# Privacy-Preserving Energy Storage Sharing with Blockchain

Nan Wang
vincent.wang@anu.edu.au
Australian National University

Sid Chi-Kin Chau
sid.chau@anu.edu.au
Australian National University

Yue Zhou
yue.zhou@anu.edu.au
Australian National University

## ABSTRACT

Energy storage provides an effective way of shifting temporal energy demands and supplies, enabling significant cost reduction under dynamic energy pricing. Despite its promising benefits, the cost of present energy storage remains expensive, presenting a major obstacle to practical deployment. A more viable solution to improve cost-effectiveness is by sharing energy storage, such as community sharing, cloud energy storage and peer-to-peer sharing. However, revealing private energy demand data in energy storage sharing may compromise user privacy, susceptible to data misuses and breaches. In this paper, we explore a novel approach to support energy storage sharing with privacy protection, based on privacy-preserving blockchain and secure multi-party computation. We present an integrated solution to enable privacy-preserving energy storage sharing, such that energy storage service scheduling and cost-sharing can be attained without the knowledge of individual users' demands. It also supports auditing and verification by the grid operator via blockchain. Furthermore, our privacy-preserving solution can safeguard against a dishonest majority of users, who may collude in cheating, without requiring trusted third-parties. We implemented our solution as a smart contract on real-world Ethereum blockchain platform, with empirical evaluation.

## CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Hardware → Reusable energy storage**;

## KEYWORDS

Privacy-preserving, Energy Storage Sharing, Blockchain

## 1 INTRODUCTION

Energy storage can buffer energy in a medium, which is useful for temporal shifting of energy demands and supplies. In addition to absorbing excessive renewable energy, energy storage can effectively reduce the consumption cost under dynamic time-of-use (ToU) energy pricing by storing energy during off-peak periods. However, the cost of present energy storage remains expensive. Energy storage also incurs considerable maintenance cost over time, with only limited life cycles. Hence, despite its benefits, current users are reluctant to adopt energy storage in a wide scale. Rather than postponing the use of energy storage, a more viable solution to improve cost-effectiveness of present energy storage is by *sharing energy storage* among multiple users or out-sourcing to a third-party energy storage service. Time-sharing and out-sourcing have been popular concepts in cloud computing. Energy storage may also become an out-sourcible resource in a similar fashion.

Currently, there are multiple possible paradigms of energy storage sharing. First, in *community sharing*, [25] a group of local users, who do not own individual energy storage, can connect to a shared energy storage facility. The shared energy storage will be utilized by the users based on a coordination mechanism. The associated cost will be split among the users in a fair manner. Second, a non-local third-party energy storage operator can provide an outsourcing service as *cloud energy storage* [27]. The energy storage operator can offset the energy consumption of remote users by exporting energy from its energy storage facility. Third, the users, who have their own energy storage, can pool their energy storage resources together to support each other in *peer-to-peer sharing* [7].

All of these energy storage sharing paradigms can be effectively supported by *virtual net metering* (VNM) [15, 34], which is a flexible bill crediting system for transferring the credits or debits of a user's energy account to another, even they do not share the same physical metering infrastructure. By virtual net metering, energy storage operators can transfer the credits of their energy export to offset the debits of energy import of other users. Virtual net metering has been used to enable community solar energy sharing in practice [16]. It also enables energy storage sharing among geographically distributed users and energy storage operators.

Although sharing can improve cost-effectiveness, there is also a heightened concern of user privacy nowadays. Users may need to disclose private energy demand data to a third-party in order to schedule the use of shared energy storage. This may reveal sensitive personal data (e.g., working patterns, number of occupants, and vacation periods). Potential misuses and breaches of personal data may lead to serious criminal consequences. Stricter privacy legislations are being introduced in some countries to restrict personal information revelations (e.g., GDPR in Europe), which will have an impact to third-party services like energy storage sharing.

In this paper, we propose the concept of "*privacy-preserving energy storage sharing*", by which a third-party energy storage operator should be given only minimal information for its energy storage service without compromising personal data for other purposes. The key question is how to design an effective mechanism to properly enable energy storage service scheduling and cost-sharing among users, without knowing individual users' demands.

To support privacy-preserving energy storage sharing, we draw on several technologies. First, there is a disruptive paradigm of *blockchain* (e.g., Bitcoin, Ethereum), which enables decentralized

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*e-Energy '21, June 25–28, 2021, Virtual Event, Italy*
© 2021 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-6671-7/19/06…$15.00
https://doi.org/10.1145/3307772.3328312

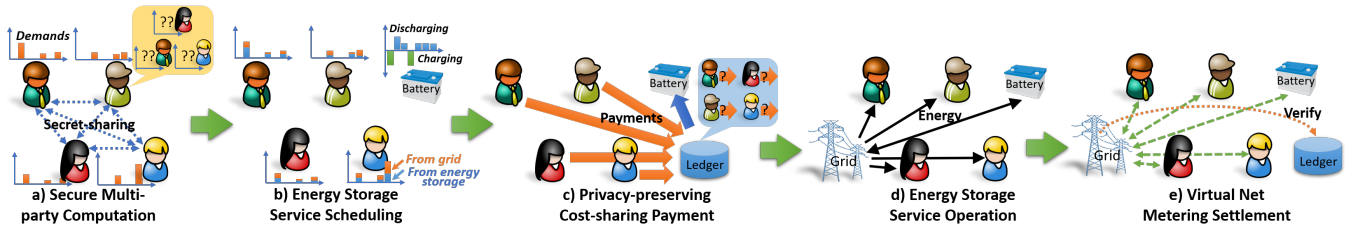



**Figure 1: An illustration of the stages of our solution for privacy-preserving energy storage sharing.**

verifiable applications without trusted intermediaries by integrating a tampering-resistant ledger with a distributed consensus protocol. Blockchain is an effective platform to support transparent energy storage sharing and auditable virtual net metering with grid operators. But blockchain by default does not ensure privacy, and transaction data is entirely disclosed on the ledger. Recently, there is a trend of supporting privacy on blockchain (e.g., privacy-oriented cryptocurrencies like ZCash, Monero [4, 31]). These blockchain platforms utilize zero-knowledge proofs for privacy-preserving digital asset management, without revealing them. In this paper, we utilize privacy-preserving blockchain to support cost-sharing and virtual net metering for privacy-preserving energy storage sharing.

Second, secure *multiple-party computation* has been a subject of extensive research [10], which provides a general framework to allow multiple parties to jointly compute a certain function while concealing the private inputs. Recently, efficient multi-party computation protocols based on secret-sharing (e.g., SPDZ) have been applied to many practical applications like privacy-preserving machine learning [8]. In this paper, we apply multiple-party computation to energy storage service scheduling with concealed individual users' demands. Moreover, we integrate multiple-party computation with privacy-preserving blockchain to support confidential cost-sharing and verifiable virtual net metering settlement.

In summary, this paper presents an integrated solution to enable privacy-preserving energy storage sharing in all the stages, as outlined in the following (and also illustrated in Figure 1):

(a),(b) **Multi-party Computation for Scheduling**: First, the users can compute their aggregate day-ahead demands by secure multi-party computation, without revealing individual demands. Then, they can derive the optimal energy storage service schedule subject to energy storage service constraints.

(c) **Privacy-preserving Cost-sharing Payment**: The users can split the cost of energy storage service based on a fair cost-sharing scheme in a privacy-preserving manner. The users can make energy storage service payments via privacy-preserving blockchain, without disclosing individual transactions. After receiving the payments, the energy storage operator will issue verifiable receipts on blockchain ledger.

(d),(e) **Operation & Virtual Net Metering Settlement**: The users and energy storage operator will follow the energy storage service schedule. They do not need to exchange energy directly, and the energy flows through the grid. They will settle their energy accounts via virtual net metering. With verifiable receipts on blockchain ledger, the users can offset their energy consumption by the energy export from energy storage, which will be audited by the grid operator.

We ensure that privacy is protected throughout the integrated process of scheduling, cost-sharing, payment and auditing, without requiring a trusted third-party. However, privacy also poses a significant challenge to the correctness and integrity of operations. Particularly, because of concealing their demands, dishonest users may attempt to cheat by paying less to energy storage service or claim more in virtual net metering than what they ought to. These dishonest users may collude to coordinate their actions. Hence, it is critical to safeguard against dishonest users. Remarkably, our privacy-preserving solution can safeguard against a *dishonest majority* of users (namely, more than 50% of users may be dishonest).

This paper is organized as follows. We first review the related work in Section 2. We formulate the problem and models in Section 3, and present the basics of cryptographic components and multi-party computation in Sections 4-C.4. The privacy-preserving solution is presented in Section 6. We provide an evaluation of our implementation on Ethereum blockchain platform in Section 7.

## 2 RELATED WORK

Optimizing energy storage under dynamic pricing is a popular research topic [14, 19, 30]. Recent studies proposed energy storage sharing among multiple users, for instance, cloud energy storage [27], virtual community sharing [25] and peer-to-peer sharing [7]. Noteworthily, there are many studies about privacy in smart grid in other aspects. For example, [21, 23] employed energy storage to hide private consumption behavior by mixing random energy storage charging and discharging to mask the consumption patterns. [35] presents privacy-preserving data aggregation for smart meters that aggregates users demands. None of these studies addressed the privacy aspect in energy storage sharing. This is the first paper to address the problem of privacy-preserving energy storage sharing.

The applications of blockchain to energy systems have received increasing attention. For example, the study [18] applied blockchain to mitigate trust in peer-to-peer electric vehicle charging. Blockchain has been applied to microgrid energy exchange and wholesale markets by prosumers [29]. Renewable energy credits and emissions trading are also applications of blockchain [22]. In these applications, the goal of blockchain is to improve transparency and reduce settlement times, since blockchain can ensure integrity and consistency of transactions and settlement on an open ledger. See a recent survey about blockchain applications to energy systems [1]. However, none of these studies considered the privacy of blockchain, even transaction data on the ledger is entirely disclosed to the public.

Our work is one of the first studies to explicitly address privacy in blockchain applications of energy systems. Supporting privacy on blockchain is a crucial research topic in cryptography and security. There are several privacy-preserving blockchain platforms with support of privacy (e.g., ZCash, Monero, Zether [4, 6, 31]). This work draws on similar concepts from privacy-preserving blockchain, but also integrates with the application of energy storage sharing.

We also survey various approaches of privacy-preserving techniques in the literature. There are two major approaches: (1) data obfuscation that masks private data with random noise, (2) secure multi-party computation that hides private data while allowing the data to be computed confidentially. Differential privacy [13], a main example of data obfuscation, is often used in privacy-preserving data mining to extract certain data properties in a relatively large dataset. There is an intrinsic trade-off between accuracy and privacy in differential privacy. On the other hand, secure multi-party computation [12, 17] traditionally employed garbled circuits [20], which have a high computational complexity, and homomorphic cryptosystems [9, 28], which need a trusted setup for key generation. Recently, information-theoretical secret-sharing [10, 11] has been utilized for secure multi-party computation, which provides high efficiency and requires no trusted third-party setup.

# 3 MODELS AND FORMULATION

In the following, we first formulate the energy storage sharing model without considering privacy. In the subsequent sections, we will incorporate privacy protection in the model.

## 3.1 Problem Setup

First, we describe several key components in the model (and list some key notations in Table 1):

(1) **Time-Varying Energy Pricing Plan:** We consider discrete timeslots, indexed by $t \in \{1, ..., T\}$, where $T$ is the number of timeslots in a day. The energy prices of a time-varying time-of-use (ToU) pricing plan at timeslot $t$ is denoted by $p(t)$. We suppose that the next-day ToU prices $(p(t))_{t=1}^{T}$ are announced before the end of today to all users and energy storage operator, such that they can plan their consumption in a day-ahead manner.

(2) **Energy Users:** There are a group of $N$ ($\geq 3$) users, indexed by $i \in \{1, ..., N\}$. Each user $i$ has certain energy demand over time, represented by a non-negative demand function $a_i(t)$. We assume that each user $i$ has non-zero demand, such that $a_i(t) > 0$ for some $t$. The users aim to reduce their energy costs by utilizing a third-party energy storage service that stores energy at lower energy prices beforehand. We consider day-ahead energy storage scheduling, whereby each user $i$ forecasts his planned energy demand $a_i(t)$ in advance, and requests energy storage service in a day-ahead manner. If the energy from energy storage service is insufficient, then user $i$ will need to acquire additional energy from the grid for the residual consumption rate denoted by $y_i(t)$ at the respective price $p(t)$.

(3) **Energy Storage Service:** The energy storage service is provided by an energy storage operator, who has energy storage characterized by capacity $\mathtt{B}(t)$, which is time-varying for modeling dynamic energy storage capacity. The energy storage is constrained by charging efficiency ratio $\mathtt{e_c} \leq 1$ and discharging efficiency ratio $\mathtt{e_d} \geq 1$, charge rate (i.e., ramp-up) constraint $\mathtt{r_c}$ and discharge rate (i.e., ramp-down) constraint $\mathtt{r_d}$. Let $b(t)$ be the current state-of-charge in the energy storage at time $t$, and $x^+(t)$ be the charging rate from the grid to the energy storage, whereas $x_i^-(t)$ be the discharging rate from the energy storage to user $i$. When the energy storage is utilized, there is a per-unit service fee at each timeslot, $\mathtt{p_s}$, which allows the energy operator to cover the wear-and-tear and maintenance cost.

Next, we will describe energy storage service scheduling in Section 3.2 and fair energy storage service cost-sharing in Section 3.3. We will present the blockchain model in Section 3.4, and incorporate privacy protection in the security and threat models in Section 3.5.

