encrypted answers in the context of HITs.

Construction and security analysis. Here is an efficiency-driven way to constructing PoQoEA for the quality function $\mathsf{Quality}(\boldsymbol{a}_j; G, G_s)$ that was defined in §4.3. We can reduce the problem to the standard notion of verifiable decryption. More precisely, given the established VPKE scheme $(\mathsf{Enc}_h, \mathsf{Dec}_k, \mathsf{Prove}_k, \mathsf{Verify}_h)$, PoQoEA can be constructed as illustrated in Figure 4.3.

Lemma 1. *Given any verifiable public key encryption* VPKE*, the algorithm in Figure 4.3 satisfies the definition of* PoQoEA *regarding the quality function defined in §4.3.*

Proof. (sketch) The completeness is immediate from the definition of quality function, the correctness of encryption, and the completeness of VPKE. To prove the upper-

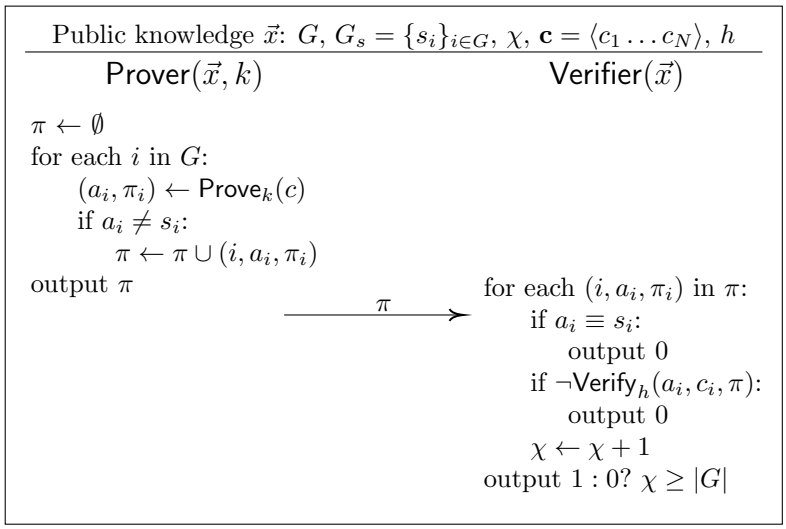

Figure 4.3 The construction of PoQoEA for the quality defined in §4.3.

bound soundness, we assume by contradiction to let an adversary break it, then the adversary can immediately break the soundness of VPKE. The special zero-knowledge is also clear: considering $|G|$ and the range of each a_i are constants, the permutation $\binom{|G|}{\chi}$ would be constant, indicating that there exists a P.P.T. simulator $\mathcal{S}$ invoking at most polynomial number of $\mathcal{S}_{\mathsf{VPKE}}$ (on input c_i, h, and guessed $a_i \in \mathsf{range} \setminus \{s_i\}$) to simulate all VPKE proofs [94], thus simulating the PoQoEA proof. □

4.4.2 HIT Contract and HIT Protocol

Now we are ready to present our concretely efficient decentralized protocol Π_{hit} for HIT. Our design centers around a smart contract $\mathscr{C}_{hit}^{\mathcal{L}}$, which is formally described in Figure 4.4. The contract $\mathscr{C}_{hit}^{\mathcal{L}}$ is the crux to take best advantage of the rather limited abilities of blockchain to make our protocol securely realize the ideal functionality $\mathcal{F}_{hit}^{\mathcal{L}}$. Thus given contract $\mathscr{C}_{hit}^{\mathcal{L}}$, our HITs protocol Π_{hit} can be defined among the requester, the worker and the contract, as formally illustrated in Figure 4.5. Informally, our HIT protocol Π_{hit} proceeds as follows.

Publishtask. The requester $\mathcal{R}$ announces her public key h, and publishes a task $\mathcal{T}$ of N multi-choice questions to crowdsource K answers for the task. Each question

HITs contract functionality $\mathscr{C}_{hit}^{\mathcal{L}}$

Given accesses to $\mathcal{L}$, $\mathscr{C}_{hit}$ interacts with $\mathcal{R}$, $\{\mathcal{W}_j\}$, and $\mathcal{A}$.

Phase 1: Publish Task

- Upon receiving $(\mathsf{publish}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, h, \mathsf{comm}_{gs})$ from $\mathcal{R}$, leak the message and $\mathcal{R}$ to $\mathcal{A}$, until the beginning of next clock, proceed with the delayed executions as follows:
  - send $(\mathsf{freeze}, \mathcal{P}_i, \ddot{\mathbb{B}})$ to $\mathcal{L}$, if returns $(\mathsf{frozen}, \mathcal{F}_{hit}^{\mathcal{L}}, \mathcal{P}_i, \ddot{\mathbb{B}})$:
    * store N, $\ddot{\mathbb{B}}$, K, range, Θ, h and comm_{gs}
    * initialize $\mathsf{answers} \leftarrow \emptyset$, $\mathsf{comms} \leftarrow \emptyset$
    * send $(\mathsf{published}, \mathcal{R}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, h, \mathsf{comm}_{gs})$ to all entities, and goto phase 2-a

Phase 2-a: Collect Answers (Commit phase)

- Upon receiving $(\mathsf{commit}, \mathsf{comm}_{\boldsymbol{c}_j})$ from $\mathcal{W}_j$, leak the message and $\mathcal{W}_j$ to $\mathcal{A}$, then proceed with the following delayed executions until the beginning of next clock, with consulting $\mathcal{A}$ to re-order all received commit messages:
  - for each received commit message (sent from $\mathcal{W}_j$):
    * if $(\mathcal{W}_j, \cdot) \notin \mathsf{comms}$ and $(\cdot, \mathsf{comm}_{\boldsymbol{c}_j}) \notin \mathsf{comms}$:
      · let $\mathsf{comms} \leftarrow \mathsf{comms} \cup (\mathcal{W}_j, \mathsf{comm}_{\boldsymbol{c}_j})$
      · if $|\mathsf{comms}| = K$, send $(\mathsf{committed}, \mathsf{comms})$ to all entities, and goto the reveal phase

Phase 2-b: Collect Answers (Reveal phase)

- Upon entering this phase, leak all received messages and their senders to $\mathcal{A}$, till the next clock period, proceed as:
  - for each $\mathcal{W}_j \in \{\mathcal{W}_j \mid (\mathcal{W}_j, \cdot) \in \mathsf{comms}\}$:
    * if receiving the message $(\mathsf{reveal}, \boldsymbol{c}_j, \mathsf{key}_j)$ from $\mathcal{W}_j$ s.t. $\mathsf{Open}(\mathsf{comm}_{\boldsymbol{c}_j}, \boldsymbol{c}_j, \mathsf{key}_j) = 1$:
      · $\mathsf{answers} \leftarrow \mathsf{answers} \cup (\mathcal{W}_j, \boldsymbol{c}_j)$
    * else $\mathsf{answers} \leftarrow \mathsf{answers} \cup (\mathcal{W}_j, \perp)$
  - send $(\mathsf{revealed}, \mathsf{answers})$ to all, and goto the next phase

Phase 3: Evaluate Answers

- Upon entering this phase, leak all received messages and their senders to $\mathcal{A}$, till the next clock period, proceed as:
  - if receiving $(\mathsf{golden}, G, G_s, \mathsf{key}_{gs})$ from $\mathcal{R}$ s.t. $\mathsf{Open}(\mathsf{comms}_{gs}, G||G_s, \mathsf{key}_{gs}) = 1$:
    * for each $\mathcal{W}_j \in \{\mathcal{W}_j \mid (\mathcal{W}_j, \cdot) \in \mathsf{answers}\}$:
      · if receiving $(\mathsf{outrange}, \mathcal{W}_j, i, a_{(i,j)}, \pi_i)$ from $\mathcal{R}$:
        send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$, if $a_{(i,j)} \in \mathsf{range}$ or $\mathsf{Verify}_h(a_{(i,j)}, c_{(i,j)}, \pi_i) = 0$
      · else if receiving $(\mathsf{evaluate}, \mathcal{W}_j, \chi_j, \pi)$ from $\mathcal{R}$:
        send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$, if $\chi_j \geq \Theta$ or $\mathsf{Verifier}_h(\mathbf{c}_j, \chi_j, \pi, G, G_s) = 0$
      · else if $\boldsymbol{c}_j \neq \perp$, send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$
  - otherwise, for each $\mathcal{W}_j \in \{\mathcal{W}_j \mid (\mathcal{W}_j, \cdot) \in \mathsf{answers}\}$, send $(\mathsf{pay}, \mathcal{W}_j, \ddot{\mathbb{B}}/K)$ to $\mathcal{L}$

Figure 4.4 The ideal functionality of the (stateful) HITs contract.

Protocol description of the HITs Π_{hit}

Π_{hit} is among the requester $\mathcal{R}$, the workers $\{\mathcal{W}_j\}$ and $\mathscr{C}_{hit}$

Phase 1: Publish Task

- Requester $\mathcal{R}$:
  - $(\mathsf{Enc}_h, \mathsf{Dec}_k) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - Upon receiving the parameters G, G_s, Θ, N, range, $\ddot{\mathbb{B}}$, K of a HIT to publish:
    - $\mathsf{key}_{sg} \xleftarrow{\$} \{0,1\}^\lambda$
    - $\mathsf{comm}_{gs} \leftarrow \mathsf{Commit}(G||G_s, \mathsf{key}_{sg})$
    - send $(\mathsf{publish}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, h, \mathsf{comm}_{gs})$ to $\mathscr{C}_{hit}$

Phase 2: Collect Answers

- Worker $\mathcal{W}_j$:
  - Upon receiving $(\mathsf{published}, \mathcal{R}, N, \ddot{\mathbb{B}}, K, \mathsf{range}, \Theta, h, \mathsf{comm}_{gs})$ from $\mathscr{C}_{hit}$:
    - get the answer $\boldsymbol{a}_j = (a_{(1,j)}, \cdots, a_{(N,j)})$
    - $\boldsymbol{c}_j \leftarrow (\mathsf{Enc}_h(a_{(1,j)}), \cdots, \mathsf{Enc}_h(a_{(1,N)}))$
    - $\mathsf{comm}_{\boldsymbol{c}_j} \leftarrow \mathsf{Commit}(\boldsymbol{c}_j, \mathsf{key}_j)$, where $\mathsf{key}_j \xleftarrow{\$} \{0,1\}^\lambda$
    - send $(\mathsf{commit}, \mathsf{comm}_{\boldsymbol{c}_j})$ to $\mathscr{C}_{hit}$
  - Upon receiving $(\mathsf{committed}, \mathsf{comms})$ from $\mathscr{C}_{hit}$:
    - if $(\mathcal{W}_j, \cdot) \in \mathsf{comms}$, send $(\mathsf{reveal}, \boldsymbol{c}_j, \mathsf{key}_j)$ to $\mathscr{C}_{hit}$

Phase 3: Evaluate Answers

- Requester $\mathcal{R}$:
  - Upon receiving $(\mathsf{revealed}, \mathsf{answers})$ from $\mathscr{C}_{hit}$:
    - send $(\mathsf{golden}, G, G_s, \mathsf{key}_{gs})$ to $\mathcal{R}$
    - for each $(\mathcal{W}_j, \boldsymbol{c}_j) \in \mathsf{answers}$:
      - decrypt each item in $\boldsymbol{c}_j$ to get $\boldsymbol{a}_j = (a_{(1,j)}, \cdots, a_{(N,j)})$
      - if $\exists a_{(i,j)} \in \boldsymbol{a}_j$ s.t. $a_{(i,j)} \notin \mathsf{range}$:
        - $(a_{(i,j)}, \pi_i) \leftarrow \mathsf{Prove}_k(c_{(i,j)})$
        - send $(\mathsf{outrange}, \mathcal{W}_j, i, a_{(i,j)}, \pi_i)$ to $\mathscr{C}_{hit}$
      - else if $\chi_j = \mathsf{Quality}(\mathsf{Dec}(\boldsymbol{c}_j, sk_{\mathcal{R}}); G, G_s) < \Theta$:
        - $\pi \leftarrow \mathsf{Prover}_k(\mathbf{c}_j, \chi_j, G, G_s)$
        - send $(\mathsf{evaluate}, \mathcal{W}_j, \chi_j, \pi)$ to $\mathscr{C}_{hit}$

Figure 4.5 The formal description of the decentralized HITs protocol Π_{hit}.

in $\mathcal{T}$ is specified to have some options in range. The task mixes some golden standard questions, whose indexes G and ground truth G_s are committed to comm_{gs}. Also, $\mathcal{R}$ places $\ddot{\mathbb{B}}$ as deposit to cover her budget, which promises that a worker would get a reward of $\ddot{\mathbb{B}}/K$, if submitting an answer beyond a specified quality standard Θ.

Commitanswer. Once the task is published, the workers can commit their answers (encrypted to the requester) in the task. To prevent against copy-and-paste attacks, duplicated commitments are rejected. The contract moves to the next phase, once K distinct workers commit.

Revealanswers. After K workers commit their answers, these workers can start to reveal their answers in form of ciphertexts encrypted to the requester. Note that the submissions of answers explicitly contain two subphases, namely, committing and revealing, which is the crux to prevent the network adversary from taking advantages by adversarially scheduling the order of submissions.

Evaluateanswers. Eventually, the requester is supposed to instruct the blockchain to correctly pay the encrypted answers for the critical fairness. To this end, the protocol leverages our novel notion of PoQoEA. The requester can efficiently prove to the contract to reject a certain answer, if the worker does not meet the pre-specified quality standard Θ. If an answer is out of the specified range, the requester is allowed to use verifiable encryption VPKE to reveal that to reject payment.

4.4.3 Instantiating Cryptographic Building Blocks

For sake of completeness, here give the constructions of cryptographic building blocks that are called by Figures 4.4 and 4.5. Let $\mathcal{G} = \langle g \rangle$ be a cyclic group of prime order p, where g is a random generator of $\mathcal{G}$.

Short range verifiable decryption is based on exponential ElGamal. The private key $k \xleftarrow{\$} \mathbb{Z}_p$, the public key $h = g^k$, the encryption $\mathsf{Enc}_h(m) = (c_1, c_2) = (g^r, g^m h^r)$, and the decryption $\mathsf{Dec}_k((c_1, c_2)) = \log(c_2/c_1^k)$ where log is to brute-force the short

plaintext range to obtain m; if decryption fails to output $m \in \mathsf{range}$, then c_2/c_1^k is returned. In addition, to efficiently augment the above $(\mathsf{Enc}_h, \mathsf{Dec}_k)$ to be *verifiable*, we adopt a variant of Schnorr protocol [130] (for Diffie-Hellman tuples) with Fiat-Shamir transform in the random oracle model.

In detail, the Prove_k algorithm and the Verify_h algorithm can be described as follows:

- $\mathsf{Prove}_k((c_1, c_2))$. Run $\mathsf{Dec}_k((c_1, c_2))$ to obtain $m \in \mathsf{range}$ (or g^m if $m \notin \mathsf{range}$). Let $x \xleftarrow{\$} \{0,1\}^\lambda$. Compute $A = c_1^x$, $B = g^x$, $C = \mathcal{H}(A||B||g||h||c_1||c_2||g^m)$, $Z = x + kC$, and $\pi = (A, B, Z)$. If $m \in \mathsf{range}$, output (m, π); else, output (g^m, π).

- $\mathsf{Verify}_h(M, (c_1, c_2), \pi)$. Parse $\pi = (A, B, Z)$. If $M \in \mathsf{range}$, compute $C' = \mathcal{H}(A||B||g||h||c_1||c_2||g^M)$, and then verify $g^{M \cdot C'} \cdot c_1^Z \equiv A \cdot c_2^{C'}$ and $g^Z \equiv B \cdot h^{C'}$, output 1 if the verification passes and 0 otherwise; else if $M \in \mathcal{G}$, compute $C' = \mathcal{H}(A||B||g||h||c_1||c_2||M)$ and verify $M^{C'} \cdot c_1^Z \equiv A \cdot c_2^{C'} \wedge g^Z \equiv B \cdot h^{C'}$, output 1 iff the verification passes and 0 otherwise.

Proof of quality of encrypted answer is built by invoking the above VPKE in a black-box manner, due to the reduction from PoQoEA to VPKE in subsection 4.4.1.

Commitment scheme is instantiated according to the well-known efficient folklore construction in the random oracle model [25, 48]: (i) $\mathsf{Commit}(\mathsf{msg}, \mathsf{key}) = \mathcal{H}(\mathsf{msg}||\mathsf{key})$; (ii) $\mathsf{Open}(\mathsf{comm}, \mathsf{msg}', \mathsf{key}') = [\mathcal{H}(\mathsf{msg}'||\mathsf{key}') \equiv \mathsf{comm}]$, where $[\cdot]$ is Iverson bracket from a proposition to 1 (true) or 0 (false).

4.4.4 Security Analysis

Theorem 3. *Conditioned on hardness of DDH problem and static corruptions, the stand-alone instance of* Π_{hit} *securely realizes* $\mathcal{F}_{hit}$ *in* $\mathscr{C}_{hit}^{\mathcal{L}}$*-hybrid, random oracle model.*

Proof. (sketch) Let $\mathbb{C}$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$, and let $\mathbb{H}$ denote the set of rest honest parties. For any P.P.T. adversary $\mathcal{A}$ in the real world, we can sketch a P.P.T. simulator $\mathcal{S}$ in the ideal world to interact with the ideal functionality $\mathcal{F}_{hit}$ and corrupted parties, such that $\mathcal{S}$ can emulate the actions of honest parties and the contract $\mathscr{C}_{hit}$. Detailedly, $\mathcal{S}$ proceeds as follows.

PublishTask *(Phase 1).* If $\mathcal{R} \in \mathbb{C}$, considering that the corrupted $\mathcal{R}$ sends the publish message to $\mathscr{C}_{hit}$ in the real world, $\mathcal{S}$ can trivially simulate that with interacting with $\mathcal{F}_{hit}$. If $\mathcal{R} \in \mathbb{H}$, when the honest $\mathcal{R}$ sends the publish message to $\mathcal{F}_{hit}$, $\mathcal{S}$ is informed and thus allows $\mathcal{S}$ to simulate the real-world scripts of *publish task*.

CollectAnswers *(Phase 2).* In the real world, the P.P.T. adversary $\mathcal{A}$ has the following abilities: (i) she can corrupt a set of parties $\mathbb{C}$ up to including the requester and a set of the workers, and (ii) she has to be consulted to reorder the so-far-undelivered messages sent to $\mathscr{C}_{hit}$ (till the next clock). The basic strategy to emulate $\mathcal{A}$ is that: $\mathcal{S}$ invokes the adversary $\mathcal{A}$ to obtain how $\mathcal{A}$ is re-ordering the commit messages (sent from workers), let $\mathbb{W}$ to represent the set of workers whose commit messages are scheduled as the first K to deliver; then $\mathcal{S}$ delays all answer messages that are not sent from the workers in $\mathbb{W}$. Then, $\mathcal{S}$ internally simulates the ciphertexts sent via reveal messages to open commitments. If $\mathcal{R} \in \mathbb{H}$, the ciphertexts can be simulated as they are indistinguishable from the uniform distribution over the ciphertext space; if $\mathcal{R} \in \mathbb{C}$, $\mathcal{S}$ is informed about all answer submissions sent from the workers, thus can internally simulate the submissions of the workers in the real world. Moreover, if $\mathcal{A}$ corrupts a worker whose commit message is scheduled in the first K to deliver but does not send any the reveal message to open the commitment, the simulator $\mathcal{S}$ can simulate that since it can let the corrupted worker to send an answer message containing $\perp$ with $\mathcal{F}_{hit}$. In addition, it is trivial to see $\mathcal{S}$ can internally simulate the parties as well as $\mathscr{C}_{hit}$, when the adversary $\mathcal{A}$ corrupts a worker to submit duplicated commitment.

EvaluateAnswers *(Phase 3).* The simulation becomes clear, if considering the security requirements of commitment scheme, VPKE, and PoQoEA. If the requester $\mathcal{R} \in \mathbb{C}$, the simulator $\mathcal{S}$ invokes $\mathcal{A}$ to obtain all outrange and/or evaluate messages sent to $\mathscr{C}_{hit}$, and then simulates the interactions. If the requester $\mathcal{R} \in \mathbb{H}$, whenever $\mathcal{R}$

sends outrange and/or evaluate messages to $\mathcal{F}_{hit}$, $\mathcal{S}$ is informed and hence is allowed to simulate the interactions between $\mathscr{C}_{hit}$ and $\mathcal{R}$ in the real world.

4.4.5 Implementation and Evaluation

To demonstrate the feasibility of our protocol, we implement it to build Dragoon, and then use the system to launch a typical image annotation task for ImangeNet [136, 127] atop Ethereum.

System overview. Dragoon consists of an on-chain part and an off-chain part: the on-chain smart contract is deployed in Ethereum ropsten network; the requester client and worker clients are implemented in Python 3.6. The off-chain clients are installed in a PC that uses Ubuntu 14.04 LTS and equips Intel Xeon E3-1220V2 CPU and 16 GB main memory. We demonstrate our system through an ImageNet task [136, 127], which is specified as: each task is made of 106 binary questions, 100 out of which are non-gold-standard questions, while the remaining 6 questions are requester's gold-standard challenges; 4 workers are allowed to participate; if a worker cannot correctly answer at least four golden standard questions, his submission will be rejected without being paid, otherwise he deserves to get the payment. The hash function is instantiated by keccak256. We choose the cyclic group $\mathcal{G}$ by using the $\mathcal{G}_1$ subgroup of BN-128 elliptic curve, over which all concrete public key primitives are instantiated. The code of our prototype is available at `https://github.com/njit-bc/dragoon`. An experiment instance is atop Ethereum ropsten network.

Implementation details. Many non-trivial on- and off-chain optimizations are particularly made for practicability.

The requester end warps: (i) an Ethereum node to interact with the blockchain, e.g., publish task, download workers' submissions, etc; (ii) the prover of verifiable encryption to generate necessary proofs to instruct the contract to reward workers; (iii) a Swarm API to publish the detailed questions of each crowdsourcing task. Swarm

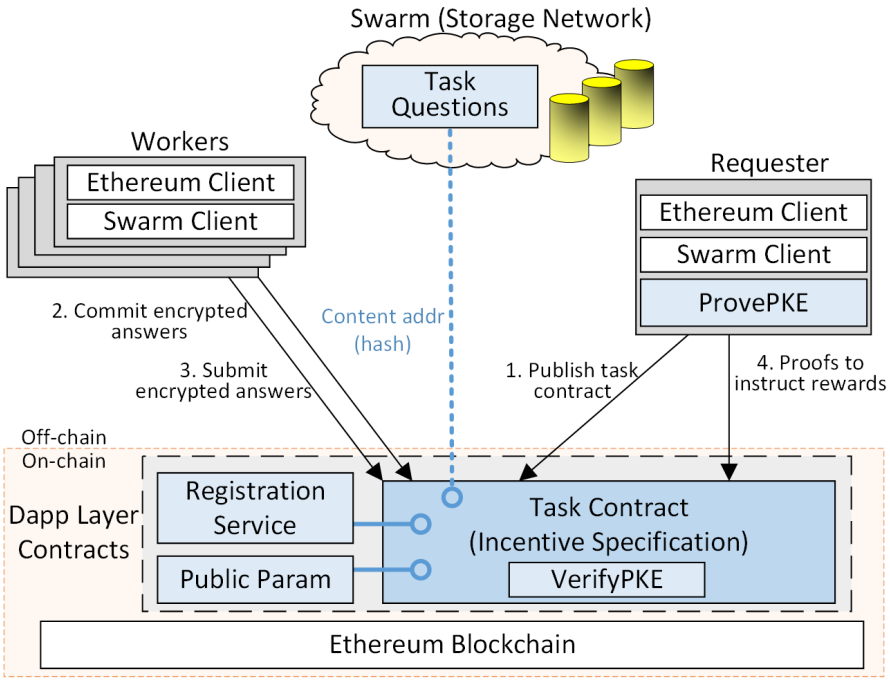

Figure 4.6 The schematic diagram of Dragoon at a high-level.

