Journal of Blockchain Research Volume 1 (2022) 19–27

Scalable and secure crowdsourcing atop blockchain via off-chain payment

Anxin Zhou, Cong Wang, and Xiaohua Jia*

Blockchain facilitates atomic exchange by serving as a decentralized coordinator. It recently has been applied to crowdsourcing for settling the payment between the mutually untrusted data collector and data providers. However, this is still not practical due to the blockchain's poor performance. This paper presents a crowdsourcing system that uses the blockchain in a secure and scalable fashion. A crowdsourcing task can involve an unlimited number of data providers but with a constant transaction cost. For this purpose, we design an off-chain payment system based on trusted hardware. In this way, the data collector can use off-chain payment for crowdsourcing, and the blockchain mainly performs lightweight verification to ensure honest payment. We implement and evaluate an application of text data collection. The evaluation shows the desirable performance.

Keywords and phrases: Blockchain, Off-chain payment, Crowdsourcing, Trusted hardware.

1. INTRODUCTION

Crowdsourcing is a common approach for data collection. A data collector can solicit data from many different sources. A known problem in crowdsourcing is false reporting and free riding [12]. If the data collector receives data before payment, it may pay less or even reject payment. Conversely, if the data collector pays first, data providers may submit low-quality data or even reject that. In practice, this conflict is handled by a crowdsourcing platform [17, 18] that is trusted to deliver the right data and payments.

However, such centralized trust is not always reliable. Incidents such as in-house misbehavior [11] and bias in resolving disputes [12] are still occurring. Recent works (e.g., [11, 12]) instead use the blockchain to settle payment and thus makes the trust spread out. However, this meanwhile hurts scalability. Blockchains such as Ethereum and Bitcoin still have low throughput and high transaction fees. It is not practical to heavily use a blockchain for crowdsourcing.

This paper presents a crowdsourcing system that uses the blockchain to settle payment in a secure and scalable fashion. A crowdsourcing task can involve an unlimited number of data providers but with a constant transaction cost. Our idea is to use off-chain payment [3, 8]. An untrusted server, which could be a crowdsourcing platform, helps collect encrypted data from data providers. The data collector makes payments with the off-chain funds that can later be transferred to the blockchain. The blockchain verifies the off-chain payment and punishes a dishonest data collector.

Unfortunately, existing off-chain payment systems do not consider the payment verification for a blockchain application. Thus, we design a new one based on the popular commodity hardware, Intel SGX [2], which allows creating a protected memory region, called enclave. The untrusted server is required to use an enclave program to manage off-chain funds. The enclave ensures that these funds can only be updated by the prescribed operations. It can also provide an attestation for the blockchain to verify off-chain payment.

The remaining challenge is to support secure persistent storage for off-chain balance. The enclave only has volatile memory. Although the bulletin sealing mechanism can persist the balance outside the enclave with integrity protection, this mechanism is vulnerable to rollback attacks [13]. An enclave cannot know whether the sealed balance are latest or not. This allows the untrusted server to spend the same funds twice. To address this issue, we follow the idea in [8] and use the blockchain to decide if an update on the balance can be accepted. The blockchain tracks the latest balance and will reject an update on an old version.

We implement an application for a data collector to collect text data. We use the state-of-the-art unsupervised algorithm, TextTruth [22], to guarantee the data quality. We implement TextTruth as an enclave program so that the untrusted server can help evaluate data quality but without seeing plaintext data. According to the evaluation on our local Ethereum testnet and a commodity PC, a crowdsourcing task has a gas cost of 275.1k, about 13x of the minimum transaction cost 21k on Ethereum. Off-chain payment has a throughput of 9k/s, which outperforms Ethereum's throughput that is about 15/s.

In summary, we make the following contributions.

- We design a blockchain-based crowdsourcing system that can settle the payment in crowdsourcing with a constant transaction cost.
- We design an off-chain payment system that allows the payment verification for a blockchain application.
- We implement and evaluate an application of text data collection.

*Corresponding author.

The remainder of this paper is organized as follows. Section 2 formulates our problem. Section 3 introduces SGX enclaves. Section 4 presents the off-chain payment system. Section 5 presents the crowdsourcing system. Section 6 describes the application of text data collection. Section 8 discusses related work. Section 9 discusses our work. Section 10 concludes this paper.

2. PROBLEM FORMULATION

2.1 System model

The data collector is the entity that posts a crowdsourcing task to request data and make off-chain payments. Data providers are the entities that provide data to the task for receiving payments. The enclave server is the entity that provides crowdsourcing and off-chain payment services. The blockchain supports smart contracts for verifying payment and punishing a dishonest data collector.

Fig. 1 shows the crowdsourcing workflow. Data providers first submit encrypted data to the server, which notifies the blockchain about the amount that the data collector should pay. Then the data collector makes off-chain payments, and the blockchain verifies the payment. Finally, the server sends the encrypted data to the data collector.

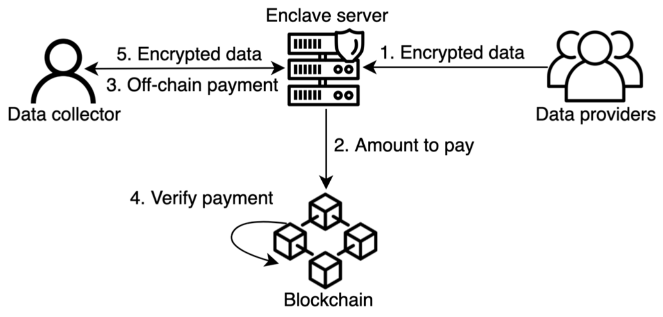


Figure 1. Crowdsourcing system.