## 3.2 Energy Storage Service Scheduling

| | |
|---|---|
| $N$ | Total number of users |
| $i$ | The $i$-th user |
| $p(t)$ | Energy price of time-varying pricing scheme at timeslot $t$ |
| $\mathtt{B}(t)$ | Capacity of energy storage at timeslot $t$ |
| $\mathtt{p_s}$ | Per-unit service fee of energy storage at each timeslot |
| $b(t)$ | State-of-charge of energy storage at timeslot $t$ |
| $\mathtt{e_c}, \mathtt{e_d}$ | Charging and discharging efficiency ratios |
| $\mathtt{r_c}, \mathtt{r_d}$ | Charging and discharging rate constraints |
| $x^+(t)$ | Charging rate from the grid to the energy storage |
| $x_i^-(t)$ | Discharging rate from the energy storage to user $i$ |
| $y_i(t)$ | User $i$'s residual consumption rate from the grid at $t$ |
| $y(t)$ | Total residual consumption rate of all users at timeslot $t$ |
| $\mathtt{Cost_{ess}}$ | Total cost of energy storage service |
| $\mathtt{Cost}_i$ | User $i$'s partial original cost without energy storage service |
| $P_i^{\mathtt{pp}}$ | User $i$'s payment under proportional cost-sharing scheme |
| $P_i^{\mathtt{ega}}$ | User $i$'s payment under egalitarian cost-sharing scheme |
| $\Delta_i$ | $(= \mathtt{Cost}_i - P_i)$ User $i$'s saving from energy storage service |

**Table 1: Table of key symbols and notations.**

First, we formulate the optimization problem of energy storage service scheduling in (P1).

$$\text{(P1)} \quad \min \sum_{t=1}^{T} \Big( p(t) \cdot \Big( x^+(t) + \sum_{i=1}^{N} y_i(t) \Big) + \mathtt{p_s} \cdot x^+(t) \Big) \tag{1}$$

$$\text{s.t. } b(t+1) - b(t) = \mathtt{e_c} x^+(t) - \mathtt{e_d} \Big( \sum_{i=1}^{N} x_i^-(t) \Big), \tag{2}$$

$$0 \leq b(t) \leq \mathtt{B}(t), b(0) = 0, b(T+1) = 0, \tag{3}$$

$$x^+(t) \leq \mathtt{r_c}, \tag{4}$$

$$\sum_{i=1}^{N} x_i^-(t) \leq \mathtt{r_d}, \tag{5}$$

$$x_i^-(t) + y_i(t) = a_i(t), \tag{6}$$

$$\text{var. } b(t) \geq 0, x_i^-(t) \geq 0, y_i(t) \geq 0, x^+(t) \geq 0 \tag{7}$$

$$\forall t \in \{1, ..., T\}, \forall i \in \{1, ..., N\}$$

The objective of (P1) is the total cost, including energy storage charging $x^+(t)$ and residual consumption $y_i(t)$ at the respective energy price $p(t)$ of timeslot $t$, as well as the energy storage service fee $\mathtt{p_s} \cdot x^+(t)$. Cons. (2) updates the state-of-charge considering charging and discharging efficiency ratios. Cons. (3) ensures feasible state-of-charge. Assume the initial and final state-of-charge are 0. Cons. (4)-(5) ensure the charging and discharging rates within the respective rate constraints. Cons. (6) ensures the balance of demands, such that each user's demands are satisfied completely.

We note that (P1) however relies on the knowledge of individual user's demand $a_i(t)$. Hence, we present an alternate problem (P2).

$$\text{(P2)} \quad \min \sum_{t=1}^{T} \Big( p(t) \cdot \big( x^+(t) + y(t) \big) + \mathtt{p_s} \cdot x^+(t) \Big) \tag{8}$$

$$\text{s.t. } b(t+1) - b(t) = \mathtt{e_c} x^+(t) - \mathtt{e_d} x^-(t), \tag{9}$$

$$0 \le b(t) \le \mathtt{B}(t), b(0) = 0, b(T+1) = 0, \tag{10}$$

$$x^+(t) \le \mathtt{r_c}, \tag{11}$$

$$x^-(t) \le \mathtt{r_d}, \tag{12}$$

$$x^-(t) + y(t) = a(t), \tag{13}$$

$$\text{var. } b(t) \ge 0, x^-(t) \ge 0, y(t) \ge 0, x^+(t) \ge 0 \qquad \forall t \in \{1, ..., T\}$$

(P2) considers the total demand $a(t) = \sum_{i=1}^{N} a_i(t)$, total discharging rate $x^-(t) \triangleq \sum_{i=1}^{N} x_i^-(t)$ and total consumption rate $y(t) = \sum_{i=1}^{N} y_i(t)$, as well as the balance of the total demand in Cons. (13). By Theorem 1, energy storage service scheduling can be solved by (P2), instead of (P1), involving no individual demand $a_i(t)$.

**Theorem** 1. *If $\big(x^-(t), y(t)\big)_{t=1}^{T}$ is an optimal solution of* (P2), *then $\Big( (x_i^-(t), y_i(t))_{i=1}^{N} \Big)_{t=1}^{T}$, where $x_i^-(t) = \frac{a_i(t)}{a(t)} \cdot x^-(t)$ and $y_i(t) = \frac{a_i(t)}{a(t)} \cdot y(t)$, is an optimal solution of* (P1).

See Appendix. A for the proof.

**Remarks**: Note that when the energy storage discharges at rate $x^-(t)$, it can simultaneously compensate the users' consumption at the same rate. This can be attained via virtual net metering. We assume that the energy storage operator announces the parameters $\mathtt{p_s}, \mathtt{e_c}, \mathtt{e_d}, \mathtt{r_c}, \mathtt{r_d}, (p(t), \mathtt{B}(t))_{t=1}^{T}$ in advance. Everyone can compute the solution to (P2) with the knowledge of $(a(t))_{t=1}^{T}$.

## 3.3 Fair Cost-sharing of Energy Storage Service

Next, we formulate how the cost of energy storage service should be shared among users in a fair manner. In (P2), in addition to the cost that is paid directly by the users to the grid (i.e., $\sum_{t=1}^{T} p(t) \cdot y(t)$), there is a cost incurred by the energy storage service as follows:

$$\mathtt{Cost_{ess}} \triangleq \sum_{t=1}^{T} \big( p(t) + \mathtt{p_s} \big) \cdot x^+(t) \tag{14}$$

$\mathtt{Cost_{ess}}$ should be split fairly among the users. There are several possible ways of dividing the energy storage service cost $\mathtt{Cost_{ess}}$. Particularly, we are interested in the fair ways that take into consideration of the individual rationality of each user.

We note that without energy storage service, each user should originally pay the following cost:

$$\sum_{t=1}^{T} p(t) \cdot a_i(t) = \sum_{t=1}^{T} p(t) \cdot \big( x_i^-(t) + y_i(t) \big) \tag{15}$$

Let $\mathtt{Cost}_i \triangleq \sum_{t=1}^{T} p(t) \cdot x_i^-(t)$ be the partial original cost of user $i$ in Eqn. (15) that would have been covered by energy storage service, which provides a basis on how to split $\mathtt{Cost_{ess}}$. Note that the other part in Eqn. (15) (i.e., $\sum_{t=1}^{T} p(t) \cdot y_i(t)$) will be paid regardless of energy storage service. Noteworthily, if a user does not get any benefit from energy storage service (i.e., $a_i(t) > 0$ only when $p(t)$ is the lowest), then we have $x_i^-(t) = 0$ in (P1) and $\mathtt{Cost}_i = 0$.

Suppose that each user $i$ contributes payment $P_i$ to cover the energy storage service cost $\mathtt{Cost_{ess}}$. A cost-sharing scheme denoted by $(P_i)_{i=1}^{N}$ is called *budget-balanced*, if $\sum_{i=1}^{N} P_i = \mathtt{Cost_{ess}}$, whereas it is called *weakly budget-balanced*, if $\sum_{i=1}^{N} P_i \ge \mathtt{Cost_{ess}}$. A cost-sharing scheme $(P_i)_{i=1}^{N}$ is called *individually rational*, if $\mathtt{Cost}_i \ge P_i$ for all $i \in \{1, ..., N\}$. Evidently, each user would prefer an individually rational cost-sharing scheme. Otherwise, some users would rather not to utilize energy storage service, as it will cost more.

We next define two fair cost-sharing schemes (based on the ideas in [7]), and show them to be individually rational by Theorem 2.

### 3.3.1 *Proportional Cost-sharing Scheme.*

One simple fair way is that each user $i$ should pay proportionally to $\mathtt{Cost}_i$. Namely,

$$P_i^{\mathtt{pp}} \triangleq \mathtt{Cost_{ess}} \cdot \frac{\mathtt{Cost}_i}{\sum_{i=1}^{N} \mathtt{Cost}_i} = \sum_{t=1}^{T} \big( p(t) + \mathtt{p_s} \big) \cdot x^+(t) \cdot \frac{\sum_{t=1}^{T} x_i^-(t) \cdot p(t)}{\sum_{t=1}^{T} x^-(t) \cdot p(t)}$$

Thus, each user has the same ratio of payment over individual cost (i.e., $\frac{P_i^{\mathtt{pp}}}{\mathtt{Cost}_i} = \frac{\mathtt{Cost_{ess}}}{\sum_{i=1}^{N} \mathtt{Cost}_i}$). It is easy to check that proportional cost-sharing is budget-balanced (i.e., $\sum_{i=1}^{N} P_i^{\mathtt{pp}} = \mathtt{Cost_{ess}}$). Note that the payments are always non-negative (i.e., $P_i^{\mathtt{pp}} \ge 0$).

### 3.3.2 *Egalitarian Cost-sharing Scheme.*

Given a payment to energy storage service $P_i$, define the user's *saving* of utilizing energy storage service by $\Delta_i \triangleq \mathtt{Cost}_i - P_i$. Another fair cost-sharing scheme is that each user should split $\mathtt{Cost_{ess}}$ in a way that attains the same saving for every user. Namely,

$$\begin{aligned} P_i^{\mathtt{ega}} &\triangleq \mathtt{Cost}_i - \frac{\sum_{i=1}^{N} \mathtt{Cost}_i - \mathtt{Cost_{ess}}}{N} \\ &= \sum_{t=1}^{T} x_i^-(t) \cdot p(t) - \frac{\sum_{t=1}^{T} x^-(t) \cdot p(t) - \sum_{t=1}^{T} \big( p(t) + \mathtt{p_s} \big) \cdot x^+(t)}{N} \end{aligned} \tag{16}$$

Thus, each user $i$ attains the same saving as: $\Delta_i^{\mathtt{ega}} \triangleq \frac{\sum_{i=1}^{N} \mathtt{Cost}_i - \mathtt{Cost_{ess}}}{N}$. It is easy to check that egalitarian cost-sharing is also budget-balanced (i.e., $\sum_{i=1}^{N} P_i^{\mathtt{ega}} = \mathtt{Cost_{ess}}$).

As a comparison, proportional cost-sharing guarantees the same percentage of savings (i.e., $\frac{\Delta_i}{\mathtt{Cost}_i}$) among users, whereas egalitarian cost-sharing guarantees the same savings (i.e., $\Delta_i$) among users.

**Theorem** 2. *If $\big(x^+(t), x^-(t)\big)_{t=1}^{T}$ is an optimal solution of* (P2) *and let $x_i^-(t) = \frac{a_i(t)}{a(t)} \cdot x^-(t)$ and $y_i(t) = \frac{a_i(t)}{a(t)} \cdot y(t)$, then proportional and egalitarian cost-sharing schemes are individually rational.*

*Let* $\hat{\mathsf{p}}(t) \triangleq \frac{x^-(t)\cdot p(t)}{a(t)}$ *and* $\mathtt{Cost}_{\mathsf{org}} \triangleq \sum_{t=1}^{T} x^-(t)\cdot p(t)$. *The proportional and egalitarian cost-sharing payments are given as follows:*

$$\begin{cases} P_i^{\mathsf{pp}} = \frac{\mathtt{Cost}_{\mathsf{ess}}}{\mathtt{Cost}_{\mathsf{org}}} \cdot \sum_{t=1}^{T} a_i(t)\cdot\hat{\mathsf{p}}(t), \\ P_i^{\mathsf{ega}} = \sum_{t=1}^{T} a_i(t)\cdot\hat{\mathsf{p}}(t) - \frac{\mathtt{Cost}_{\mathsf{org}}-\mathtt{Cost}_{\mathsf{ess}}}{N} \end{cases} \tag{17}$$

See Appendix. A for the proof.

**Remarks**: Egalitarian cost-sharing may have negative payments (i.e., $P_i^{\mathsf{ega}} < 0$), when $\mathtt{Cost}_i < \Delta_i^{\mathsf{ega}}$. Namely, a user may be paid by other users who have larger original costs, in order to maintain equal savings among all users. In this case, such a user is not benefited sufficiently from energy storage service because of the presence of other users and capacity constraint, and hence, will be compensated by other users in egalitarian cost-sharing.

One may argue whether proportional cost-sharing is better than egalitarian cost-sharing, because it rules out negative payments. Here, we provide a solution to support both cost-sharing schemes. We will leave the decision of adopting which scheme to the users.

## 3.4 Blockchain Model

We consider an account-based blockchain model like Ethereum (which is a general-purpose blockchain platform [32]), whereas Bitcoin operates with a different transaction-output-based model for cryptocurrency transactions only. Smart contracts are programming code on a blockchain that can provide customized computation tasks to each transaction (e.g., verification, data processing). Our payment system can be implemented as a smart contract.

The payment and auditing of energy storage service are carried out on a blockchain. Each user has an account on the blockchain. Users can top-up their accounts in advance. For cost-sharing, the users can initiate a joint payment transaction to the energy storage operator. The transaction records on the blockchain will also be used to verify VNM settlement by the grid operator.

Our blockchain model is based on a common model in the cryptography literature (e.g., Zether [6] that was built on Ethereum), which can be incorporated with privacy protection to conceal the transaction records. The blockchain consists of several components:

(1) **Ledger**: An append-only ledger on a blockchain holds the records of all accounts and transactions. Note that by default, there is no privacy protection to the ledger, such that the account details and transaction histories are visible to the public. On Ethereum, one can create tokens on the ledger to represent certain digital assets. Our payment system is implemented by tokens, which allows us to incorporate privacy protection. To pay for energy storage service, users are required to purchase tokens that will be subsequently transfered to the energy storage operator and redeemed.

(2) **Accounts**: A user account is identified by a public key $K^{\mathsf{p}}$ and an address $\mathtt{ad}$, which is the hash of the public key: $\mathtt{ad} = \mathcal{H}(K^{\mathsf{p}})$, where $\mathcal{H}(\cdot)$ is a cryptographic hash function. The user manages the account by the corresponding private key $K^{\mathsf{s}}$. Each account holds a balance of tokens, denoted by $\mathtt{Bal}(\mathtt{ad})$, which by default is a plaintext visible to the public. We will subsequently conceal the balances. Each user $i \in \{1,\ldots,N\}$ has an account associated with a tuple $(\mathtt{ad}_i, K_i^{\mathsf{p}}, K_i^{\mathsf{s}}, \mathtt{Bal}(\mathtt{ad}_i))$. We denote the energy storage operator's account address by $\mathtt{ad}_{\mathtt{ess}}$.