[137] is an off-chain storage network, where the questions of HIT is stored; in addition, to ensure integrity of HIT questions, the digest of the questions is committed in the contract, which significantly reduces on-chain cost, without violating securities. The worker client wraps Ethereum to interact with the blockchain to read task and submit answers, and also incorporates Swarm client to allow download task questions.

We cautiously perform a few non-trivial system-level optimizations to lighten the task contract: (i) we implement all public key schemes over $\mathcal{G}_1$ subgroup of BN-128 [12], since we can use some precompiled contracts in Ethereum to do algebraic operations there cheaply [53]; (ii) it is expensive to store ciphertexts in the contract as internal variables, while we make the contract store their 256-bit hashes instead and let the actual ciphertexts included in the chain as emitted event logs [152].

Evaluations. We conduct intensive experiments to measure the concrete performance, and discuss the system feasibilities from the on-chain side and the off-chain side.

Off-chain costs. First, Dragoon enables the requester to manage only one private-public key pair throughout all her tasks, because all protocol scripts are simulatable without secret key and therefore leak nothing relevant. More importantly, the off-chain cost of proving relevant cryptographic proofs is significantly reduced by removing unnecessary generality.

Table 4.1 Off-Chain Proving Cost of VPKE and PoQoEA

	Statement to Prove	Time	Peak Memory
Ours	VPKE	3 ms	53 MB
	PoQoEA	10 ms	53 MB
Generic ZKP*	VPKE	37 s	3.9 GB
	PoQoEA	112 s	10.3 GB

* Through our evaluations, generic zk-proofs are instantiated by zk-SNARK, which is the only generic zk-proof feasibly supported by existing blockchains to our knowledge.

Table 4.1 clarifies the requester suffers from hindersome off-chain burden of generating generic zk-proofs. The concrete construction removes the bottleneck of proving in generic zk-proof. First, the requester can generate a proof to reject a worker's submission within only a few milliseconds, which costs nearly 2 minutes if using generic zk-proof. Second, the concretely efficient constructions also save in memory usage. For example, by generic zk-proof, rejecting a submission requires a peak memory usage of 10 GB, which is reduced to only 53 MB by concrete constructions.

Table 4.2 On-chain Verification Cost of VPKE and PoQoEA

	Statement to Verify	Verifying Time
Ours†	VPKE	1 ms
	PoQoEA	2 ms
Generic ZKP‡	VPKE	11 ms
	PoQoEA	17 ms

† The implementation of BN-128 curve is from `https://github.com/scipr-lab/libff`.
‡ The evaluations for generic ZKP (SNARK) are performed due to constructions from 2048-bit RSA-OAEP over $\mathbb{Z}_{pq}$ instead of ElGamal over the $\mathcal{G}_1$ subgroup of BN-128.

On-chain costs. We measure the critical on-chain performance from many angles including the cost of verifying zk-proofs and the on-chain gas usage of the whole protocol. First, we compare the verifying cost of concrete and generic constructions for VPKE and PoQoEA (six golden standards) in Table 4.2. The concrete proofs are faster, even compared to the generic zk-proof (SNARK) known for efficient verification. Moreover, the overall handling fee of running a concrete ImageNet instance is summarized in Table 4.3. To estimate the cost of on-chain usages, we apply a gas price at 1.5×10^{-9} Ether per gas, and an Ether price at 115 USD per Ether, which are the safe-low price of gas [51] and the market price of Ether on March/17th/2020, respectively. Under the above exchange rate, the on-chain handling fee paid by each worker is about \$0.48, which is used to submit an answer. In addition, thanks to the efficient verification of PoQoEA, the requester can spend *few cents* to reject each low-quality answer. The overall on-chain handling cost of the entire HIT is about two US dollars. In contrast, when MTurk facilitates the same ImageNet task, it charges a handling fee at least \$4 currently [131, 6].

Table 4.3 On-Chain Overall Handling Fees of the Concrete ImageNet Task

Handling fee of	Gas Usage	In USD
Publish task (by requester)	∼1293 k	\$0.22
Submit answers (by worker)	∼2830 k	\$0.48
Verify PoQoEA to reject an answer	∼180 k	\$0.03
Overall (best-case: reject no submission)	∼12164 k	\$2.09
Overall (worst-case: reject all submissions)	∼12877 k	\$2.22

To summarize, Dragoon is practical. Our experiment even reveals that Dragoon's on-chain handling cost can be economically cheaper than the the handling fee charged by third-party platforms such as MTurk. In addition, Dragoon is compatible with many alternative chains (e.g., Cardano [30]) other than Ethereum, as long as the blockchains are using Ethereum Virtual Machine (EVM) as the running environment

of smart contracts. As a consequence, our system can be deployed in these alternative chains to further reduce the handling cost.

4.5 Summary

This Chapter answers the unresolved problem of the earlier Chapter, and presents a *practical* private decentralized HITs protocol for the major tasks of crowdsourcing human knowledge. In sum, its core technical contributions are three-fold.

First, to achieve practical private decentralized HITs, we explore various non-trivial optimizations to avoid the cumbersome generic-purpose zk-proof framework, and reduce the protocol to the specific verifiable decryption. As such, we attain concrete improvements by orders of magnitude, regarding both the proving and verification. (i) For proving, our approach is *two orders of magnitude* better than generic zk-proof [5]; For the same HIT, the proving in our protocol costs 50 MB memory and 10 msec, while the generic proof costs 10 GB and 2 min. (ii) For verifying, our result improves upon the generic solution by *nearly an order of magnitude*. The on-chain cost of verifying a proof for the quality of an answer to 106 batched binary questions is reduced to ∼180k gas in Ethereum (much smaller than verifying SNARK proofs) and typically few US cents.

Second, we further implement our protocol to instantiate a *practical* private decentralized crowdsourcing system Dragoon, the handling cost of which could be even less than the existing centralized platforms such as MTurk. Dragoon is launched atop Ethereum to conduct a typical HIT adopted by ImageNet [136] to solicit large-scale image annotations. To handle the task, Dragoon attains an on-chain (handling) cost ∼$2 US dollars at the time of writing. In comparison, for the same task, the handling fee of MTurk is at least $4 currently [6, 131]. Our result provides an insight that the on-chain *handling fee* (characterizing the users' financial expense) in the

[5]Generic zk-proof refers zk-SNARK in our context, since the only generic zk-proof that can be feasibly supported by existing blockchains is zk-SNARK.

decentralized setting can approximate or even less than the handling fee charged by centralized platforms. This indicates the *de facto* users can financially benefit from decentralization, though it is not contradictory to the common belief [132] that decentralization is more expensive w.r.t. the overall computational cost of the system.

Along the way, we firstly formulate the ideal functionality of decentralized HITs. The rigorous security model clearly defines what a HIT shall be and allows us to use the simulation-based paradigm to prove security against subtle adversaries in the blockchain. In contrast, existing decentralized HITs [98, 99] have quite different property-based definitions on "securities", which at least makes the lack of well-defined benchmark to compare them. Even worse, many of them are "flawed", as failing to capture all respects of the subtle adversary in the blockchain; for example, they allow the corrupted requester to reap data without paying, if being given the standard ability of adversarially re-ordering message deliveries, while our approach precisely defines the security requirement against this subtle attack.

CHAPTER 5

ON CROWDSOURCING FOR MACHINE LEARNING TASKS

5.1 Background

It has been a common practice for small companies and individual citizens to crowdsource their machine learning (ML) tasks to experts [122], as these small parities want to understand their own data but no clue on how to do that. Such an increasing demand makes it enticing to establish a marketplace of crowdsourcing ML tasks. In such a marketplace, the requesters can be small startups or even individuals, and they will publish their ML tasks with promising some payments; the workers can be resourceful experts, and can resolve the published ML tasks to earn the well-deserved rewards. But there lacks such a trusted marketplace currently, and the crowdsourcing of ML tasks is usually instantiated in the form of an open challenge launched by the requester herself. Usually, the requesters have to be trusted to 1) maintain their own challenges at web servers, and 2) pay promised rewards to the experts whoever solve the challenges.

5.1.1 Motivation

But this challenge-based framework suffers from a couple of major issues [50]: first, the ML challenges could be too expensive to instantiate and maintain by individual citizens, as they have to design their own ML challenges and run their own web servers to maintain them; second, the expert workers have to rely on that the requesters are fully trusted, as the dishonest requesters can always lie and reject to pay workers, which could be a rather serious concern, if the requesters are small business owners or even individuals. The above problems are consequences of the lack of trusted third-party in the particular use-case of crowdsourced ML, and it become natural to

consider the global computer , i.e., the blockchain, as the existing infrastructure to empower the crowdsourcing of ML tasks.

This research therefore will focus on a broad variety of machine learning tasks such as the training/validation of ML models, and proposes a simple and novel incentive based approach to realize the crowdsourcing of such ML tasks.

5.1.2 Challenges

The smart contract in (permissionless) blockchain is still at its infant stage, and suffers from several inherent limitations that hinder its wider adoption in crowdsourcing of ML tasks. Among several fundamental challenges, one that we are particularly interested in is that currently smart contract can only support very light computation: the complexity of computation tasks of each smart contract is usually strictly bounded. The reason lies in the fact that: during the blockchain mining procedure (the new block generation), when some output of a smart contract is expected to be recorded (that might also affect the validity of future blocks), honest miners are required to execute the program in order to validate the correctness of the outcome. If such a program is computationally intensive, crafty adversarial nodes may simply skip such verification step (or ignore putting the output at all), and go ahead to propose new blocks. Doing this gives the adversarial nodes substantial advantage of winning the chance for proposing new blocks, as honest nodes would not be able to propose any block until the execution of the smart contract finishes. Such an undesirable feature was known as verifier's dilemma [102].

There is another fundamental challenge for smart contracts that they cannot support randomized computations: since randomization lets even honest nodes have inconsistent outputs for the same program with the same input, it obviously prevents the central goal of the blockchain to ensure the consensus among all honest nodes.

Those limitations of smart contracts seriously hinder the applicability to broader interesting scenarios, especially in the settings that involve the execution of machine learning programs. Since those machine learning tasks are often computational costly and randomized (e.g., deep learning using stochastic gradient descent algorithm [88]), it becomes elusive how we can enable the machine learning tasks codified to be auto-executed in decentralized applications. The severe tension between the demands of applications and the restrictions of smart contract is widely acknowledged.

Observing that the security of the consensus protocol restricts only the *on-chain* computations, thus naturally we could turn to the area of verifiable computation for mitigation. In particularly, one may employ advanced cryptographic tools for verifiable computation such as SNARK [85, 98]. The full nodes could simply "outsource" the execution of the smart contracts to some service providers, and ask them further attach a succinct proof to the output. The full nodes now only verify the correctness of the proof during consensus. Unfortunately, although the nice feature of SNARK enables a cheap verification which reduces the on-chain cost (few elliptic curve pairings essentially), the computation and memory cost of generating the SNARK proofs in general for complex programs are still astronomically large, which makes it in fact infeasible in the scenarios of our interests as the machine programs are complex. Another line of researches explored the attestation capability of recently emerged trusted hardwares such as Intel's SGX to carry such outsourcing [158]. The basic idea is to load the program into the protected memory (called trusted enclave), execute the program and sign the output using the secret key hardcoded in the chip. However, the size of the enclave is pretty small, and therefore considerable performance penalties can be brought during attesting memory intensive tasks such as large-scale deep learning and random forest etc. More seriously, a recent attack on SGX could actually extract the attestation key completely [146], making the status of secure hardware is currently unclear.

5.1.3 Problem Formulation

In this subsection, the problem will be presented more precisely along with the security requirement, in the game-theory model.

The overview of system. As briefly illustrated in Figure 3.1, there are three roles in the system. The *blockchain* network mimics a trusted third-party that is also computationally-restricted. A blockchain user, called *requester*, can request the blockchain to execute some small programs called smart contracts. The chain will internally execute the programs, and "deliver" the computing results to all blockchain users, which is fully trustful. But, the requester cannot expect the blockchain run a computationally-intensive smart contract, as heavy smart contracts are hindered by the intrinsic limit of the blockchain [102]. Alternatively, the blockchain/requester would like to outsource the computations to some external off-chain service providers, called *workers*.

Remark that we will consider the outsourced program has a large output space, and the correct output is unpredictable, namely, for each given input, one cannot guess the output without computing the program with that input. Note that for most machine learning tasks (e.g., the training of DNN classifier for large-scale dataset), such properties can be observed. Also note that there could be some trivial outputs such as all zeros, all ones, the input or a part of the input etc., we remark that such a set of common knowledge denoted by $\mathbf{E}_{ck}$ (also a subset of the output space) is considered in the dissertation to capture these publicly known false outputs.[1]

The *blockchain* is always "good" for availability and correctness [57, 85]. The blockchain can faithfully execute some pre-specified programs called smart contracts to compute, transfer money, and/or broadcast execution result. But, smart contracts have to be light and deterministic, due to the intrinsic limitations of the underlying

[1]Remark that the correct output will not fall into the common-knowledge set $\mathbf{E}_{ck}$, as we consider a program whose output is unpredictably placed into a large output space that is much more considerable than $|\mathbf{E}_{ck}|$.

blockchain [102]. Worse still, the blockchain is transparent and promise no privacy guarantee, as all its internal states are public to everyone. It is worth to noticing that we consider the smart contract can detect any particular false output belonging to $\mathbf{E}_{ck}$, as $\mathbf{E}_{ck}$ is actually common knowledge. This is still a scenario much more stringent than the presence of a TTP who has to actively show up to resolve any dispute.

The *requester* requests the blockchain to execute a smart contract, which in practice is done through sending the blockchain a transaction pointing to the contract's program code[2]. Also, the requester promises some pre-specified monetary rewards to incentivize the blockchain to enforce the smart contract[3]. But the *requester* cannot request the blockchain to run heavy and randomized computations to facilitate his/her business. In such case, s/he will seek to outsource the task to off-chain service providers. Let the outsourced program be randomized, and represent it as $\vec{y} = P(\vec{x}; \vec{r})$, where $\vec{x}$ is the inputs, $\vec{y}$ is the outputs, and $\vec{r}$ is the random coin. We remark our assumption that the outputs $\vec{y}$ will not be predictable as P can extract considerable entropy from the random coin $\vec{r}$ (or from the input $\vec{x}$ as in some cases $\vec{r}$ can be empty), which actually captures many machine learning algorithms. Also, the requester promises an incentive mechanism pre-specified via the blockchain to incentivize the external computation services to compute his/her task, and we can view the requester as "good", since all his/her behaviors are codified into self-enforced contract.

As the blockchain cannot enforce the execution of complex and randomized computations for the requester, the requester/blockchain will further outsource these computation tasks to some external computing services, called *workers*, with announcing a pre-specified incentive mechanism via the blockchain. When workers

[2]After a smart contract is deployed in the blockchain, it will have a unique blockchain address, such that one can point it out through that address.

[3]In practice, the incentive mechanism of smart contract is much more detailed, c.f. the gas mechanism of Ethereum for details.

claim to compute a outsourced task, they have to transfer enough deposits to the blockchain, such that "penalties" can be applied, when the blockchain realizes undesirable behaviors of the workers. More importantly, any worker is rational, that means a work only prefers to maximize its utility. If some workers form a coalition, these workers will prefers to maximize the coalition's utility as well.

Monetary parameters. As we consider the problem in the game-theoretic setting, the monetary parameters should be clearly defined. In the following, we will explain the related parameters, and clarify the rationale behind:

- r represents the reward for a task, which is promised by the requester and facilitated by the blockchain;
- d corresponds the deposit should be funded by a worker, if the worker claims to do the outsourced task;
- c means the cost of a worker to do the task, which corresponds to the cost of running a program in a computer.

Let us consider a situation that $r \gg c$, i.e., the payment of the requester should be significantly larger than the cost of computing. That can be observed from the real-world blockchain such as Ethereum, where users have to pay pretty considerable premiums for on-chain computation resources. The requester in our case should be fine with such a surcharge as well, because his/her purpose is to let the correct computing result shown in the blockchain to further facilitate his/her business, which is much more valuable than the tiny cost of computing. Therefore, we will ignore c in the remaining of this dissertation.

Security goal. Next is to specify the security requirements for outsourcing the expensive and randomized computation to off-chain services in the rational setting.

Detailedly, we consider that a requester outsources the execution of a computation task to n workers denoted by $W_1, \ldots, W_n$. Our protocol and incentive design essentially instantiate a game Γ joined by these workers, as the workers have different utilities under different joint strategies. For a work W_k, we let its strategy set denoted

by $\mathbf{S}_k$. Also, there is always a strategy of "always sending the true result", denoted by σ_t, in per each worker's strategy set.

Our security requires that the joint strategy $\vec{\sigma} = \{\sigma_t, \ldots, \sigma_t\}$ is the only one that survives iterated elimination of weakly dominated strategies (IEWDS). The intuition of such an IEWDS refining equilibrium is quite clear: although sending the correct result might not always bring better utilities, when other workers do not send the correct result, but there is always no harm for a worker to send the correct one, and there even is a situation where sending the correct result brings the dominating utility. At such a refined equilibrium, everyone worker will apply the strategy of "always sending the true result". When this security requirement is satisfied, we can say our protocol realizes a *practical mechanism* $(\Gamma, \vec{\sigma} = \{\sigma_t, \ldots, \sigma_t\})$, as it can realize the desired Nash equilibrium $\vec{\sigma}$ surviving IEWDS in the game Γ.

5.2 Prior Art

The insufficiencies of related work, such as verifiable computation, trusted hardware, and existing game-theoretic approaches, are briefly summarized as follows.

A verifiable computation [59] allows a prover to produce an output along with a cryptographic proof to convince a verifier that the output is obtained through correctly executing a pre-defined computation. Recent constructive developments of zk-SNARKs [15] enables very efficient verification with the price that proving takes tremendous time and memory. For many heavy tasks, it is infeasible to generate the proof in practice. For example, in [98], we use more than 250 GB memory+swap and 15 hours to prove the majority of 11 RSA-OAEP encrypted answers.

The recent rise of trusted computation hardware such as Intel SGX [36] brings new techniques to allow people outsource their particular computation tasks [158] to untrusted third-party. However, to "enjoy" such new developments, one has to first trust the manufacturer. Worse still, the design rationale of SGX is a minimalist

trusted machine that has a restricted amount of enclave, so it becomes unclear how to avoid the huge performance penalty during attesting memory intensive programs.

Outsourced computation have been intensively discussed in game theoretic settings before [45, 121, 116, 14, 87]. Most of them rely on a TTP (or an implicit one launched by the requester) to resolve disputes and de-collude coalitions [45, 14, 87]. Some of them consider the absence of TTP under more optimistic scenario where all computing services are non-colluding [121, 116]. We will consider a more stringent setting: n computing services can form any coalition size up to $n-1$, and also there is no TTP to resolve mismatched computing results.

5.3 Protocol for Crowdsourcing ML Tasks

5.3.1 Protocol and Analysis: Two Non-Colluding Workers

In this section, we consider a fundamental scenario where the complex and randomized computation task is outsourced to two independent workers. Also, the absence of a trusted third-party (TTP) arbiter is taken into consideration. We observe that the state-of-the-art incentive mechanism will suffer from deviation by free-riding due to the absence of TTP arbiter, which can be intuitively understood as that a worker always prefers to copy and paste the result reported by the other worker. We present our protocol to address this critical issue, and analyze the security in game-theoretic setting.

Intuitions. Our basic idea is to outsource the computations of a smart contract to two *independent* rational workers, such that the workers can perform the computation *off-chain* and the blockchain nodes therefore get rid of doing the heavy work. At the same time, a simple incentive mechanism should be hosted by the blockchain to incentivize the workers submit the correct output (and prevent false output as well).

A naive solution may allow the two workers submit their computing results to the blockchain anytime, which implies someone can submit earlier, and the other one

can report the computing result later. Unfortunately, the blockchain is transparent, so the latter worker can free-ride by copying the first submission without doing any computation; worse still, the second worker will submit the same, even if knowing the earlier submission is wrong.

Intuitively, the major challenge is that the free-riding problem discourages the two workers to compute the result independently. To solve this critical issue, our idea is to ensure that the two workers reveal their computing results to the blockchain simultaneously. For the purpose, our protocol involves a *commit* phase, and a *reveal* phase to let the results "simultaneously" posted on the blockchain (or once committing a result, one cannot change the value even if revealed later). Such that, the incentive mechanism can ensure: (i) rational workers always submit their results "simultaneously", otherwise there will be significant penalty; (ii) rational workers have no interest to deviate by committing a false result.

Protocol details. Our protocol can be described as follows.

TaskPublish. *The requester announces the computing task via the blockchain.* When the requester R would like to run intensive computations atop the blockchain, she sends a smart contract that clearly specifies the task and the incentive mechanism. The computing task can be viewed as a randomized program denoted by $P(\vec{x};\vec{r})$, where $\vec{x}$ is the inputs and also stored on the chain, and $\vec{r}$ is the random coin which will be announced by the requester in the next phase.

TaskPrepare. *All participants fund enough deposits, and the random coin is released if necessary.* The budget (i.e., the promised rewards, denoted by r) of the requester will be deposited to the same contract. Then, a worker[4] that is interested in the task should send its deposit, denoted by d. Once both two workers, denoted by W_i, W_j, send their deposits. Random coin can be released, denoted by $\vec{r}$, if necessary.

[4]We remark that we ignore how to select two workers out of all qualified services. Any proper worker selection method should be compatible here, for example, we can either let the requester randomly choose, or let the workers bid, or based on the reputation score (if there is).

In practice, this coin can be randomly picked by the requester, or be derived by a future block [20]. After all these are completed, the protocol can move on. Remark that the blockchain addresses of W_i, W_j are recorded by the contract, such that the contract can require the workers to authenticate their transactions by signing under the corresponding secret keys later.

Commit. *The workers send the commitments encapsulating their computing results, respectively.* A worker, for example W_i, figures out a result, denoted by $\vec{y_i}$ for the task. If the result is correctly compute, we can expect $\vec{y_i}$ same to $\vec{y} = P(\vec{x}; \vec{r})$. Then, W_i will send the commitment of $\vec{y_i}$, denoted by c_i, to the smart contract. The commitment c_i is gotten via a standard commitment scheme[5], namely, $c_i = \mathsf{Commit}(\vec{y_i}, , K_i)$, where K_i is a randomly chosen secret of W_i until to open the commitment. For the other worker W_j, it also commits its result to the blockchain, which can be done through the same commitment scheme. Assuming that all computations and commitments can be done within a-priori delay Δ_c (in unit of block number), Δ_c can be pre-specified by the contract. If any worker misses the time to commit, its deposit is taken away. We also remark that the execution of the outsourced computation is done *off-chain*, which will hurt the consensus of the underlying blockchain.

Reveal. *The workers open the commitment to reveal their computing results, respectively.* When both workers commit results, a worker, e.g., W_i can reveal the result $\vec{y_i}$ to the blockchain. If the blockchain receives K_i, it computes $\mathsf{Open}(c_i, K_i)$ to get $\vec{y_i}$. If the Open algorithm outputs $fail$, W_i will lose all its deposit. Also, supposing that reveal should be done within a-priori countdown timer Δ_o, a countdown timer can be instantiated by the contract, if any worker fails to reveal in time, its deposit will not be returned.

[5]Remark that the commitment scheme itself is not necessarily non-malleable, as the transactions encapsulating commitments are accepted by the contract only if they are validly signed by workers.

Reward. *The blockchain rewards/punishes the workers according to the results they revealed.* Once both the commitments are open or the countdown timer for revealing stage expires, the smart contract will check the equality of result(s), and make rewards or penalties, according to the pre-defined incentive clauses. More detailedly, the clauses can be abstracted as:

- If any worker fails to commit or de-commit or submit a result in the common-knowledge set $\mathbf{E}_{ck}$, it loses all deposits, and the other party gets only half of its deposit refunded (if it submits a result); [6]
- If $\vec{r}_i = \vec{r}_j$, the two workers will split the reward, i.e., W_i gets $r/2$, and W_j gets $r/2$; [7]
- If $\vec{r}_i \neq \vec{r}_j$, both the two workers will lose half of their deposits, i.e., W_i gets $-d/2$, and W_j gets $-d/2$.