2.2 Threat model

The data collector is rational and will pay honestly if the blockchain can punish incorrect payment.

Data providers may submit low-quality data to cheat for payments. This assumption could be adjusted based on how to detect low-quality data. For example, in a truth discovery algorithm [22], some data providers need to be reliable.

The server is a covert adversary that will not misbehave if its operations are verifiable. The server may be a crowdsourcing platform that aims for best services but may misbehave covertly. As in [5, 14], we also assume that the server will not collude with the data collector. Otherwise, the data collector could receive data without making payments.

The enclave and blockchain are trusted for their functionalities. Similar to many enclave applications [4], we do not consider side-channel attacks.

2.3 Problem definition

Our first goal is to design a crowdsourcing system where a data collector can post a task to request data, and data providers can submit data to receive payments. The system should satisfy:

Atomicity: A task succeeds if the data collector receives data, and data providers receive payments. A task fails if neither occurs.
Scalability: A task has a constant transaction cost.
Confidentiality: Only the data collector can receive plaintext data.

Our second goal is to design an off-chain payment system where a user can transfer on-chain funds to its off-chain account for payment and transfer the off-chain funds back on demand. The system should satisfy:

Integrity: Only the prescribed operations that manage the funds can take effect.
Atomicity: An operation either succeeds according to its specification or fails without changing the funds.
Verifiability: The blockchain can verify off-chain payment.

3. BACKGROUND

SGX [2] is an instruction set built into some Intel CPUs, which allows an enclave program to define protected regions of memory called enclaves, and only allows the enclave program to access the code and data inside an enclave.

Remote attestation and abstraction. SGX provides remote attestation for users to verify that an enclave program is correctly running on an untrusted server. The enclave program first generates a report containing the enclave identity and the application data (digest) that can specify the program outputs to be attested. Next, the quote enclave (provided by Intel) on the same server converts the report to an attestation. Finally, the users or the server can send the attestation to Intel Attestation Service (IAS), which checks the attestation and returns a report that is publicly verifiable.

To avoid frequent contact with IAS, as in [21], an enclave program can generate a key pair of a digital signature scheme and run the remote attestation once to attest the public key. The enclave program later uses its signature to attest execution.

Motivated by [15], we abstract the above remote attestation procedure as $\Sigma_{\mathsf{att}} = (\mathsf{Attest}, \mathsf{Verify})$:

- $\Sigma_{\mathsf{att}}.\mathsf{Attest}(\mathsf{eid}, \mathsf{fid}, \mathsf{outps}) \rightarrow \sigma_{\mathsf{att}}$. The algorithm attests the outputs of an enclave function.
- $\Sigma_{\mathsf{att}}.\mathsf{Verify}(\mathsf{eid}, \mathsf{fid}, \mathsf{outps}, \sigma_{\mathsf{att}}) \rightarrow 0/1$. The algorithm verifies the attestation of the function outputs.

In the remaining part of this paper, we will omit the enclave identity and function identity for simplicity.

Data persistence and deficiency. Due to the volatile enclave memory, the data inside an enclave may get lost for many reasons (e.g., a power outage). To support secure persistent storage, SGX allows an enclave to seal data outside the enclave, and only the enclave or enclave author (in some design) can retrieve the sealed data.

However, the sealing mechanism is vulnerable to rollback attacks [13]. The enclave program cannot know whether the sealed data is the latest after a restart. Although an enclave can use the bulletin hardware counters to track versions, this mechanism still has performance and security issues [13].

4. OFF-CHAIN PAYMENT SYSTEM

4.1 Overview

A user of the system can request the server to execute three operations. The two operations, On2Off and Off2On, allow a user to transfer funds between its on-chain and off-chain accounts. The other operation Pay allows making payments with off-chain funds.

Fig. 2 shows the workflow. The user first issues a request to execute an operation. Then the enclave server updates the off-chain balance and commits the update on the blockchain. If the current operation is On2Off or Off2On, the blockchain also updates the on-chain balance.

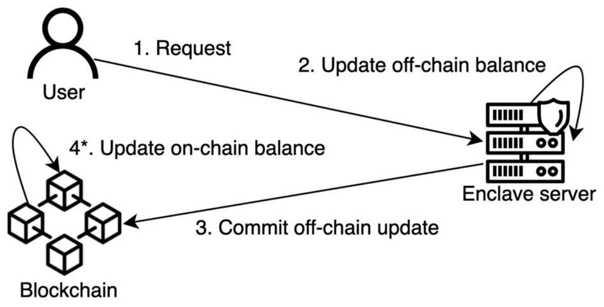

1. Request
User
2. Update off-chain balance
4*. Update on-chain balance
Enclave server
3. Commit off-chain update
Blockchain

Figure 2. Off-chain payment system. Note that off-chain payment does not include Step 4.*

In the above workflow, the server is required to commit the off-chain update on the blockchain so that the enclave program can securely persist off-chain balance outside the enclave. In this way, the server cannot replay an old version of the balance because the blockchain can check the version before accepting the update.

For checking the version, the blockchain will checkpoint the off-chain balance. In more detail, the enclave program will record the off-chain balance in a ledger L (e.g., Merkle tree) and use the ledger digest $\mathsf{H(L)}$ as a checkpoint. (H is a hash function). In normal cases, the blockchain and enclave program will agree on the same digest. So, if the server uses an old version of the balance, the blockchain can detect that due to the different digest.

In the following, we will present the design of the operations (Fig. 3), where the server uses the enclave program (Fig. 4) to update the off-chain balance and uses the smart contract (Fig. 5) to commit off-chain updates and update the on-chain balance. We only consider one user for simplicity. It is easy to extend our design to many users.