(3) **Transactions**: To initiate a transaction of tokens from $\mathtt{ad}_i$ to $\mathtt{ad}_{i'}$ with transaction value $\mathtt{val}$, the user submits a transaction request to the blockchain: $\mathtt{tx} = (\mathtt{ad}_i, \mathtt{ad}_{i'}, \mathtt{val})$, along with a signature $\mathtt{sign}_{K_i^{\mathsf{s}}}(\mathtt{tx})$ using the private key $K_i^{\mathsf{s}}$ associated with $\mathtt{ad}_i$. The transaction request will be executed[1] if $\mathtt{Bal}(\mathtt{ad}_i) \geq \mathtt{val}$. A multi-transaction can also be requested. Let $\mathtt{mtx} = (\mathtt{ad}_i, \mathtt{ad}_{i'}, \mathtt{val}_i)_{i=1}^N$. $\mathtt{mtx}$ will be executed, only if $\mathtt{Bal}(\mathtt{ad}_i) \geq \mathtt{val}_i$ for all $i$ and multi-signature $\mathtt{sign}_{(K_i^{\mathsf{s}})_{i=1}^N}(\mathtt{mtx})$ is present. Depending on the cost-sharing scheme, a user will pay either $P_i^{\mathsf{pp}}$ or $P_i^{\mathsf{ega}}$ to the energy storage operator. Each transaction request by default is a plaintext visible to the public. We will subsequently conceal the transaction records.

(4) **Receipts**: The recipient of a transaction can attach a receipt on the ledger, which may include additional information for further verification and auditing by a third-party. In VNM settlement, the grid operator will need to audit the amount of energy that a user can be offset from energy storage service, which can be verified from the receipts associated with transaction records.

Note that there may be a negative flow of payment in egalitarian cost-sharing, such that $\mathtt{val}_i < 0$. Hence, we need to ensure the corresponding transaction on a blockchain still functions correctly.

**Theorem** 3. *Consider a multi-transaction* $\mathtt{mtx} = (\mathtt{ad}_i, \mathtt{ad}_{\mathtt{ess}}, \mathtt{val}_i)_{i=1}^N$, *where* $\mathtt{val}_i$ *may be negative. Namely, every* $\mathtt{ad}_i$ *pays to the energy storage operator* $\mathtt{ad}_{\mathtt{ess}}$. *If* $\sum_{i=1}^N \mathtt{val}_i > 0$, *then* $\mathtt{mtx}$ *can be handled on a blockchain by the following transaction operations:*

$$\mathtt{Bal}(\mathtt{ad}_i) \leftarrow \mathtt{Bal}(\mathtt{ad}_i) - \mathtt{val}_i, \textit{for all } i \tag{18}$$

$$\mathtt{Bal}(\mathtt{ad}_{\mathtt{ess}}) \leftarrow \mathtt{Bal}(\mathtt{ad}_{\mathtt{ess}}) + \sum_{i=1,\ldots,N} \mathtt{val}_i \tag{19}$$

See Appendix. A for the proof.

## 3.5 Security & Threat Models

We next consider privacy protection in our problem. We assume synchronously authenticated communications among the parties, including users, blockchain, energy storage/grid operator, where the protocols proceed in rounds and the parties can authenticate each other properly so that there is no man-in-the-middle attack.

### 3.5.1 *Security Requirements.*

Our system aims to satisfy the following security requirements:

(S1) **Demand Concealment**: The individual user's demand $\left(a_i(t)\right)_{t=1}^T$ is private information, which should not be revealed to other users or energy storage operator in energy storage service scheduling, cost-sharing and payment. But the parameters, such as $\mathtt{p_s}, \mathtt{e_c}, \mathtt{e_d}, \mathtt{r_c}, \mathtt{r_d}, \left(p(t), \mathtt{B}(t)\right)_{t=1}^T$, are publicly known to all users. We need to ensure the operations of scheduling, cost-sharing and payment can be achieved correctly without leaking any information about $\left(a_i(t)\right)_{t=1}^T$ of a user to another. Specifically, given $\left(\left(a_i(t)\right)_{t=1}^T\right)_{i=1}^N$, we need a

[1] We skip some practical issues of a blockchain transaction, like nonce to prevent replay attack, account-locking against front-running attack, etc. But our model can easily incorporate the solutions from the security literature (e.g., [6]) to address these issues.

privacy-preserving summation function for the aggregate demand:

$$\mathtt{Sum}_{\mathtt{prv}}\left[\left(\left(a_i(t)\right)_{t=1}^{T}\right)_{i=1}^{N}\right] = \left(a(t)\right)_{t=1}^{T}.$$

No user should learn any information from $\mathtt{Sum}_{\mathtt{prv}}[\cdot]$ other than his own inputs and the final outputs.

(S2) **Zero-knowledge Cost-Sharing & Payment**: With $\left(a(t)\right)_{t=1}^{T}$, one can compute the energy storage service schedule $\left((x^+(t), x^-(t), y(t)\right)_{t=1}^{T}$ by Theorem 1. Then, each user $i$ can compute and make his payment $P_i (= P_i^{\mathsf{pp}}$ or $P_i^{\mathsf{ega}})$ by Theorem 2. Since $a_i(t)$ is only known to user $i$, we need verifiable "zero-knowledge" proofs in the payment transactions to show the following properties without revealing $a_i(t)$ or $P_i$:

(S2.1) *Non-negativity of user demands*: $a_i(t) \geq 0$ for all $t$. s

(S2.2) *Correctness of payment*: $P_i$ is computed correctly according to Theorem 2 for each user $i$.

(S2.3) *Sufficient balance of payment*: $\mathtt{Bal}(\mathtt{ad}_i) \geq P_i$, where $\mathtt{ad}_i$ is the account address of user $i$.

(S2.4) *Budget balance of energy storage service*: $\sum_{i=1}^{N} P_i = \mathtt{Cost}_{\mathtt{ess}}$. These zero-knowledge proofs will be crucial to safeguard against dishonest users in cost-sharing payments.

(S3) **Auditing for Virtual Net Metering**: The grid operator needs to verify the agreed energy flows from the energy storage operator to users, namely, $\left(\left(x_i^-(t)\right)_{t=1}^{T}\right)_{i=1}^{N}$ and $\left(x^-(t)\right)_{t=1}^{T}$. To enable auditing, the energy storage operator needs to provide a receipt for each user $i$ to certify his corresponding schedule $\left(x_i^-(t)\right)_{t=1}^{T}$, without the knowledge of $\left(x_i^-(t)\right)_{t=1}^{T}$.

#### 3.5.2 Threat Model.

The users may be dishonest, who may try to cheat by paying less to energy storage service or claim more in VNM than what they ought to. The dishonest users may collude to coordinate their actions. We aim to ensure the privacy of honest users and the correctness of scheduling, cost-sharing and payment in the presence of an adaptively malicious (active) adversary[2] corrupting a dishonest-majority of users up to $N-2$ [3]. Note that a malicious adversary is more intractable than a classical semi-honest user due to his ability of deliberately deviating from the protocols for prying into others' privacy or sabotage the protocols. In case of any dishonest actions being detected, our system will abort and notify all the users. Note that our system is not required to identify individual dishonest user and it is fundamentally impossible [3] to identify a dishonest user in multi-party computation with a majority of dishonest users[4].

[2] Adaptive adversary model provides a stronger security guarantee than static one, where the adversary will corrupt users at any time during the protocols rather than before the protocols.

[3] We achieve the security up to $N-2$ since we consider the fact that the total energy demands are known to all users.

[4] There are secure multi-party computation protocols [9] that can identify a dishonest user, but requiring a majority of honest users and considerable computational overhead. On the other hand, we can impose further measures to mitigate dishonesty. For example, requiring proper user authentication to prevent shilling. Or, we can require each user to pay a deposit in advance, which will be forfeited if any dishonesty is detected.

# 4 CRYPTOGRAPHIC COMPONENTS

Our privacy-preserving solution relies on several basic components from cryptography. We briefly explain them in this section. More details can be found in a standard cryptography textbook (e.g., [5]).

Denote by $\mathbb{Z}_p = \{0, ..., p-1\}$ the set of integers modulo $p$, for encrypting private data. For brevity, we simply write "$x + y$" and "$x \cdot y$" for modular arithmetic without explicitly mentioning"$\mathtt{mod}\ p$". We consider a usual finite group $\mathbb{G}$ of order $p$. We pick $g, h$ as two generators of $\mathbb{G}$, such that they can generate every element in $\mathbb{G}$ by taking proper powers, namely, for each $e \in \mathbb{G}$, there exist $x, y \in \mathbb{Z}_p$ such that $e = g^x = h^y$. The classical discrete logarithmic assumption states that given $g^x$, it is computationally hard to obtain $x$, which underlies the security of many cryptosystems.

## 4.1 Cryptographic Commitments

A cryptographic commitment allows a user to hide a secret (e.g., to hide the balances and transactions on a blockchain). We use Pedersen commitment, which is perfectly hiding (i.e., a computionally unbounded adversary cannot unlock the secret) and computationally binding (i.e., an adversary cannot associate with another secret in polynomial time). To commit secret value $x \in \mathbb{Z}_p$, a user first picks a random number $\mathtt{r} \in \mathbb{Z}_p$ to mask the commitment. Then, the user computes the commitment by:

$$\mathtt{Cm}(x, \mathtt{r}) = g^x \cdot h^{\mathtt{r}} \pmod{p} \tag{20}$$

where $g$ is a generator of a multiplicative group $\mathbb{Z}_p^*$, $h = g^k \pmod{p}$, $k$ is a secret value and $p$ is a large prime number.

Note that Pedersen commitment satisfies homomorphic property: $\mathtt{Cm}(x_1 + x_2, \mathtt{r}_1 + \mathtt{r}_2) = \mathtt{Cm}(x_1, \mathtt{r}_1) \cdot \mathtt{Cm}(x_2, \mathtt{r}_2)$. Sometimes, we simply write $\mathtt{Cm}(x)$ without specifying random $\mathtt{r}$. Next, we use $\Sigma$-protocol to construct zero-knowledge proofs for several useful properties of cryptographic commitments.

## 4.2 Zero-knowledge Proofs

In a zero-knowledge proof (of knowledge), a prover convinces a verifier of the knowledge of a secret without revealing the secret. For example, to show the knowledge of $(x, \mathtt{r})$ for $\mathtt{Cm}(x, \mathtt{r})$ without revealing $(x, \mathtt{r})$. A zero-knowledge proof of knowledge should satisfy completeness (i.e., the prover always can convince the verifier if knowing the secret), soundness (i.e., the prover cannot convince a verifier if not knowing the secret) and zero-knowledge (i.e., the verifier cannot learn the secret).

#### 4.2.1 $\Sigma$-Protocol.

$\Sigma$-Protocol is a general approach to construct zero-knowledge proofs. This can prove the knowledge of $x$ such that $f(x) = y$ with concealed $x$, where $y$ and $f(\cdot)$ are revealed and $f(\cdot)$ is not computationally invertible. Suppose $f(\cdot)$ satisfies homomorphic property: $f(a + b) = f(a) + f(b)$. The $\Sigma$-protocol proceeds as follows:

(1) First, the prover sends a commitment $y' = f(x')$, for a random $x'$, to the verifier.
(2) Next, the verifier generates a random challenge $\beta$ and sends it to the prover.
(3) The prover replies with $z = x' + \beta \cdot x$ (which does not reveal $x$).
(4) Finally, the verifier checks whether $f(z) \stackrel{?}{=} y' + \beta \cdot y$.

*4.2.2 Σ-Protocol Based Zero-knowledge Proofs.*

Next, we present four crucial instances of zero-knowledge proofs based on Σ-protocol. The details of these zero-knowledge proofs as well as the security proofs are in Appendix. B.

- **Zero-knowledge Proof of Commitment (**$\mathtt{zkpCm}$**)**: Given $\mathtt{Cm}(x, r)$, a prover wants to convince a verifier of the knowledge $x$ without revealing $(x, r)$. Denote a zero-knowledge proof of commitment for $\mathtt{Cm}(x, r)$ by $\mathtt{zkpCm}[x]$.
- **Zero-knowledge Proof of Summation (**$\mathtt{zkpSum}$**)**: Given commitments $\left(\mathtt{Cm}(x_1, \mathtt{r}_1), ..., \mathtt{Cm}(x_n, \mathtt{r}_n)\right)$ and $y$, a prover wants to convince a verifier of the knowledge of $y = \sum_{i=1}^{n} x_i$ without revealing $(x_1, ..., x_n)$. Denote a zero-knowledge proof of summation for $\left(\mathtt{Cm}(x_i, \mathtt{r}_i)\right)_{i=1}^{n}$ by $\mathtt{zkpSum}[y, (x_i)_{i=1}^{n}]$.
- **Zero-knowledge Proof of Membership (**$\mathtt{zkpMbs}$**)**: Given a set $\mathcal{X} = \{x_1, ..., x_n\}$ and $\mathtt{Cm}(x, \mathtt{r})$, a prover wants to convince a verifier of the knowledge of $x \in \mathcal{X}$ without revealing $x$. Denote a zero-knowledge proof of membership for $x \in \mathcal{X}$ by $\mathtt{zkpMbs}[x, \mathcal{X}]$.
- **Zero-knowledge Proof of Non-Negativity (**$\mathtt{zkpNN}$**)**: Given $\mathtt{Cm}(x, \mathtt{r})$, a prover wants to convince a verifier of the knowledge of $x \geq 0$ without revealing $x$. Denote a zero-knowledge proof of the non-negativity of $x$ by $\mathtt{zkpNN}[x]$.

## 4.3 Non-interactive Zero-knowledge Proofs

An interactive zero-knowledge proof of knowledge can be converted to a non-interactive one by Fiat-Shamir heuristic to remove the verifier-provided challenge. Let $\mathcal{H}(\cdot) \mapsto \mathbb{Z}_p$ be a cryptographic hash function. Given a list of commitments $(\mathtt{Cm}_1, ..., \mathtt{Cm}_r)$, one can map to a single hash value by $\mathcal{H}(\mathtt{Cm}_1|...|\mathtt{Cm}_r)$, where the input is the concatenated string of $(\mathtt{Cm}_1, ..., \mathtt{Cm}_r)$. In a Σ-protocol, one can set the challenge $\beta = \mathcal{H}(\mathtt{Cm}_1|...|\mathtt{Cm}_r)$, where $(\mathtt{Cm}_1, ..., \mathtt{Cm}_r)$ are all the commitments generated by the prover prior to the step of verifier-provided challenge (Step 2 of Σ-protocol). Hence, the prover does not wait for the verifier-provided random challenge, and instead generates the random challenge himself. The verifier will generate the same challenge following the same procedure for verification. Denote the non-interactive versions of the previous zero-knowledge proofs by $\mathtt{nzkpCm}$, $\mathtt{nzkpSum}$, $\mathtt{nzkpMbs}$, $\mathtt{nzkpNN}$, respectively.

## 4.4 Public-Private Key Signatures

Cryptographic signatures can verify the authenticity of some given data. Suppose a signer has a pair of public and private keys $(K^{\mathsf{p}}, K^{\mathsf{s}})$ for an asymmetric key cryptosystem (e.g., RSA). To sign a message $m$, the signer first maps $m$ by a cryptographic hash function $\mathcal{H}(m)$ (e.g., SHA-3). Then the signature of $m$ is the encryption $\mathtt{sign}_{K^{\mathsf{s}}}[m] = \mathtt{Enc}_{K^{\mathsf{s}}}[\mathcal{H}(m)]$. Given $(m, K^{\mathsf{p}})$, anyone can verify the signature $\mathtt{sign}_{K^{\mathsf{s}}}[m]$ by checking whether the decryption $\mathtt{Dec}_{K^{\mathsf{p}}}[\mathtt{sign}_{K^{\mathsf{s}}}[m]] \stackrel{?}{=} \mathcal{H}(m)$.