Intuitively, the blockchain can only know the following information: (i) the equality of two results (if there are two); (ii) anyone who fails to submit a result. And our incentive mechanism are trying to make best usage of these information to deter the submission of incorrect results.

Security analysis (sketch). Here we briefly discuss the security of the above protocol in game-theoretic setting. Let us firstly check all possible pure strategies of each worker. Considering that the workers are non-colluding, there are three pure strategies for each worker in general:

- "Do not commit a result in the Commit phase, or do not open the commitment in the Decommit phase, or submit a result in $\mathbf{E}_{ck}$", denoted by σ_e;
- "Reveal the correct result to the blockchain", denoted by σ_t;
- "Reveal some false output chosen out of $\mathbf{E}_{ck}$", denoted by σ_f.

[6] In practice, we may have valid heuristics to efficiently check weather an output is in $\mathbf{E}_{ck}$. For example, the false ones in $\mathbf{E}_{ck}$ usually have small entropy (except some cases like outputting inputs), while the output of programs is unpredictable in our setting, which infers much more entropy. Such that the requester can provide a list of random coins and then let the workers to compute the program for several times under different random coins, which will allow the smart contract to easily filter out low-entropy results as they are suspiciously chosen from the set of $\mathbf{E}_{ck}$.

[7] One may wonder a trivial strategy that both of workers output some junks in the set of $\mathbf{E}_{ck}$. But these have been captured by the contract, and are seen as failures of submitting.

Table 5.1 The Game of Two Workers in Normal Form

Utility of W_i,W_j / W_i \ W_j	σ_e	σ_t	σ_f
σ_e	$-d, -d$	$-d, -d/2$	$-d, -d/2$
σ_t	$-d/2, -d$	$r/2, r/2$	$-d/2, -d/2$
σ_f	$-d/2, -d$	$-d/2, -d/2$	$-d/2, -d/2$

Since we consider a program having large output space, where the possible correct results are placed in an unpredictable way. When two non-colluding workers are submitting two junks out of $\mathbf{E}_{ck}$ without coordination, their best way is to sampling random junks, as the unpredictable output makes them have no advantage other than guessing. Also, for large output space, *there is no chance for such two junks to be the same*, i.e., the contract can always find two different submissions, if both workers makes junks out of $\mathbf{E}_{ck}$.

Now we are ready to show the security of our protocol. In Table.5.1, the utilities of two workers are shown for different joint strategies, as the incentive clauses are clearly defined. We can leverage IEWDS method to check our game:

- For any worker, the pure strategy of playing σ_e is strictly dominated (i.e., worse utility for sure), so σ_e is "deleted" for both workers, i.e., both workers will at least submit;

- For any worker, the pure strategy of playing σ_f is weakly dominated (i.e., not better utility for sure), so σ_f is "deleted" for both workers, i.e., sending the true result is at least not worse than sending random junk.

The only left IEWDS refining Nash equilibrium is the joint strategy $\{\sigma_t, \sigma_t\}$, i.e., "both workers send the correct result", and we can claim our protocol realizes a *practical mechanism* $(\Gamma, \{\sigma_t, \sigma_t\})$, which clearly realizes our security goal.

5.3.2 Protocol & Analysis: n Workers ($|\textbf{Coalition}| \leq n-1$)

The protocol for two workers rely on a key assumption that they are independent, i.e., non-colluding. Now we are ready to present our protocol in a more general scenario where n workers instead of two workers can hired to compute the outsourced computations. More generally, we will allow the n workers to collude, as long as the size of their coalitions is up to $n-1$ (inclusively). Detailedly, in such a coalition, the colluding workers there can make a strategy pre-agreed by all of coalition members to deviate the game. If the coalition strategy is not weakly dominated by "always sending the correct", the mechanism fails. Otherwise, the mechanism is still a $(n-1)$-resilient practical mechanism $(\Gamma, \vec{\sigma} = \{t, \ldots, t\})$ that can tolerate these coalitions. In this section, we will show the existence of such a $(n-1)$-resilient practical mechanism. Remark that this more general conclusion essentially captures the situation of two independent workers (i.e., 1-resilient practical mechanism).

Protocol brief. The protocol for n workers also has TaskPublish, TaskPrepare, Commit, Reveal and Reward phases, which are similar to the protocol for two workers. Therefore we only focus on the incentive mechanism for n workers, and present the clauses only:

- If any worker fails to commit or de-commit or submit a result in the common-knowledge set $\mathbf{E}_{ck}$, it loses all deposits, and all the other workers who submit will only get a refund of half deposit;
- If n workers successfully report the same computing result, the n workers will share the reward equally, i.e., each worker gets r/n;
- If n workers submit different results, i.e., there is as least one submission differs from others, all workers will lose half of their deposits;

The above incentive mechanism for n workers is a straightforward extension of the one for two workers. Again, the blockchain can only know the equality of reported results, along with who fails to submit. Our incentive mechanism is designed to use these little information to deter false results.

Table 5.2 Utility of an Arbitrary Coalition C ($|C| \leq n-1$) in the Game of n Workers

C's utility / Else's strategy / C's strategy	Situation **I**	Situation **II**	Situation **III**
σ_1	$-kd$	$-kd$	$-kd$
σ_2	$-kd/2$	kr/n	$-kd/2$
σ_3	$-kd/2$	$-kd/2$	$-kd/2$
$\boldsymbol{\sigma}_4$	$-pd/2-ld$	$-pd/2-ld$	$-pd/2-ld$
$\boldsymbol{\sigma}_5$	$-qd/2-ld$	$-qd/2-ld$	$-qd/2-ld$
$\boldsymbol{\sigma}_6$	$-(p+q)d/2$	$-(p+q)d/2$	$-(p+q)d/2$
$\boldsymbol{\sigma}_7$	$-(p+q)d/2-ld$	$-(p+q)d/2-ld$	$-(p+q)d/2-ld$

Security analysis (sketch). Here we briefly discuss the security of the protocol for n workers. For a coalition C of size up to k ($1 \leq k \leq n-1$), it roughly has the following strategies:

- Let all player in it play e, i.e., the coalition will submit nothing, denoted by σ_1;
- Let all members play t, i.e., the coalition will submit all correct results, denoted by σ_2;
- Let all members "submit the same pre-agreed randomly chosen junk", denoted by σ_3;
- Let p members send the correct results, and the other $l = k - p$ members send nothing, and the family of strategies can be denoted by $\boldsymbol{\sigma}_4$;
- Let q members send the random junks, and the other $l = k - q$ members send nothing, and we denote the family of strategies by $\boldsymbol{\sigma}_5$;
- Let p members send correct results, and the other $q = k - p$ members send junks, denoted by $\boldsymbol{\sigma}_6$;
- Let q members to send junks, some p members send correct results, and the remaining $l = k-p-q$ members send nothing, and this strategy family denoted by $\boldsymbol{\sigma}_7$.

Notice that a coalition's strategy can be enforced for all its member workers. For example, the same randomly chosen junk can be agreed by all its members. But for

another worker who is out of the coalition C, it cannot know what the random junk agreed within C. When C is making a decision on what a strategy to play, its utility can be discussed by three situations. In each of the situation, the out-of-coalition worker(s) might play different strategies:

- Situation **I**: there is at least one out-of-coalition worker submits nothing (including to submit a result in $\mathbf{E}_{ck}$);
- Situation **II**: there is at least one out-of-coalition worker submits a randomly chosen junk (out of $\mathbf{E}_{ck}$);
- Situation **III**: all out-of-coalition worker(s) submit the correct result.

When the coalition C applies different strategy in different situations, the utility of this coalition can be derived and shown in Table 5.2. Essentially, for any coalition formed by up to $n-1$ workers, its utility can be represented in a similar table. After performing IEWDS, the only survived joint strategy of all coalitions is "to send the correct results". In details, the iterative deletion of the weakly dominated strategies can be discussed as follows:

- For any coalition, the pure strategy σ_1 is strictly dominated, so σ_1 is deleted, also, situation **I** can be removed, as a consequence of the previous deletion;
- For any coalition, the pure strategies in $\boldsymbol{\sigma}_4$, $\boldsymbol{\sigma}_5$ and $\boldsymbol{\sigma}_7$ are strictly dominated, and therefore these strategies can be removed one by one;
- For any coalition, the pure strategies in $\boldsymbol{\sigma}_6$ are weakly dominated, and therefore can be removed, also after this deletion, the situation **II** disappears.

For any coalition C ($1 \leq |C| \leq n-1$), the only left pure strategy is to "send all correct result(s)" after applying IEWDS. Actually, the game even can guarantee that no matter how a coalition plays, no single worker in that coalition can get better utility than "sending the true result". Here we omit the trivial steps of proving this conclusion. As such, all n workers will submit the correct results, even if we admit that these workers can collude to make up any coalitions size up to $n-1$. In sum, we can realize a practical mechanism $(\Gamma, \{\sigma_t, \ldots, \sigma_t\})$ which is $(n-1)$ resilient.

5.4 Summary

To enable the feasibility of using smart contracts to crowdsource computation intensive and sometimes randomized programs such as machine learning tasks, with the given limitations on all existing verifiable computation, we take a game theoretic approach. Instead of providing absolute confidence of verifying the correct execution of outsourced "smart contract", we design a simple incentive mechanism (that is enforced by smart contract itself) so that dishonest execution could be deterred. We present a general protocol to allow the chain to outsource heavy and randomized computations to some external computing service providers (called workers). More specifically, our contributions are threefold.

First, to enable the execution of heavy computations off-chain, we first consider a classical (non-cooperative) game-theoretic setting with two workers that can audit each other. The chain would take results that both workers agree on. Most state-of-the-art solutions along this line assume an opportunistic approach that a trusted party (TTP) comes online to arbitrate, in case an agreement on the result cannot be reached. As opposed, we would get rid of this assumption. Besides properly setting the incentives/payoffs, another subtle point remains due to the *transparency* of the open blockchain: if one worker submits his answer, the other worker can simply send the element without doing any work. This is known as the free-riding problem in crowdsourcing which was also noted in previous works such as [98]. We leverage a cryptographic tool of commitment scheme to tackle this problem. Taking all the issues into account, we design a simple incentive mechanism and prove there is a desirable refinement of Nash Equilibria that both workers do the correct computation and return the right answer (c.f. Sec. 2.4 for the Equilibria refinement). We also consider a class of randomized algorithms that can be considered as programs taking inputs and an extra random coin (which can be supplied by the chain to ensure the consistency for different workers). We remark that in our protocol, the

expensive and randomized computation is crowdsourced via the application layer to be "asynchronously" executed and separated from the consensus layer. The full nodes/miners can simply put the output into the coming block as the result is submitted, they do not need to wait the execution to finish during mining, and thus the consensus security will not be influenced.

Second, we then give two concrete instantiations regarding different type of machine learning tasks, and apply our protocol to enable the "execution" of them over the blockchain. The first example is about checking the quality of a crowdsourced machine learning model, i.e., conditional payment based on the performance on the testing data. The second example is directly for outsourcing machine learning training via the smart contract, which can be an important piece of the social computing puzzle. These two interesting instances capture the essence of most potential decentralized crowdsourcing for machine learning tasks. We also note that our solution is actually general to carry a wide range of complex (or randomized) programs as well. Finally, we set force to consider potential coalition(s) in the general setting of more (≥ 3) workers. In particular, we design a protocol that can tolerate any coalition(s), as long as there is no coalition consisting of all workers.

CHAPTER 6

ON RECRUITING RELAYS FOR BLOCKCHAINS' LIGHT CLIENTS

6.1 Background

6.1.1 Motivation

The emerging blockchain technology, in particular a permissionless blockchain, as nice resultants of cryptography, distributed consensus and economic incentive, enables an append only digital ledger to be jointly maintained and replicated consistently across Internet peers. It enables many fantastic paradigms such as cryptocurrency and smart contracts. As such, attractive decentralized ecosystems can be envisioned to reduce the reliance on undesired third-parties, and therefore various promising decentralized applications (DApps), such as novel financial instruments [114], peer-to-peer storage outsourcing [108, 124], and data crowdsourcing [98] etc., can be implemented atop blockchains.

Normally, a blockchain is considered as an abstracted layer so that higher layer applications can simply interact with the ledger to read data from or write data into the ledger [57, 85]. This basic abstraction (of reading data)[1] implicitly assumes the higher layer application is within a *full node* of the blockchain who has a complete copy of the ledger thus can read data locally. A blockchain full node need always try to download, verify, and store a replica of the ledger, so that he can catch up with the consensus reached within the blockchain network.

On the other hand, there is a huge demand [37, 142, 98, 11] of blockchain's *lightweight nodes* (sometimes we also call them lightweight clients or light clients) that may not have the capability to maintain the whole copy of the ledger. In the setting of DApps, they could be hosted in smartphones, browser extensions, IoT nodes

[1] We remark that to write valid messages into the blockchain could be trivial, as one can always gossip with some blockchain nodes to diffuse its messages to the whole blockchain network, and then the liveness of the blockchain will ensure the appearance of these messages in the ledger [57].

or even an external blockchain's smart contract. For example, it is highly desirable that a user can use his mobile cryptocurrency wallet to do online sale and verify the settlement of transactions sent to her in the blockchain. This demand calls for a protocol specifically designed for lightweight nodes, such that they can be exempt from running the consensus protocol to keep up with the whole ledger. What's more, such a desirable protocol is also promising to resolve the critical problem of efficient "cross-chain" communication [80, 11] between two different chains (or among multiple chains), in which a peer may not be interested to keep the complete ledger of both of the ledgers; instead, he could be a full node of one blockchain, while plays the role as a lightweight node of another external blockchain. Let alone for resourceful DApps users, they have few motivations to setup full nodes to maintain a ledger, and might also look for lightweight node protocol, as their main purpose is just to use DApps.

6.1.2 Challenges

The difficulty of designing a lightweight friendly protocol comes from the fact that without a replica of the complete chain, a lightweight node essentially has to rely on some full nodes to relay the blockchain's states. Many of the existing systems simply rely on some *trusted* relay nodes [104]. Two threats emerge: (i) a dishonest relay node (especially the one has financial transactions with the lightweight client) may simply return false information about the ledger to the lightweight client; (ii) not like miners maintaining the ledger to obtain rewards, full nodes have little incentive to participate in the relay service [39]. Current efforts on lightweight node protocols [96, 80, 114, 91] have been focusing on providing succinct cryptographic proofs to convince the lightweight nodes. The general idea is to allow the lightweight nodes to keep with some small amount of metadata (which usually are consensus-dependent such as the proof-of-work in Nakamoto's blockchains [114, 80, 96]) to pick up the correct blockchain branch.

These methods have several drawbacks: first, most of them require the lightweight nodes to store and keep updating some metadata to verify the relayed blockchain readings, which can be unfavorable in resource-starved environments such as IoT sensors and/or the multi-chain scenarios [96]. Worse still, as these methods usually require lightweight nodes to validate some *consensus-dependent* metadata, and therefore are always specifically designed with considering the underlying consensus protocols. For example, for proof-of-stake type of consensus, currently one of few candidates [91] is specifically designed for Algorand [63], which is not suitable for blockchains that utilize other proof-of-stake protocols. This also creates prohibitive obstacles to interesting use-cases such as multi-chain clients. See Related Work below for more detailed discussions about the insufficiencies of the current solutions.

"*How to realize a lightweight protocol that is generic as well as friendly to extremely resource-starved environments*" is still a valuable open problem.

6.2 Prior Art

The SPV client is the first lightweight protocol for PoW blockchains, proposed as early as Bitcoin [114]. Following the protocol, resource-constrained devices need to download, verify and store a chain of block headers, and then can verify the existence of any transaction, with the help of other full nodes. The main weaknesses of SPV client is that the block headers to download, verify and store will grow linearly with the number of the blocks of the chain, which nowadays corresponds 40 MB in Bitcoin and more than 2 GB in Ethereum. For this reason, the concept of Proofs of Proof-of-Work (PoPoW) was formulated in [80, 79], through which one can store only the genesis block header, and then verify the existence of any transaction at a communication cost sub-linear to the length of PoW chain. All above schemes cannot be applied to a proof-of-stake (PoS) blockchain, because in a PoS blockchain, the validity of a block relies on the signature(s) of stakeholder(s), whose validities further depend on the distribution

of all "stakes" recorded in the blockchain ledger. Some fast bootstrap methods such as [91] was proposed for lightweight nodes, but only works for a particular PoS chain; moreover the bootstrapping is still at a substantial cost which is still unaffortable to most resource-constrained nodes.

Vitalik Buterin [22] proposed to avoid forks in PoS blockchain using incentives: a selected committee will be punished if he proposed two different blocks in one epoch. In this way, the lightweight node was claimed to be supported as receiving a block (and then send back to the blockchain network) different with the one on chain could be viewed as a malicious behavior of the relay node. However, as committees rotate periodically, it is unclear whether the protocol still works if the relay node is a committee for a recent period, while the lightweight client query is about some "ancient" block which was generated before the relay node becomes a committee. Another recent attempt in [39] focused on implementing a prototype protocol to support lightweight nodes by using economic incentives way, while our work provides a formally game-theoretic study in the more general setting.

Verifiable computation allows a prover to produce a cryptographic proof to convince a verifier that the output is obtained through correctly executing a pre-defined computation [59]. Recent constructive developments of zk-SNARKs [15] enables very efficient verification with the price that proving takes tremendous time and memory. For heavy tasks such as to prove the length (and even more complex properties) of a blockchain, it is usually infeasible in practice [96].

Attestation via trusted hardwares has attracted many attentions recently [36, 158]. But recent Foreshadow attacks [146, 150] put SGX's attestation keys in danger of leakage, and thus allow the adversary to forge attestations, which hints us again that the whole assumption of trusted hardwares could be arguable.

Outsourced computation via incentive games have been discussed earlier [121, 87]. Some recent studies even consider using the blockchain to facilitate such

outsourcing games [45, 141]. All these studies assume an implicit game mediator (e.g., the blockchain) who can speak to and listen to all involving parties, including the requester, the workers and the arbiter. However, in our setting, we have to resolve a special issue that there is no such a game mediator, since the blockchain, as a potential candidate, can not speak to the light node (i.e., the requester of computing).

6.3 Warmup: Security in Extensive-Form Game

Here gives a simple interactive "protocol" to exemplify the game-theoretic setting (see Chapter 2 for more rigor definitions of the relevant game-theory preliminaries).

6.3.1 Interactive Protocol as Extensive-Form Game

Consider an oversimplified "light-client protocol": Alice is a cashier of a pizza store; her client asks a full node (i.e., relay) to check a transaction's (non)existence, and simply terminates to output what is forwarded by the relay.

Strategy, action, history, and information set. Let the *oversimplified* "protocol" proceed in synchronous round. In each round, the parties will execute its *strategy*, i.e., a probabilistic polynomial-time ITM in our context, to produce and feed a string to the protocol, a.k.a., take an *action*. During the course of the protocol, a sequence of actions would be made, and we say it is a *history* by convention of the game theory literature; moreover, when a party acts, it might have learned some (incomplete) information from earlier actions taken by other parties, so the notion of *information sets* can be used to characterize what has and has not been learned by each party. Concretely speaking, the *oversimplified* "light-client protocol" can be described by the extensive-form game as shown in Figure 6.1, and it proceeds in the next three rounds.

Round 1 (chance acts). A definitional virtual party called *chance* sets the ground truth, namely, it determines $True$ or $False$ to represent whether the

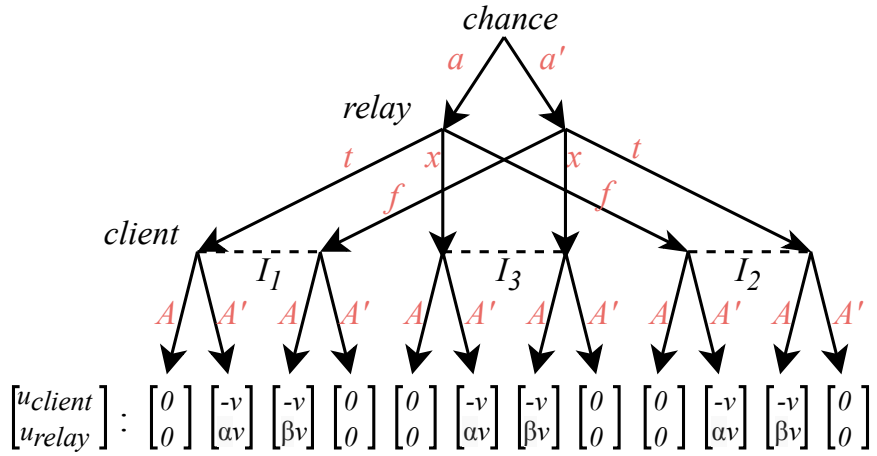

Figure 6.1 The extensive game of an oversimplified light-client "protocol". The utility function is an example to clarify *insecurity* of the trivial idea.

transaction exists (denoted by a or a', respectively). To capture the uncertainty of the ground truth, the chance acts arbitrarily.

Round 2 (relay acts). Then, the relay is activated to forward $True$ or $False$ to the light client, which states whether the transaction exists or not. Note the strategy chosen by the relay is an ITM that can produce arbitrary strings in this round, we need to map the strings into the admissible actions, namely, t, f and x. For definiteness, we let the string of ground truth be interpreted as the action t, the string of the opposite of ground truth be interpreted as the action f, and all other strings (including abort) be interpreted as x.

Round 3 (client acts). Finally, the client outputs $True$ (denoted by A) or $False$ (denoted by A') to represent whether the transaction exists or not, according to the (incomplete) information acquired from the protocol. Note the client knows how the relay acts, but cannot directly infer the action of *chance*. It faces three distinct information sets I_1, I_2 and I_3, which respectively represent the client receives $True$, $False$ and others in Round 2. Note that the client cannot distinguish which history it reaches within each of its information sets.

Utility function. After the protocol terminates, its game reaches a so-called *terminal history*. A well-defined *utility function* specifies the economic outcome of each party, for each terminal history induced by the extensive game.

In practice, the utility function is determined by some economic factors of the parties and the protocol itself [69, 45]. For example, the rationale behind the utility function in Figure 6.1 can be understood as: (i) the relay is motivated to fool the client to believe the nonexistence of an existing transaction, because this literally "censors" Alice to harm her business by a loss of $\$v$, which also brings a malicious benefit $\$\alpha \cdot v$ to the relay; (ii) the relay also prefers to fool the client to believe the existence of a non-existing transaction, so the relay gets free pizzas valued by $\$\beta \cdot v$, which causes Alice lose $\$v$ (i.e., the amount supposed to be transacted to purchase pizzas), (iii) after all, the oversimplified protocol itself does not facilitate any punishment/reward, so will not affect the utility function.

6.3.2 Security via Sequential Equilibrium

Putting the game structure and the utility function together, we can argue the (in)security due to the equilibria in the game. In particular, we can adopt the strong notion of *sequential equilibrium* for extensive games [70, 71, 45, 118] to demonstrate that the rational parties would not deviate, at each stage during the execution of the protocol. As a negative lesson, the oversimplified "light-client game" in Figure 6.1 is *insecure* in the game-theoretic setting, as the relay can unilaterally deviate to fool the client for higher utility. In contrast, if the protocol is *secure* in game-theoretic settings, its game shall realize desired equilibrium, such that rational parties would not diverge from the protocol for highest utilities.

6.4 Problem Formulation

The light-client protocol involves a light client, some relay full nodes (e.g., one or two), and an ideal functionality (i.e., "arbiter" contract). The light client relies on the relays to "read" the chain, and the relays expect to receive correct payments.