4.2 Off-chain payment

A user can issue a request $\mathsf{req_{pay}} = \{(\mathsf{toAcnt_i}, \mathsf{amnt_i})_{i=1}^{k}, \mathsf{fromAcnt}, \mathsf{msg}\}$, to transfer the amount $\mathsf{amnt_i}$ from the account $\mathsf{fromAcnt}$ to the account $\mathsf{toAcnt_i}$. The message msg could be used in crowdsourcing for a data collector to specify which task to pay. Here we assume the request has been signed by the account $\mathsf{fromAcnt}$.

Update off-chain balance. In this step, the server calls the enclave function Pay to update the off-chain balance. The function transfers the amount $\mathsf{amnt_i}$ from the account $\mathsf{fromAcnt}$ to the account $\mathsf{toAcnt_i}$ and returns the attestation σ_{pay} that attests $\{\mathsf{H(L_{old})}, \mathsf{H(L_{new})}, \mathsf{H(req_{pay})}\}$. The two digests, $\mathsf{H(L_{old})}$ and $\mathsf{H(L_{old})}$, are the digests before and after the off-chain update. The digest $\mathsf{H(L_{old})}$ will be used to verify that the current update uses the correct ledger. The digest $\mathsf{H(L_{new})}$ will be used to verify the next update.

Commit off-chain update. In this step, the server invokes the smart contract function CmtPay, which will do some checks before committing the off-chain update. The function first checks the attestation σ_{att} to ensure that the function inputs are from the enclave program. Then the function checks if the digest $\mathsf{H(L_{old})}$ is already on-chain to ensure that the off-chain update uses the correct ledger. Finally, the function stores $\{\mathsf{H(L_{new})}, \mathsf{H(req_{pay})}\}$ on the blockchain. The digest $\mathsf{H(L_{new})}$ will be used to verify the next off-chain update. The digest $\mathsf{H(req_{pay})}$ allows one (e.g., the user or the blockchain in the crowdsourcing system) to verify the corresponding payment.

4.3 On2Off/Off2On transfer

We only present the design of the operation Off2On because the two operations are similar. A user can issue a request $\mathsf{req_{f2n}} = \{\mathsf{offAcnt}, \mathsf{onAcnt}, \mathsf{amnt}\}$ to transfer the amount amnt from the off-chain account $\mathsf{offAcnt}$ to the on-chain account onAcnt. Here we assume the request has been signed by the account $\mathsf{offAcnt}$. We only consider one transfer in this request since it is trivial to batch many transfers.

Update off-chain balance. In this step, the server calls the enclave function F2N to update off-chain balance. The function debits the amount amnt from the account $\mathsf{offAcnt}$ and returns the attestation σ_{f2n} that attests $\{\mathsf{H(L_{old})}, \mathsf{H(L_{new})}, \mathsf{H(req_{f2n})}\}$.As in the operation Pay, the digest $\mathsf{H(L_{old})}$ will be used to verify that the current off-chain update uses the correct ledger, and the digest $\mathsf{H(L_{new})}$ will be used to verify the next update.

Commit off-chain update & Update off-chain balance. In this step, the server invokes the smart contract function CmtF2N, which will do some checks before committing

```
**Operation** Pay
1: The user sends to the server the request $\mathsf{req_{pay}} = \{(\mathsf{toAcnt_i}, \mathsf{amnt_i})_{i=1}^{k}, \mathsf{fromAcnt}, \mathsf{msg}\}$.
2: The server calls the enclave function $\mathsf{Pay}(\mathsf{req_{pay}})$, which transfers the amount $\mathsf{amnt_i}$ from the account $\mathsf{fromAcnt}$ to the
   account $\mathsf{toAcnt_i}$ for each $i = 1, .., k$. The function returns $\{\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}}), \sigma_{\mathsf{pay}}\}$.
3: The server calls the smart contract function CmtOffPay with the above outputs. After checking that $\sigma_{\mathsf{pay}}$ is correct and
   $\mathsf{H}(\mathsf{L_{old}})$ is on-chain, the function stores $\{(\mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}})\}$ on the blockchain.

**Operation** On2Off/Off2On
1: The user sends to server the request $\mathsf{req_{n2f/f2n}} = (\mathsf{offAcnt}, \mathsf{onAcnt}, \mathsf{amnt})$.
2: The server calls the enclave function $\mathsf{NF2/F2N}(\mathsf{req_{n2f/f2n}})$, which credits/debits the amount amnt to/from the account
   offAcnt. The function returns $\{\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}}), \sigma_{\mathsf{n2f/f2n}}\}$.
3: The server calls the smart contract function CmtN2F/CmtF2N with the above outputs. After checking that $\sigma_{\mathsf{n2f/f2n}}$ is
   correct and $\mathsf{H}(\mathsf{L_{old}})$ is on-chain, the function stores $\{(\mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}})\}$ on the blockchain and debits/credits the
   amount amnt from/to the account onAcnt.
```