# 5 MULTI-PARTY COMPUTATION PROTOCOL

This section presents a simplified multi-party computation protocol called SPDZ [10, 11], which allows multiple parties to jointly compute a certain function while concealing the private inputs. SPDZ can safeguard against a dishonest majority (i.e., all but one party can be dishonest), and does not require a trusted dealer for setup.

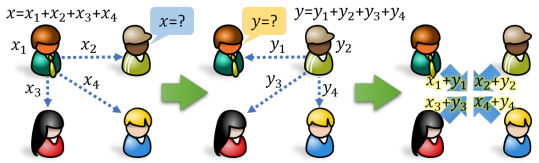



**Figure 2: An example of information-theoretical secret-sharing. Let us compute $x+y$, without revealing $x$ or $y$. Party 1 splits $x$ into $(x_i)_{i=1}^{4}$ and secretly shares each with one party. So is party 2 for $y$. Next, each party $i$ computes $z_i = x_i + y_i$ locally. Then, $x + y = \sum_{i=1}^{4} z_i$, if $(z_i)_{i=1}^{4}$ are revealed.**

## 5.1 Information-theoretical Secret Sharing

SPDZ relies on information-theoretical secret-sharing, whereby private data will be distributed to multiple parties, such that each party only knows a share of the data, without complete knowledge of other shares. Hence, computation of individual shares of data will not reveal the original data, unless all shares are revealed for output or verification. Several distributed computation operations can be performed locally, while preserving the secret sharing property.

We consider the computation of a function of an arithmetic circuit consisting of only additions and multiplications. Suppose a private number $x$ is distributed to $n$ parties, such that each party $i$ knows a share $x_i$ only, where $x = \sum_{i=1}^{n} x_i$, but not knowing other shares $x_j$, where $j \neq i$. Note that a party is unable to construct $x$, without knowing all the shares. In the following, we write $\langle x \rangle$ as a *secretly shared* number, meaning that there is a vector $(x_1, ..., x_n)$, such that each party $i$ knows only $x_i$. Given secretly shared $\langle x \rangle$ and $\langle y \rangle$, and a public known constant $c$, the following operations can be attained by local computation at each party, and then the outcome can be assembled from the individual shares:

A1) $\langle x \rangle + \langle y \rangle$ can be computed by $(x_1 + y_1, ..., x_n + y_n)$.
A2) $c \cdot \langle x \rangle$ can be computed by $(c \cdot x_1, ..., c \cdot x_n)$.
A3) $c + \langle x \rangle$ can be computed by $(c + x_1, x_2, ..., x_n)$.

To reveal $\langle x \rangle$, each party $i$ broadcasts $x_i$ to other parties. Then each party can reconstruct $x = \sum_{i=1}^{n} x_i$. See an example in Figure 2.

Multiplications can also be computed by SPDZ, and the detailed description can be found in Appendix. D. With additions and multiplications, one can construct a large class of computation functions (including comparison and branching conditions).

However, some parties may be dishonest, who may not perform the correct local computation. To safeguard against dishonest parties, an information-theoretical message authentication code (MAC) can be used for verification. Every secretly shared number is encoded by a MAC as $\gamma(x)$, which is also secretly shared as $\langle \gamma(x) \rangle$. The basic idea is that if a dishonest party wants to modify his share $x_i$, then he also needs to modify $\gamma(x)_i$ consistently. This allows dishonesty to be detectable by checking the corresponding MAC in the final output. The detailed description of MAC can be found in Appendix. D. In the following, we write $\langle\langle x \rangle\rangle$ meaning that both $\langle x \rangle$ and the respective MAC $\langle \gamma(x) \rangle$ are secretly shared among users.

## 5.2 Overview of SPDZ Protocol

The SPDZ consists of three phases, as outlined as follows:

(1) *Pre-processing Phase*: In this phase, a collection of shared random numbers will be constructed to mask the private input numbers. For each private input number of party $i$, there needs a shared random number $\langle\!\langle r^i \rangle\!\rangle$, where $r^i$ is revealed to party $i$ only, but not to other parties.
(2) *Online Phase*: To secretly shares a private input number $x^i$ using $\langle\!\langle r^i \rangle\!\rangle$, without revealing $x^i$, it proceeds as follows:
   1) Party $i$ computes and reveals $z^i = x^i - r^i$ to all parties.
   2) Every party sets $\langle\!\langle x^i \rangle\!\rangle \leftarrow z^i + \langle\!\langle r^i \rangle\!\rangle$ (see A3).

   Any computation function in terms of additions or multiplications can be computed by proper local computations (e.g., A1-A3). The MACs are updated accordingly to preserve the consistency. See Appendix. C for details.
(3) *Output and Validation Phase*: All MACs will be revealed for validation. If there is any inconsistency in MACs, then abort.

The details of SPDZ protocol can be found in Appendix. C.

## 6 PRIVACY-PRESERVING ENERGY STORAGE SHARING

This section presents an integrated solution for privacy-preserving energy sharing, with blockchain, zero-knowledge proofs and SPDZ.

### 6.1 Privacy-Preserving Ledger

First, we incorporate privacy protection to hide transaction records on the ledger, while still allowing proper verifications for cost-sharing and VNM. As in other privacy-preserving blockchain (e.g., Zether [6]), we conceal the balances and transaction values in the ledger by the respective cryptographic commitments instead of plaintext values. The accounts in the ledger will become as follows:

| | | |
|---|---|---|
| $\mathsf{ad}_i$ | $K_i^{\mathsf{pb}}$ | $\mathtt{Cm}(\mathtt{Bal}(\mathsf{ad}_i))$ |
| $\mathsf{ad}_j$ | $K_j^{\mathsf{pb}}$ | $\mathtt{Cm}(\mathtt{Bal}(\mathsf{ad}_j))$ |
| ... | ... | ... |

A multi-transaction will be concealed as $\mathtt{mtx} = (\mathsf{ad}_i, \mathsf{ad}_{i'}, \mathtt{Cm}(\mathtt{val}_i))_{i=1}^N$. Because of concealed balance and transaction value, each user must also provide $\mathtt{nzkpNN}[\mathtt{Bal}(\mathsf{ad}_i) - \mathtt{val}_i]$ associated with each transaction to prove the non-negativity of the resultant balance.

### 6.2 Privacy-Preserving Protocol $\Pi_{\mathsf{pess}}$

We design protocol $\Pi_{\mathsf{pess}}$ for privacy-preserving energy storage service scheduling, cost-sharing, payment and VNM. Before presenting the detailed protocol, we first outline some high-level ideas:

(1) First, the users need to secretly share private individual demands $\langle\!\langle a_i(t) \rangle\!\rangle$. Then, they can compute aggregate demand $\langle\!\langle a(t) \rangle\!\rangle$ via SPDZ in a privacy-preserving manner.
(2) To enable later verification of payment transactions and VNM, each user also needs to announce commitment $\mathtt{Cm}(a_i(t))$ to each other. However, dishonest users may use inconsistent $\mathtt{Cm}(a_i(t))$ with respect to secretly shared $\langle\!\langle a_i(t) \rangle\!\rangle$. To show the equivalence, all users need to create zero-knowledge proof of commitment $\mathtt{zkpCm}[a_i(t)]$ via SPDZ using secretly shared $\langle\!\langle a_i(t) \rangle\!\rangle$. One can also show the non-negativity of $\langle\!\langle a_i(t) \rangle\!\rangle$ by zero-knowledge proof of non-negativity $\mathtt{nzkpNN}[a_i(t)]$.
(3) After verifying $\mathtt{zkpCm}[a_i(t)]$ and $\mathtt{nzkpNN}[a_i(t)]$, the users reveal $\langle\!\langle a(t) \rangle\!\rangle$ and verify the corresponding MAC to ensure the integrity of $a(t)$. Then, the users compute the energy storage service schedule with the knowledge of $\left(a(t)\right)_{t=1}^T$.
(4) Next, each user $i$ can compute and make his payment $P_i$ (= $P_i^{\mathsf{pp}}$ or $P_i^{\mathsf{ega}}$) by Theorem 2. The users jointly compute the total payments $\sum_i^N P_i$ via SPDZ in a privacy-preserving manner to ensure that the difference between $\mathtt{Cost}_{\mathtt{ess}}$ and $\sum_i^N P_i$ is within a negligible rounding error $\varepsilon$, such that $|\mathtt{Cost}_{\mathtt{ess}} - \sum_i^N P_i| < \varepsilon$. Then the users agrees and sets $\mathtt{Cost}_{\mathtt{ess}} = \sum_i^N P_i$.
(5) To make cost-sharing payments for energy storage service on the ledger, the users need to create a zero-knowledge proof that $\sum_{i=1}^N P_i = \mathtt{Cost}_{\mathtt{ess}}$ via SPDZ. Each user also creates $\mathtt{nzkpNN}[\mathtt{Bal}(\mathsf{ad}_i) - P_i]$ locally.
(6) After the completion of multi-transaction of payments, the energy storage service schedule is executed. Afterwards, the energy storage operator signs $\mathtt{Cm}(x_i^-(t))$ as a receipt on the ledger for each user. Note that $\mathtt{Cm}(x_i^-(t))$ can be generated based on $\mathtt{Cm}(a_i(t))$ and the energy storage service schedule.
(7) Finally, the users requests VNM settlement with the grid operator. The grid operator will verify $x_i^-(t)$ from the signed $\mathtt{Cm}(x_i^-(t))$ on the ledger.

The details of the privacy-preserving protocol $\Pi_{\mathsf{pess}}$, consisting of three stages ($\Pi_{\mathsf{pess}}^{(1)}$-$\Pi_{\mathsf{pess}}^{(3)}$), are presented as follows:

**Stage 0: Initialization.** In this stage, system parameters are chosen and the pre-processing phase of SPDZ is executed among the users. See Appendix. C for detailed SPDZ pre-processing phase.

*Initialization*:
(1) Choose and announce a multiplicative group $\mathbb{Z}_p^*$, two generators $g, h \in \mathbb{Z}_p^*$ and hash function $\mathcal{H}(\cdot) \mapsto \mathbb{Z}_p$ as public information to all users. Note that $g$ and $h$ can be obtained via a coin-tossing protocol [33] among the users such that $\log_g h$ is unknown due to the hardness of the discrete logarithm.
(2) The energy storage operator announces $\mathtt{p_s}, \mathtt{e_c}, \mathtt{e_d}, \mathtt{r_c}, \mathtt{r_d}$, $\left(p(t), \mathtt{B}(t)\right)_{t=1}^T$ as public information to all users.
(3) Initialize SPDZ pre-processing phase among all users.

**Stage 1: Pre-operation Scheduling.** In this stage, the users will compute their aggregate day-ahead demands via SPDZ. The users also need to make commitments of their individual demands $\left(a_i(t)\right)_{t=1}^T$, which will be used for auditing in VNM. We ensure that the individual demands shared via SPDZ match the ones being committed. This can be accomplished by computing zero-knowledge proof of commitment $\mathtt{zkpCm}[a_i(t)]$ using the secretly shared $\langle\!\langle a_i(t) \rangle\!\rangle$. Next, the users will compute the optimal energy storage service schedule in (P2) based on aggregate demands $\left(a(t)\right)_{t=1}^T$.

*Protocol* $\Pi_{\mathsf{pess}}^{(1)}$:
(1) User $i$ commits $C_i(t) = \mathtt{Cm}(a_i(t), \mathtt{r}_i(t))$ for all $t$ and announces $\left(C_i(t)\right)_{t=1}^T$ to all users with $\left(\mathtt{nzkpNN}[a_i(t)]\right)_{t=1}^T$, where $\mathtt{r}_i(t)$ is a random masking number. All users verify $\mathtt{nzkpNN}[a_i(t)]$. If verification of $\mathtt{nzkpNN}[a_i(t)]$ fails, announce Abort.

(2) User $i$ secretly shares $\langle\langle a_i(t)\rangle\rangle$ via SPDZ for all $t$.
(3) To show the equality of $a_i(t)$ in $\langle\langle a_i(t)\rangle\rangle$ and $C_i(t)$, user $i$ constructs an $\mathtt{zkpCm}[a_i(t)]$ distributedly via SPDZ:
  (a) User $i$ randomly generates $\left(a_i'(t), \mathtt{r}_i'(t)\right) \in \mathbb{Z}_p$ and secretly shares as $\langle\langle a_i'(t)\rangle\rangle$ and $\langle\langle \mathtt{r}_i'(t)\rangle\rangle$. User $i$ announces $C_i'(t) = \mathtt{Cm}(a_i'(t), \mathtt{r}_i'(t))$ to all users. Note that not until all the users have completed this step do they proceed to the next step to produce a common random challenge $\beta(t)$.
  (b) All the users conduct a coin-tossing protocol to obtain a random challenge $\beta(t)$. Firstly, each user announces a commitment $C_i''(t)$ of a randomly generated number $\mathtt{r}_i''(t) \in \mathbb{Z}_p$. Then all the users reveal $\mathtt{r}_i''(t)$ and compute a random challenge $\beta(t) = \sum_{i=1}^N \mathtt{r}_i''(t)$.
  (c) All users compute and reveal $\langle\langle z_{a_i(t)}\rangle\rangle = \langle\langle a_i'(t)\rangle\rangle + \beta(t) \cdot \langle\langle a_i(t)\rangle\rangle$ and $\langle\langle z_{\mathtt{r}_i(t)}\rangle\rangle = \langle\langle \mathtt{r}_i'(t)\rangle\rangle + \beta(t) \cdot \langle\langle \mathtt{r}_i(t)\rangle\rangle$ for user $i$.
  (d) This creates $\mathtt{zkpCm}[a_i(t)] = \{C_i'(t), z_{a_i(t)}, z_{\mathtt{r}_i(t)}\}$. All users verify $\mathtt{zkpCm}$ by checking
$$g^{z_{a_i(t)}} \cdot h^{z_{\mathtt{r}_i(t)}} \stackrel{?}{=} C_i'(t) \cdot \mathtt{Cm}\Big(a_i(t), \mathtt{r}_i(t)\Big)^{\beta(t)}$$
  (e) If the verification of $\mathtt{zkpCm}$ fails, announce $\mathtt{Abort}$.
(4) The users compute $\langle\langle a(t)\rangle\rangle \leftarrow \sum_{i=1}^N \langle\langle a_i(t)\rangle\rangle$ for all $t$ via SPDZ. Reveal $\langle\langle a(t)\rangle\rangle$. Check MAC of $\langle\langle a(t)\rangle\rangle$. If the MAC check fails, announce $\mathtt{Abort}$.
(5) The users solve (P2) using $\left(a(t)\right)_{t=1}^T$ for $\left(x^+(t), x^-(t), y(t)\right)_{t=1}^T$
(6) All users compute $\mathtt{Cost}_{\mathtt{ess}} \triangleq \sum_{t=1}^T \left(p(t) + \mathtt{p_s}\right) \cdot x^+(t)$ and $\mathtt{Cost}_{\mathtt{org}} \triangleq \sum_{t=1}^T p(t) \cdot x^-(t)$, $\hat{\mathtt{p}}(t) \triangleq \frac{x^-(t) \cdot p(t)}{a(t)}$.