The basic functionality of our light-client protocol is to allow the resource-starved clients to evaluate the falseness or trueness about some statements over the blockchain [80]. This aim is subtly broader than [83], whose goal is restricted to prevent the client from deciding trueness when the statement is actually false.

Chain predicate. The dissertation focuses on a general class of chain predicates whose trueness (or falseness) can be induced by up to l transactions' inclusions in the chain, such as "whether the transaction with identifier txid is in the blockchain $\mathsf{C}[0:N]$ or not". Formally, we focus on the chain predicate in the form of:

$$\mathsf{P}^{\ell}(\mathsf{C}[0:N]) = \begin{cases} False, \text{ otherwise} \\ True, \exists \mathsf{C}' \subset \mathsf{C}[0:N] \text{ s.t. } \mathsf{D}^{\ell}(\mathsf{C}') = True \end{cases}$$

or equivalently, there is $\mathsf{Q}(\cdot) = \neg\mathsf{P}(\cdot)$:

$$\mathsf{Q}^{\ell}(\mathsf{C}[0:N]) = \begin{cases} False, \exists \mathsf{C}' \subset \mathsf{C}[0:N] \text{ s.t. } \mathsf{D}^{\ell}(\mathsf{C}') = True \\ True, \text{ otherwise} \end{cases}$$

where C' is a subset of the blockchain $\mathsf{C}[0:N]$, and $\mathsf{D}^{\ell}(\cdot)$ is a computable predicate taking C' as input and is writable as:

$$\mathsf{D}^{\ell}(\mathsf{C}') = \begin{cases} True, \exists\ \{\mathsf{tx}_i\} \text{ that } |\{\mathsf{tx}_i\}| \leq \ell\text{:} \\ \qquad f(\{\mathsf{tx}_i\}) = 1\ \wedge\ \forall\ \mathsf{tx}_i \in \{\mathsf{tx}_i\}, \\ \qquad\quad \exists\ \mathsf{C}[t] \in \mathsf{C}' \text{ and P.P.T. computable } \pi_i \text{ s.t.} \\ \qquad\qquad \mathsf{VrfyMTP}(\mathsf{C}[t].\mathsf{root}, \mathsf{H}(\mathsf{tx}_i), \pi_i) = 1 \\ False, \text{ otherwise} \end{cases}$$

where $f(\{\mathsf{tx}_i\}) = 1$ captures that $\{\mathsf{tx}_i\}$ satisfies a certain relationship, e.g., "the hash of each tx_i equals a specified identifier txid_i", or "each tx_i can pass the membership test of a given bloom filter", or "the overall inflow of $\{\mathsf{tx}_i\}$ is greater than a given value". We let P^{ℓ}_{N} and Q^{ℓ}_{N} be short for $\mathsf{P}^{\ell}(\mathsf{C}[0:N])$ and $\mathsf{Q}^{\ell}(\mathsf{C}[0:N])$, respectively.

Examples of chain predicate. The seemingly complicated definition of chain predicate actually has rather straightforward intuition to capture a wide range of blockchain "readings", as for any predicate under this category, either its trueness or its falseness can be succinctly attested by up to ℓ transactions' inclusion in the chain. For $\ell = 1$, some concrete examples are:

- "A certain transaction tx is included in $\mathsf{C}[0:N]$", the trueness which can be attested by tx's inclusion in the chain.
- "A set of transactions $\{\mathsf{tx}_j\}$ are *all* incoming transactions sent to a particular address in $\mathsf{C}[0:N]$", the falseness of which can be proven, if $\exists$ a transaction tx s.t.: (i) $\mathsf{tx} \notin \{\mathsf{tx}_j\}$, (ii) tx is sent to the certain address, and (iii) tx is included in the chain $\mathsf{C}[0:N]$.

Limits. A chain predicate is a binary question, whose trueness (or falseness) is reducible to the inclusion of some transactions. Nevertheless, its actual meaning depends on how to concretely specify it. Intuitively, a "meaningful" chain predicate might need certain specifications from an external party outside the system. For example, the cashier of a pizza store can specify a transaction to evaluate its (non)existence, only if the customer tells the txid.

"Handicapped" verifiability of chain predicate. W.lo.g., we will focus on the chain predicate in form of P^{ℓ}_{N}, namely, whose trueness is provable instead of the falseness for presentation simplicity. Such the "handicapped" verifiability can be well abstracted through a tuple of two algorithms $(\mathsf{evaluate}, \mathsf{validateTrue})$:

- $\mathsf{evaluate}(\mathsf{P}^{\ell}_{N}) \to \sigma$ or $\perp$: The algorithm takes the replica of the blockchain as auxiliary input and outputs σ or $\perp$, where σ is a proof for $\mathsf{P}^{\ell}_{N} = True$, and $\perp$ represents its falseness; note the proof σ here includes: a set of transactions $\{\mathsf{tx}_i\}$, a set of Merkle proofs $\{\pi_i\}$, and a set of blocks C';

- $\mathsf{validateTrue}(\sigma, \mathsf{P}_N^\ell) \rightarrow 0$ or 1: This algorithm takes $\mathsf{blockhashes}$ as auxiliary input and outputs 1 (accept) or 0 (reject) depending on whether σ is deemed to be a valid proof for $\mathsf{P}_N^\ell = True$.

The above algorithms satisfy: (i) *Correctness*, that means for any chain predicate P_N^ℓ, the probability $\Pr[\mathsf{validateTrue}(\mathsf{evaluate}(\mathsf{P}_N^\ell), \mathsf{P}_N^\ell) = 1 \mid \mathsf{P}_N^\ell = true]$ is equal to 1, and (ii) *Verifiability*, which means for any P.P.T. $\mathcal{A}$ and P_N^ℓ, there is $\Pr[\mathsf{validateTrue}(\sigma \leftarrow \mathcal{A}(\mathsf{P}^\ell), \mathsf{P}_N^\ell) = 1 \mid \mathsf{P}_N^\ell = false] \leq negl(\lambda)$, where $\mathsf{evaluate}$ implicitly takes the blockchain replica as input, and $\mathsf{validateTrue}$ implicitly inputs $\mathsf{blockhashes}$. The abstraction can also be slightly adapted for Q_N^ℓ whose falseness is the provable side, though we omit that for presentation simplicity. Through the remaining of the dissertation, $\mathsf{evaluate}$ can be seen as a black-box callable by any full nodes that have the complete replica of the blockchain, and $\mathsf{validateTrue}$ is a subroutine that can be invoked by the smart contracts that can access the dictionary $\mathsf{blockhashes}$.

6.4.1 System and Adversary Model

The system explicitly consists of a light client, some relay(s) and an arbiter contract. All of them are computationally bounded to perform only polynomial-time computations. The messages between them can deliver synchronously within a-priori known delay ΔT, via point-to-point channels. In details, each system participant can be abstracted in the following way.

The rational lightweight client $\mathcal{LW}$ is abstracted as follows: (i) It is rational and selfish; (ii) It is computationally bounded, i.e., it can only take an action computable in probabilistic polynomial-time; (iii) It opts out of consensus; to capture this, we assume: The client can *send* messages to the contract due to the network diffusion functionality [57, 85]; and the client cannot receive messages from the contract except a short setup phase, which can be done in practice because the client user can temporarily boost a personal full node by fast-bootstrapping protocols.

The rational full node $\mathcal{R}_i$ is modeled as: (i) It is rational, and the full node $\mathcal{R}_i$ might (or might not) cooperate with another full node $\mathcal{R}_j$; The cooperative full nodes form a coalition to maximize the total utility, as they can share all information, coordinate all actions and transfer payoffs, etc. [117]; essentially, we follow the conventional notion to view the cooperative relays as **a single** party [13]; Non-cooperative full nodes maximize their own utilities independently in a selfish manner due to the standard non-cooperative game theory, which can be understood as that they are not allowed to choose some ITMs to communicate with each other [90]; (ii) It can only take P.P.T. computable actions at any stage of the protocol; (iii) It runs the consensus, such that it stores the complete replica of the latest blockchain and can send/receive messages to/from the smart contract; (iv) It can send messages to the light client via an off-chain private channel.[2]

The arbiter contract $\mathcal{G}_{ac}$ follows the standard abstraction of smart contracts [85, 82], with a few slight extensions. First, it would not send any messages to the light client except during a short setup phase. Second, it can access a dictionary $\mathsf{blockhashes}$ [106, 54], which contains the hashes of all blocks. The latter abstraction allows the contract to invoke $\mathsf{validateTrue}$ to verify the proof attesting the trueness of any predicate P_N^{ℓ}, in case the predicate is actually true.

6.4.2 Economic Factors

It is necessary to clarify the economic parameters of the **rational** parties to complete our game-theoretic model. We present those economic factors and argue the rationale behind them as follows.

Parameter c: It represents how much the client spends to maintain its (personal) trusted full node during the repeatable query phase. Note c does not mean the

[2]Such the assumption can be granted if considering the client and the relays can set up private communication channels on demand. In practice, this can be done because (i) the client can "broadcast" its network address via the blockchain [101], or (ii) there is a trusted name service that tracks the network addresses of the relays.

security relies on a trusted full node and only characterizes the cost of maintaining the trusted full node. For example, $c \to \infty$ would represent that the client cannot connect any available trusted full node, once the protocol has been set up and the client disconnects any personal full node. Note c does not characterize the cost of the relay full node to run the consensus, and is considered to argue that rational users are willing to employ our protocol instead of maintaining their own personal full nodes (or subscribing some other relaying services).

Parameter v: The factor means the "value" attached to the chain predicate under query. If the client incorrectly evaluates the predicate, it loses v. For example, the cashier Alice is evaluating the (non)existence of a certain transaction; if Alice believes the existence of a non-existing transaction, she loses the amount to be transacted; if Alice believes the nonexistence of an existing transaction, her business is harmed by such the censorship.

Parameter $v_i(\mathsf{P}_N^\ell, \mathsf{C}) \to [0, v_i]$: This function characterizes the motivation of the relay $\mathcal{R}_i$ to cheat the light client. Namely, it represents the extra (malicious) utility that the relay $\mathcal{R}_i$ earns, if fooling the client to incorrectly evaluate the chain predicate. We explicitly let $v_i(\mathsf{P}_N^\ell, \mathsf{C})$ to have an upper-bound v_i s.t. $\sum_{\mathcal{R}_i} v_i \leq v$, which means the malicious utilities acquired by all relay nodes when the light client is fooled shall not be greater than the "value" attached to the chain predicate to query.

Parameter ϵ: When a party chooses a strategy (i.e., a P.P.T. ITM) to break underlying cryptosystems, we let ϵ represent the expected utility of such a strategy, where ϵ is a negligible function in cryptographic security parameter [44].

In addition, all communications and P.P.T. computations can be done costlessly w.r.t. the economic aspect (unless otherwise specified).

6.4.3 Security Goals

The aim of the light-client protocol in the game-theoretic model is to allow a rational light client employ some rational relaying full nodes (e.g., two) to correctly *evaluate* a few chain predicates, and these recruited full nodes are correctly paid as pre-specified. In details, we require such the light-client protocol $\Pi_{\mathcal{LW}}$ to satisfy the following *correctness* and *security* properties.

Correctness. If all parties are honest, we require: (i) the relay nodes are correctly paid; (ii) the light client correctly evaluates some chain predicates under the category of $\mathsf{P}^{\ell}(\cdot)$, regarding the chain $\mathsf{C}[0:T]$ (i.e., the chain at the time of evaluating). Both requirements shall hold with probability 1.

Security. We adopt a strong game-theoretic security notion of *sequential equilibrium* [71, 45, 70] for incomplete-information extensive games. Consider an extensive-form game Γ that models the light-client protocol $\Pi_{\mathcal{LW}}$, and let $(\mathbf{Z_{bad}}, \mathbf{Z_{good}})$ as a partition of the terminal histories $\mathbf{Z}$ of the game Γ. Given a *ϵ-sequential equilibrium* of Γ denoted by σ, the probability of reaching each terminal history $z \in \mathbf{Z}$ can be induced, which can be denoted by $\rho(\sigma, z)$. Our security goal would require: there is a ϵ-sequential equilibrium σ of Γ where ϵ is at most a negligible function in cryptographic security parameter λ, such that under the ϵ-equilibrium σ, the game Γ always terminates in $\mathbf{Z_{good}}$.

Remark. The traditional game-theory analysis captures only computationally unbounded players. But it becomes natural to consider computationally-bounded players in an interactive protocol using cryptography, so will we do through the dissertation. In such the setting, a strategy of a party can be a P.P.T. ITM to break the underlying cryptosystems. However, this strategy succeeds with only negligible probability. Consequently, our security goal (i.e., ϵ-sequential equilibrium) can be refined into a computational variant to state the rational players switch strategies, only if the gain of deviation is non-negligible.

6.5 A Simple Light-Client Protocol

We dedicatedly design a simple light-client protocol, in which a light client ($\mathcal{LW}$) can leverage it to employ two (or one) relays to evaluate the chain predicates P_N^ℓ.

6.5.1 Arbiter Contract and High-Level of the Protocol

The simple light-client protocol is centering around an arbiter smart contract $\mathcal{G}_{ac}$ as shown in Figure 6.2. It begins with letting all parties place their initial deposits in the arbiter contract $\mathcal{G}_{ac}$. Later, the client can ask the relays to forward some readings about the blockchain, and then feeds what it receives back to the contract. As such, once the contract hears the feedback from the client, it can leverage the initial deposits to facilitate some proper incentive mechanism, in order to prevent the parties from deviating by rewards and/or punishments, which becomes the crux of our protocol.

For security in the rational setting, the incentive mechanism must be powerful enough to precisely punish misbehaviors (and reward honesty). Our main principle to realize such the powerful incentive is letting the arbiter contract to learn as much as possible regarding how the protocol is actually executed off-chain, so it can precisely punish and then deter any deviations.

Nevertheless, the contract has "handicapped" abilities. We have to carefully design the protocol to circumvent its limits, for the convenience of designing the powerful enough incentive mechanism later.

First, the contract $\mathcal{G}_{ac}$ does not know what the relay nodes forward to the light client off-chain. The contract $\mathcal{G}_{ac}$ has to rely on the client to know what the relays did. At the first glance, the client might cheat the contract, by claiming that it receives nothing from the relays or even forging the relays' messages, in order to avoid paying. To deal with the issue, we require that: (i) the relays authenticate what they forward to the client by digital signatures, so the contract later can verify whether a message was originally sent from the relays, by checking the attached signatures;

The arbiter contract $\mathcal{G}_{ac}$ for m relays ($m = 1$ or 2)

Init. Let state := INIT, deposits := {}, relays := {}, pubKeys := {}, ctr := 0, predicate := $\emptyset$, predicate.N := 0, $T_{end} := 0$

Setup phase

Create. On receiving the message $(\mathsf{create}, k, p, e, d_L, d_F, \Delta T)$ from $\mathcal{LW}$:

- assert state = INIT and ledger$[\mathcal{LW}] \geq \$k \cdot d_L$
- store k, p, e, r, d_L, d_F, and ΔT as internal states
- ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] - \$k \cdot d_L$
- ctr := k and state := CREATED
- send $(\mathsf{deployed}, k, p, e, d_L, d_F, \Delta T)$ to all

Join. On receiving (join, pk_i) from $\mathcal{R}_i$ for first time:

- assert state = CREATED and ledger$[\mathcal{R}_i] \geq \$k \cdot d_F$
- ledger$[\mathcal{R}_i]$:= ledger$[\mathcal{R}_i] - \$k \cdot d_F$
- pubKeys := pubKeys $\cup (\mathcal{R}_i, pk_i)$
- state := READY, if $|$pubKeys$| = m$

Queries phase

Request. On receiving $(\mathsf{request}, \mathsf{P}^\ell)$ from $\mathcal{LW}$:

- assert state = READY and ledger$[\mathcal{LW}] \geq \$(p + e)$
- ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] - \$(p + e)$
- predicate := P^ℓ_T //Note T is the current chain height
- $T_{end} := T + \Delta T$
- send $(\mathsf{quering}, \mathsf{ctr}, \mathsf{predicate})$ to each full node registered in pubKeys
- state := QUERYING

Feedback. On receiving (feedback, responses) from $\mathcal{LW}$ for first time:

- assert state = QUERYING
- store responses for the current ctr

Timer. Upon $T \geq T_{end}$ and state := QUERYING:

- call Incentive(responses, predicate) subroutine
- let ctr := ctr $- 1$
- if ctr > 0 then state := READY
- else state := EXPIRED

Figure 6.2 The contract $\mathcal{G}_{ac}$ written in the conventional pseudocode notations.

(ii) the contract requires the light client to deposit an amount of $\$e$ for each query, which is returned to the client, only if the client reports some forwarded blockchain readings signed by the relays.

Second, the contract has a "handicapped" verifiability, which allows it to efficiently verify a claim of $\mathsf{P}_N^\ell = True$, if being give a succinct proof σ. To leverage the property, the protocol is designed to let the relays attach the corresponding proof σ whenever claiming the provable trueness. Again, such the design is a simple yet still useful way to allow the contract "learn" more about the protocol execution, which later allows us to design powerful incentive mechanisms to precisely punish deviations.

6.5.2 The Light-Client Protocol

In the presence of the contract $\mathcal{G}_{ac}$, our light-client protocol can be formally described as Figure 6.3. To make an oversimplified summary, it first comes with a one-time setup phase, during which the relay(s) and client make initial deposits, which later can be leveraged by the incentive mechanism to fine tune the payoffs. Then, the client can work independently and request the relays to evaluate a few chain predicates up to k times, repeatedly. Since the payoffs are well adjusted, "following the protocol" becomes the rational choice of everyone in each query.

Setup phase. As shown in Figure 6.3, the user of a lightweight client $\mathcal{LW}$ connects to a trusted full node in the setup phase, and announces an "arbiter" smart contract $\mathcal{G}_{ac}$. After the contract $\mathcal{G}_{ac}$ is deployed, some relay full nodes (e.g., one or two) are recruited to join the protocol by depositing an amount of $\$k \cdot d_F$ in the contract. The public keys of the relay(s) are also recorded by contract $\mathcal{G}_{ac}$.

Once the setup phase is done, each relay full node places the initial deposits $\$k \cdot d_F$ and the light client deposit $\$k \cdot d_L$, which will be used to deter their deviations from the protocol. At the same time, $\mathcal{LW}$ records the public keys of the relay(s), and then disconnects the trusted full node to work independently.

The light-client protocol $\Pi_{\mathcal{LW}}$ (where are m relays)

Setup phase

- Protocol for the light client $\mathcal{LW}$:

Create. On instantiating a protocol instance:
- decide $k, p, e, d_L, d_F, \Delta T$ and let $\mathsf{ctr}_{lw} := k$
- send $(\mathsf{create}, k, p, e, d_L, d_F, \Delta T)$ to $\mathcal{G}_{ac}$

Off-line. On receiving $(\mathsf{initialized}, \mathsf{pubKeys})$ from $\mathcal{G}_{ac}$:
- record $\mathsf{pubKeys}$ and disconnect the trusted full node

- Protocol for the relay $\mathcal{R}_i$:

Join. On receiving $(\mathsf{deployed}, k, p, e, d_L, d_F, \Delta T)$ from $\mathcal{G}_{ac}$:
- generate a key pair (sk_i, pk_i) for signature scheme
- send (join, pk_i) to $\mathcal{G}_{ac}$

Queries phase

- Protocol for the light client $\mathcal{LW}$:

Request. On receiving a message (from the higher level app) to evaluate the predicate P^ℓ:
- $T_{feed} := T + 2\Delta T$, and send $(\mathsf{request}, \mathsf{P}^\ell)$ to $\mathcal{G}_{ac}$

Evaluate. On receiving $(\mathsf{response}, \mathsf{ctr}_i, \mathsf{result}_i, \mathsf{sig}_i)$ from the relay $\mathcal{R}_i$:
- assert $T \leq T_{feed}$ and $\mathsf{ctr}_i = \mathsf{ctr}_{lw}$
- assert $\mathsf{vrfySig}(\langle \mathsf{result}_i, \mathsf{ctr} \rangle, \mathsf{sig}_i, pk_i) = 1$
- $\mathsf{responses} := \mathsf{responses} \cup (\mathsf{result}_i, \mathsf{sig}_i)$
- if $|\mathsf{responses}| = m$ then
  output $b \in \{True, False\}$, if responses claim b

Feedback. Upon the global clock $T = T_{feed}$:
- $\mathsf{ctr}_{lw} := \mathsf{ctr}_{lw} - 1$
- send $(\mathsf{feedback}, \mathsf{responses})$ to $\mathcal{G}_{ac}$

- Protocol for the relay $\mathcal{R}_i$:

Respond. On receiving $(\mathsf{quering}, \mathsf{ctr}, \mathsf{P}^\ell_N)$ from $\mathcal{G}_{ac}$:
- $\mathsf{result}_i := \mathsf{evaluate}(\mathsf{P}^\ell_N)$
- $\mathsf{sig}_i := \mathsf{sign}(\langle \mathsf{result}_i, \mathsf{ctr} \rangle, sk_i)$
- send $(\mathsf{response}, \mathsf{ctr}, \mathsf{result}_i, \mathsf{sig}_i)$ to $\mathcal{LW}$

Figure 6.3 The light-client protocol $\Pi_{\mathcal{LW}}$ among the relay(s) and client.

In practice, the setup can be done by using many fast bootstrap methods [123, 91, 80], which allows the user to efficiently launch a personal trusted full node in the PC. The light client (e.g., a smart phone) can connect to the PC to sync. Remark that, besides the cryptographic security parameter λ, the protocol is specified with some other parameters:

- k: The protocol is expired, after the client requests the relay(s) to evaluate some chain predicates for k times.
- $k \cdot d_L$: This is the deposit placed by the client to initialize the protocol.
- $k \cdot d_F$: The initial deposit of a full node to join the protocol as a relay node.
- p: Later in each query, the client shall place this amount to cover the well-deserved payment of the relay(s).
- e: Later in each query, the client shall place this deposit e in addition to p.

Repeatable query phase. Once the setup is done, $\mathcal{LW}$ disconnects the trusted full node, and can ask the relay(s) to query some chain predicates repeatedly. During the queries, $\mathcal{LW}$ can message the arbiter contract, but cannot read the internal states of $\mathcal{G}_{ac}$. Informally, each query proceeds as "request-response-evaluate-payout".

Request. In each query, $\mathcal{LW}$ firstly sends a $\mathsf{request}$ message to the contract $\mathcal{G}_{ac}$, which encapsulates detailed specifications of a chain predicate $\mathsf{P}^{\ell}(\cdot)$, along with a deposit denoted by $\$(p+e)$, where $\$p$ is the promised payment and $\$e$ is a deposit refundable only when $\mathcal{LW}$ reports what it receives from the relays. Once $\mathcal{G}_{ac}$ receives the $\mathsf{request}$ message from $\mathcal{LW}$, $\mathcal{G}_{ac}$ further parameterizes the chain predicate P^{ℓ} as P^{ℓ}_{N}, where $N \leftarrow T$ represents the current global time (i.e., the latest blockchain height).

Response. The the relay full node(s) can learn the predicate P^{ℓ}_{N} under query (whose ground truth is fixed since N is fixed and would not be flipped with the growth of the global timer T), and the settlement of the deposit $\$(p+e)$ by reading the arbiter contract. Then, the relay node can evaluate the predicate P^{ℓ}_{N} with using its local blockchain replica as auxiliary input. When $\mathsf{P}^{\ell}_{N} = True$, the relay node

shall send the client a response message including a proof σ for trueness, which can be verified by the arbiter contract but not the light client; in case $\mathsf{P}_N^\ell = False$, the "honest" full node shall reply to the light client with a response message including $\perp$. In addition, when the relay sends a response message to $\mathcal{LW}$ off-chain,[3] it also authenticates the message by attaching its signature (which is also bounded to an increasing only counter to prevent replaying).