Figure 3. Operation Pay, On2Off, *and* Off2On *in off-chain payment system.*

```
**Function** Pay
  • Input: request $\mathsf{req_{pay}}$
  • Output:
      – digest $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}})$
      – attestation $\sigma_{\mathsf{pay}}$
1: Let $\mathsf{H}(\mathsf{L_{old}})$ be the ledger digest
2: for $i = 1 : k$ do
3:     Transfer $\mathsf{amnt_i}$ from fromAcnt to $\mathsf{toAcnt_i}$
4: Let $\mathsf{H}(\mathsf{L_{new}})$ be the ledger digest
5: Let $\sigma_{\mathsf{pay}} = \Sigma_{\mathsf{att}}.\mathsf{Attest}(\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}}))$
6: Output $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}}), \sigma_{\mathsf{pay}}$

**Function** N2F/F2N
  • Input: request $\mathsf{req_{n2f/f2n}}$
  • Output:
      – digest $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}})$
      – attestation $\sigma_{\mathsf{n2f/f2n}}$
1: Let $\mathsf{H}(\mathsf{L_{old}})$ be the ledger digest
2: Credit/Debit amnt to/from offAcnt
3: Let $\mathsf{H}(\mathsf{L_{new}})$ be the ledger digest
4: Let $\sigma_{\mathsf{n2f/f2n}} = \Sigma_{\mathsf{att}}.\mathsf{Attest}(\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}}))$
5: Output $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}}), \sigma_{\mathsf{n2f/f2n}}$
```

Figure 4. Enclave program in off-chain payment system.

```
**Function** CmtPay
  • Input:
      – digest $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}})$
      – attestation $\sigma_{\mathsf{pay}}$
1: Assert $\Sigma_{\mathsf{att}}.\mathsf{Verify}(\sigma_{\mathsf{pay}}, \mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}}))$
2: Assert $\mathsf{H}(\mathsf{L_{old}})$ is on-chain
3: Store $\mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{pay}})$ on the blockchain

**Function** CmtN2F/CmtF2N
  • Input:
      – digest $\mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}})$
      – request $\mathsf{req_{n2f/f2n}}$
      – attestation $\sigma_{\mathsf{n2f/f2n}}$
1: Assert $\Sigma_{\mathsf{att}}.\mathsf{Verify}(\sigma_{\mathsf{n2f/f2n}}, \mathsf{H}(\mathsf{L_{old}}), \mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}}))$
2: Assert $\mathsf{H_{old}}$ is on-chain
3: Debit/Credit amnt from/to onAcnt
4: Store $\mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{n2f/f2n}})$ on the blockchain
```

Figure 5. Smart contract in off-chain payment system.

the off-chain update and updating the on-chain balance. As in the operation Pay, the function checks the attestation σ_{f2n} to ensure that the function inputs are from the enclave program; and checks that the digest $\mathsf{H}(\mathsf{L_{old}})$ has been stored on the blockchain to ensure that the off-chain update uses the correct ledger. Finally, the function stores $\{\mathsf{H}(\mathsf{L_{new}}), \mathsf{H}(\mathsf{req_{f2n}})\}$ on the blockchain and credits the amount amnt to the account onAcnt.

Note that the operation On2Off still has a potential issue. This is because the off-chain update is before the on-chain update. The operation could fail if the user consumes its on-chain balance before the on-chain update starts. This does not affect the security of our design because, in this case, the blockchain will not accept the off-chain update. When this issue becomes a concern, the operation may start with locking the on-chain balance.

4.4 Analysis

It is easy to see our system satisfies verifiability ($2.2). We argue that the system also satisfies integrity and atomicity.

The system achieves integrity because only the smart contract and enclave program can update the on-chain and off-chain balance. Although the server can replay a stale ledger to the enclave program, this will not take effect because the blockchain can verify that with the ledger digest.

To see how the system achieves atomicity, we analyze the possible results of an operation. In the system, an operation will first be executed by the enclave program and then the smart contract. The part executed by the enclave program will not take effect unless it is committed on the blockchain. In this case, the smart contract will finish the remaining part of the execution. If the execution can be finished, the operation succeeds. Otherwise, the operation fails without changing the on-chain or off-chain funds.

5. CROWDSOURCING SYSTEM

5.1 Overview

To settle the payment in crowdsourcing efficiently, our idea is to use off-chain payment. The data collector makes payments with off-chain funds. A rationale data collector will behave honestly because the blockchain can verify that and punish dishonest payment.

As shown in the previous section, the off-chain payment system allows the blockchain to verify payment by checking the corresponding request's digest $\mathsf{H}(\mathsf{req}_{\mathsf{pay}})$ is on-chain. The blockchain still needs to know the correct request to be used for crowdsourcing. In other words, it should know how much the data collector should pay each data provider.

An intuitive solution is to let data providers submit data to the blockchain so that the blockchain can compute the amount to pay each data provider. However, this breaks confidentiality because the blockchain is transparent. This is also not scalable because the transaction cost is linear with the number of data providers.

Instead, we let the server receive encrypted data from data providers, compute the payment amount via the enclave program, and stores on the blockchain a digest about the payment amount. The blockchain can verify that with the enclave's attestation. In this way, the confidentiality is preserved, and the transaction cost becomes constant.

Note that for simplicity, we omit the details about punishing dishonest payment. In practice, the data collector may be required to deposit collateral on the blockchain. In case of dishonest payment, the collateral may be used to compensate data providers.

5.2 Design details

The following describes the design of the crowdsourcing procedures (Fig. 6), where the enclave program (Fig. 7) is used to compute the payment amount, and the smart contract (Fig. 8) is used to verify the payment.

Post task. The data collector calls the smart contract function $\mathsf{PostTask}$ to set up the task on the blockchain. The function stores on the blockchain the task information $\mathsf{taskInfo} = \{\mathsf{pk_c}, \mathsf{cAcnt}, \mathsf{budget}, *\}$. The public key $\mathsf{pk_c}$ is to disclose encrypted data after crowdsourcing. The account cAcnt is for off-chain payment. The budget is the amount that the data collector is willing to pay. The symbol $*$ denotes other application-specific data.

Submit data. Data providers send to the server $\{\mathsf{data}, \mathsf{pAcnts}\}$, where pAcnts are the accounts to receive payments. We assume that before $\{\mathsf{data}, \mathsf{pAcnts}\}$ is sent out, it is already encrypted and authenticated by the secret key shared with the enclave program. This could be implemented by remote attestation and key exchange [15].