**Stage 2: Cost-sharing Payment & Operation.** In this stage, the users split the cost of energy storage service based on proportional or egalitarian cost-sharing scheme via SPDZ. The users compute the payment commitments and verify the validity of $\mathtt{Cost}_{\mathtt{ess}}$ by comparing with $\sum_{i=1}^N P_i$. Before issuing the multi-transaction, the users compute $\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N]$ to satisfy Theorem 3. The users then make energy storage service payments via privacy-preserving blockchain. After receiving the payments, the energy storage operator will issue verifiable receipts on the ledger.

*Protocol* $\Pi_{\mathtt{pess}}^{(2)}$:

(1) Each user $i$ computes $P_i = P_i^{\mathtt{pp}}$ (or $P_i^{\mathtt{ega}}$) by Theorem 2, and announces commitment $\mathtt{Cm}(P_i, \mathtt{r}_i)$ to all users, and secretly shares $\langle\langle \mathtt{r}_i\rangle\rangle$ via SPDZ.
(2) The users also compute the total payments via SPDZ by
$$\sum_{i=1}^N P_i = \begin{cases} \sum_{i=1}^N P_i^{\mathtt{pp}} = \sum_{t=1}^T \left(\frac{\mathtt{Cost}_{\mathtt{ess}} \cdot \hat{\mathtt{p}}(t)}{\mathtt{Cost}_{\mathtt{org}}} \cdot \sum_{i=1}^N \langle\langle a_i(t)\rangle\rangle\right), \\ \sum_{i=1}^N P_i^{\mathtt{ega}} = \sum_{t=1}^T \left(\sum_{i=1}^N \left(\langle\langle a_i(t)\rangle\rangle \cdot \hat{\mathtt{p}}(t) - \frac{\mathtt{Cost}_{\mathtt{org}} - \mathtt{Cost}_{\mathtt{ess}}}{N}\right)\right) \end{cases} \tag{21}$$
and check $|\mathtt{Cost}_{\mathtt{ess}} - \sum_{i=1}^N P_i| < \varepsilon$, where $\varepsilon$ is a small fault-tolerant factor, which confines the division rounding error, arising from computing $\frac{\mathtt{Cost}_{\mathtt{ess}}}{\mathtt{Cost}_{\mathtt{org}}}$ or $\hat{\mathtt{p}}(t)$. If so, the users take $\mathtt{Cost}_{\mathtt{ess}} = \sum_{i=1}^N P_i$. Otherwise, announce $\mathtt{Abort}$.
(3) The users then compute $\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N]$ via SPDZ distributely as follows:
  (a) User $i$ randomly generates and secretly shares $\langle\langle \mathtt{r}_i'\rangle\rangle \in \mathbb{Z}_p$ before announcing $\mathtt{Cm}(0, \mathtt{r}_i')$.
  (b) All users conduct a coin-tossing protocol to obtain a random challenge $\beta$ as in $\Pi_{\mathtt{pess}}^{(1)}$. Then all users compute $\langle\langle z_{\mathtt{r}}\rangle\rangle = \sum_{i=1}^n \langle\langle \mathtt{r}_i'\rangle\rangle + \beta \cdot \sum_{i=1}^n \langle\langle \mathtt{r}_i\rangle\rangle$ and $C' = \prod_{i=1}^N \mathtt{Cm}(0, \mathtt{r}_i')$.
  (c) This creates $\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N] = \{C', z_{\mathtt{r}}\}$. All users verify $\mathtt{zkpSum}$ by checking:
$$g^{\beta \cdot \mathtt{Cost}_{\mathtt{ess}}} \cdot h^{z_{\mathtt{r}}} \stackrel{?}{=} C' \cdot \prod_{i=1}^n \mathtt{Cm}(P_i, \mathtt{r}_i)^{\beta}$$
  If the verification of $\mathtt{zkpSum}$ fails, announce $\mathtt{Abort}$.
(4) Each user computes $\mathtt{nzkpNN}[\mathtt{Bal}(\mathtt{ad}_i) - P_i]$ based on $\mathtt{Cm}(P_i)$.
(5) Let the account address of energy storage operator be $\mathtt{ad}_{\mathtt{ess}}$. The users submit a multi-transaction request
$$\mathtt{mtx} = \left(\mathtt{ad}_i, \mathtt{ad}_{\mathtt{ess}}, \mathtt{Cm}(P_i)\right)_{i=1}^N$$
to the ledger, along with
$$\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N] \text{ and } \mathtt{nzkpNN}[\mathtt{Bal}(\mathtt{ad}_i) - P_i]_{i=1}^N$$
(6) The ledger verifies $\mathtt{zkpSum}$ and $\mathtt{nzkpNN}$ before proceeding the transaction. If the verification fails, announce $\mathtt{Abort}$.
(7) The users provide the schedule $\left(a(t), x^+(t), x^-(t), y(t)\right)_{t=1}^T$ to the energy storage operator. After transaction completes, the energy storage operator will execute the schedule.

**Stage 3: Post-operation VNM Settlement.** In this stage, the energy storage operator signs the receipts of individual energy storage service $\left(x_i^-(t)\right)_{t=1}^T$. The receipts will be stored on the ledger. The users will request VNM settlement with the grid operator. The grid operator will verify their claims with the receipts on the ledger.

*Protocol* $\Pi_{\mathtt{pess}}^{(3)}$:

(1) The users upload $\left(\mathtt{Cm}(a_i(t))_{t=1}^T\right)_{i=1}^N$ to the ledger.
(2) The energy storage operator computes commitment $\mathtt{Cm}(x_i^-(t)) = \mathtt{Cm}(a_i(t))^{\frac{x^-(t)}{a(t)}}$ (according to Theorem 2) for all $t$ and $i$.
(3) Let the public-private keys of the energy storage operator be $(K_{\mathtt{ess}}^{\mathtt{p}}, K_{\mathtt{ess}}^{\mathtt{s}})$. The energy storage operator signs $\mathtt{sign}_{K_{\mathtt{ess}}^{\mathtt{s}}}[\mathtt{Cm}(x_i^-(t))]$ along with $\mathtt{Cm}(x_i^-(t))$ to be stored on the ledger.
(4) The energy storage operator prepares VNM and provides energy export profile $\left(x^-(t)\right)_{t=1}^T$ to the grid operator.
(5) Each user $i$ submits a claim for reimbursement by referring to receipt $\mathtt{Cm}(x_i^-(t))$ and $\mathtt{sign}_{K_{\mathtt{ess}}^{\mathtt{s}}}[\mathtt{Cm}(x_i^-(t))]$ on the ledger. Also, reveal $x_i^-(t)$ to the grid operator to prove the validity.
(6) The grid operator verifies $\mathtt{sign}_{K_{\mathtt{ess}}^{\mathtt{s}}}[\mathtt{Cm}(x_i^-(t))]$ by public key $K_{\mathtt{ess}}^{\mathtt{p}}$, and compares the energy demand profile $\left(a_i(t)\right)_{t=1}^T$ with $\left(x_i^-(t)\right)_{t=1}^T$. If the verification is consistent, the grid operator will deduct $\sum_{t=1}^T x_i^-(t) \cdot p(t)$ from user $i$'s total payment to the grid.

See Appendix. D for the security analysis of $\Pi_{\mathtt{pess}}$ for satisfying security requirements S1-S3.

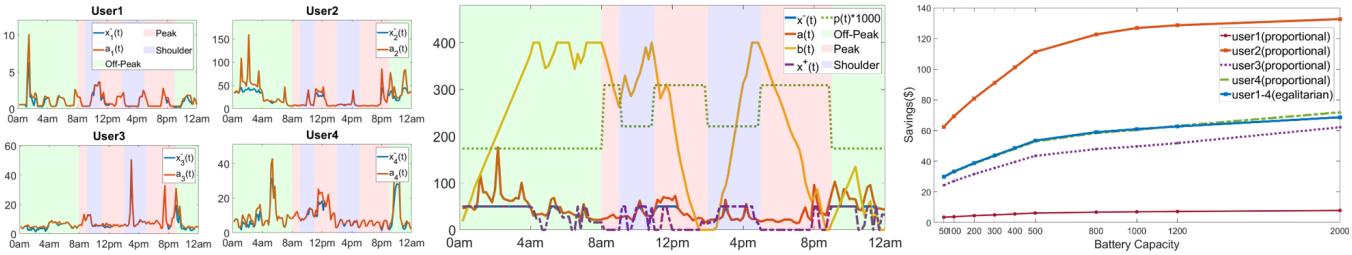
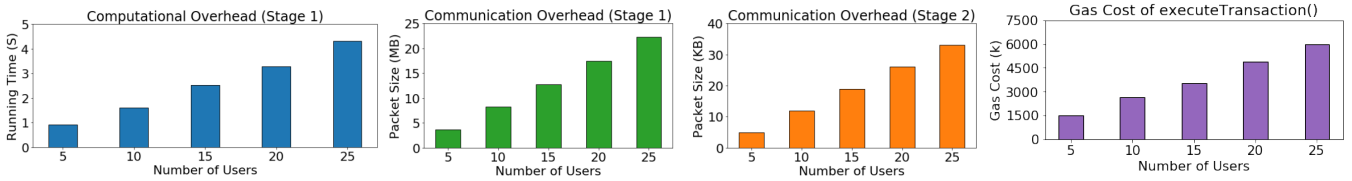



**Figure 3: Data trace of energy storage service schedule.**




**(a) Computational overhead** **(b) Communication overhead** **(c) Gas cost of `executeTransaction()`**

**Figure 4: System performance and smart contract gas costs.**

# 7 EVALUATION

## 7.1 Energy Storage Service Scheduling

We first evaluate the effectiveness of energy storage sharing. We selected 120 users from the Smart* microgrid dataset [2]. We consider a single 24-hour period, from midnight to next midnight.

In particular, we present the temporal data trace of scheduled energy storage services for 4 users. We observe that each of the user can utilize energy storage discharging during peak-hour. Most users acquire energy from the grid during off-peak-hour, and partially during shoulder-hour. Next, we study the saving ($\Delta_i$) of each user. In Figure 3, we study different cost-sharing schemes of energy storage service for 4 users. We observe that proportional cost-sharing gives each user at the same percentage of saving of 28.78% when energy storage $B$ is 400 kWh, whereas different users have different percentages of saving with egalitarian cost-sharing, but with the same amount of saving of $37.90.

## 7.2 SPDZ Performance

Next, we evaluate the performance of SPDZ in stages 1-2 of $\Pi_{\text{pess}}$. We skip stage 3 due to its negligible performance compared with stages 1-2. We consider 144 time slots in a single 24-hour period. All the results were averaged over 20 instances.

*7.2.1 Computational Overhead.* We scaled the number of users from 5 to 25. Figure 4a displays the average running time incurred at each user in the stage 1. The running time shows a linearly growing trend with increasing number of users. The running time starts from about 0.91 seconds with 5 users to around 4.32 seconds with 25 users. We skip the displaying of the stage 2 due to the negligible computational overhead of only several milliseconds.

*7.2.2 Communication Overhead.* Figure 4b shows the average total volume of the transmission data in stages 1 and 2. It is evident that the total transmission amount scale linearly with the growing number of users in both stages. The data volume increases from 3.62 MB with 5 users to about 26.89 MB with 25 users in stage 1. In contrast, the data volume in stage 2 starts from a merely 5 KB with 5 users to about 40 KB with 25 users. Thus, the total data volume in stage 1 dominates the entire protocol.

## 7.3 Ethereum Smart Contract Gas Costs

We implemented the payment systems as a smart contract[5] on real-world Ethereum blockchain platform. The smart contract is specified by Solidity programming language [24]. We outline some implementation components as follows:

(1) *Pedersen.* This component aims to realize the underlying Pedersen commitment scheme.
(2) *ESToken.* We create a Ethereum-based cryptocurrency *ESToken* for our energy sharing scenario, whereby users are able to pay the energy cost without revealing the true payments.
(3) *MultiSignature.* This component allows the users to submit a multi-transaction request, where the transaction will proceed unless all the involved users validate the transaction. There are three main methods: `submitTransaction()`, `executeTransaction()`, `confirmTransaction()` .

See sample code of Solidity-based smart contracts in Appendix. E.

Distributed miners will execute the compiled bytecode of the smart contracts in *Ethereum Virtual Machine*. Miners will charge additional Ether/ETH (Ethereum native crytocurrency) called gas costs, because the extra computational tasks incurred by smart contracts will be broadcast throughout the blockchain. Gas costs are used to measure the amount of computational resources to execute the operations required by a transaction. We measured the incurred gas costs by our smart contracts and used a 255-bit prime number $q$ for Pedersen commitment since Solidity supports at most 256-bit numbers. We employed Truffle Suite [37] as the Ethereum development framework to test and measure the average gas costs[6] of Multi-Signature methods, shown in Table 2, where $N$ is the number of users and $N_b$ indicates the number of bits to represent the plaintext payment in the `nzkpMbs`. A transaction initiator must pay

[5]The source code will be released publicly, when this paper is published.
[6]The actual gas costs may vary based on the random generated parameters of the zero-knowledge proofs. Here, we show the average gas costs.

sufficient amount of gas costs to the miner, who creates transaction blocks on the network. The *gasPrice* in a transaction allows the transaction initiator to set the gas price that he is willing to pay. The higher the gas prices, the higher probability the transaction will be chosen by the miner in a block. We use the standard gas price 54 Gwei to calculate the transaction cost in Ether (see [36]).

| | Gas Cost | Ether | \|Input\| (bytes) |
|---|---|---|---|
| `submitTransaction()` | 106k | 0.0057 | $64 + 64 \times N$ |
| `confirmTransaction()` | 3600k | 0.1944 | $128 + 180 \times N_b$ |

**Table 2: Gas costs for Multi-Signature methods**

Figure 4c presents the gas cost of `executeTransaction()` function. We observe that the gas cost is linearly proportional to the number of involved users, starting from 1437k (0.0776 ether) with 5 users to 5986k (0.3232 ether) with 25 users, since the verification of `zkpSum` depends on the number of users. Overall, we observe only moderate incurred costs by our smart contract.