Evaluate & Feedback. Upon receiving a response message from the relay $\mathcal{R}_i$, $\mathcal{LW}$ firstly verifies that it is authenticated by a valid signature sig_i. If sig_i is valid, $\mathcal{LW}$ parses the response message to check whether $\mathcal{R}_i$ claims $\mathsf{P}_N^\ell = True$ or $\mathsf{P}_N^\ell = False$. If receiving consistent response message(s) from all recruited relay(s), the light client decides this consistently claimed $True/False$. Then the client sends a feedback message to the contract $\mathcal{G}_{ac}$ with containing these signed response message(s). Remark we do *not* assume the client follows the protocol to output and feeds back to the contract. Instead, we focus on proving "following the protocol to decide an output" is the sequential rational strategy of the client.

Payout. Upon receiving the feedback message sent from the light client, the contract $\mathcal{G}_{ac}$ shall invoke the Incentive subroutine to facilitate some payoffs. Functionality-wise, the payoff rules of the incentive subroutine would punish and/or reward the relay node(s) and the light client, such that none of them would deviate from the protocol.

Remark on correctness. It is immediate to see the correctness: when all parties are honest, the relay(s) receive the payment pre-specified due to incentive mechanism in the contract, and the client always outputs the ground truth of chain predicate.

Remark on security. The security would depend on the payoffs clauses facilitated by the incentive subroutine, which will be elaborated in later subsections as we

[3] Note that we assume the off-chain communication can be established on demand in the dissertation, which in practice can be done through a name service or "broadcasting" encrypted network addresses through the blockchain [101].

intentionally decouple the protocol and the incentive design. Intuitively, if the incentive subroutines does nothing, there is no security to any extent; since following the protocol is not any variant of equilibrium. Thus, the incentive mechanism must be carefully designed to finely tune the payoffs, in order to make the sequential equilibrium to be following the protocol.

6.6 Adding Incentives for Security

Without a proper incentive subroutine, our simple light-client protocol is seemingly insecure to any extent, considering at least the relay nodes are well motivated to cheat the client. To mitigate the issue, this section formally treats the light-client protocol as an extensive game, and then studies on how to squeeze most out of the "handicapped" abilities of the arbiter contract to design proper incentives, such that the utility function of the game can be well adjusted to deter any party from deviating at any stage of the protocol's extensive game.

6.6.1 Challenges of Designing Incentives

The main challenge of designing proper incentives to prevent the parties from deviating is the "handicapped" abilities of the arbiter contract $\mathcal{G}_{ac}$: there is no proof for a claim of $\mathsf{P}_N^{\ell} = False$, so $\mathcal{G}_{ac}$ cannot directly catch a liar who claims bogus $\mathsf{P}_N^{\ell} = False$. We conquer the above issue in the rational setting, by allowing the contract $\mathcal{G}_{ac}$ to believe unverifiable claims are correctly forwarded by rational relay(s), even if no cryptographic proofs for them. Our solution centers around the fact: if a claim of $\mathsf{P}_N^{\ell} = False$ is actually fake, there shall exist a succinct cryptographic proof for $\mathsf{P}_N^{\ell} = True$, which can falsify the bogus claim of $\mathsf{P}_N^{\ell} = False$. As such, we derive the basic principles of designing proper incentives in different scenarios: (i) When there are two non-cooperative relays, we create an incentive to leverage non-cooperative parties to audit each other, so sending fake $\mathsf{P}_N^{\ell} = False$ become

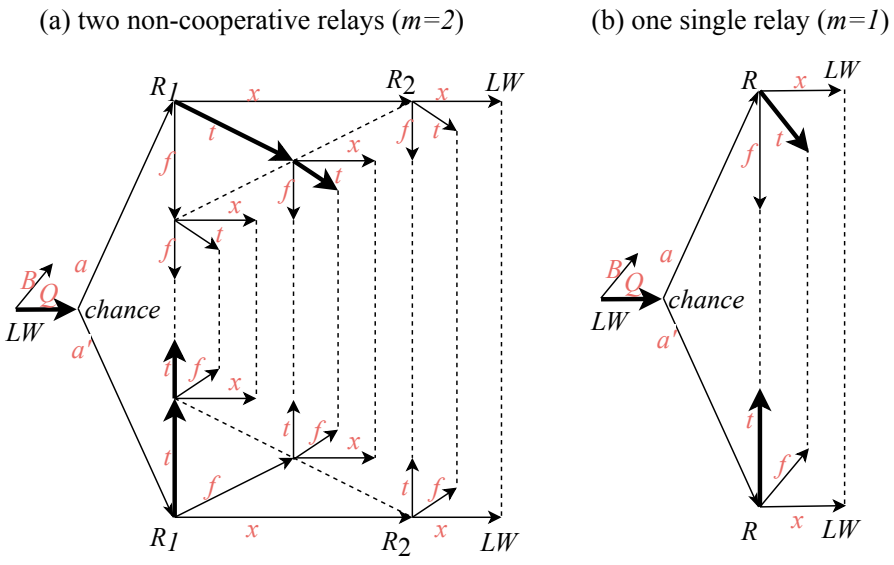

Figure 6.4 The repetition structure of the light-client game in one query: (a) two non-cooperative relays (i.e., Γ_2); (b) one single relay (i.e., Γ_1). The last actions of the client are not shown for presentation simplicity.

irrational and would not happen; (ii) When there is only one relay node (which models that there are no non-cooperative relays at all), we somehow try an incentive design to let the full node "audit" itself, which means: the relay would get a higher payment, as long as it presents a verifiable claim instead of an unverifiable claim. So the relay is somehow motivated to "audit" itself.

6.6.2 "Light-Client Game" of the Protocol

Here we present the structure of "light-client game" for the simple light-client protocol presented in the earlier section. We would showcase how the extensive game does capture (i) all polynomial-time computable strategies and (ii) the incomplete information received during the course of the protocol.

Game structure for two relays. For the case of recruiting two (non-cooperative) relays, we denote the "light-client" game as Γ_2^k. It has a repetition structure (i.e., a stage game Γ_2) that can be repeated up to k times as shown in Figure 6.4 (a), since the client can raise queries for up to k times in the protocol. More precisely, for each

query, the protocol proceeds as the incomplete-information extensive stage game Γ_2 that can be described as follows.

Client acts by making a query or not. The client moves, with two optional actions Q and B. Q denotes "sending a request message to query", and B denotes "others" (including abort). The game only proceeds when the light client acts Q.

Chance acts by choosing the truth. At the history Q, the special player "*chance*" moves, with two possible actions a and a'. Let a represent $\mathsf{P}_N^\ell = True$, and a' for $\mathsf{P}_N^\ell = False$. The occurrence of a and a' follows an arbitrary distribution $[\rho, 1-\rho]$. Note the action of *chance* can be observed by the relay full nodes but not the client.

Relay acts by responding the client. At histories $Q(a|a')$, [4] the relay node $\mathcal{R}_1$ acts, with three available actions $\{t, f, x\}$:

- The action t means $\mathcal{R}_1$ forwards the ground truth of P_N^ℓ to $\mathcal{LW}$ (with attaching correct "proofs" if there are any).[5]
- The action f represents that $\mathcal{R}_1$ forwards the opposite of the ground truth of chain predicate to $\mathcal{LW}$.
- The action x means as others, including abort and some attempts to break the cryptographic primitives.

The other relay acts by responding the relay. At histories $Q(a|a')(t|f|x)$, it is the turn of $\mathcal{R}_2$ to move. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are non-cooperative, histories $Qa(t|f|x)$ make of an information set of $\mathcal{R}_2$ denoted by I_1, and similarly, $Qa'(t|f|x)$ is another information set I_2. At either I_1 or I_2, $\mathcal{R}_2$ has three actions $\{t, f, x\}$, which can be understood as same as the actions of $\mathcal{R}_1$ at $Q(a|a')$, since $\mathcal{R}_1$ and $\mathcal{R}_2$ are exchangeable notations.

Client acts by feeding back and evaluating. Then the game Γ_2 reaches one of the histories $Q(a|a')(t|f|x)(t|f|x)$. As shown in Figure 6.4, the client $\mathcal{LW}$ is facing

[4] Remark that we are using standard regular expressions to denote the histories and information set. For example, $Q(a|a')$ represents $\{Qa, Qa'\}$

[5] There exists another strategy to claim the truth of the predicate when the predicate is indeed true, but with invalid proof. This strategy is strictly dominated and would not be adopted at all, since it neither fools the client, nor get through the verification of contract to get any reward. We therefore omit it.

nine information sets[6]: $I_1^{\mathcal{LW}} = Q(att|a'ff)$, $I_2^{\mathcal{LW}} = Q(atf|a'ft)$, $I_3^{\mathcal{LW}} = Q(at|a'f)x$, $I_4^{\mathcal{LW}} = Q(aft|a'tf)$, $I_5^{\mathcal{LW}} = Q(aff|a'tt)$, $I_6^{\mathcal{LW}} = Q(af|a't)x$, $I_7^{\mathcal{LW}} = Q(axt|a'xf)$, $I_8^{\mathcal{LW}} = Q(axf|a'xt)$, $I_9^{\mathcal{LW}} = Q(a|a')xx$. At these information sets, the light client shall choose a probabilistic polynomial-time ITM to: (i) send a feedback message back to the contract, and (ii) decide an output. As such, the available actions of the client at each information set can be interpreted as:

- From $I_1^{\mathcal{LW}}$ to $I_5^{\mathcal{LW}}$, the client receives two response messages from both relays in time, and it can take an action out of $\{T, L, R, X\} \times \{A, A', O\}$: T means to report the arbiter contract $\mathcal{G}_{ac}$ both of the response; L (or R) represents that $\mathcal{LW}$ reports to the contract $\mathcal{G}_{ac}$ only one response message sent from $\mathcal{R}_1$ (or $\mathcal{R}_2$); X represents others, including abort; A means to output $True$; A' is to output $False$; O denotes to output nothing.

- Through $I_3^{\mathcal{LW}}$ to $I_8^{\mathcal{LW}}$, the client $\mathcal{LW}$ receives only one response message from $\mathcal{R}_1$ (or $\mathcal{R}_2$), and can take an action out of $\{T, X\} \times \{A, A', O\}$: T means to report the contract $\mathcal{G}_{ac}$ the only response message that it receives; X means to do others, including abort; A, A' and O have the same concrete meaning as before.

- At $I_9^{\mathcal{LW}}$, $\mathcal{LW}$ receives nothing from the relays in time, it can take an action out of $\{T, X\} \times \{A, A', O\}$: T can be translated as to send the contract nothing until the contract times out, X represents others (for example, trying to crack digital signature scheme); A, A' and O still have the same meaning as before.

After all above actions are made, the protocol completes one query, and can go to the next query, as long as it is not expired or the client does not abort. The protocol's game Γ_2^k (capturing all k queries) can be inductively defined by repeating the above structure up to k times.

Game structure for one relay. For the case of recruiting only one relay to request up to k queries, we denote the protocol's "light-client" game as Γ_1^k. As shown in Figure 6.4 (b), it has a repetition structure (i.e., the stage game Γ_1) similar to the game Γ_2, except few differences related to the information sets and available actions of the light client. In particular, when the client receives response from the only relay,

[6]Remark that the histories $Qatt$ and $Qa'ff$ cannot be distinguished by the light client, because for the light client, both of them correspond that two claims of $True$. All the nine information sets of the light client can be translated similarly.

it would face three information sets, namely, $Q(at|a'f)$, $Q(af|a't)$ and $Q(ax|a'x)$, instead of nine in Γ_2. At each information set, the client always can take an action out of $\{T, X\} \times \{A, A', O\}$, which has the same interpretation in the game Γ_2. Since Γ_1 is extremely similar to Γ_2, we omit such details here.

What if no incentive? If the arbiter contract facilitates no incentive, the possible execution result of the protocol can be concretely interpreted due to the economic aspects of our model, which are:

- *When the client is fooled.* The client loses $\$v$, and the relay $\mathcal{R}_i$ earns $\$v_i$. Note $\$v$ is related to the value attached to the chain predicate under query, say the transacted amount, due to our economic model; and $\$v_i$ is the malicious benefit earned by $\mathcal{R}_i$ if the client if fooled.

- *When the client outputs the ground truth.* The relay would not earn any malicious benefit, and the client would not lose any value attached to the chain predicate either.

- *When the client outputs nothing.* The relay would lose $\$c$, which means it will launch its own (personal) full node to query the chain predicate. In such case, the relay would not learn any malicious benefit.

It is clear to see that without proper incentives to tune the above outcomes, the game cannot reach a desired equilibrium to let all parties follow the protocol, because at least the relays are well motivated to cheat the client. Thus we leverage the deposits placed by the client and relay(s) to design simple yet still useful incentives in next subsections, such that we can fine tune the above outcome to realize a utility function obtaining desired equilibrium, thus achieving security.

6.6.3 Basic Incentive Mechanism

The incentive subroutine takes the feedback message sent from the client as input, and then facilitates rewards/punishments accordingly. After that, the utility function of the "light-client game" is supposed to be well tuned to ensure security. Here we will present such the carefully designed incentive subroutine, and analyze the incentive makes the "light-client game" secure to what extent.

Basic incentive **for two relays**. If two non-cooperative relays can be recruited, the incentive subroutine takes the feedback message from the client as input, and then facilitates the incentives following hereunder general principles:

- It firstly verifies whether the feedback from the light client indeed encapsulates some responses that were originally sent from $\mathcal{R}_1$ and/or $\mathcal{R}_2$ (w.r.t. the current chain predicate under query). If feedback contains two validly signed responses, return $\$e$ to the client; If feedback contain one validly signed response, return $\$e/2$ to the client.

- If a relay claims $\mathsf{P}_N^\ell = True$ with attaching an invalid proof σ, its deposit for this query (i.e., $\$d_F$) is confiscated and would not receive any payment.

- When a relay sends a response message containing $\perp$ to claim $\mathsf{P}_N^\ell = False$, there is no succinct proof attesting the claim. The incentive subroutine checks whether the other relay full node provides a proof attesting $\mathsf{P}_N^\ell = True$. If the other relay proves $\mathsf{P}_N^\ell = True$, the cheating relay loses its deposit this query (i.e., $\$d_F$) and would not receive any payment. For the other relay that falsifies the cheating claim of $\mathsf{P}_N^\ell = False$, the incentive subroutine assigns it some extra bonuses (e.g., doubled payment).

- After each query, if the contract does not notice a full node is misbehaving (i.e., no fake proof for truthness or fake claim of falseness), it would pay the node $\$p/2$ as the basic reward (for the honest full node). In addition, the contract returns a portion of the client's initial deposit (i.e., $\$d_L$). Moreover, the contract returns a portion of each relay's initial deposit (i.e., $\$d_F$), if the incentive subroutine does not observe the relay cheats during this query.

The rationale behind the incentive design is straightforward. First, during any query, the rational light client will always report to the contract whatever the relays actually forward, since the failure of doing so always causes strictly less utility, no matter the strategy of the relay full nodes; Second, since the two relay full nodes are non-cooperative, they would be incentivized to audit each other, such that the attempt of cheating the client is deterred. To demonstrate above general reward/punishment principles of the incentive mechanism are implementable, we concretely instantiate its pseudocode that are deferred to subsection A.2.1.

Basic incentive **for one relay**. When any two recruited relays might collude, the situation turns to be pessimistic, as the light client is now requesting an unknown

information from only a single distrustful coalition. To argue security in such the pessimistic case, we consider only one relay in the protocol. To deal with the pessimistic case, we tune the incentive subroutine by incorporating the next major tuning (different from the the incentive for two relays):

- If the relay claims $\mathsf{P}_N^{\ell} = False$, its deposit is returned, but it receives a payment less than $\$p$, namely, $\$(p - r)$ where $\$r \in [0, p]$ is a parameter of the incentive.
- Other payoff rules are same to the basic incentive mechanism for two non-cooperative relays.

To demonstrate the above delicately tuned incentive is implementable, we showcase its pseudocode that is deferred to subsection A.2.2.

6.6.4 Security Analysis for Basic Incentive

Utility function. Putting the financial outcome of protocol executions together with the incentive mechanism, we can eventually derive the utility functions of game Γ_2^k and game Γ_1^k, inductively. The formal definitions of utilities are deferred to Appendix A.3. Given such utility functions, we can precisely analyze the light-client game Γ_2^k (and the game Γ_1^k) to precisely understand our light-client protocol is secure to what extent.

Security theorems of basic incentive. For the case of two non-cooperative relays, the security can be abstracted as Theorem 4:

Theorem 4. *If the relays that join the protocol are non-cooperative, there exists a $negl(\lambda)$-sequential equilibrium of Γ_2^k that can ensure the game Γ_2^k terminates in a terminal history belonging $\mathbf{Z}_{\mathbf{good}} := (QattTA|Qa'ttTA')\{k\}$ (i.e., no deviation from the protocol), conditioned on $d_F + p/2 > v_i$, $d_L > (p + e)$, and $c > p$. In addition, the rational client and the rational relays would collectively set up the protocol, if $p > 0$.*

For the case of one single relay (which models cooperative relays), the security of the basic incentive mechanism can be abstracted as:

Theorem 5. *In the pessimistic case where is only one single relay (which models a coalition of relays), there exists a $negl(\lambda)$-sequential equilibrium that can ensure the game Γ_1^k terminates in $\mathbf{Z_{good}} := (QatTA|Qa'tTA')\{k\}$ (i.e., no deviation from the protocol), conditioned on $d_F+p-r > v_i$, $r > v_i$, $d_L > (p+e)$, and $c > p$. Moreover, the rational client and the rational relay would collectively set up the protocol, if $p-r > 0$ and $p > 0$.*

Interpretations of Theorem 4. The theorem reveals that: conditioned on there are two non-cooperative relays, the sufficient conditions of security are: (i) the initial deposit d_F of relay node is greater than its malicious benefit v_i that can be obtained by fooling the client; (ii) the initial deposit d_L of the client is greater than the payment p plus another small parameter e; (iii) for the light client, it. The above conclusion essentially hints us how to safely set up the light-client protocol to instantiate a cryptocurrency wallet in practice, that is: let the light client and the relays finely tune and specify their initial deposits, such that the client can query the (non)existence of any transaction, as long as the transacted amount of the transaction is not greater than the initial deposit placed by the relay nodes.

Interpretations of Theorem 5. The theorem states that: even in an extremely hostile scenario where only one single relay exists, deviations are still prevented when fooling the light client to believe the non-existence of an existing transaction does not yield better payoff than honestly proving the existence. The statement presents a feasibility region of our protocol that at least captures many important DApps (e.g., decentralized messaging apps) in practice, namely: fooling the client is not very financially beneficial for the relay, and only brings a payoff v_i to the relay; so as long the client prefers to pay a little bit more than v_i to read a record in the blockchain, no one would deviate from the protocol.

6.6.5 Augmented Incentive

This subsection further discusses the pessimistic scenario that no non-cooperative relays can be identified, by introducing an extra assumption that: at least one public full node (denoted by $\mathcal{PFN}$) can monitor the internal states of the arbiter contract at a tiny cost ε w.r.t. economic factor (say zero through the dissertation for the convenience of analysis), and does not cooperate with the only recruited relay. This extra rationality assumption can boost an incentive mechanism to deter the relay and client from deviating from the light-client protocol. Here we present this augmented incentive design, and analyze its security guarantees.

Augmented incentive **for one relay**. The tuning of the incentive mechanism stems from the observation that: if there is *any* public full node that does not cooperate with the recruited relay (and monitor the internal states of the arbiter contract), it can stand out to audit a fake claim about $\mathsf{P}_N^\ell = False$ by producing a proof attesting $\mathsf{P}_N^\ell = True$. Thus, we slightly tune the incentive subroutine (by adding few lines of pseudocode), which can be summarized as:

- When a relay forwards a response message containing $\perp$ to claim $\mathsf{P}_N^\ell = False$, the incentive subroutine shall wait few clock periods (e.g., one). During the waiting time, the public full node is allowed to send a proof attesting $\mathsf{P}_N^\ell = True$ in order to falsify a fake claim of $\mathsf{P}_N^\ell = False$; in this case, the initial deposit d_F of the cheating relay is confiscated and sent to the public full node who stands out to prove the cheating behavior.

- Other payoff rules are same to the basic incentive mechanism, so do not involve the public full node.

We defer the formal instantiation of the above augmented incentive mechanism in the standard pseudocode format to subsection A.2.3.

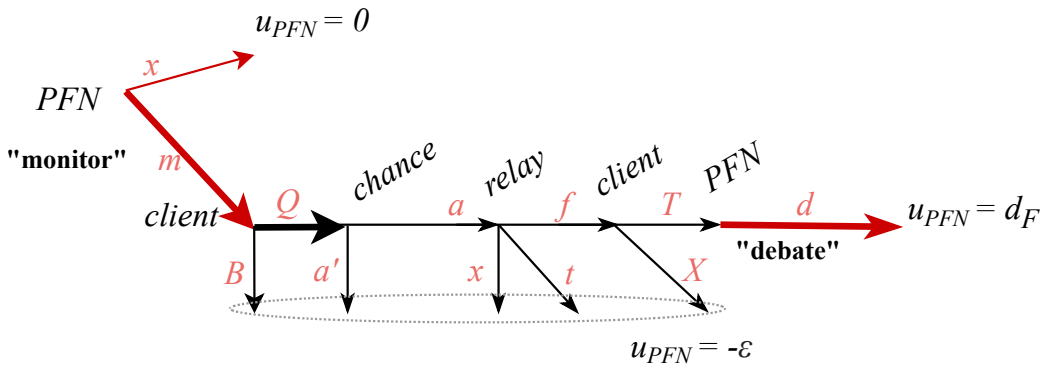

Figure 6.5 The induced game G_1, if having a non-cooperative public full node.

6.6.6 Security Analysis for Augmented Incentive

Augmented "light-client game". By the introduction of the extra incentive clause, the "light-client game" Γ_1 is extended to the augmented light-client game G_1. As shown in Figure 6.5, the major differences from the original light-client game Γ_1 are two aspects: (i) the public full node ($\mathcal{PFN}$) can choose to monitor the arbiter contract (denoted by m) or otherwise (x) in each query, which cannot be told by the relay node due to the non-cooperation, and (ii) when the ground truth of predicate is true, if the relay cheats, $\mathcal{PFN}$ has an action "debate" by showing the incentive mechanism a proof attesting the predicate is true, conditioned on having taken action m.

The security intuition thus becomes clear: if the recruited relay chooses a strategy to cheat with non-negligible probability, the best strategy of the public full node is to act m, which on the contrary deters the relay from cheating. In the other word, the relay at most deviate with negligible probability.

Security theorem of augmented incentive. Now, in the augmented game G_1, if the recruited relay deviates when the predicate is true with non-negligible probability, the rational $\mathcal{PFN}$ would act m and then d, which will confiscate the initial deposit of the

relay and deters it from cheating. More precisely, the security due to the augmented incentive mechanism can be summarized as:

Theorem 6. *Given the augmented incentive mechanism, there exists a* $negl(\lambda)$*-sequential equilibrium of the augmented light-client game* G_1 *such that it can ensure the client and the relay would not deviate from the protocol except with negligible probability, conditioned on* $d_F > v_i$, $d_L > (p + e)$, $c > p$ *and a non-cooperative public full node that can "monitor" the arbiter contract costlessly. Also, the client and the relay would set up the protocol, if* $p > 0$.

Interpretations of Theorem 6. The economics behind the theorem can be translated similarly to Theorem 4.

6.7 Instantiation of the Light-Client Protocol

Here we shed light on the concrete instantiation of the protocol in practice and emphasize some tips towards feasibility. Though the current permissionless blockchains (e.g., Ethereum) are suffering from many baby-age limitations (e.g., high cost of on-chain resources, low throughput, and large latency), a straight instantiation of our light-client protocol has been arguably practical.

On- and off-chain feasibility. As shown in Table 6.1, we instantiate the protocol atop Ethereum (with recruiting one relay and using the basic incentive mechanism, c.f., Section 6.6.4), and measure the costs of repeatedly evaluating five chain predicates about the (non)existence of different Ethereum transactions.