Compute payment amount. The server first calls the enclave function $\mathsf{CmptPayAmnt}$. We assume the server will use the correct $\mathsf{taskInfo}$ as the function input because before submitting data, data providers can verify that via remote attestation. This assumption prevents the server from using its own public key in $\mathsf{taskInfo}$ to decrypt the output $\mathsf{data_{ct}}$ that is supposed to be decrypted only by the data collector.

The enclave function first uses a data valuation algorithm $\mathsf{ValuateData}(\mathsf{data}, \mathsf{pAcnts}, \mathsf{budget}) \rightarrow (\mathsf{pAcnt_i}, \mathsf{amnt_i})_{\mathsf{i}=1}^{\mathsf{k}}$ to filter out low-quality data and compute the amount $\mathsf{amnt_i}$ that the data collector should transfer to the account $\mathsf{pAcnt_i}$. This algorithm could be customized by the data collector before crowdsourcing. The function finally outputs

Post task

1: The data collector calls the smart contract function $\mathsf{PostTask}(\mathsf{taskInfo})$, which stores on the blockchain the task information $\mathsf{taskInfo} = (\mathsf{pk_c}, \mathsf{cAcnt}, \mathsf{budget}, *)$.

Submit data

1: Data providers send to the server $\{\mathsf{data}, \mathsf{pAcnts}\}$, which are assumed to be encrypted and authenticated by the secret key shared with the enclave program.

Compute payment amount

1: The server calls the enclave function $\mathsf{CmptPayAmnt}(\mathsf{taskInfo}, \mathsf{data}, \mathsf{pAcnts})$, which computes the payment amount according to the data valuation algorithm $\mathsf{ValuateData}$. The function returns $\{\mathsf{req_{pay}}, \mathsf{data_{ct}}, \sigma_{\mathsf{att}}\}$.
2: The server calls the smart contract function $\mathsf{SubmPayAmnt}(\mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}}), \sigma_{\mathsf{att}})$, which verifies the attestation σ_{att} and stores $\{\mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}})\}$ on the blockchain.

Make off-chain payments

1: The server sends the request $\mathsf{req_{pay}}$ to the data collector, which verifies the request by checking that $\mathsf{H}(\mathsf{req_{pay}})$ is on-chain.
2: The data collector makes off-chain payments with the request and calls the smart contract function $\mathsf{VerifyPay}()$. The function verifies the payment by checking that $\mathsf{H}(\mathsf{req_{pay}})$ is in the smart contract of the off-chain payment system.
3: The server sends the ciphertext $\mathsf{data_{ct}}$ to the data collector, which checks that $\mathsf{H}(\mathsf{data_{ct}})$ is on the blockchain.

Figure 6. Crowdsourcing procedures.

```
Function CmptPayAmnt
  • Input:
      – task information taskInfo
      – data data
      – accounts pAcnts
  • Output:
      – request $\mathsf{req_{pay}}$
      – ciphertext $\mathsf{data_{ct}}$
      – attestation $\sigma_{att}$
1: Parse taskInfo as $(\mathsf{pk_c}, \mathsf{cAcnt}, \mathsf{budget}, *)$
2: Let $(\mathsf{pAcnt_i}, \mathsf{amnt_i})_{i=1}^{k} = \mathsf{ValuateData}(\mathsf{data}, \mathsf{pAcnts}, \mathsf{budget})$
3: Let $\mathsf{req_{pay}} = \{(\mathsf{pAcnt_i}, \mathsf{amnt_i})_{i=1}^{k}, \mathsf{cAcnt}, \mathsf{taskInfo}\}$
4: Let $\mathsf{data_{ct}}$ be the result of encrypting data with $\mathsf{pk_c}$
5: Let $\sigma_{att} = \Sigma_{att}.\mathsf{Attest}(\mathsf{taskInfo}, \mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}}))$
6: Output $\mathsf{req_{pay}}$, $\mathsf{data_{ct}}$, $\sigma_{att}$
```

Figure 7. Enclave program in crowdsourcing system.

```
Function PostTask
  • Input: task information taskInfo
1: Store taskInfo on the blockchain

Function SubmPayAmnt
  • Input:
      – digest $\mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}})$
      – attestation $\sigma_{att}$
1: Assert $\Sigma_{att}.\mathsf{Verify}(\sigma_{att}, \mathsf{taskInfo}, \mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}}))$
2: Store $\{\mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}})\}$ on the blockchain

Function VerifyPay
1: Assert $\mathsf{H}(\mathsf{req_{pay}})$ is in the smart contract of the off-chain
   payment system
```

Figure 8. Smart contract in crowdsourcing system.

$\{\mathsf{req_{pay}}, \mathsf{data_{ct}}, \sigma_{att}\}$. The request $\mathsf{req_{pay}}$ is based on the above payment amount and will be used for the data collector to make off-chain payments. The ciphertext $\mathsf{data_{ct}}$ is encrypted with the public key $\mathsf{pk_c}$ of the data collector.

Then the server calls the smart contract function SubmPayAmnt. The function first verifies the attestation σ_{att} to ensure that function inputs are from the enclave program. Then the function stores the two digests $\{\mathsf{H}(\mathsf{req_{pay}}), \mathsf{H}(\mathsf{data_{ct}})\}$ on the blockchain. The digest $\mathsf{H}(\mathsf{req_{pay}})$ will be used to verify the off-chain payment. The digest $\mathsf{H}(\mathsf{data_{ct}})$ will be used for the data collector to verify the data received from the server.