## 8 CONCLUSION

In this paper, we provide a novel approach to support third-party energy storage sharing without compromising the privacy of individual users. In our privacy-preserving solution, an energy storage operator is only revealed the minimal information to schedule energy storage operations, without knowing users' private demands. At the same time, the users can divide the cost of energy storage service fairly among themselves without knowing each other's demands. Our solution can effectively safeguard against a dishonest majority of users, without requiring trusted third-parties. We implemented our solution as a smart contract on Ethereum blockchain platform, which incurs low overhead and gas costs in practice.

## REFERENCES


[1] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David P. Jenkins, Peter McCallum, and Andrew Peacock. 2019. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews* 100 (2019), 143–174.

[2] Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, Prashant Shenoy, and Jeannie Albrecht. 2012. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. In *SustKDD*.

[3] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Annual ACM Symposium on Theory of Computing (STOC)*.

[4] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, and Eran Tromerand Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE Symposium on Security and Privacy*.

[5] William J. Buchanan. 2017. *Cryptography*. River Publishers.

[6] Benedikt Bunz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In *Financial Cryptography and Data Security (FC)*.

[7] Sid Chi-Kin Chau, Jiajia Xu, Wilson Bow, and Khaled Elbassioni. 2019. Peer-to-Peer Energy Sharing: Effective Cost-Sharing Mechanisms and Social Efficiency. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[8] Valerie Chen, Valerio Pastro, and Mariana Raykova. 2018. Secure Computation for Machine Learning With SPDZ. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[9] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. 2001. Multiparty computation from threshold homomorphic encryption. In *Intl. conference on the theory and applications of cryptographic techniques*.

[10] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. 2015. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press. Cambridge Books Online.

[11] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority - or: Breaking the SPDZ Limits. In *European Symposium on Research in Computer Security (ESORICS)*.

[12] Wenliang Du and Mikhail J Atallah. 2001. Secure multi-party computation problems and their applications: a review and open problems. In *the Workshop on New Security Paradigms*.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer.

[14] Neda Edalat, Mehul Motani, Jean Walrand, and Longbo Huang. 2014. Control of systems that store renewable energy. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[15] Pacific Gas and Electric Company. 2018. Understanding the Virtual Net Energy Metering Program A guide for statements and bills. (2018).

[16] J. Heeter R. Gelman and L. Bird. 2014. *Status of Net Metering: Assessing the Potential to Reach Program Caps*. Technical Report.

[17] Oded Goldreich. 1998. Secure multi-party computation. *Manuscript. Preliminary version* 78 (1998).

[18] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. 2019. Using a Blockchain to Mitigate Trust in Electric Vehicle Charging. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[19] Mohammad H. Hajiesmaili, Minghua Chen, Enrique Mallada, and Chi-Kin Chau. 2017. Crowd-Sourced Storage-Assisted Demand Response in Microgrids. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[20] Carmit Hazay and Yehuda Lindell. 2010. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Springer.

[21] Zhichuan Huang, Ting Zhu, Yu Gu, and Yanhua Li. 2016. Shepherd: sharing energy for privacy preserving in hybrid AC-DC microgrids. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[22] Fabian Knirsch, Clemens Brunner, Andreas Unterweger, and Dominik Engel. 2020. Decentralized and permission-less green energy certificates with GECKO. *Energy Informatics* 3, 2 (2020).

[23] Fabian Laforet, Erik Buchmann, and Klemens BÃűhm. 2016. Towards provable privacy guarantees using rechargeable energy-storage devices. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[24] Solidity Programming Language. 2020. https://docs.soliditylang.org. (2020).

[25] Stephen Lee, Prashant Shenoy, Krithi Ramamritham, and David Irwin. 2018. vSolar: Virtualizing Community Solar and Storage for Energy Sharing. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[26] Yehuda Lindell. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. In *IACR Cryptol. ePrint Arch.*

[27] Jingkun Liu, Ning Zhang, Chongqing Kang, Daniel Kirschen, and Qing Xia. 2017. Cloud energy storage for residential and small commercial consumers: A business case study. *Applied Energy* 188 (2017), 226–236.

[28] Lingjuan Lyu, Sid Chi-Kin Chau, Nan Wang, and Yifeng Zheng. 2020. Cloud-based Privacy-Preserving Collaborative Consumption for Sharing Economy. *IEEE Trans. Cloud Computing* (2020).

[29] Esther Mengelkamp, Johannes Garttner, Kerstin Rock, Scott Kessler, Lawrence Orsini, and Christof Weinhardt. 2018. Designing microgrid energy markets: A case study: The Brooklyn Microgrid. *Applied Energy* 210 (2018), 870–880.

[30] Aditya Mishra, David Irwin, Prashant Shenoy, Jim Kurose, and Ting Zhu. 2012. SmartCharge: cutting the electricity bill in smart homes with energy storage. In *ACM Intl. Conf. on Future Energy Systems (e-Energy)*.

[31] Monero. 2021. http://getmonero.org. (2021).

[32] The Ethereum Yellow Paper. 2014. https://ethereum.github.io/yellowpaper/paper.pdf. (2014).

[33] Quantiki. 2020. https://quantiki.org/wiki/coin-tossing. (2020).

[34] Damian Shaw-Williams and Connie Susilawati. 2020. A techno-economic evaluation of Virtual Net Metering for the Australian community housing sector. *Applied Energy* 261 (2020).

[35] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*.

[36] ETH Gas Station. 2020. https://ethgasstation.info. (2020).

[37] Truffle Suite. 2020. https://www.trufflesuite.com. (2020).


## APPENDIX

## A PROOFS

**Theorem 1.** *If* $\left(x^-(t), y(t)\right)_{t=1}^{T}$ *is an optimal solution of* (P2), *then* $\left((x_i^-(t), y_i(t))_{i=1}^{N}\right)_{t=1}^{T}$, *where* $x_i^-(t) = \frac{a_i(t)}{a(t)} \cdot x^-(t)$ *and* $y_i(t) = \frac{a_i(t)}{a(t)} \cdot y(t)$, *is an optimal solution of* (P1).

Proof. Since $\left(x^-(t), y(t)\right)_{t=1}^{T}$ is an optimal solution of (P2), it satisfies the condition $x^-(t) + y(t) = a(t)$.

We let $x_i^-(t) = \frac{a_i(t)}{a(t)} \cdot x^-(t)$ and $y_i(t) = \frac{a_i(t)}{a(t)} \cdot y(t)$, then $\left(x_i^-(t), y_i(t)\right)_{t=1}^T$ satisfies $x_i^-(t) + y_i(t) = a_i(t)$ for all $i \in \{1, ..., N\}$ and $t \in \{1, ..., T\}$, and hence, is a feasible solution of (P1).

Next, we argue that $\left((x_i^-(t), y_i(t))_{i=1}^N\right)_{t=1}^T$ is an optimal solution of (P1) by contradiction. Suppose that there exists a better solution $\left((x'^-_i(t), y'_i(t))_{i=1}^N\right)_{t=1}^T$ with a lower total cost in (P2). Then $\left(x^-(t), y(t)\right)_{t=1}^T$ is not an optimal solution, because we can find another better solution by considering $\left(\sum_{i=1}^N x'^-_i(t), \sum_{i=1}^N y'_i(t)\right)_{t=1}^T$ instead, which is also a feasible solution of (P2). This will violate the optimality of $\left(x^-(t), y(t)\right)$. □

**Theorem** 2. *If $\left(x^+(t), x^-(t)\right)_{t=1}^T$ is an optimal solution of (P2) and let $x_i^-(t) = \frac{a_i(t)}{a(t)} \cdot x^-(t)$ and $y_i(t) = \frac{a_i(t)}{a(t)} \cdot y(t)$, then proportional and egalitarian cost-sharing schemes are individually rational.*

*Let $\hat{\mathsf{p}}(t) \triangleq \frac{x^-(t) \cdot p(t)}{a(t)}$ and $\mathtt{Cost}_{\mathsf{org}} \triangleq \sum_{t=1}^T x^-(t) \cdot p(t)$. The proportional and egalitarian cost-sharing payments are given as follows:*

$$\begin{cases} P_i^{\mathsf{pp}} = \frac{\mathtt{Cost}_{\mathsf{ess}}}{\mathtt{Cost}_{\mathsf{org}}} \cdot \sum_{t=1}^T a_i(t) \cdot \hat{\mathsf{p}}(t), \\ P_i^{\mathsf{ega}} = \sum_{t=1}^T a_i(t) \cdot \hat{\mathsf{p}}(t) - \frac{\mathtt{Cost}_{\mathsf{org}} - \mathtt{Cost}_{\mathsf{ess}}}{N} \end{cases} \tag{22}$$

Proof. For proportional cost sharing, user $i$'s saving will be

$$\Delta_i^{\mathsf{pp}} = \sum_{t=1}^T x_i^-(t) \cdot p(t) - \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t) \cdot \frac{\sum_{t=1}^T x_i^-(t) \cdot p(t)}{\sum_{t=1}^T x^-(t) \cdot p(t)} \tag{23}$$

$$= \left(\sum_{t=1}^T x^-(t) \cdot p(t) - \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)\right) \cdot \frac{\sum_{t=1}^T x_i^-(t) \cdot p(t)}{\sum_{t=1}^T x^-(t) \cdot p(t)} \tag{24}$$

Next, we show $\sum_{t=1}^T x^-(t) \cdot p(t) \geq \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)$ by contradiction. Suppose $\sum_{t=1}^T x^-(t) \cdot p(t) < \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)$, then $\left(x^+(t), x^-(t)\right)_{t=1}^T$ is not an optimal solution of (P2) because one can always find a better solution by not charging energy storage according to $x^+(t)$. Instead, drawing energy at the time it is needed will only cost $\sum_{t=1}^T x^-(t) \cdot p(t)$, which is cheaper than the cost of charging and subsequently discharging from energy storage ($\sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)$). Hence, we conclude that $\sum_{t=1}^T x^-(t) \cdot p(t) \geq \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)$ and $\Delta_i^{\mathsf{pp}} \geq 0$.

For egalitarian cost sharing, user $i$'s saving will be

$$\Delta_i^{\mathsf{ega}} = \frac{\sum_{t=1}^T x^-(t) \cdot p(t) - \sum_{t=1}^T \left(p(t) + \mathsf{p}_{\mathsf{s}}\right) \cdot x^+(t)}{N} \tag{25}$$

Following a similar approach by contradiction, we can similarly show that $\Delta_i^{\mathsf{ega}} \geq 0$. □

**Theorem** 3. *Consider a multi-transaction $\mathtt{mtx} = (\mathsf{ad}_i, \mathsf{ad}_{\mathsf{ess}}, \mathtt{val}_i)_{i=1}^N$, where $\mathtt{val}_i$ may be negative. Namely, every $\mathsf{ad}_i$ pays to the energy storage operator $\mathsf{ad}_{\mathsf{ess}}$. If $\sum_{i=1}^N \mathtt{val}_i > 0$, mtx can be handled on a blockchain by the following transaction operations:*

$$\mathtt{Bal}(\mathsf{ad}_i) \leftarrow \mathtt{Bal}(\mathsf{ad}_i) - \mathtt{val}_i, \textit{ for all } i \tag{26}$$

$$\mathtt{Bal}(\mathsf{ad}_{\mathsf{ess}}) \leftarrow \mathtt{Bal}(\mathsf{ad}_{\mathsf{ess}}) + \sum_{i=1}^N \mathtt{val}_i \tag{27}$$

Proof. It is straightforward to see that Eqn. (26) applies to the case when $\mathtt{val}_i$ is negative. As long as $\sum_{i=1}^N \mathtt{val}_i > 0$, there is no net out-going payment from $\mathsf{ad}_{\mathsf{ess}}$. Therefore, mtx can be handled properly. □

# B ZERO-KNOWLEDGE PROOFS OF KNOWLEDGE

## B.1 Zero-knowledge Proof of Commitment (zkpCm)

Given $\mathtt{Cm}(x, \mathsf{r})$, a prover wants to convince a verifier of the knowledge of $(x, \mathsf{r})$. We can apply Σ-protocol as follows:

(1) The prover randomly generates $(x', \mathsf{r}') \in \mathbb{Z}_p^2$ and sends the commitment $\mathtt{Cm}(x', \mathsf{r}')$ to the verifier.
(2) The verifier sends a random challenge $\beta \in \mathbb{Z}_p$ to the prover.
(3) The prover replies with $z_x = x' + \beta \cdot x$ and $z_{\mathsf{r}} = \mathsf{r}' + \beta \cdot \mathsf{r}$.
(4) The verifier checks whether $g^{z_x} \cdot h^{z_{\mathsf{r}}} \stackrel{?}{=} \mathtt{Cm}(x', \mathsf{r}') \cdot \mathtt{Cm}(x, \mathsf{r})^\beta$.

Denote a zero-knowledge proof of commitment for $\mathtt{Cm}(x, \mathsf{r})$ by $\mathtt{zkpCm}[x]$.

## B.2 Zero-knowledge Proof of Summation

Given commitments $\left(\mathtt{Cm}(x_1, \mathsf{r}_1), ..., \mathtt{Cm}(x_n, \mathsf{r}_n)\right)$ and $y$, a prover wants to convince a verifier of the knowledge of $y = \sum_{i=1}^n x_i$ without revealing $(x_1, ..., x_n)$. We can apply Σ-protocol as follows:

(1) The prover randomly generates $\mathsf{r}' \in \mathbb{Z}_p$ and sends the commitment $\mathtt{Cm}(0, \mathsf{r}')$ to the verifier.
(2) The verifier sends a random challenge $\beta \in \mathbb{Z}_p$ to the prover.
(3) The prover replies with $z_{\mathsf{r}} = \mathsf{r}' + \beta \cdot \sum_{i=1}^n \mathsf{r}_i$.
(4) The verifier checks whether $g^{\beta y} \cdot h^{z_{\mathsf{r}}} \stackrel{?}{=} \mathtt{Cm}(0, \mathsf{r}') \cdot \prod_{i=1}^n \mathtt{Cm}(x_i, \mathsf{r}_i)^\beta$

Denote a zero-knowledge proof of summation for $\left(\mathtt{Cm}(x_i, \mathsf{r}_i)\right)_{i=1}^n$ by $\mathtt{zkpSum}[y, (x_i)_{i=1}^n]$.