Due to the simple nature of our protocol, the off-chain cost of the light client is constant and essentially tiny, as it only needs: (i) to store two public keys, (ii) to instantiate two secure channels to connect the relay nodes (e.g., the off-chain response message is $<$ 1KB), (iii) to verify two signatures and to compute a few hashes to few verify Merkle tree proof(s) in the worst case.

Table 6.1 An Instantiation (Basic Incentive of One Relay for 5 Ether Transactions)

txid queried for evaluating (non)existence	Gas of request ($\mathcal{LW} \rightarrow \mathcal{G}_{ac}$)	Size of response ($\mathcal{R}_i \rightarrow \mathcal{LW}$)	Gas of feedback ($\mathcal{LW} \rightarrow \mathcal{G}_{ac}$)
`0x141989127035`...	71,120 gas	947 Byte	199,691 gas
`0x0661d6e95ab1`...	41,120 gas	951 Byte	251,480 gas
`0x949ae094deb0`...	41,120 gas	949 Byte	257,473 gas
`0x1e39d5b4b46d`...	41,120 gas	985 Byte	339,237 gas
`0xfe28a4dffb8e`...	41,120 gas	951 Byte	248,119 gas

Note: The code is available at https://github.com/yylluu/rational-light-client. These five transactions under query are included by the Ethereum blockchain, and we choose them from the blocks having various sizes. For example, the transaction `0x1e39`... is in a block having 263 transactions, which indicates the evaluation has captured some worst cases of reading from large blocks. In lieu of EIP-210 [54] which currently is not available in EVM, we hardcore the needed blockhashes (in the contract) to measure the actual on-chain overhead (as if EIP-210 is available).

Besides the straightforward off-chain efficiency, the on-chain cost is also low. Particularly, the client only sends two messages (i.e., request and feedback) to the contract, which typically costs mere 300k gases in the worst case as shown in Table 6.1. At the time of writing (Jan/13/2020), ether is $143 each [35], and the average gas price is 10 Gwei [51], which corresponds to a cost of only $0.43.

Latency. If the network diffuse functionality [57] can approximate the latency of global Internet [10, 58], the delay of our light-client protocol will be dominated by the limitations of underlying blockchain. The reasons are: (i) many existing blockchains have limited on-chain resources, and the miners are more willing to pack the transactions having higher transaction fees [114, 152, 10], and (ii) messaging the contract suffers from the intrinsic delay caused by underlying consensus. For example, in Ethereum network at the time of writing, if the light client sets its transactions at the average gas price (i.e., 9 Gwei), the latency of messaging the contract on average will include: (i) 10 blocks (about two minutes) [51] for being mined, plus (ii) a few more blocks for confirmations (another a few minutes) [62]. If the client expects the protocol to proceed faster, it can set higher gas price (e.g., 22 Gwei per gas), which causes its messages to be included after 2-3 blocks on average (i.e., about

30-45 seconds) [51], though the on-chain cost increases by 144%. After all, once the underlying blockchain goes through the baby-age limitations, the protocol's latency can be further reduced to approximate the actual Internet delay.

Who are the relays? The light-client protocol can be deployed in any blockchain supporting smart contracts. The relays in the protocol can be the full nodes of the chain (e.g., the full nodes of two competing mining pools) that are seeking the economic rewards by relaying blockchain readings to the light clients, so it is reasonable to assume that they can maintain the full nodes to evaluate chain predicates nearly costlessly. Even in the extremely adversarial environment where the light client has no confidence in the non-collusion of any two full nodes, the protocol can still be finely tuned (e.g., increase the rewards) to support at least a wide range of useful low-value chain predicates.

The initial setup. We explicitly decouple the presentations of "protocol" and "incentive mechanism" to provide the next insight: the initial deposit is not necessary to be cryptocurrency as our design, and it can be any form of "collateral", such as business reputations and subscriptions; especially, if the "deposit" is publicly known off the chain, the setup phase also becomes arguably removable, as the light client has no need to rely on a personal full node to verify the correct on-chain setup of the initial deposit anymore.

The amount of initial deposits. One might worry that the amount of initial deposit, especially when considering that the needed initial deposit is linear to the number of queries to be asked. In practice, a few instantiations can avoid the deposit from being too large to be feasible. One of those is to let the light client and the relay node(s) to negotiate before (or during) the setup phase to choose a moderate number of queries to support, and then they can periodically reset the protocol, which is feasible as the light client user can afford to periodically reset her personal full node for a short term to handle the setups. Another possibility, as already mentioned, is

relying on some external "collateral" (e.g., reputations and subscriptions) to replace the deposit of cryptocurrency in the protocol.

6.8 Summary

Different from the existing light client protocols in cryptographic settings, this dissertation takes a different path and systematic study the problem in the game-theoretic setting and solve it via mechanism design. Assuming lightweight nodes and relaying full nodes are rational, it leverages the smart contract to facilitate a simple incentive mechanism such that being honest is their best choice, i.e., for their highest utility, full nodes must faithfully relay blockchain's states to the light client for being paid.

In a bit more details, after a one time setup with a trusted full node checking the relay nodes have correctly deposit to the dedicated designed incentive smart contract, the lightweight client can repeated query. First, it posts detailed specifications about the query along with a deposit (which is larger than the payment promised for the query) to the incentive smart contract via a transaction (which can be done because writing to blockchain is trivial for any Internet nodes including the lightweight ones); The relay full nodes will see the above transaction in its local ledger replica, and relay the query results to the lightweight node off-chain for the promised payment; The lightweight node is incentivized to report to the contract all relayed results that it receives, otherwise it gets a fine; The contract verifies the correctness of these relayed results and pay the full nodes accordingly. The amount of the required deposits and economic punishment if cheating is finely tuned such that the game achieves a desired refinement of Nash equilibrium. This means, it is guaranteed that (i) the rational full nodes will relay the blockchain's states correctly and (ii) a rational lightweight node would like to pay the full nodes as promised.

These procedures empower a superlight protocol that enables a light client to recruit several relay full nodes (e.g., one or two) to securely evaluate a general class

of predicates about the blockchain. To summarize, the core technical contributions of this Chapter are four-fold.

First, our light-client protocol can be bootstrapped in the rational setting, efficiently and generically. The protocol is superlight, in the sense that the client can go off-line and wake up any time to *evaluate* a general class of chain predicates at a tiny constant computationally cost; as long as the truthness or falseness of these chain predicates is reducible to few transactions' inclusion in the blockchain.

Second, this generic protocol gets rid of the dependency on consensuses and can be deployed in nearly any permissionless blockchain (e.g., Turing-complete blockchains [23, 152]) without even velvet forks [157], thus supporting the promising PoS type of consensuses.

Third, it conducts a systematic study to understand whether, or to what extent, the light-client protocol is secure in the rational setting (without trusted third-parties). It makes non-trivial analyses of the incomplete-information extensive game induced by our light-client protocol and conduct a comprehensive study to understand how to design the incentives to achieve security in different scenarios, from the standard setting of non-cooperative full nodes to the pessimistic setting of colluding full nodes.

Finally, the protocol enables the rational client to *evaluate* non-existence of a given transaction besides the existence, i.e., the rational client can be convinced by the rational full nodes that a given transaction is *not* in the blockchain. That provides a simple way to performing non-existence "proof". In contrast, relevant studies in the cryptographic setting either give up non-existence proof [114, 83] or have to heavily modify the blockchain's data structure [18, 110].

CHAPTER 7

OTHER PERTINENT RESULTS

Besides focusing on decentralized applications for specific crowdsourcing scenarios, it also initiates a few other relevant studies, ranging from various high-level decentralized application to the underlying core blockchain techniques, thus inferring a much larger scope of the research (c.f., Chapter 8 for more details on the future vision). Here down below are some brief discussions on these preliminary results.

Decentralized content delivery with strong fairness [72]. P2P content delivery is thought of a cost-saving alternative to replace existing content delivery network (CDN), the latter of which suffers from extremely high cost. Nevertheless, to ensure P2P content delivery to function as expected, it does require to enforce carefully designed incentive mechanism to ensure fairness among the participants, that means each participant earns proportional to what it contributes. Most existing P2P content delivery frameworks either only achieve weaker versions of fairness or rely on unrealistic assumptions that are elusive in practice. Here it designs a decentralized content delivery system atop the blockchain. This system satisfies the fairness for the content owner, the deliverer, and the consumer, each of which can enjoy guaranteed fairness against the other two (probably colluding) parties. For the first time, it achieves a stronger and realistically meaningful notion of fairness, such that the content deliverer would be paid (nearly) proportional to the bandwidth that it uses, despite the influence of the malicious consumer and the malicious content owner. This system still remains high efficiency and attains low on-chain computational cost, which is close to optimal. Especially in the optimistic case that all parties are honest, the on-chain is as small as a few US cents.

Privacy-preserving decentralized retailer gift card ledger [100]. Though the retailer gift card has been an ultra-practical marketing tactic to attract customers to spend more, it on the contrary also places a great number of customers in troublesome situations due to its current limitations. First, dealing with unwanted gift cards is often time-consuming, costly or even risky due to the frequent occurrences of gift card resale frauds. Worse still, the issuance and redemption of gift cards happen inside the retailer as in a "black-box", indicating that a compromised retailer can cheat customers (or even third-party auditors) to deny the issuances of some unredeemed gift cards. This paper proposes a practical middle-layer solution based on blockchain to address the fundamental issues of the existing gift card system, with incurring minimal changes to the current infrastructure. In addition, it enjoy other needed security requirements such as (i) keeping the gift card balances confidential against corrupted blockchain nodes and (ii) maintaining easy giftability such that any cards can be gifted without knowing the recipient's public key.

Optimal validated asynchronous Byzantine agreement (VABA) [97]. VABA is fundamental for critical atomic broadcast in the asynchronous network [24], and turns to be a needed building block for achieving consortium blockchain in the unstable global Internet environment. It was left as an open problem to asymptotically reduce the $O(\ell n^2 + \lambda n^2 + n^3)$ communication (where n is the number of parties, ℓ is the input length, and λ is the security parameter). Recently, [1] removed the n^3 term to partially answer the question only if the input is small in size. However, in many other typical use-cases, for example, building atomic broadcast around VABA, the needed input length (of VABA) ℓ shall be larger than λn, and thus the communication is still dominated by the ℓn^2 term so does not fully answer the open problem raised in [24]. This work fills the gap and answers the remaining part of the open problem by presenting two VABA protocols with $O(\ell n + \lambda n^2)$ communicated bits without scarifying optimal fault-tolerance and optimal message complexity, which

immediately corresponds to a better way to realizing robust yet still high-performing consortium blockchain suitable for the fluctuating real-world Internet.

CHAPTER 8

SUMMARY OF THE DISSERTATION

8.1 Conclusion

To reduce over-reliance on third-party platforms in the traditional crowdsourcing sector, it becomes enticing to use the novel blockchain technologies and seek for the rising decentralization paradigm. However, if decentralized crowdsourcing is naively engineered, the inherent restrictions of the blockchain might backfire and greatly harm the basic utilities of these trivial designs. There still remains a huge gap between the ideal end-goals of decentralized crowd-sharing economy and the problematic prior art of blockchain-based crowdsourcing applications.

Noticing that huge gap, this dissertation proposes a package of secure yet still efficient solutions to go through the intrinsic issues of blockchains, such that one can achieve truely meaningful decentralized crowdsourcing in practice, which means: (i) the basic utilities can be guaranteed with high security assurance, despite of the influence of attacks launched by (probably any) Internet nodes through the open blockchain network; (ii) the system remains highly feasible, although necessary security-driven designs have to be added for the provably secure utilities. In short, the dissertation focuses on the following two main categories of applications.

On the one hand, Chapters 3 and 4 initiate the study of *private knowledge solicitation atop the blockchain*. This line of research identifies privacy as an indispensable security requirement to make decentralized knowledge solicitation meaningful; otherwise, the well-known transparency of blockchain would open new attack surface to allow literally anyone to free-ride (namely, reap credits without contributing), which further causes that no rational users would contribute high-quality data anymore and results in sorta tragedy of the commons to completely fail

the system. Chapter 3 proposes ZebraLancer to adapt the advanced zero-knowledge proof framework (zk-SNARK) to let the requester prove the quality of encrypted answers, which achieves critical on-chain efficiency due to constant proof size and nearly constant verification time; Chapter 4 proposes Dragoon to further improve this result by designing a special-purpose proving scheme for the concrete quality rules widely adopted by Amazon's MTurk, thus improving the efficiency by a few orders of magnitude without scarifying privacy. More surprisingly, Dragoon is so efficient that it is on-chain handling cost can be less than the handling fee charged by Amazon MTurk for the same ImageNet tasks.

On the other hand, Chapters 5 and 6 initiate systematic studies on *decentralized solicitation of "computing resources"*. The challenges of decentralized crowdsourcing of computing power stem from the fact that the blockchain has to validate the (dis)honesty of computations off the blockchain, otherwise the recruited workers can earn rewards without committing any actual efforts. Traditional solutions to resolve the challenges (in the cryptographic setting) require advanced cryptographic primitives such as verifiable computation, the state-of-art of which, unfortunately, is far from being practical for complex computations. For real-world practicality, this dissertation explicitly deviates from conventional cryptographic approaches, and seeks for mitigation in the game-theoretic setting. At high-level, the selfishness of non-cooperative workers is leveraged to audit each other. When some workers is misbehaving, the other non-colluding workers would report to a dedicatedly designed smart contract about the misbehavior and instruct the contract to punish. Eventually, the designs realize some desirable refinements of Nash equilibrium, under which no rational parties would deviate from being honest and thus ensure the desired outcome despite of adversarial behaviors.

These results not only bridge the gap towards the end-goal of deploying meaningful decentralized crowdsourcing in reality, but also turn to be promising to

bootstrap blockchains themselves to conquer their own issues, for example, to support complicated smart contracts and to enable superlight clients.

8.2 Reflection

More generally, this dissertation develops and demonstrates two distinct methodologies to simultaneously achieve the critical security and efficiency required by the decentralized applications atop (permissionless) blockchains.

In the cryptographic model, it showcases that **specific-purpose optimizations for concrete decentralization** can be dedicated to significantly reduce the cryptographic overhead incurred by the indispensable security requirements such as privacy (see Chapters 3 and 4 for hints). In particular, by considering the specific demands of real-world use-cases, some concretely efficient solutions can be worked out for real-world problems. The idea behind the methodology is to remove needless generality in specific circumstances, thus achieving high efficiency while remaining any indispensable security guarantees.

In the game-theoretic setting, it demonstrates that **secure decentralization can be efficiently attained via proper incentive mechanisms** through using smart contracts and cryptocurrencies. In contrast to the cryptographic setting where an implicit adversary can corrupt and fully control some participating parties to arbitrarily misbehave, the game-theoretic approach considers that each party neither completely honest nor arbitrarily malicious, but is rational to seek for its own highest benefits. Embracing this game-theoretic methodology, Chapters 5 and 6 consider how to design proper incentives atop the blockchain to (i) recruit computing power for machine learning tasks and (ii) employ blockchain full nodes to relay blockchain readings, respectively. These game-theoretic solutions are much more efficient than the best so-far solutions in the conventional cryptographic setting.

The previous two methodologies essentially correspond to a couple of general ways that can be applied for a broader array of decentralized applications to achieve indispensable security as well as real-world practicability.

In greater detail, the research can immediately benefits many categories of decentralized applications in much larger scope. First, the methodologies in this research could be leveraged to handle the decentralized crowdsourcing of various physical resources. One remarkable example is the decentralized crowdsourcing of Internet bandwidth, which could be further augmented to attain the ideal goal of decentralized peer-to-peer content delivery network [72] in order to reduce the high usage cost of existing content delivery networks. Second, the results about decentralized knowledge crowdsourcing can be directly borrowed to bootstrap the desired decentralized data marketplace, since the key methodologies would also be valid solutions to solve the critical privacy and fairness issues in the decentralized version of data marketplace. As such, individuals can expect a secure and anonymous way to sell their personal information to certain parties. Third, this research (in particular, Zebralancer) provides an insight on how to achieve an anonymous blockchain-based auction platform in a meaningful way. In an anonymous auction protocol, the major challenge is that some malicious users might leverage the anonymity to flood bids to manipulate the auction' outcome. The common-prefix linkable anonymous authentication scheme could be the key to prevent those malicious behaviors. Finally, the methods (e.g., the special-purpose cryptographic optimizations) also shed light on how to implement a few interesting applications in the sector of decentralized finance, securely yet still efficiently. One example can be private blockchain-based gift cards [100]. This dedicatedly optimized system can enable the retailers to shift their gift card management databases on the blockchain, without scarifying the indispensable confidentiality or the critical efficiency.

8.3 Future Vision

Open problems. Though this dissertation presents a few newest results on decentralized crowdsourcing, this novel paradigm (along with the underlying blockchain technology) is still quickly developing, and the relevant area is also largely unexplored. In particular, a few immediate follow-up studies can be carried out to extend the functionalities and securities from various perspectives.

One promising direction is to conduct more specific-purpose designs to efficiently decentralize distinct flavors of crowdsourcing. For example, can we design a concretely efficient protocol to decentralize participatory crowd-sensing that is minimally meaningful with the needed fairness and privacy? Such the problem is challenging, since there is no explicit requester to "prove" the quality of encrypted data anymore. Unfortunately, letting the blockchain learn encrypted data's quality (without a prover) falls into the category of (multi-input) functional encryption [66, 19], which is unclear how can be solved practically till today.

In addition, this dissertation considers the security of decentralized knowledge crowdsourcing due to conventional cryptographic notions, in which the corrupted parties are fully controlled by an adversary and the honest parties will be honest to follow the protocol independently. This model has an inherent drawback to explain why rational workers would not deviate (e.g., by colluding together to free-ride). To address this realistic concern, an "incentive-compatible" protocol is required, so "following the protocol" is a Nash equilibrium (or its refinement) that can deter rational workers from deviating.

Another appealing future work is to explore decentralized crowdsourcing for more physical resources, for example, the bandwidth. In particular, the decentralized crowdsourcing of bandwidth resources could be considered as a novel P2P content delivery paradigm based on blockchain for the needed fairness, namely, an Internet users can be well paid iff it honestly delivers contents on behalf of the content owners.

Such a bandwidth crowdsourcing paradigm could be further considered as the final piece of the puzzle to complete the P2P storage network [124, 108], which only guarantees the storage of content but not the delivery.

Finally, it becomes an urgent open problem to consider the composability of collateral in crypto-economic protocols. Say all protocols in this dissertation requires participants to place considerable deposits for security. What is more, many crypto-economic protocols (e.g., PoS blockchains [81, 38, 63] and payment channels [107, 49] already introduce a few locked deposits, and it becomes enticing to explore the composability of using the same collateral in many crypto-economic protocols, without scarifying the securities of all. Collateral composability is critical to bootstrap the wider adoption of crypto-economic protocols in reality, as it allows the participants to conveniently enjoy the benefits of all protocols after merely placing a single piece of deposit.

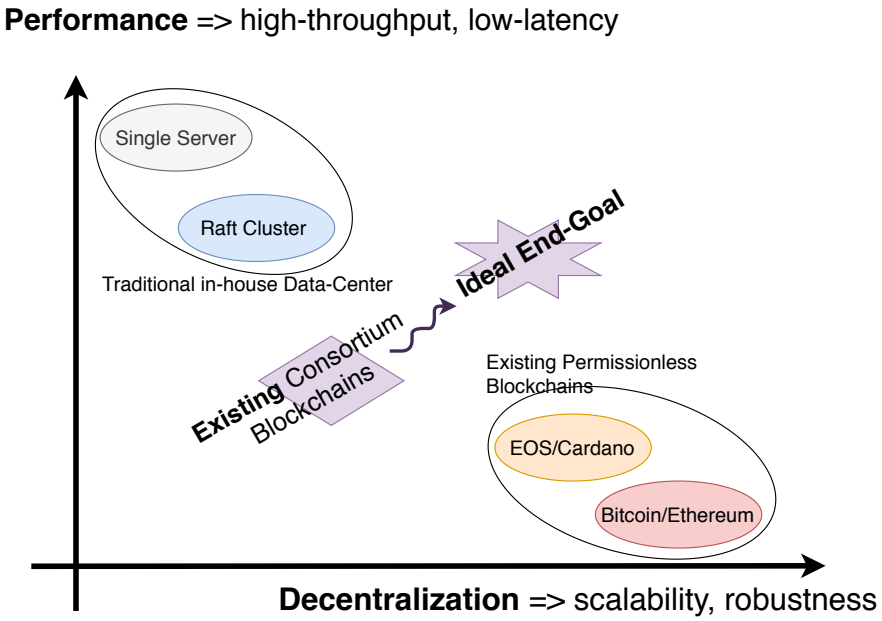

Figure 8.1 Trade-off between performance and decentralization.

Permissionless v.s. permissioned? Though the main results of this dissertation are initiated to adapt the extremely adversarial environment of so-called permis-

sionless blockchains (in which any Internet node can dynamically join and leave), they can be directly migrated to fit the so-called consortium blockchains (e.g., permissioned blockchains) where the consensuses are maintained collectively by a set of pre-selected parties with known identities.

Taking this in mind, it is straightforward to ask: when some practitioners are implementing the protocols in the dissertation, which underlying infrastructure shall they choose, the permissionless or the permissioned? Currently, it is actually a choice between performance (e.g., throughput and latency) and decentralization (e.g., scalability and robustness), as briefly illustrated in Figure 8.1. In particular, most existing permissionless blockchains are low-performing while being highly robust and scalable; in contrast, most existing permissioned blockchains are usually better performing but is much less robust and small in scale. Thus, in performance-aware scenarios, the permissioned blockchains could be preferred, while in robustness-critical cases, the permissionless infrastructure would be more desired.

Nevertheless, it is arguable that the blockchains, no matter the permissioned or the permissionless, are at their baby age and might be greatly improved through more groundbreaking efforts. In particular, in light of a few recent breakthroughs of Byzantine fault-tolerant protocols [68, 97, 156, 109], it is rather promising to improve the scalability of permissioned blockchains for large-scale. It, therefore, could be envisioned that the (permissioned) blockchains can soon break the current barrier of performance-decentralization trade-off, so in the near future, practitioners no longer have to sacrifice either decentralization or performance as they have to nowadays.

The previous two methodologies essentially correspond to a couple of general ways that can be applied for a broader array of decentralized applications to achieve indispensable security as well as real-world practicaliblity, instead of being restricted to the crowdsourcing use-cases discussed in this dissertation.

APPENDIX

SUPPLEMENTAL MATERIALS OF CHAPTER 6

A.1 Merkle Tree Algorithms

Figure A.1, A.2 and A.3 are the deferred algorithms related to Merkle tree. A Merkle tree (denoted by MT) is a binary tree: (i) the i-th leaf node is labeled by $\mathsf{H}(\mathsf{tx}_i)$, where H is a cryptographic hash function; (ii) any non-leaf node in the Merkle tree is labeled by the hash of the labels of its siblings. $\mathsf{BuildMT}(\cdot)$ takes a sequence of transactions as input, and outputs a Merkle tree whose root commits these transactions. GenMTP takes a Merkle tree MT and a transaction tx as input, and can generate a Merkle tree proof π for the inclusion of tx in the tree. Finally, $\mathsf{VrfyMTP}$ takes $\mathsf{H}(tx)$, the tree root, and the Merkle proof π, and can output whether $\mathsf{H}(tx)$ labels a leaf of the Merkle tree MT.

$\mathsf{BuildMT}$ algorithm

$\mathsf{BuildMT}(\mathsf{TX} = (tx_1, \cdots, tx_n))$:

- if $|\mathsf{TX}| = 1$:
  - $label(\mathsf{root}) = \mathsf{H}(tx_1)$
- else:
  - $lchild = \mathsf{BuildMT}(tx_1, \ldots, tx_{\lceil n/2 \rceil})$
  - $rchild = \mathsf{BuildMT}(tx_{\lceil n/2 \rceil + 1}, \ldots, tx_n)$
  - $label(\mathsf{root}) = \mathsf{H}(label(lchild)||label(rchild))$
- return Merkle tree MT with root

Figure A.1 $\mathsf{BuildMT}$ that generates a Merkle tree with root for $\mathsf{TX} = (tx_1, \cdots, tx_n)$.