Make off-chain payments. The server sends the request $\mathsf{req_{pay}}$ to the data collector, which verifies the request by checking that the digest $\mathsf{H}(\mathsf{req_{pay}})$ is on-chain.

The data collector then makes off-chain payments with the request $\mathsf{req_{pay}}$ and calls the smart contract function VerifyPay, which verifies the payment by checking that the digest $\mathsf{H}(\mathsf{req_{pay}})$ is already in the smart contract of the off-chain payment system. Note that the digest $\mathsf{H}(\mathsf{req_{pay}})$ stored in the previous procedure only belongs to the smart contract of the crowdsourcing system. Although it is feasible to use the same smart contract, this will limit the application scope of the off-chain payment system.

Finally, the server sends the ciphertext $\mathsf{data_{ct}}$ to the data collector, which verifies the ciphertext by checking that $\mathsf{H}(\mathsf{data_{ct}})$ is on-chain. The data collector can decrypt the ciphertext $\mathsf{data_{ct}}$ with its public key $\mathsf{pk_c}$.

5.3 Analysis

It is easy to verify that the system achieves confidentiality and scalability ($2.2). We argue that the system also achieves atomicity.

If the data collector does not make honest payments, it cannot receive data from the server because, by assumption, the server will not collude with the data collector. On the other hand, after the data collector makes payments, it will receive the correct data from the server because, by assumption, the server is a covert adversary that will not misbehave when its operations are verifiable. So, we only need to show that the server can verify the payment, and the data collector can verify the data.

The server can know that after the data collector completes the smart contract function VerifyPay. The data collector can know that after the server puts $\mathsf{H}(\mathsf{data_{ct}})$ on the blockchain with the smart contract function SubmPayAmnt.

6. TEXT DATA COLLECTION

Many works [14, 23] have studied crowdsourcing with privacy protection and quality assurance. But they treat text data equally as numerical data, making the quality evaluation less effective [22]. On the other hand, the quality evaluation algorithm [22] dedicated to text data is often too complex to be made privacy-preserving by crypto-based approaches. This motivate us to implement an application of text data collection based on our crowdsourcing system, where the quality of text data is evaluated inside the enclave so as to achieve both privacy and efficiency.

We adopt the state-of-the-art unsupervised algorithm TextTruth [22] to evaluate the quality of text data. In the context of TextTruth, a crowdsourcing task consists of many text questions, such as “What are the symptoms of flu?”. TextTruth somehow uses the majority to decide if an answer is reliable, so each text question will be answered by multiple data providers.

In our work, we use TextTruth to implement the data valuation algorithm ValudateData. Based on the data quality from TextTruth, we allocate the payment amount to each data provider proportionally; we regard it as an independent interest to use a different incentive mechanism. To make TextTruth more efficient, we adopt the suggestion in its paper and set the concentration parameter of the vMF distribution to ∞ as in [16].

7. EVALUATION

7.1 Methodology

In the crowdsourcing system and off-chain payment system, the smart contracts and enclave programs play a key role. Thus, we aim to answer the following questions:

- What are the transaction fee and maximum throughput of each smart contract function?
- What is the throughput of each enclave function?

Here we target the maximum throughput of each smart contract function because in practice there could be other applications using the same blockchain.

For Ethereum, the transaction fee of a smart contract function is computed from the gas cost, which in turn is computed according to the low-level opcodes used by the function. Specifically, the transaction fee is computed by:

$$\text{Tx fee} = \text{Gas cost} * \text{Gas price} * \text{ETH price}$$

The maximum throughput is estimated by:

$$\text{Max tx/s} = \frac{\text{Block gas limit}}{\text{Gas cost} * \text{Block time}}$$

We will use the average gas price of $9.1 * 10^{-8}$, ETH price of \$2868.5, block gas limit of 30M, and block time of 13.2s as per February, 2022 [19].

To evaluate the throughput of the enclave function CmptPayAmnt that relies on TextTruth, we synthesize a dataset as follows. Each data provider submits a 10-keyword answer to each question. Each keyword is a word vector of length 100 and is randomly generated.

The smart contracts are implemented with solidity 0.5.2 and tested on our local Ethereum testnet. The enclave programs are implemented with C++ and Intel SGX SDK v2.11 and tested on a PC equipped with Intel i7-9700K @ 3.60GHz CPU, 16GB RAM, and Ubuntu 18.04 LTS. The ledger in the off-chain payment system is implemented as a Merkle tree. The hash function is implemented as SHA-256.

7.2 Crowdsourcing system

Smart contract. Table 1 shows the performance of each smart contract function in the crowdsourcing system, including the function PostTask, SubmPayAmnt, and VerifyPay. Based on the result, we can estimate the transaction fee of a crowdsourcing task. In a task, each above function will be called once. Besides, the function CmtPay will be used once to make off-chain payments. In total, a task has a gas cost of 275.1k, which is about 13x of the minimum gas cost 21k of a transaction. However, a task still has a high transaction fee of \$71.8 due to the high price of ETH. We regard it as an independent interest to use a cheaper blockchain to reduce the transaction fee.

Enclave program. Table 2 shows the execution time of the enclave function CmptPayAmnt. The main part of this function is TextTruth, which has a time complexity linear with the number of questions or data providers. When the number of providers is 50, the function could process questions at a speed of 230/s, which is enough for many crowdsourcing tasks. When this throughput becomes a concern, one may optimize our implementation or use a more efficient algorithm to evaluate the quality of text data.

Table 1. Performance of smart contract functions

Function	Gas cost	Tx fee	Max tx/s
PostTask	129.4k	\$33.8	17.6
SubmPayAmnt	75.9k	\$19.8	29.9
VerifyPay	21.8k	\$5.7	104.3
CmtN2F/CmtF2N	69.8k	\$18.2	32.6
CmtPay	48.0k	\$12.5	47.3

Note that the minimum gas cost of a transaction is 21k on Ethereum.