## B.3 Zero-knowledge Proof of Membership

Given a set $\mathcal{X} = \{x_1, ..., x_n\}$ and $\mathtt{Cm}(x, \mathsf{r})$, a prover wants to convince a verifier of the knowledge of $x \in \mathcal{X}$ without revealing $x$. We can apply Σ-protocol as follows:

(1) Suppose $x = x_i \in \mathcal{X}$. The prover first randomly generates $(x'_j, \mathsf{r}'_j) \in \mathbb{Z}_p$ and computes the commitment $\mathtt{Cm}(x'_j, \mathsf{r}'_j)$ for all $j \in \{1, ..., n\}$. Then, the prover randomly generates $\beta_j \in \mathbb{Z}_p$ for each $j \in \{1, ..., n\} \backslash \{i\}$, and computes

$$z_{x_j} = \begin{cases} x'_j + (x_i - x_j)\beta_j, & \text{if } j \in \{1, ..., n\} \backslash \{i\} \\ x'_i, & \text{if } j = i \end{cases}$$

Next, the prover sends $(\mathtt{Cm}(x'_j, \mathsf{r}'_j), z_{x_j})_{j=1}^n$ to the verifier.
(2) The verifier sends a random challenge $\beta \in \mathbb{Z}_p$ to the prover.
(3) The prover sets $\beta_i = \beta - \sum_{j \neq i} \beta_j$, then computes $z_{\mathsf{r}_j} = \mathsf{r}'_j + \mathsf{r} \cdot \beta_j$ for all $j \in \{1, ..., n\}$, and sends $(\beta_j, z_{\mathsf{r}_j})_{j=1}^n$ to the verifier.

(4) The verifier checks whether $\beta \stackrel{?}{=} \sum_{i=1}^{n} \beta_j$ and

$$g^{z_{x_j}} \cdot h^{z_{r_j}} \stackrel{?}{=} \mathtt{Cm}(x'_j, \mathtt{r}'_j) \cdot \left(\frac{\mathtt{Cm}(x, \mathtt{r})}{g^{x_j}}\right)^{\beta_j} \text{for all } j \in \{1, ..., n\}$$

Denote a zero-knowledge proof of membership for $x \in \mathcal{X}$ by $\mathtt{zkpMbs}[x, \mathcal{X}]$.

## B.4 Zero-knowledge Proof of Non-Negativity

Given $\mathtt{Cm}(x, \mathtt{r})$, a prover wants to convince a verifier of the knowledge of $x \geq 0$ without revealing $x$. Suppose $x < 2^m$. We aim to prove there exist $(b_1, ..., b_m)$ such that $b_i \in \{0, 1\}$ for $i \in \{0, ..., m\}$ and $\sum_{i=1}^{m} b_i \cdot 2^{i-1} = x$. We can apply $\Sigma$-protocol as follows:

(1) The prover sends $(\mathtt{Cm}(b_i, \mathtt{r}_i))_{i=1}^{m}$ to the verifier, and provides $\mathtt{zkpMbs}[b_i, \{0, 1\}]$ for each $b_i$ to prove that $b_i \in \{0, 1\}$. Also, the prover randomly generates $\mathtt{r}' \in \mathbb{Z}_p$ and sends the commitment $\mathtt{Cm}(0, \mathtt{r}')$ to the verifier.
(2) The verifier sends a random challenge $\beta \in \mathbb{Z}_p$ to the prover.
(3) The prover replies with $z_{\mathtt{r}} = \mathtt{r}' + \beta \cdot (\sum_{i=1}^{m} \mathtt{r}_i \cdot 2^{i-1} - \mathtt{r})$.
(4) The verifier checks whether $h^{z_{\mathtt{r}}} \stackrel{?}{=} \mathtt{Cm}(0, \mathtt{r}') \cdot \mathtt{Cm}(x, \mathtt{r})^{-\beta} \cdot \prod_{i=1}^{m} \mathtt{Cm}(b_i, \mathtt{r}_i)^{\beta \cdot 2^{i-1}}$.

Denote a zero-knowledge proof of $x \geq 0$ by $\mathtt{zkpNN}[x]$.

## B.5 Security Proof

It is straightforward to prove the completeness of these protocols. We will provide detailed proofs on their soundness and honest-verifier zero-knowledge properties below.

*B.5.1 Soundness Proof.* There exists a *Knowledge Extractor* that makes the prover successfully answer two random challeges $\beta_1$ and $\beta_2$, such that:

- $\mathtt{zkpCm}$. For $x$ we have $(z'_x = x' + \beta_1 \cdot x, z''_x = x' + \beta_2 \cdot x)$ and for $r$ we have $(z'_r = r' + \beta_1 \cdot r, z''_r = r' + \beta_2 \cdot r)$. Then we obtain $x = \dfrac{z'_x - z''_x}{\beta_1 - \beta_2}$ and $r = \dfrac{z'_r - z''_r}{\beta_1 - \beta_2}$. Finally, we can check that $g^x h^r = \mathtt{Cm}(x, r)$.
- $\mathtt{zkpSum}$. $z_1 = r' + \beta_1 \cdot \sum_{i=1}^{N} r_i$ and $z_2 = r' + \beta_2 \cdot \sum_{i=1}^{N} r_i$. Then we have $\sum_{i=1}^{N} r_i = \dfrac{z_1 - z_2}{\beta_1 - \beta_2}$. Finally, we can check that $g^y h^{\sum_{i=1}^{N} r_i} = \prod_{i=1}^{n} \mathtt{Cm}(x_i, \mathtt{r}_i)$ to prove that $y = \sum_{i=1}^{N} x_i$.
- $\mathtt{zkpMbs}$. $z'_{r_j} = r'_j + \beta_1 \cdot r$ and $z''_{r_j} = r' + \beta_2 \cdot r$. Then we have $r = \dfrac{z'_{r_j} - z''_{r_j}}{\beta_1 - \beta_2}$. Finally, we can check for each $j$ that $g^{z_{x_j}} h^r = \dfrac{\mathtt{Cm}(x, \mathtt{r})}{g^{x_j}}$ to prove that $x \in \mathcal{X}$.
- $\mathtt{zkpNN}$. $z'_r = r' + \beta_1 \cdot (\sum_{i=1}^{m} \mathtt{r}_i \cdot 2^{i-1} - \mathtt{r})$ and $z''_r = r' + \beta_2 \cdot (\sum_{i=1}^{m} \mathtt{r}_i \cdot 2^{i-1} - \mathtt{r})$. Then we have $(\sum_{i=1}^{m} \mathtt{r}_i \cdot 2^{i-1} - \mathtt{r}) = \dfrac{z'_{r_j} - z''_{r_j}}{\beta_1 - \beta_2}$. Finally, we can check that $h^{\sum_{i=1}^{m} \mathtt{r}_i \cdot 2^{i-1} - \mathtt{r}} = \prod_{i=1}^{m} \mathtt{Cm}(b_i, \mathtt{r}_i)^{2^{i-1}} \cdot \mathtt{Cm}(x, \mathtt{r})^{-1}$ to prove that $x = \sum_{i=1}^{N} b_i \cdot 2^{i-1} \geq 0$.

*B.5.2 Honest-verifier zero-knowledge Proof.* There exists a *simulator* that chooses random $z, \beta \in Z_p$ and checks:

- $\mathtt{zkpCm}$. $\mathtt{Cm}(x', r') = g^{z_x} \cdot h^{z_r} \cdot \mathtt{Cm}(x, r)^{-\beta}$.
- $\mathtt{zkpSum}$. $\mathtt{Cm}(0, \mathtt{r}') = g^{\beta y} \cdot h^{z_{\mathtt{r}}} \cdot \prod_{i=1}^{N} \mathtt{Cm}(x_i, \mathtt{r}_i)^{-\beta}$.
- $\mathtt{zkpMbs}$. $\mathtt{Cm}(x'_j, \mathtt{r}'_j) = g^{z_{x_j}} \cdot h^{z_{r_j}} \cdot \left(\frac{\mathtt{Cm}(x, \mathtt{r})}{g^{x_j}}\right)^{-\beta_j}$ for all $j \in \{1, ..., n\}$.
- $\mathtt{zkpNN}$. $\mathtt{Cm}(0, \mathtt{r}') = h^{z_{\mathtt{r}}} \cdot \mathtt{Cm}(x, \mathtt{r})^{\beta} \cdot \prod_{i=1}^{m} \mathtt{Cm}(b_i, \mathtt{r}_i)^{-\beta \cdot 2^{i-1}}$.

# C SPDZ PROTOCOL

In the following, we present a simplified version of SPDZ for the clarity of exposition. The full version can be found in [10, 11].

There are three phases in SPDZ protocol: (1) pre-processing phase, (2) online phase, and (3) output and validation phase. We write $\langle x \rangle$ as a *secretly shared* number, meaning that there is a vector $(x_1, ..., x_n)$, such that each party $i$ knows only $x_i$. To reveal secretly shared number $\langle x \rangle$, each party $i$ broadcasts $x_i$ to other parties. Then each party can reconstruct $x = \sum_{i=1}^{n} x_i$. We write $\langle\langle x \rangle\rangle$ meaning that both $\langle x \rangle$ and the respective MAC $\langle \gamma(x) \rangle$ are secretly shared.

## C.1 Online Phase

In the online phase, the parties can jointly compute an arithmetic circuit, consisting of additions and multiplications with secretly shared input numbers.

*C.1.1 Addition.*

Given secretly shared $\langle x \rangle$ and $\langle y \rangle$, and a public known constant $c$, the following operations can be attained by local computation at each party, and then the outcome can be assembled from the individual shares:

A1) $\langle x \rangle + \langle y \rangle$ can be computed by $(x_1 + y_1, ..., x_n + y_n)$.
A2) $c \cdot \langle x \rangle$ can be computed by $(c \cdot x_1, ..., c \cdot x_n)$.
A3) $c + \langle x \rangle$ can be computed by $(c + x_1, x_2, ..., x_n)$.

*C.1.2 Multiplication.*

Given secretly shared $\langle x \rangle$ and $\langle y \rangle$, computing the product $\langle x \rangle \cdot \langle y \rangle$ involves a given multiplication triple. A multiplication triple is defined by $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, where $a, b$ are some unknown random numbers and $c = a \cdot b$, are three secretly shared numbers already distributed among the parties. The triple is assumed to be prepared in a pre-processing phase. To compute $\langle x \rangle \cdot \langle y \rangle$, it follows the below steps of operations (A4):

A4.1) Compute $\langle \epsilon \rangle = \langle x \rangle - \langle a \rangle$ (by A1). Then, reveal $\langle \epsilon \rangle$, which does not reveal $x$.
A4.2) Compute $\langle \delta \rangle = \langle y \rangle - \langle b \rangle$. Then, reveal $\langle \delta \rangle$.
A4.3) Finally, compute $\langle x \rangle \cdot \langle y \rangle = \langle c \rangle + \epsilon \cdot \langle b \rangle + \delta \cdot \langle a \rangle + \epsilon \cdot \delta$ (by A1-A3).

*C.1.3 Message Authentication Code.*

To safeguard against dishonest parties, who may perform incorrect computation, an information-theoretical message authentication code (MAC) can be used for verification. We write a MAC key as a global number $\widetilde{\alpha}$, which is unknown to the parties, and is secretly shared as $\langle \widetilde{\alpha} \rangle$. Every secretly shared number is encoded by a MAC as $\gamma(x) = \widetilde{\alpha} x$, which is secretly shared as $\langle \gamma(x) \rangle$. For each $\langle x \rangle$, each party $i$ holds a tuple $(x_i, \gamma(x)_i)$ and $\widetilde{\alpha}_i$, where $x = \sum_{i=1}^{n} x_i$, $\widetilde{\alpha} = \sum_{i=1}^{n} \widetilde{\alpha}_i$ and $\gamma(x) = \widetilde{\alpha} x = \sum_{i=1}^{n} \gamma(x)_i$. If any party tries to modify his share $x_i$ uncoordinatedly, then he also needs to modify $\gamma(x)_i$ accordingly. Otherwise, $\gamma(x)$ will be inconsistent. However, it is difficult to modify $\gamma(x)_i$ without coordination among the parties, such that $\widetilde{\alpha} x = \sum_{i=1}^{n} \gamma(x)_i$. Hence, it is possible to detect incorrect computation (possibly by dishonest parties) by checking the MAC.

To check the consistency of $x$, there is no need to reveal $\langle\widetilde{\alpha}\rangle$. One only needs to reveal $\langle x\rangle$, and then reveals $\widetilde{\alpha}_i - x\cdot\gamma(x)_i$ from each party $i$. One can check whether $\sum_{i=1}^{n}(\widetilde{\alpha}_i - x\cdot\gamma(x)_i) \stackrel{?}{=} 0$ for consistency. To prevent a dishonest party from modifying his share $x_i$ after learning other party's $x_j$. Each party needs to commit his share $x_i$ before revealing $x_i$ to others.

To maintain the consistency of MAC for operations A1-A4, the MAC needs to be updated accordingly as follows:

- B1) $\langle x\rangle + \langle y\rangle$: Update MAC by $(\gamma(x)_1 + \gamma(y)_1, ..., \gamma(x)_n + \gamma(y)_n)$.
- B2) $c\cdot\langle x\rangle$: Update MAC by $(c\cdot\gamma(x)_1, ..., c\cdot\gamma(x)_n)$.
- B3) $c + \langle x\rangle$: Update MAC by $(c\cdot\alpha_1 + \gamma(x)_1, ..., c\cdot\alpha_n + \gamma(x)_n)$.
- B4) $\langle x\rangle\cdot\langle y\rangle$: Update MAC at each individual step of A4.1-A4.3 accordingly by B1-B3.

The additions and multiplications of $\langle\!\langle x\rangle\!\rangle$ and $\langle\!\langle y\rangle\!\rangle$ follow A1-A4 and the MACs will be updated accordingly by B1-B4.

To verify the computation of a function, it only requires to check the MACs of the revealed values and the final outcome, which can be checked all efficiently together in a batch at the final stage by a technique of called "random linear combination".

## C.2 Pre-processing Phase

In the pre-processing phase, all parties need to prepare a collection of triplets $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ where $c = a\cdot b$, each for a required multiplication operation. Assume that the parties hold secretly shared numbers $a = \sum_{i=1}^{N} a_i$ and $b = \sum_{i=1}^{N} b_i$ (which has been generated by local random generation). Note that $a\cdot b = \sum_{i=1}^{N} a_i b_i + \sum_{i=1}^{N}\sum_{j=i"i\neq j}^{N} a_i b_j$. $a_i b_i$ can be computed locally. To distribute $a_i b_j$, one can use partial homomorphic cryptosystems, with encryption function $\mathsf{Enc}[\cdot]$ and decryption function $\mathsf{Dec}[\cdot]$ using party $i$'s public and private $(K_i^{\mathsf{p}}, K_i^{\mathsf{p}})$. First, party $i$ sends $\mathsf{Enc}_{K_i^{\mathsf{p}}}[a_i]$ to party $j$, who responds by $C_i = b_j\mathsf{Enc}_{K_i^{\mathsf{p}}}[a_i] - \mathsf{Enc}_{K_i^{\mathsf{p}}}[\tilde{c}_j]$, where $\tilde{c}_j$ is a random share generated by party $j$ and is encrypted by party $i$'s public key $K_i^{\mathsf{s}}$. Then party $i$ can obtain $\tilde{c}_j = \mathsf{Dec}_{K_i^{\mathsf{s}}}[C_j]$. Hence, $a_i b_j = \tilde{c}_i + \tilde{c}_j$, which are secret shares $a_i b_j$. The above generation assumes honest parties. To prevent cheating by dishonest parties, one would need to use proper zero-knowledge proofs before secret sharing [10, 11].