GenMTP algorithm

GenMTP(MT, tx):

- $x \leftarrow$ the leaf node labeled by $\mathsf{H}(tx)$
- while $x \neq label(\mathsf{MT.root})$:
  - $lchild \leftarrow x.parent.lchild$
  - $rchild \leftarrow x.parent.rchild$
  - if $x = lchild$, $b_i \leftarrow 0$, $l_i = label(rchild)$
  - otherwise, $b_i \leftarrow 1$, $l_i = label(lchild)$
  - $x \leftarrow x.parent$
- return $\pi = ((l_i, b_i))_{i \in [1,n]}$

Figure A.2 GenMTP that generates a Merkle tree proof.

VrfyMTP algorithm

VrfyMTP$(lable(\mathsf{root}), \pi, \mathsf{H}(tx))$:

- parse π as a list $((l_i, b_i))_{i \in [1,n]}$
- $x = \mathsf{H}(tx)$
- for i in $[1, n]$:
  - if $b_i = 0$, $x \leftarrow \mathsf{H}(x||l_i)$, else $x \leftarrow \mathsf{H}(l_i||x)$
- if $x \neq lable(\mathsf{root})$, return $False$, otherwise return $True$

Figure A.3 VrfyMTP that verifies a Merkle tree proof.

A.2 Deferred Formal Description of Incentive Subroutines

Here are the deferred proofs for the security theorems in Chapter 6. These proofs complete the analysis for the basic incentive mechanism for two relays, the basic incentive for single relay, and the augmented incentive for single relay.

A.2.1 Basic Incentive for the Protocol with Two Relays

Figure A.4, A.5 and A.6 showcase that the basic incentive mechanism for two relays is implementable, when given the validateTrue algorithm. Figure A.5 presents the detailed incentive clauses when the client feeds back two validly signed response messages from both recruited relays (clause 1-6). Figure A.6 presents how to deal with the scenario where the client only feeds back one validly signed response message from only relay (clause 7-9). Note that $\$r$ is a parameter used to deter the collusion of two relays, which could be better illustrated later by the incentive for one single relay (that models colluding relays essentially).

Incentive **subroutine** for two non-cooperative relays

Incentive(responses, P_N^ℓ):

if $|\mathsf{responses}| = 2$ then

parse $\{(\mathsf{result}_i, \mathsf{sig}_i)\}_{i\in\{1,2\}} := \mathsf{responses}$

if $\mathsf{vrfySig}(\langle \mathsf{result}_i, \mathsf{ctr}\rangle, \mathsf{sig}_i, pk_1) = 1$ for each $i \in [1, 2]$

call Payout($\mathsf{result}_1, \mathsf{result}_2, \mathsf{P}_N^\ell$) subroutine in Figure A.5

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$d_L$

return

if $|\mathsf{responses}| = 1$ then

parse $\{(\mathsf{result}_i, \mathsf{sig}_i)\} := \mathsf{responses}$

if $\mathsf{vrfy}(\mathsf{result}_i||\mathsf{ctr}, \mathsf{sig}_i) = 1$

call Payout$'$($\mathsf{result}_i, \mathsf{P}_N^\ell$) subroutine in Figure A.6

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$d_L$

return

$\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$d_F$

$\mathsf{ledger}[\mathcal{R}_{|1-i|}] := \mathsf{ledger}[\mathcal{R}_{|1-i|}] + \$d_F$

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$d_L$

Figure A.4 Incentive subroutine (two non-cooperative relays).

Payout **subroutine**

Payout(result$_1$, result$_2$, P_N^ℓ):

if result$_1$ can be parsed as σ_1 then

if validateTrue($\sigma_1, \mathsf{P}_N^\ell$) $= 1$ then

if result$_2$ can be parsed as σ_2 then

if validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 1$ then // Clause 1

ledger$[\mathcal{R}_i]$:= ledger$[\mathcal{R}_i] + \$(\frac{p}{2} + d_F), \forall i \in \{1, 2\}$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e)$

else // i.e., validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 0$ // Clause 2

ledger$[\mathcal{R}_1]$:= ledger$[\mathcal{R}_1] + \$(p + \frac{3}{2}d_F)$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e + \frac{d_F}{2})$

else // i.e., result$_2 = \perp$ // Clause 3

ledger$[\mathcal{R}_1]$:= ledger$[\mathcal{R}_1] + \$(p + \frac{3}{2}d_F)$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e + \frac{d_F}{2})$

else // i.e., validateTrue($\sigma_1, \mathsf{P}_N^\ell$) $= 0$

if result$_2$ can be parsed as σ_2 then

if validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 1$ then // Clause 2

ledger$[\mathcal{R}_2]$:= ledger$[\mathcal{R}_2] + \$(p + \frac{3}{2}d_F)$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e + \frac{d_F}{2})$

else // i.e., validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 0$ // Clause 4

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(p + e + 2d_F)$

else // i.e., result$_2 = \perp$ // Clause 5

ledger$[\mathcal{R}_1]$:= ledger$[\mathcal{R}_1] + \$(\frac{p}{2} - r + d_F)$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(\frac{p}{2} + e + r + d_F)$

else // i.e., result$_1 = \perp$

if result$_2$ can be parsed as σ_2 then

if validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 1$ then // Clause 3

ledger$[\mathcal{R}_2]$:= ledger$[\mathcal{R}_2] + \$(p + \frac{3d_F}{2})$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e + \frac{d_F}{2})$

else // i.e., validateTrue($\sigma_2, \mathsf{P}_N^\ell$) $= 0$ // Clause 5

ledger$[\mathcal{R}_2]$:= ledger$[\mathcal{R}_2] + \$(\frac{p}{2} - r + d_F)$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(\frac{p}{2} + e + r + d_F)$

else // i.e., result$_2 = \perp$ // Clause 6

ledger$[\mathcal{R}_i]$:= ledger$[\mathcal{R}_i] + \$(\frac{p}{2} - r + d_F), \forall i \in \{1, 2\}$

ledger$[\mathcal{LW}]$:= ledger$[\mathcal{LW}] + \$(e + 2r)$

Figure A.5 Payout subroutine called by Figure A.4.

Payout′ subroutine

Payout′($\mathsf{result}_i, \mathsf{P}_N^\ell$):

if result_i can be parsed as σ_i then

if $\mathsf{validateTrue}(\sigma_i, \mathsf{P}_N^\ell) = 1$ then // Clause 7

$\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$(p + d_F)$

$\mathsf{ledger}[\mathcal{R}_{|1-i|}] := \mathsf{ledger}[\mathcal{R}_{|1-i|}] + \$(d_F)$

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$(\frac{e}{2})$

else // i.e., $\mathsf{validateTrue}(\sigma_i, \mathsf{P}_N^\ell) = 0$ // Clause 8

$\mathsf{ledger}[\mathcal{R}_{|1-i|}] := \mathsf{ledger}[\mathcal{R}_{|1-i|}] + \$(d_F)$

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$(\frac{p+e}{2} + \frac{d_F}{2})$

else // i.e., $\mathsf{result}_i = \perp$ // Clause 9

$\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$(\frac{p}{2} - r + d_F)$

$\mathsf{ledger}[\mathcal{R}_{|1-i|}] := \mathsf{ledger}[\mathcal{R}_{|1-i|}] + \$(d_F)$

$\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$(\frac{e}{2} + r)$

Figure A.6 Payout′ subroutine called by Figure A.4.

A.2.2 Basic Incentive for the Protocol with Single Relay

When the protocol is participated by the light client and one single relay node (which models that the client does not believe there are non-cooperative relays), we tune the incentive to let the only relay node to audit itself by "asymmetrically" pay the proved claim of truthness and the unproved claim of falseness (with an extra protocol parameter r), and thus forwarding the correct evaluation result of the chain predicate becomes dominating. In general, the incentive mechanism for one single relay is similar to the case of two relays. Moreover, as shown earlier, the incentive parameter r can also be incorporated into the two-relay case to deter the collusion of relays. Here we describe the case of one single relay as an over-simplified modeling to capture the effect of two completely cooperative relays (that can act as a single coalition), which allows us to better illustrate the idea of using r to deter collusion.

Incentive **subroutine** for one single relay

```
Incentive(responses, P_N^ℓ):
  if |responses| = 1 then
    parse {(result_i, sig_i)} := responses
    if vrfy(result_i||ctr, sig_i) = 1
      if result_i can be parsed as σ_i then
        if validateTrue(σ_i, P_N^ℓ) = 1 then
          ledger[R_i] := ledger[R_i] + $(p + d_F)
          ledger[LW] := ledger[LW] + $e
        else // i.e., validateTrue(σ_i, P_N^ℓ) = 0
          ledger[LW] := ledger[LW] + $(p + e + d_F)
      else // i.e., result_i = ⊥
        ledger[R_i] := ledger[R_i] + $(p − r + d_F)
        ledger[LW] := ledger[LW] + $(e + r)
      return
  ledger[R_i] := ledger[R_i] + $d_F
  ledger[LW] := ledger[LW] + $d_L
```

Figure A.7 Incentive subroutine for the protocol with (one single relay).

A.2.3 Augmented Incentive for the Protocol with Single Relay

Here is the augmented incentive mechanism for the protocol with one single relay. Different from the aforementioned idea of using "asymmetric" incentive to create "self-audition", we explicitly allow the additional public full nodes to audit the relay node in the system. To do so, the incentive subroutine has to wait for some "debating" message sent from the public full nodes (that are monitoring the internal states of the contract at a tiny cost), after it receives the feedback from the client about what the relay node does forward. If the relay node is cheating to claim a fake unprovable side, the public can generate a proof attesting that the relay was dishonest, thus allowing the contract to punish the cheating relay and reward the public node and return the payments to the light client.

Incentive **subroutine** for one single relay

Incentive(responses, P_N^ℓ):
  if $|\mathsf{responses}| = 1$ then
    parse $\{(\mathsf{result}_i, \mathsf{sig}_i)\} := \mathsf{responses}$
    if $\mathsf{vrfy}(\mathsf{result}_i||\mathsf{ctr}, \mathsf{sig}_i) = 1$
      if result_i can be parsed as σ_i then
        if $\mathsf{validateTrue}(\sigma_i, \mathsf{P}_N^\ell) = 1$ then
          $\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$(p + d_F)$
          $\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$e$
        else // i.e., $\mathsf{validateTrue}(\sigma_i, \mathsf{P}_N^\ell) = 0$
          $\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$(p + e + d_F)$
      else // i.e., $\mathsf{result}_i = \perp$
        $T_{debate} := T + \Delta T$
        $\mathsf{debate} := \mathsf{P}_N^\ell$
      return
  $\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$d_F$
  $\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$d_L$

Debate. Upon receiving $(\mathsf{debating}, \mathsf{P}_N^\ell, \sigma_{pfn})$ from $\mathcal{PFN}$:
  assert $\mathsf{debate} \neq \emptyset$ and $\mathsf{debate} = \mathsf{P}_N^\ell$
  if $\mathsf{validateTrue}(\sigma_{pfn}, \mathsf{P}_N^\ell) = 1$ then:
    $\mathsf{ledger}[\mathcal{PFN}] := \mathsf{ledger}[\mathcal{PFN}] + \$d_F$
    $\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$(p + e)$
    $\mathsf{debate} := \emptyset$

Timer. Upon $T \geq T_{debate}$ and $\mathsf{debate} \neq \emptyset$:
  $\mathsf{ledger}[\mathcal{R}_i] := \mathsf{ledger}[\mathcal{R}_i] + \$(d_F + p)$
  $\mathsf{ledger}[\mathcal{LW}] := \mathsf{ledger}[\mathcal{LW}] + \$e$
  $\mathsf{debate} := \emptyset$

Figure A.8 Augmented Incentive subroutine (a single relay) with assuming a non-colluding public full node (in the whole blockchain network).

A.3 Inductive Definition of Utility Functions

Let h denote a history in Γ_2^k the to represent the beginning of a reachable query, at which the $\mathcal{LW}$ moves to determine whether to abort or not, and then *chance*, $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{LW}$ sequentially move. Then the utilities of $\mathcal{LW}$, $\mathcal{R}_1$ and $\mathcal{R}_2$ can be defined recursively as shown in Table A.1, A.2 and A.3.

Table A.1 Recursive Definition of Utility of Γ_2^k (Continued in Table A.2)

Info Set of LW	Histories of LW	Actions of LW	Utility of LW	Utility of R_1	Utility of R_2
I_{LW}^1	$hQatt$	TA	$u_{LW}(h) + d_L - p$	$u_{R_1}(h) + \frac{p}{2} + d_F$	$u_{R_2}(h) + \frac{p}{2} + d_F$
		LA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	-	-
		RA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	-	-
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'ff$	TA	$u_{LW}(h) + d_L - v + 2d_F$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2$
		LA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^2	$hQatf$	TA	$u_{LW}(h) + d_L - p + d_F$	$u_{R_1}(h) + p + 3\frac{d_F}{2}$	$u_{R_2}(h)$
		LA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	-	-
		RA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'ft$	TA	$u_{LW}(h) + d_L - v - \frac{p}{2} + r + d_F$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2 + \frac{p}{2} - r + d_F$
		LA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^3	$hQatx$	TA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	$u_{R_1}(h) + p + d_F$	$u_{R_2}(h) + d_F$
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'fx$	TA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2 + d_F$
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^4	$hQaft$	TA	$u_{LW}(h) + d_L - p + \frac{d_F}{2}$	$u_{R_1}(h)$	$u_{R_2}(h) + p + 3\frac{d_F}{2}$
		LA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		RA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	-	-
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'tf$	TA	$u_{LW}(h) + d_L - v - \frac{p}{2} + r + d_F$	$u_{R_1}(h) + v_1 + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + v_2$
		LA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		RA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^5	$hQaff$	TA	$u_{LW}(h) + d_L - p + 2r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		RA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'tt$	TA	$u_{LW}(h) + d_L - v - p + 2r$	$u_{R_1}(h) + v_1 + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + v_2 + \frac{p}{2} - r + d_F$
		LA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		RA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^6	$hQafx$	TA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + d_F$
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'tx$	TA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	$u_{R_1}(h) + v_1 + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + v_2 + d_F$
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^7	$hQaxt$	TA	$u_{LW}(h) + d_L - p - \frac{e}{2}$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + p + d_F$
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'xf$	TA	$u_{LW}(h) + d_L - v - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h) + v_1 + d_F$	$u_{R_2}(h) + v_2$
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^8	$hQaxf$	TA	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'xt$	TA	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	$u_{R_1}(h) + v_1 + d_F$	$u_{R_2}(h) + v_2 + \frac{p}{2} - r + d_F$
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
I_{LW}^9	$hQaxx$	TA	$u_{LW}(h) + d_L - p - e$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + d_F$
		XA	$u_{LW}(h) + d_L - p - e$	-	-
	$hQa'xx$	TA	$u_{LW}(h) + d_L - v - p - e$	$u_{R_1}(h) + v_1 + d_F$	$u_{R_2}(h) + v_2 + d_F$
		XA	$u_{LW}(h) + d_L - v - p - e$	-	-
-	h	B	$u_{LW}(h)$	-	-

Table A.2 Recursive Definition of Utility of Γ_2^k (Continued from Table A.1)

Info Set of LW	Histories of LW	Actions of LW	Utility of LW	Utility of R_1	Utility of R_2
I_{LW}^1	$hQatt$	TA'	$u_{LW}(h) + d_L - v - p$	$u_{R_1}(h) + v_1 + \frac{p}{2} + d_F$	$u_{R_2}(h) + v_2 + \frac{p}{2} + d_F$
		LA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	-	-
		RA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	-	-
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'ff$	TA'	$u_{LW}(h) + d_L + 2d_F$	$u_{R_1}(h)$	$u_{R_2}(h)$
		LA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^2	$hQatf$	TA'	$u_{LW}(h) + d_L - v - p + \frac{d_F}{2}$	$u_{R_1}(h) + v_1 + p + 3\frac{d_F}{2}$	$u_{R_2}(h) + v_2$
		LA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	-	-
		RA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'ft$	TA'	$u_{LW}(h) + d_L - \frac{p}{2} + r + d_F$	$u_{R_1}(h)$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^3	$hQatx$	TA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	$u_{R_1}(h) + v_1 + p + d_F$	$u_{R_2}(h) + v_2 + d_F$
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'fx$	TA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h)$	$u_{R_2}(h) + d_F$
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^4	$hQaft$	TA'	$u_{LW}(h) + d_L - v - p + \frac{d_F}{2}$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2 + p + 3\frac{d_F}{2}$
		LA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		RA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	-	-
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'tf$	TA'	$u_{LW}(h) + d_L - \frac{p}{2} + r + d_F$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h)$
		LA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		RA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^5	$hQaff$	TA'	$u_{LW}(h) + d_L - v - p + 2r$	$u_{R_1}(h) + v_1 + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + v_2 + \frac{p}{2} - r + d_F$
		LA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		RA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	-	-
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'tt$	TA'	$u_{LW}(h) + d_L - p + 2r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		RA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	-	-
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^6	$hQafx$	TA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	$u_{R_1}(h) + v_1 + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + v_2 + d_F$
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'tx$	TA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + d_F$
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^7	$hQaxt$	TA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2}$	$u_{R_1}(h) + v_1 + d_F$	$u_{R_2}(h) + v_2 + p + d_F$
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'xf$	TA'	$u_{LW}(h) + d_L - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h) + d_F$	$u_{R_2}(h)$
		XA'	$u_{LW}(h) + d_L - p - e$	-	-
I_{LW}^8	$hQaxf$	TA'	$u_{LW}(h) + d_L - v - p - \frac{e}{2} + r$	$u_{R_1}(h) + v_1 + d_F$	$u_{R_2}(h) + v_2 + \frac{p}{2} - r + d_F$
		XA'	$u_{LW}(h) + d_L - v - p - e$	-	-
	$hQa'xt$	TA'	$u_{LW}(h) + d_L - p - \frac{e}{2} + r$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		XA'	$u_{LW}(h) + d_L - e$	-	-
I_{LW}^9	$hQaxx$	TA'	$u_{LW}(h) + d_L - v - p - e$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2$
	$hQaxx$	XA'	$u_{LW}(h) + d_L - v - p - e$	$u_{R_1}(h) + v_1$	$u_{R_2}(h) + v_2$
	$hQa'xx$	TA'	$u_{LW}(h) + d_L - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
	$hQa'xx$	XA'	$u_{LW}(h) + d_L - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
-	h	B	$u_{LW}(h)$	-	-

Table A.3 Recursive Definition of Utility of Γ_2^k (Continued from Tables A.1 and A.2)

Info Set of LW	Histories of LW	Actions of LW	Utility of LW	Utility of R_1	Utility of R_2
I_{LW}^1	$hQatt$	TO	$u_{LW}(h) - c - p$	$u_{R_1}(h) + \frac{p}{2} + d_F$	$u_{R_2}(h) + \frac{p}{2} + d_F$
		LO	$u_{LW}(h) - c - p - e$	-	-
		RO	$u_{LW}(h) - c - p - e$	-	-
		XO	$u_{LW}(h) - c - p - e + \epsilon$	-	-
	$hQa'ff$	TO	$u_{LW}(h) - c + 2d_F$	$u_{R_1}(h)$	$u_{R_2}(h)$
		LO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XO	$u_{LW}(h) - c - p - e + \epsilon$	-	-
I_{LW}^2	$hQatf$	TO	$u_{LW}(h) - c - p + \frac{d_F}{2}$	$u_{R_1}(h) + p + 3\frac{d_F}{2}$	$u_{R_2}(h)$
		LO	$u_{LW}(h) - c - p - \frac{e}{2}$	-	-
		RO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'ft$	TO	$u_{LW}(h) - c - \frac{p}{2} + r + d_F$	$u_{R_1}(h)$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		RO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^3	$hQatx$	TO	$u_{LW}(h) - c - p - \frac{e}{2}$	$u_{R_1}(h) + p + d_F$	$u_{R_2}(h) + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'fx$	TO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h)$	$u_{R_2}(h) + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^4	$hQaft$	TO	$u_{LW}(h) - c - p + \frac{d_F}{2}$	$u_{R_1}(h)$	$u_{R_2}(h) + p + 3\frac{d_F}{2}$
		LO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		RO	$u_{LW}(h) - c - p - \frac{e}{2}$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'tf$	TO	$u_{LW}(h) - c + \frac{p}{2} + r + d_F$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h)$
		LO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		RO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^5	$hQaff$	TO	$u_{LW}(h) - c - p + 2r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		RO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'tt$	TO	$u_{LW}(h) - c - p + 2r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		LO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		RO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	-	-
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^6	$hQafx$	TO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'tx$	TO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	$u_{R_1}(h) + \frac{p}{2} - r + d_F$	$u_{R_2}(h) + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^7	$hQaxt$	TO	$u_{LW}(h) - c - p - \frac{e}{2}$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + p + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'xf$	TO	$u_{LW}(h) - c - \frac{p}{2} - \frac{e}{2} + \frac{d_F}{2}$	$u_{R_1}(h) + d_F$	$u_{R_2}(h)$
		XO	$u_{LW}(h) - c - p - e$	-	-
I_{LW}^8	$hQaxf$	TO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		XO	$u_{LW}(h) - c - p - e$	-	-
	$hQa'xt$	TO	$u_{LW}(h) - c - p - \frac{e}{2} + r$	$u_{R_1}(h) + d_F$	$u_{R_2}(h) + \frac{p}{2} - r + d_F$
		XO	$u_{LW}(h) - c - e$	-	-
I_{LW}^9	$hQaxx$	TO	$u_{LW}(h) - c - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
	$hQaxx$	XO	$u_{LW}(h) - c - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
	$hQa'xx$	TO	$u_{LW}(h) - c - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
	$hQa'xx$	XO	$u_{LW}(h) - c - p - e$	$u_{R_1}(h)$	$u_{R_2}(h)$
-	h	B	$u_{LW}(h)$	-	-

We make the following remarks about the recursive definition of the utilities: (i) when $h := \emptyset$, set $u_{\mathcal{LW}}(h) := 0$, $u_{\mathcal{R}_1}(h) := 0$, and $u_{\mathcal{R}_2}(h) := 0$; (ii) $u_{\mathcal{LW}}(h) = u_{\mathcal{LW}}(h) + d_L$ and the utility functions depict the final outcomes of the players, when $|h| = 6k$ (i.e., the protocol expires); (iii) when the *chance* choose a and the lightweight client outputs A', or when the *chance* choose a' and the lightweight client outputs A, the light client is fooled, which means its utility increment shall subtract v, and the utility increments of $\mathcal{R}_1$ and $\mathcal{R}_2$ shall add $v_1 := v_1(\mathsf{P}_h^1, \mathsf{C})$ and $v_2 := v_2(\mathsf{P}_h^1, \mathsf{C})$ respectively; (iv) when the light client outputs O, which means the client decides to setup its own full node and c is subtracted from its utility increment to reflect the cost. Also note that if two non-cooperative relays $\mathcal{R}_1$ and $\mathcal{R}_2$ share no common conflict of cheating the light client, one more constrain is applied to ensure $v_1(\mathsf{P}_h^1, \mathsf{C}) \cdot v_2(\mathsf{P}_h^1, \mathsf{C}) = 0$.

The utility functions of game Γ_1^k and G_1^k can be inductively defined similarly.