Table 2. Execution time of enclave function CmptPayAmnt

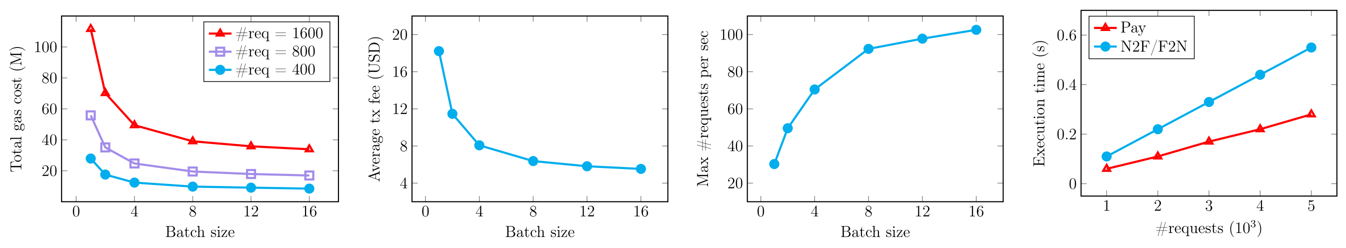
#Providers = 50	#Questions	1000	2000	3000	4000
	Exec. time	4.3s	9.0s	13.0s	16.8s
#Questions = 50	#Providers	300	900	1500	2100
	Exec. time	1.6s	4.4s	7.4s	9.7s

7.3 Off-chain payment system

Smart contract. Table 1 shows the performance of each smart contract function in the off-chain payment system, including the function CmtN2F/CmtF2N and CmtPay. The result suggests that the On2Off/Off2On operation is inefficient because every operation needs a transaction. Although it is also true for the operation Pay, a user can make an limited number of payments in one operation. This could be further improved by extending our design to multiple users.

To improve the operation On2Off/Off2On, we study the impact of batching, that is, let one function process multiple requests. We evaluate the performance of the function under different batch sizes. Fig. 9(a)-9(c) suggests that a large batch size can reduce the total gas cost and the average transaction fee for requests and increase the maximum throughput of processing requests. In more detail, batching could reduce ≈70% total gas cost and average transaction fee and increase 2.4x maximum throughput. Batching can take effect because every transaction on Ethereum has the minimal gas cost of 21k. This also implies the effect of batching is limited since it cannot avoid the other gas cost.

Enclave program. Fig. 9(d) shows the execution time of the enclave function Pay and N2F/F2N. Both the functions have a time complexity that is linear with the number of requests. Their throughput is 9k/s and 18k/s respectively.



(a) Total gas cost w.r.t. batch size under different #requests.

(b) Average transaction fee for requests w.r.t. batch size.

(c) Max throughput of processing requests w.r.t. batch size.

(d) Execution time w.r.t. #requests (10^3).

Figure 9. ***(a)-(c):*** *Performance of smart contract function* CmtN2F*.* ***(d):*** *Performance of enclave function* Pay *and* N2F/F2N*.*

Although the function N2F/F2N has a relative good performance, the operation On2Off/Off2On may still be inefficient, because the complexity of the smart contract function CmtN2F/CmtF2N is linear with the number of requests. Thus, to utilize the system, it is better for users to keep using off-chain payments.

8. RELATED WORK

Many works [11, 12] have used the blockchain for crowdsourcing, while few of them consider the scalability issue. NF-Crowd [10] is among the first to make the transaction cost constant. However, it is only applicable to specific tasks, where only a few data providers will receive payment. Besides, it relies on voting for quality evaluation while we allow the algorithm to be customized.

Existing off-chain payment systems could be categorized based on how off-chain payment is committed. The first kinds of works, payment channels [3] and ZK-rollups [1], need to submit the payment result to the blockchain. The transaction cost is linear with the number of data providers, thus not suitable for our purpose.

The second kinds of works, commit-chains [6, 8], commit off-chain payment with checkpoints. Khalil et al. [8] submit to the blockchain only the digest of the payment result. The transaction cost thus becomes constant. CommiTEE [6] utilizes enclaves and makes checkpoints off-chain. It requires no transactions to commit payment. However, commit-chains also come at a price. The users rely on a malicious server to manage off-chain funds. In case the server misbehaves, each user needs to be periodically online to ask the server for the off-chain balance and the proof regarding the latest checkpoint. In our scenario, the server is only a covert adversary. Thus, our off-chain payment system removes the above overhead for the users. Besides, we also allow the payment verification for a blockchain application, and compared to CommiTEE that also use enclaves, we also support secure persistent storage for off-chain balance.

The last kinds of works, side-chains (e.g., [7, 20]), have a trusted majority to manage off-chain funds. This avoids the transaction cost to commit off-chain payment on the (main) blockchain. We do not consider these works since they rely on more entities and a trusted majority.

9. DISCUSSION

Issues caused by stronger adversaries. Our design could fail if the adversaries are more powerful than in our threat model ($2.2). In the crowdsourcing system, if the data collector can collude with the server, it may receive data without making payments. If the server can deny services, the data collector may not be able to receive data after payment. For the off-chain payment system, if the server can deny services, the users may not be able to spend their off-chain funds or transfer the funds to the blockchain.

Comparison to (de)centralized crowdsourcing. Compared to fully decentralized crowdsourcing systems [11, 12], as a trade-off for scalability, our crowdsourcing system relies on an off-chain server. If the server is a more stronger adversary than in our threat model ($2.2), there may be issues as discussed above. Despite that, the server is still less powerful than in centralized crowdsourcing systems [17, 18]. Under our threat model, the server can only see encrypted data and cannot influence the atomic exchange between the data collector and data providers.