To generate a random mask $\langle\!\langle r^i\rangle\!\rangle$, each party $j$ needs to generate a random share $r_j^i$ locally. Then the parties follow the similar procedure of triplet generation to compute the secretly shared product $\langle\gamma(r^i)\rangle$, where $\gamma(r^i) = \widetilde{\alpha}r^i$.

## C.3 Output and Validation Phase

We describe random linear combination for batch checking. To check the MACs of a number of secretly shared numbers $\langle\!\langle x^1\rangle\!\rangle, ..., \langle\!\langle x^m\rangle\!\rangle$ in a batch, first generate a set of random $(\mathsf{r}^1, ..., \mathsf{r}^m)$. then reveal $\langle\!\langle x^1\rangle\!\rangle, ..., \langle\!\langle x^m\rangle\!\rangle$. Each party $i$ computes $\sum_{j=1}^{m}\mathsf{r}^j(\widetilde{\alpha}_i - x^j\cdot\gamma(x^j)_i)$ and reveals it. All parties check whether $\sum_{i=1}^{N}\sum_{j=1}^{m}\mathsf{r}^j(\widetilde{\alpha}_i - x^j\cdot\gamma(x^j)_i) \stackrel{?}{=} 0$ for consistency in a batch checking.

## C.4 Protocol

We summarize the SPDZ protocol as follows:

(1) *Pre-processing Phase*: In this phase, a collection of shared random numbers will be constructed that can be used to mask the private input numbers. For each private input number of party $i$, there is a shared random number $\langle\!\langle r^i\rangle\!\rangle$, where $r^i$ is revealed to party $i$ only, but not to other parties. All parties also prepare a collection of triplets $(\langle\!\langle a\rangle\!\rangle, \langle\!\langle b\rangle\!\rangle, \langle\!\langle c\rangle\!\rangle)$ where $c = a\cdot b$, each for a required multiplication operation.

(2) *Online Phase*: To secretly shares a private input number $x^i$ using $\langle\!\langle r^i\rangle\!\rangle$, without revealing $x^i$, it proceeds as follows:
   1) Party $i$ computes and reveals $z^i = x^i - r^i$ to all parties.
   2) Every party sets $\langle\!\langle x^i\rangle\!\rangle \leftarrow z^i + \langle\!\langle r^i\rangle\!\rangle$.

   To compute an arithmetic circuit, implement the required additions or multiplications by A1-A4 and the MACs are updated accordingly by B1-B4.

(3) *Output and Validation Phase*: All MACs will be checked for all revealed numbers and the final output value. It can check all in a batch using random linear combination. If there is any inconsistency in the MACs, then abort.

Note that SPDZ cannot guarantee abort with fairness – dishonest parties may learn some partial values, even when the protocol aborts. However, this is a fundamental problem for any multi-party computation protocol with a dishonest majority, where dishonest parties are not identifiable when the computation is aborted.

# D SECURITY ANALYSIS

We adopt the most common approach of security analysis in cryptography, based on the *Ideal/Real-Model Simulation* paradigm to prove security and formalize the security achieved by our protocol. We next briefly describe the simulation paradigm. The detailed explanation can be found in the tutorial [26].

In an *ideal model*, all the parties send their private inputs to a trusted third party, who performs the prescribed computations and outputs the results to each party, whereas such a trusted third party is unlikely to exist in the real model. The security is defined by comparing what an adversary can learn in the real model to that in the ideal model. If what can be learned by an adversary in the real world can be totally simulated in an ideal world, then the adversary cannot learn more information in the real world than in the ideal world, we can say that a protocol $\Pi$ is as secure as its corresponding ideal functionality $\mathcal{F}$. We give a formal definition of the security of our protocol as below:

**Theorem** 4. *Assuming the hardness of discrete logarithm problem underlying the Pedersen commitment scheme as well as the non-interactive zero-knowledge proofs of knowledge, in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model [11], the protocol $\Pi_{\mathsf{pess}}$ securely implements $\mathcal{F}_{\mathsf{pess}}$ with abort in the presence of an adaptive, active adversary in a dishonest-majority setting, if for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ for the real model, there also exists a PPT adversary $\mathcal{S}$ for the ideal model, such that for each $i \in N$:*

$$\{\mathtt{VIEW}^{\mathtt{IDEAL}}_{\mathcal{F}_{\mathsf{pess}},\mathcal{S}}\} \stackrel{stat}{\equiv} \{\mathtt{VIEW}^{\mathtt{REAL}}_{\Pi_{\mathsf{pess}},\mathcal{A}}\}$$

*where* $\mathtt{VIEW}^{\mathtt{IDEAL}}$ *and* $\mathtt{VIEW}^{\mathtt{REAL}}$ *refer to the output distributions of the corrupted and honest users in both ideal and real worlds.*

We next sketch the proof of the above theorem. Our aim is to demonstrate a simulator in the ideal model that can create a statistically indistinguishable view from that of the adversary in the real model. The simulator $\mathcal{S}$ *externally* interacts with the ideal

functionality and *internally* runs a copy of the protocol $\Pi_{\mathsf{pess}} \diamond \mathcal{F}_{\mathsf{prep}}$ feeding messages to the adversary $\mathcal{A}$. However, it is a not trivial task for a simulator to emulate an adaptively malicious adversary, who is able to corrupt users at any time during the protocol. The challenge lies in the difficulty that it must produce a consistent view of the corrupted users throughout the protocol without knowing their inputs. Firstly, we define an ideal functionality $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$ in the stage 1 *Preoperation Scheduling* computing the total energy demands $a(t)$ before presenting the corresponding simulator $\mathcal{S}^{(1)}_{\mathsf{a(t)}}$.

Functionality $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$

**Input:** On input (*input*, $U_i$, $a_i(t)$) from $U_i$, the functionality stores $(a_i(t))$.

**Output:** On input (*output*) from all honest users, the functionality computes and output $a(t) = \sum_{i=1}^{N} a_i(t)$ to all the users.

**Abort:** On input (*abort*), the functionality outputs $\emptyset$.

**Initialize:** The simulator first calls $\mathcal{F}_{\mathsf{prep}}$ to generate a sufficient number of multiplication triples and random pairs. Moreover, it has access to all the shares of the MAC key. The adversary $\mathcal{A}$ firstly corrupts a set of users, denoted by $\mathtt{C}$, whose internal states are also visible to the simulator. Then the adversary may adaptively make corruptions on other users during the protocol. Next, the simulator produces $g, h = g^k \in \mathbb{Z}_p^*$, where $k = \log_g h$ is the trapdoor to Pedersen commitment, with which the simulator is able to find out two pairs $(m, r)$, $(m', r')$, such that $\mathtt{Cm}(m, r) = \mathtt{Cm}(m', r')$.

Simulator $\mathcal{S}^{(1)}_{\mathsf{a(t)}}$

(1) For honest users $i \notin \mathtt{C}$, without the knowledge of their inputs, $\mathcal{S}$ will simply generates dummy inputs $\hat{a}_i(t) = 0, \hat{r}_i(t) \in Z_p$ and reveals a commitment $\hat{C}_i(t) = \mathtt{Cm}\Big(\hat{a}_i(t), \hat{r}_i(t)\Big)$. For the corrupted users $i \in \mathtt{C}$, the simulator can extract their inputs with the knowledge of all the shares of $\langle\!\langle r \rangle\!\rangle$.
*Remark:* From the perspective of the adversary, the inputs of the honest users is uniformly random due to the information-theoretically hiding properties of SPDZ secret-sharing, where unless all the $N$ shares are collected can the inputs be reconstructed, and of the pedersen commitment.

(2) $\mathcal{S}$ firstly calls $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$ to obtain the output $a(t)$. As $\mathcal{S}$ already computed an output $\hat{a}(t)$ using dummy inputs of the honest users, it can respectively modify the share and MAC of a random honest user by adding $a(t) - \hat{a}(t)$ and $\alpha\Big(a(t) - \hat{a}(t)\Big)$ with the MAC key $\alpha$ initialized in the preprocessing phase. Then $\mathcal{S}$ can perform the MAC check to evaluate and open $a(t)$. If the check passes, $\mathcal{S}$ calls $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$ to output $a(t)$ to all the users. Otherwise, $\mathcal{S}$ sends $\mathtt{Abort}$ to $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$.
*Remark:* No matter what inputs the adversary generates for the corrupted users, $\mathcal{S}$ can always create a statistically indistinguishable output distribution in the ideal model from that in the real model from the view of the adversary $\mathcal{A}$. For the evaluation of $a(t)$, each $i$-th share $\alpha_i a(t) - \gamma_i\Big(a(t)\Big)$ appears uniformly random to the adversary, which has exactly the same distribution in both ideal and real models.

After the simulator provided the simulated input $\Big(\hat{a}_i(t), \hat{r}_i(t), \hat{C}_i(t)\Big)$ for $i \notin \mathtt{C}$, the adversary can corrupt an honest user $i$ at any time. As aforementioned, the simulator must reveal its entire internal states, including the inputs, share of inputs and random values that are consistent with the commitment $\hat{C}_i(t)$ to simulate an adaptive adversary. It is easy to obtain the input $a_i(t)$ from $\mathcal{F}^{(1)}_{\mathsf{a(t)}}$. Regarding the random value, the simulator will take advantage of the trapdoor $k$ of the Pedersen commitment to obtain $r_i(t) = \hat{r}_i(t) + \Big(\hat{a}_i(t) - a_i(t)\Big) \cdot k^{-1}$, such that $\hat{C}_i(t) = \mathtt{Cm}(a_i(t), r_i(t)) = \mathtt{Cm}(\hat{a}_i(t), \hat{r}_i(t))$. Moreover, user $i$'s share of his initial dummy input $\hat{a}_i(t)$ is $\hat{a}_i(t) + r_i - r$. Thus, it is trivial for the simulator to reveal the share $a_i(t) + \hat{a}_i(t) - r + r_i$ by adding $a_i(t)$. (See online phase in Section C.4).

Next, we give a brief description of the SPDZ-based zero-knowledge proofs $\mathtt{zkpCm}[a_i(t)]$ in $\Pi^{(1)}_{\mathsf{pess}}$ and $\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N]$ in $\Pi^{(2)}_{\mathsf{pess}}$. For $\mathtt{zkpCm}[a_i(t)]$, $z_{a_i(t)}$ and $z_{r_i(t)}$ are collectively computed by all users and will be evaluated via MAC check to prove their correctness. A similar simulator to $\mathcal{S}^{(1)}_{\mathsf{a(t)}}$ can be constructed to emulate the ideal functionality computing $z_{a_i(t)}$ and $z_{r_i(t)}$. The challenge $\beta(t)$ is uniformly random independent of the prover's input as it is obtained by summing the random values generated by all the users. Thus, this zero-knowledge proof is secure given the proof of completeness, soundness, zero-knowledge properties in Section B.5. The same security likewise applies to $\mathtt{zkpSum}[\mathtt{Cost}_{\mathtt{ess}}, (P_i)_{i=1}^N]$.

We skip the details for the simulator $\mathcal{S}^{(2)}_{\mathtt{Cost}_{\mathtt{ess}}}$ emulating ideal functionality computing the total payment $\mathtt{Cost}_{\mathtt{ess}} = \sum_i^N P_i$ as it is similar to $\mathcal{S}^{(1)}_{\mathsf{a(t)}}$ except using a different input $P_i$.

# E ETHEREUM BLOCKCHAIN PLATFORM & SMART CONTRACTS

In this section, we provide a brief description of *Ethereum blockchain platform* and *Solidity programming language* as well as the details on the implementations of the smart contracts in our protocols.

## E.1 Background

Bitcoin was the first widely adopted digital currency on a permissionless distributed ledger. Bitcoin relies on a tampering-resistant ledger based on cryptographic signatures. Tampering-resistance ensures integrality when the ledger is maintained by a network of peer-to-peer systems called "miners". The miners are incentivized by cryptocurrency rewards for updating and validating the transaction records. Since the distributed ledger can be modified by multiple systems simultaneously, it is crucial to ensure consistency by a distributed consensus protocol among untrusted peer-to-peer systems, based on proof-of-work (by solving computational puzzles) or proof-of-stake (by demonstrating ownership of digital assets).

Subsequently, Ethereum was built on the Bitcoin ideas by expanding its functions to support general computing as smart contracts along with transactions. Bitcoin operates using a transaction-output-based system, called unspent transaction outputs (UTXOs), whereas Ethereum operates using accounts and balances in a manner called state transitions. Smart contracts, which are code programmed in high-level logic, will be compiled into byte code and executed in the virtual machine of miners. Miners will charge additional crytocurrency payments called gas costs, because the extra

computational tasks incurred by smart contracts will be broadcast throughout the blockchain. Smart contracts are implemented in a high-level programming language, such as Solidity [24].

It is worth noting that Bitcoin and Ethereum were only supposed to enable decentralization, but do not ensure privacy. In fact, the transaction histories of many crytocurrencies are visible to the public. There are certain high-profiled prosecution of darknet operators based on the evidence of Bitcoin transactions. Supporting privacy in blockchain is a crucial on-going research topic.

## E.2 Smart Contract Implementation

We next explain in details of how *Multi-Signature* smart contract can achieve the step (5) and (6) of the stage *Cost-sharing Payment* by the following methods:

(1) `submitTransaction()`. This method allows each user to submit the $\texttt{zkpSum}[\texttt{Cost}_{\texttt{ess}}, (P_i)_{i=1}^{N}]$ that they have agreed upon off the chain. The method will compare whether users have submitted the same `zkpSum`.
(2) `confirmTransaction()`. On one hand, this method allows each user to confirm that the stored `zkpSum` in the smart contract is the one that they have agreed upon off the chain. On the other, each user is required to submit a $\texttt{nzkpNN}[\texttt{Bal}(\texttt{ad}_i) - P_i]_{i=1}^{N}$, which will be validated to prove that there is sufficient balance in his account to pay for the energy cost.
(3) `executeTransaction()`. This method can only be executed by the operator unless all the users have already confirmed the transaction. The method will validate the `zkpSum` before calling *ESToken* smart contract to credit $\texttt{Cost}_{\texttt{ess}}$ to the operator's account and debit the corresponding payment from each user's account.
(4) `secretlyJointTransfer()`. This method, defined within the *EStoken* smart contract is invoked by `executeTransaction()`, which actually performs the real transfer between multiple accounts.