A.4 Deferred Proofs for Security Theorems

A.4.1 Proof for Theorem 4

Lemma 2. *If the client raises a query in the game Γ_2^k, the sequentially rational strategies of the light client $\mathcal{LW}$ (under any belief system) will not include L, R and X (i.e., the light client will always take T to report the contract the whatever it receives from the relay nodes) in this query.*

Proof. It is clear to see the Lemma from the recursive formulation of the utility function of $\mathcal{LW}$. Because no matter at any history of any information set, taking an action including L, R or X is dominated by replacing the character by T. □

Lemma 3. *At the last query (history h) in the game Γ_2^k, if the client raises the last query (i.e., reaching the history hQ), when $\mathcal{R}_1$ and $\mathcal{R}_2$ are non-cooperative, the game terminates in $hQ(attTA|a'ttTA')$, conditioned on $d_F + \frac{p}{2} > v_i$.*

Proof. Let the history h denote any history where is the turn of the light client to choose from $\{Q, B\}$. If the client raises the query due to Lemma 6, it

can be seen that the only reachable histories from h must have the prefix of: $hQ(a|a')(t|f|x)(t|f|x)T(A|A'|O)$.

Say $hQ(a|a')(t|f|x)(t|f|x)T(A|A'|O)$ terminates the game, and w.l.o.g. let $\mathcal{R}_i$ choose to deviate from t. Due to such the strategy of $\mathcal{R}_i$, a belief system of $\mathcal{R}_j$ consistent with that has to assign some non-negligible probability to the history corresponding that $\mathcal{R}_i$ takes an action off t, so the best response of $\mathcal{R}_j$ after h is also to take action t according to the utility definitions. Then we can reason the sequential rationality backwardly: in any information set of the full nodes, the best response of the relay full nodes is to take action t. Thus the strategy of $\mathcal{R}_i$ consistent to $\mathcal{R}_j$'s strategy must act t. Note the joint strategy $\vec{\sigma}$ that $\mathcal{R}_1$ and $\mathcal{R}_2$ always choose the action of t and the $\mathcal{LW}$ always chooses the strategy of T. For the light client, its belief system consistent with the fact must assign the probability of 1 to $hQatt$ out of the information set of I_1^{LW} and the probability of 1 to $hQa'tt$ out of the information set of I_5^{LW}. Conditioned on such the belief, the light client must choose TA in I_1^{LW} and choose TA' in I_5^{LW}. This completes the proof for the lemma.

□

Deferred proof for Theorem 4:

Proof. At the last query in the game Γ_2^k, the client will raise the query (namely act Q), conditioned on $d_L > (p + e)$. Additionally from Lemma 3, it nearly immediately proves the Theorem 4 due to backward reduction. To prove, there is only one more step of backward reduction to see that: (i) the rational light client must raise a query through the protocol when $c > p$, and (ii) rational parties are incentivized to setup the protocol (as the light client avoids the cost of maintaining its own personal full node and the relay full node will get positive payments). □

A.4.2 Proof for Theorem 5

Lemma 4. *If the client raises a query in the game Γ_1^k, the sequentially rational strategies of the light client $\mathcal{LW}$ (under any belief system) will not include X (i.e., the light client will always take T to report the contract the whatever it receives from the relay nodes) in this query.*

Proof. No matter at any history of any information set, taking an action including X is dominated by replacing the character by T, which is clear from the utility function. □

Lemma 5. *At the last query (history h) in the game Γ_1^k, if the client raises the last query (i.e., reaching the history hQ), the game terminates in $hQ(atTA|a'tTA')$, conditioned on $d_F + p - r > v_i$, $r > v_i$.*

Proof. Let the history h denote the beginning of the last query. If the client raises the query, the game reaches hQ. Due to Lemma 6, it can be seen that the only reachable histories from h must have the prefix of: $hQ(a|a')(t|f|x)T(A|A'|O)$. Then it is immediate to see t is dominating from the utility function. This completes the lemma.

□

Deferred proof for Theorem 5:

Proof. From Lemma 5, acting Q is strictly dominates B due to the utility function, at least in the last query. The last query would include no deviation at all. It (nearly) immediately allows us prove the Theorem 5 due to backward reduction from Lemma 5. To prove, there is only one more step of backward reduction to see that, which can be derived because: (i) the rational light client must raise a query through the protocol when $c > p$, and (ii) rational parties are incentivized to setup the protocol (as the light client avoids the cost of maintaining its own personal full node and the relay full node will get positive payments). □

A.4.3 Proof for Theorem 6

Lemma 6. *If the client raises a query in the augmented game G_1^k, the sequentially rational strategies of the light client $\mathcal{LW}$ (under any belief system) will not include X (i.e., the light client will always take T to report the contract the whatever it receives from the relay nodes) in this query.*

Proof. No matter how the relay and the public full node act, acting X is dominated by replacing the character by T in the augmented game G_1^k. □

Lemma 7. *At the last query (history h) in the game G_1^k, if the client raises the last query (i.e., reaching the history $h(m|x)Q$), the relay node would not deviate off t with non-negligible probability, conditioned on $d_F > v_i$.*

Proof. Let the history h denote any history where is the turn of the light client to choose from $\{Q, B\}$. If the client raises the query due to Lemma 6, it can be seen that the only reachable histories from h must have the prefix of: $h(m|x)Q(a|a')(t|f|x)T(A|A'|O)$. Given such fact, the relay would not deviate off t with non-negligible probability: (i) deviating from the protocol to play x is strictly dominated; (ii) deviating to play f with negligible probability will consistently cause the public full node acts m. This completes the lemma.

□

Deferred proof for Theorem 6:

Proof. From Lemma 7, acting Q is strictly dominates B due to the utility function, at least in the last query. Thus we can argue due to backward reduction from the last query to the first query. It (nearly) immediately completes the proof for Theorem 6. To prove, there is only one more step of backward reduction to see rational parties are incentivized to setup the protocol (as the light client avoids the cost of maintaining its own personal full node to evaluate chain predicates and the relay full node will get positive payments). □

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 337–346, 2019.

[2] Airbnb. `https://www.airbnb.com` (accessed: December 2017).

[3] Alibaba Cloud. Unveiled for Double 11, StarAgent: Alibaba's Automatic OM System. `https://medium.com/@Alibaba_Cloud/unveiled-for-double-11-staragent-alibabas-automatic-o-m-system-97df1b3ec616` (accessed: August 2020).

[4] Mohammad Allahbakhsh, Boualem Benatallah, Aleksandar Ignjatovic, Hamid Reza Motahari-Nezhad, Elisa Bertino, and Schahram Dustdar. Quality control in crowdsourcing systems: issues and directions. *IEEE Internet Computing*, 17(2):76–81, 2013.

[5] Amazon Mechanical Turk. `https://www.mturk.com/mturk/` (accessed: January 2018).

[6] Amazon Mechanical Turk. MTurk Pricing. `https://www.mturk.com/pricing` (accessed: January 2018).

[7] Amazon Mechanical Turk. Tutorial: How to verify crowdsourced training data using a known answer review policy. `https://blog.mturk.com/tutorial-how-to-verify-crowdsourced-training-data-using-a-known-answer-review-policy-85596fb55ed` (accessed: July 2018).

[8] Man Ho Au, Willy Susilo, and Siu-Ming Yiu. Event-oriented k-times revocable-iff-linked group signatures. In *Proceedings of Australasian Conference on Information Security and Privacy (ACISP)*, pages 223–234, 2006.

[9] Yukino Baba and Hisashi Kashima. Statistical quality estimation for general crowdsourcing tasks. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 554–562, 2013.

[10] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of ACM Conference on Electronic Commerce (EC)*, pages 56–73, 2012.

[11] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2014. `https://blockstream.com/sidechains.pdf` (accessed: January 2019).

[12] Paulo SLM Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of International Conference on Selected Areas in Cryptography (SAC)*, pages 319–331, 2005.

[13] Amos Beimel, Adam Groce, Jonathan Katz, and Ilan Orlov. Fair computation with rational players. 2011. `https://eprint.iacr.org/2011/396` (accessed: June 2020).

[14] Mira Belenkiy, Melissa Chase, C. Chris Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing outsourced computation. In *Proceedings of ACM Workshop on the Economics of Networks (NetEcon)*, pages 85–90, 2008.

[15] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, et al. SNARKs for C: verifying program executions succinctly and in zero knowledge. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 90–108, 2013.

[16] Juan Benet. Ipfs: Content addressed, versioned, p2p file system. `https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf` (accessed: June 2019).

[17] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In *Proceedings of Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 410–440, 2017.

[18] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 561–586, 2019.

[19] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 563–594, 2015.

[20] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015. `https://eprint.iacr.org/2015/1015` (accessed: June 2020).

[21] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. Tweetchain: an alternative to blockchain for crowd-based applications. In *Proceedings of International Conference on Web Engineering (ICWE)*, pages 386–393, 2017.

[22] Vitalik Buterin. Light clients and proof of stake. `https://blog.ethereum.org/2015/01/10/light-clients-proof-stake/` (accessed: December 2018).

[23] Vitalik Buterin. A next-generation smart contract and decentralized application platform. 2014. `https://github.com/ethereum/wiki/wiki/White-Paper` (accessed: May 2020).

[24] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 524–541, 2001.

[25] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 280–312, 2018.

[26] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 201–210, 2006.

[27] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 93–118, 2011.

[28] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 126–144, 2003.

[29] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[30] Cardano. `https://www.cardano.org/en/home/` (accessed: December 2018).

[31] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: confidential distributed ledger transactions via pvorm. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 701–717, 2017.

[32] David Chaum. Blind signatures for untraceable payments. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 199–203, 1983.

[33] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 319–327, 1988.

[34] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200, 2019.

[35] Coinbase. Ethereum Price. https://www.coinbase.com/price/ (accessed: July 2019).

[36] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086 (accessed: June 2020).

[37] CryptoKitties. https://www.cryptokitties.co/ (accessed: May 2019).

[38] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 23–41, 2019.

[39] Ghassan Karame Damian Gruber, Wenting Li. Unifying lightweight blockchain client implementations. In *Proceedings of Workshop on Decentralized IoT Security and Standards (DISS)*, 2018.

[40] Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 500–519, 2018.

[41] Jia Deng, Wei Dong, Richard Socher, et al. Imagenet: A large-scale hierarchical image database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[42] Jia Deng, Olga Russakovsky, Jonathan Krause, Michael S Bernstein, Alex Berg, and Li Fei-Fei. Scalable multi-label annotation. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 3099–3102, 2014.

[43] Srinivas Devarakonda, Parveen Sevusu, Hongzhang Liu, Ruilin Liu, Liviu Iftode, and Badri Nath. Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of ACM International Workshop on Urban Computing (UrbCom)*, pages 15:1–15:8, 2013.

[44] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 112–130, 2000.

[45] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 211–227, 2017.

[46] John R Douceur. The sybil attack. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.

[47] Emily Dreyfuss. A BOT panic HITs Amazon's Mechanical Turk. https://www.wired.com/story/amazon-mechanical-turk-bot-panic/ (accessed: May 2019).

[48] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 967–984, 2018.

[49] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 327–344, 2019.

[50] Fred Ehrsam. Blockchain-based machine learning marketplaces. https://medium.com/@FEhrsam/blockchain-based-machine-learning-marketplaces-cb2d4dae2c17 (accessed: May 2018).

[51] ETH Gas Station. Recommended gas prices in gwei. https://ethgasstation.info/ (accessed: July 2019).

[52] Ethereum Team. Byzantium HF Announcement. https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/ (accessed: December 2018).

[53] Ethereum Team. Ethereum Improvement Proposals 1108. https://eips.ethereum.org/EIPS/eip-1108 (accessed: January 2020).

[54] Ethereum Team. Ethereum Improvement Proposals 210. https://eips.ethereum.org/EIPS/eip-210 (accessed: January 2020).

[55] Christopher Frantz and Mariusz Nowostawski. From institutions to code: Towards automated generation of smart contracts. In *Proceedings of IEEE International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 210–215, 2016.

[56] Raghu Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[57] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 281–310, 2015.

[58] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.

[59] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 465–482, 2010.

[60] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 626–645, 2013.

[61] Craig Gentry, Zulfikar Ramzan, and Stuart Stubblebine. Secure distributed human computation. In *Proceedings of ACM Conference on Electronic Commerce (EC)*, pages 155–164, 2005.

[62] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 3–16, 2016.

[63] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[64] Stylianos Gisdakis, Thanassis Giannetsos, and Panagiotis Papadimitratos. Security, privacy, and incentive provision for mobile crowd sensing systems. *IEEE Internet of Things Journal*, 3(5):839–853, 2016.

[65] Oded Goldreich. *Foundations of Cryptography: Volume 2 Basic Applications*. Cambridge University Press, Cambridge, UK, 2009.

[66] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 578–602, 2014.

[67] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 365–377, 1982.

[68] Binyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Fast asynchronous bft protocols. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2020.

[69] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 623–632, 2004.

[70] Joseph Y Halpern and Rafael Pass. Sequential equilibrium in computational games. *ACM Transactions on Economics and Computation (TEAC)*, 7(2):1–19, 2019.

[71] Joseph Y Halpern, Rafael Pass, and Lior Seeman. Computational extensive-form games. In *Proceedings of ACM Conference on Economics and Computation (EC)*, pages 681–698, 2016.

[72] Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Wu. Enigma: Decentralized computation platform with guaranteed privacy. manuscript ready for submission.

[73] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf` (accessed: December 2017).

[74] Lilly C Irani and M Silberman. Turkopticon: Interrupting worker invisibility in amazon mechanical turk. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 611–620, 2013.

[75] Haiming Jin, Lu Su, Danyang Chen, Klara Nahrstedt, and Jinhui Xu. Quality of information aware incentive mechanisms for mobile crowd sensing systems. In *Proceedings of International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc)*, pages 167–176, 2015.

[76] Aniket Kate. Blockchain privacy: challenges, solutions, and unresolved issues. `http://www.isical.ac.in/~rcbose/blockchain2017/lecture/Kate_Slides.pdf` (accessed: December 2017).

[77] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, Boca Raton, FL, 2014.

[78] Heather Kelly. Apple account hack raises concern about cloud storage. `http://www.cnn.com/2012/08/06/tech/mobile/icloud-security-hack/` (accessed: December 2018).

[79] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 61–78, 2016.

[80] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963, 2017. `https://eprint.iacr.org/2017/963` (accessed: June 2020).

[81] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: a provably secure proof-of-stake blockchain protocol. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 357–388, 2017.

[82] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 705–734, 2016.

[83] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 21–34, 2019.

[84] Gillat Kol and Moni Naor. Games for exchanging information. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 423–432, 2008.

[85] Ahmed Kosba, Andrew Miller, Elaine Shi, et al. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 839–858, 2016.

[86] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 195–206, 2015.

[87] Alptekin Küpçü. Incentivized outsourced computation resistant to malicious contractors. *IEEE Transactions on Dependable and Secure Computing*, 14(6):633–649, 2015.

[88] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[89] Michael Z Lee, Alan M Dunn, Brent Waters, Emmett Witchel, and Jonathan Katz. Anon-pass: practical anonymous subscriptions. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 319–333, 2013.

[90] Matt Lepinksi, Silvio Micali, and Abhi Shelat. Collusion-free protocols. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 543–552, 2005.

[91] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. Vault: Fast bootstrapping for the algorand cryptocurrency. In *Proceedings of Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[92] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, and Robert H Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, 2018.

[93] Qinghua Li and Guohong Cao. Providing efficient privacy-aware incentives for mobile sensing. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, pages 208–217, 2014.

[94] Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*. Springer International Publishing, Cham, 2017.

[95] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Proceedings of Australasian Conference on Information Security and Privacy (ACISP)*, pages 325–335, 2004.

[96] Mahdi Zamani Loi Luu, Benedikt Bünz. Flyclient super light clients for cryptocurrencies. `https://scalingbitcoin.org/stanford2017/Day1/flyclientscalingbitcoin.pptx.pdf` (accessed: December 2018).

[97] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, 2020.

[98] Yuan Lu, Qiang Tang, and Guiling Wang. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, pages 853–865, 2018.

[99] Yuan Lu, Qiang Tang, and Guiling Wang. Zebralancer: Decentralized crowdsourcing of human knowledge atop open blockchain. *arXiv preprint arXiv:1803.01256*, 2019.

[100] Yuan Lu, Qiang Tang, and Guiling Wang. Enhancing the Retailer Gift Card via Blockchain: Trusted Resale and More. *Journal of Database Management*, 2020.

[101] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 17–30, 2016.

[102] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 706–719, 2015.

[103] Brian McInnis, Dan Cosley, Chaebong Nam, and Gilly Leshed. Taking a HIT: Designing around rejection, mistrust, risk, and workers' experiences in Amazon Mechanical Turk. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 2271–2282, 2016.

[104] Metamask. `https://metamask.io/` (accessed: January 2018).

[105] Katie Benner Mike Isaac and Sheera Frenkel. Uber hid 2016 breach, paying hackers to delete stolen data. `https://www.nytimes.com/2017/11/21/technology/uber-hack.html` (accessed: December 2018).

[106] Andrew Miller. Blockhash contract. `https://github.com/amiller/ethereum-blockhashes` (accessed: May 2019).

[107] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, 2019.

[108] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 475–490, 2014.

[109] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 31–42, 2016.

[110] Andrew Edmund Miller, Michael Hicks, Jonathan Katz, and Elaine Shi. Authenticated data structures, generically. In *Proceedings of ACM Symposium on Principles of Programming Languages (POPL)*, pages 411–423, 2014.

[111] Anton Muehlemann. Sentiment protocol: A decentralized protocol leveraging crowd sourced wisdom. 2017. `https://eprint.iacr.org/2017/1133.pdf` (accessed: June 2020).

[112] Phil Muncaster. China's Internet wunderkind in the dock over alleged fraud. `http://www.theregister.co.uk/2012/07/03/qihoo_fraud_traffic_comscore` (accessed: December 2018).

[113] Henry Alexander Murray. *Thematic Apperception Test*. Harvard University Press, Cambridge, MA, 1943.

[114] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. 2008. `https://bitcoin.org/bitcoin.pdf` (accessed: June 2018).

[115] Toru Nakanishi, Toru Fujiwara, and Hajime Watanabe. A linkable group signature and its application to secret voting. *Transactions of Information Processing Society of Japan*, 40(7):3085–3096, 1999.

[116] Robert Nix and Murat Kantarcioglu. Contractual agreement design for enforcing honesty in cloud outsourcing. In *Proceedings of International Conference on Decision and Game Theory for Security (GameSec)*, pages 296–308, 2012.

[117] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[118] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gaži, Joël Alwen, and Krzysztof Pietrzak. Spacemint: A cryptocurrency based on proofs of space. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 480–499, 2018.

[119] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 238–252, 2013.

[120] Dan Peng, Fan Wu, and Guihai Chen. Pay as how well you do: A quality based incentive mechanism for crowdsensing. In *Proceedings of International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc)*, pages 177–186, 2015.

[121] Viet Pham, M. H. R. Khouzani, and Carlos Cid. Optimal contracts for outsourced computation. In *Proceedings of International Conference on Decision and Game Theory for Security (GameSec)*, pages 79–98, 2014.

[122] Jack Plantin. How outsourcing can help you build a better startup. `https://medium.com/@JackPlantin/how-outsourcing-can-help-you-build-a-better-startup-5d04516fe71a`.

[123] Andrew Poelstra. Mimblewimble. `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf` (accessed: July 2018).

[124] Protocol Labs. Filecoin: A decentralized storage network. 2017. `https://filecoin.io/filecoin.pdf` (accessed: May 2019).

[125] Sazzadur Rahaman, Long Cheng, Danfeng Daphne Yao, He Li, and Jung-Min Jerry Park. Provably secure anonymous-yet-accountable crowdsensing with scalable sublinear revocation. In *Proceedings of Annual Privacy Enhancing Technologies Symposium (PETS)*, pages 384–403, 2017.

[126] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[127] Olga Russakovsky and Fei-Fei Li. Attribute learning in large-scale datasets. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 1–14, 2010.

[128] Niloufar Salehi, Lilly C Irani, Michael S Bernstein, et al. We are dynamo: Overcoming stalling and friction in collective action for crowd workers. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 1621–1630, 2015.

[129] Stefan Saroiu and Alec Wolman. I am a sensor, and I approve this message. In *Proceedings of International Workshop on Data Privacy Management and Security Assurance (DPM)*, pages 37–42, 2010.

[130] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Proceedings of Annual International Cryptology Conference (CRYPTO)*, pages 239–252, 1989.

[131] Dave Schultz. Use amazon mechanical turk with amazon sagemaker for supervised learning. `https://aws.amazon.com/blogs/machine-learning/use-amazon-mechanical-turk-with-amazon-sagemaker-for-supervised-learning/` (accessed: July 2019).

[132] Kai Sedgwick. Decentralized apps might be the future but they're not the present. `https://news.bitcoin.com/decentralized-apps-might-be-the-future-but-theyre-not-the-present/`. (accessed: May 2020).

[133] Nihar Shah and Dengyong Zhou. Double or nothing: multiplicative incentive mechanisms for crowdsourcing. *Journal of Machine Learning Research*, 17(1):5725–5776, 2016.

[134] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, 2012.

[135] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 254–263, 2008.

[136] Standford Vision Lab. ImageNet Readme. `http://image-net.org/downloads/attributes/README` (accessed: January 2018).

[137] Swarm. `http://swarm-guide.readthedocs.io` (accessed: July 2018).

[138] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[139] Cristian Tanas, Sergi Delgado-Segura, and Jordi Herrera-Joancomartí. An integrated reward and reputation mechanism for MCS preserving users' privacy. In *Proceedings of International Workshop on Data Privacy Management and Security Assurance (DPM)*, pages 83–99, 2015.

[140] Isamu Teranishi, Jun Furukawa, and Kazue Sako. K-times anonymous authentication. In *Proceedings of Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 308–322, 2004.

[141] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. 2017. `https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf` (accessed: July 2019).

[142] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, pages 393–409, 2017.

[143] Patrick P Tsang, Victor K Wei, Tony K Chan, Man Ho Au, Joseph K Liu, and Duncan S Wong. Separable linkable threshold ring signatures. In *Proceedings of International Conference on Cryptology in India (INDOCRYPT)*, pages 384–398, 2004.

[144] Turkrequesters. The BOT Problem on Mturk. `http://turkrequesters.blogspot.com/2018/08/the-bot-problem-on-mturk.html` (accessed: July 2018).

[145] Uber. `https://www.uber.com` (accessed: December 2017).

[146] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of USENIX Security Symposium (USENIX Security)*, 2018.

[147] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1):184–204, 2013.

[148] Waze. `https://status.waze.com` (accessed: January 2018).

[149] He Wei. Alipay apologizes for leak of personal info. `http://www.chinadaily.com.cn/china/2014-01/07/content_17219203.htm` (accessed: December 2018).

[150] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. Foreshadow-ng: Breaking the virtual memory abstraction with transient out-of-order execution. 2018. `https://foreshadowattack.eu/foreshadow-NG.pdf` (accessed: June 2019).

[151] Yutian Wen, Jinyu Shi, Qi Zhang, et al. Quality-driven auction-based incentive mechanism for mobile crowd sensing. *IEEE Transactions on Vehicular Technology*, 64(9):4203–4214, 2015.

[152] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. 2014. `https://ethereum.github.io/yellowpaper/paper.pdf` (accessed: May 2019).

[153] Shouhuai Xu and Moti Yung. K-anonymous secret handshakes with reusable credentials. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 158–167, 2004.

[154] Dejun Yang, Guoliang Xue, Xi Fang, and Jian Tang. Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing. In *Proceedings of International Conference on Mobile Computing and Networking (MobiCom)*, pages 173–184, 2012.

[155] Kan Yang, Kuan Zhang, Ju Ren, and Xuemin Shen. Security and privacy in mobile crowdsourcing networks: challenges and opportunities. *IEEE Communications Magazine*, 53(8):75–81, 2015.

[156] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–356, 2019.

[157] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, and William J Knottenbelt. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In *Proceedings of International Conference on Financial Cryptography and Data Security (FC)*, pages 31–42, 2018.

[158] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 270–282, 2016.

[159] Yu Zhang and Mihaela Van der Schaar. Reputation-based incentive protocols in crowdsourcing applications. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pages 2140–2148, 2012.

[160] Dong Zhao, Xiang-Yang Li, and Huadong Ma. How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pages 1213–1221, 2014.

[161] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.