Alternative trusted hardware. Although our work focuses on SGX, other trusted hardware such as Keystone [9] that supports remote attestation is also applicable.

10. CONCLUSION

This paper presents a crowdsourcing system that can settle the payment in crowdsourcing with a constant transaction cost, and presents an enclave-based off-chain payment system that allows the payment verification for a blockchain application. The crowdsourcing system may inspire other blockchain applications to reduce the transaction cost. The off-chain payment system may have its independent use.

ACKNOWLEDGEMENTS

This work was supported by the RGC of Hong Kong under Grant CityU 11213920, 11217819, 11217620, 11218521, N_CityU139/21, RFS2122-1S04, and R6021-20F, by the

NSFC under Grant 61572412, and by Shenzhen Municipality Science and Technology Innovation Commission (grant no. SGDX20201103093004019, CityU).

Received 13 October 2021

REFERENCES

[1] Buterin, V. On-chain scaling to potentially 500 tx/sec through mass tx validation. https://ethresear.ch/t/on-chain-scalingto-potentially-500-tx-sec-through-mass-tx-validation/3477. Accessed 30/03/2022.
[2] Costan, V. and Devadas, S. (2016). Intel SGX Explained. *IACR Cryptol. ePrint Arch.* **2016** 1–118.
[3] Decker, C. and Wattenhofer, R. (2015). A fast and scalable payment network with bitcoin duplex micropayment channels. In *Proc. of SSS*. MR3420253
[4] Duan, H., Wang, C., Yuan, X., Zhou, Y., Wang, Q. and Ren, K. (2019). Lightbox: Full-stack protected stateful middlebox at lightning speed. In *Proc. of ACM CCS*.
[5] Duan, H., Zheng, Y., Du, Y., Zhou, A., Wang, C. and Au, M. H. (2019). Aggregating Crowd Wisdom via Blockchain: A Private, Correct, and Robust Realization. In *Proc. of IEEE PerCom*.
[6] Erwig, A., Faust, S., Riahi, S. and Stöckert, T. (2020). CommiTEE: An Efficient and Secure Commit-Chain Protocol using TEEs. *IACR Cryptol. ePrint Arch.* **2020** 1486.
[7] Gaži, P., Kiayias, A. and Zindros, D. (2019). Proof-of-stake sidechains. In *Proc. of IEEE S&P*.
[8] Khalil, R., Zamyatin, A., Felley, G., Moreno-Sanchez, P. and Gervais, A. (2018). Commit-Chains: Secure, Scalable Off-Chain Payments. Cryptology ePrint Archive, Report 2018/642. https://eprint.iacr.org/2018/642.
[9] Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K. and Song, D. (2020). Keystone: An open framework for architecting trusted execution environments. In *Proc. of ACM EuroSys*.
[10] Li, C., Palanisamy, B., Xu, R., Wang, J. and Liu, J. (2020). NF-Crowd: Nearly-free Blockchain-based Crowdsourcing. In *Proc. of IEEE SRDS*.
[11] Lu, Y., Tang, Q. and Wang, G. (2018). ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain. In *Proc. of IEEE ICDCS*.
[12] Lu, Y., Tang, Q. and Wang, G. (2020). Dragoon: Private Decentralized HITs Made Practical. In *Proc. of IEEE ICDCS*.
[13] Matetic, S., Ahmed, M., Kostiainen, K., Dhar, A., Sommer, D., Gervais, A., Juels, A. and Capkun, S. (2017). ROTE: Rollback protection for trusted execution. In *Proc. of USENIX Security*.
[14] Miao, C., Su, L., Jiang, W., Li, Y. and Tian, M. (2017). A lightweight privacy-preserving truth discovery framework for mobile crowd sensing systems. In *Proc. of IEEE INFOCOM*.
[15] Pass, R., Shi, E. and Tramer, F. (2017). Formal abstractions for attested execution secure processors. In *Proc. of EUROCRYPT*. MR3652106
[16] Straub, J., Campbell, T., How, J. P. and Fisher, J. W. (2015). Small-variance nonparametric clustering on the hypersphere. In *Proc. of IEEE CVPR*.
[17] Appen. https://appen.com/. Accessed 30/03/2022.
[18] Amazon Mechanical Turk. https://www.mturk.com/. Accessed 30/03/2022.
[19] Etherscan. https://etherscan.io/. Accessed 30/03/2022.
[20] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A. and Knottenbelt, W. (2019). Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *Proc. of IEEE S&P*.
[21] Zhang, F., Cecchetti, E., Croman, K., Juels, A. and Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In *Proc. of ACM CCS*.
[22] Zhang, H., Li, Y., Ma, F., Gao, J. and Su, L. (2018). Texttruth: an unsupervised approach to discover trustworthy information from multi-sourced text data. In *Proc. of ACM SIGKDD*.
[23] Zheng, Y., Duan, H., Yuan, X. and Wang, C. (2017). Privacy-aware and efficient mobile crowdsensing with truth discovery. *IEEE TDSC*.

Anxin Zhou
City University of Hong Kong, Hong Kong
City University of Hong Kong Shenzhen Research Institute, China
E-mail address: anxin.zhou@my.cityu.edu.hk

Cong Wang
City University of Hong Kong, Hong Kong
City University of Hong Kong Shenzhen Research Institute, China
E-mail address: congwang@cityu.edu.hk

Xiaohua Jia
City University of Hong Kong, Hong Kong
E-mail address: csjia@cityu.edu.hk