Fair and Privacy-Respecting Bitcoin Payments for Smart Grid Data

Tassos Dimitriou, *Senior Member, IEEE*, Ameer Mohammed

***Abstract*—In this work we present $\mathcal{DPTS}$, a data payment and transfer scheme that uses bitcoin payments to reward users for detailed electricity measurements they submit to a utility provider. $\mathcal{DPTS}$ emphasizes both privacy and fairness of transactions; not only it allows participants to earn bitcoins in a way that cannot be linked to their actions or identities but also ensures that data is delivered *if and only if* an appropriate payment is received. While $\mathcal{DPTS}$ is described in the smart grid setting, the protocol can also be applied in other areas where incentives are used to increase user participation. One such important area is *participatory* or *crowd-sensing*, where individuals use their smartphones to report sensed data back to a campaign administrator and obtain a reward for it. $\mathcal{DPTS}$ allows users to enjoy the benefits of participation without compromising anonymity. The proposal is coupled with a security analysis showing the privacy-preserving character of the system along with an efficiency analysis demonstrating the feasibility of our approach.**

***Index Terms*—Smart grid, Crowd sensing, Privacy, Fairness, Data reporting, Rewarding mechanisms, Bitcoin payments, Blockchain, zkSNARKs**

I. Introduction

Smart grid technologies utilize a two way flow and transmission of electricity in an effort to improve reliability and efficiency of the electric grid. At the core of the smart grid lie 'smart meters', devices deployed at consumer locations that provide real-time monitoring of electricity consumption. This advanced metering infrastructure helps provide better situational awareness of the electric power system, thus reducing outages, improving reliability, and benefiting both users and electricity providers from a balanced utilization of energy and lower costs.

However, as meters transmit power-usage information to the utility provider, there are serious concerns with respect to the privacy-invasive character of these devices. Indeed, the frequent collection of energy data can be used to profile users and their home devices, thus revealing detailed information about behaviors, activities or preferences of inhabitants [1].

Given that most users are reluctant to submit detailed measurements, the use of *incentives or rewards* may be one of the few mechanisms to solicit user collaboration for even more detailed measurements than those allowed by regulatory frameworks. For example, instead of reporting data every hour, the user may be willing to increase the frequency and the granularity of submitted data. Rewarding schemes could then be used by utility providers to attract a large number of participants, thus better mapping the state of the grid and offering an even better service back to the users.

What makes this problem challenging is that security and privacy are two *contradictory* objectives; on one hand only authenticated meters should benefit from the use of rewarding services such as the ones used for submitting energy data. On the other hand, the identity of the meter/user or other contextual information should not serve as a unique identifier that can be used to filter the meter's transactions or link it with other activities and thus leak sensitive user information. However, even if the above issues are solved, a last remaining issue is the *fairness* problem, which in the context of secure multi-party computation protocols, states that either both the meter and provider receive their desired output or neither of them will. This is exemplified by the following scenario.

Consider a user Alice who has agreed to report more detailed measurements to a Utility Provider ($\mathcal{UP}$). Obviously Alice cannot send her data directly to the $\mathcal{UP}$ because (i) user privacy is violated, and (ii) the $\mathcal{UP}$ might mis-behave, i.e. obtain the data and then refuse to credit Alice for it. Conversely, if the $\mathcal{UP}$ first rewards Alice for some promised data, Alice might not send the data at all.

Is there a way to send electricity data in a private way so that none of the parties can cheat the other? While the privacy problem can be solved using various aggregation techniques ([2], [3], [4]), the fair exchange is a fundamental problem that *cannot* be solved in general *without* the use of a trusted third party.

In typical on-line payment systems (e.g. using credit cards or bank transfers) the buyer cannot be easily cheated since if something goes wrong the payment can be reversed. For example, if a particular product is never delivered, the buyer can claim its money back. The centralized character of the system ensures that none of the parties can be in disadvantage. However, the situation is quite different with blockchain-based currencies like Bitcoin. In such a system, once a transaction is posted and finalized in the blockchain, it cannot be reversed unless the seller voluntarily agrees to return the money back. In general, nothing prevents the seller to keep both the money and the product. This is further magnified by the fact that both parties may interact through pseudonymous IDs.

Contributions: In this work we propose $\mathcal{DPTS}$, a data payment and transfer scheme that uses Bitcoin payments for the trading of smart grid data between the owner of a smart meter and a utility provider. Our protocol is fair, ensuring no party can cheat the other; in particular, data is delivered *if and*

T. Dimitriou is affiliated with the Computer Engineering Dept. Kuwait University, Kuwait. Email: tassos.dimitriou@ieee.org (Corresponding author).
A. Mohammed is affiliated with the Computer Engineering Dept. Kuwait University, Kuwait. Email: ameer.mohammed@ku.edu.kw

only if an appropriate payment is received, essentially making the protocol *atomic*. Additionally, the payment is proportional to the value of the data as captured by a public utility function. Hence another key requirement is to compute the correct utility *prior* to the release of the user data.

We have thoroughly analyzed the security and privacy aspects of the proposal and have showed that $\mathcal{DPTS}$ is indeed privacy preserving. We have also studied the efficiency aspects of the proposal demonstrating its practicality as the bitcoin network is only used for the transfer of bitcoins while the majority of the protocol's steps take place *offchain*.

$\mathcal{DPTS}$ is complementary to all approaches that try to enhance user privacy when reporting detailed electricity measurements to a utility provider. Every time such data is submitted, our scheme can be used to ensure that an appropriate bitcoin amount will be credited to the user's untraceable address.

Our proposal also fits nicely in other areas where rewarding schemes can be used to encourage user participation. One such important area is *participatory* or *crowd-sensing*, where individuals use their smartphones (or other sensing devices) to collect data of common interest (environmental, physiological, etc.) and report them back to a campaign administrator [5], [6]. The use of appropriate rewarding mechanisms to increase user participation is critical to the success of mobile crowd-sensing applications. The generic character of $\mathcal{DPTS}$ fits nicely in this domain as well.

Organization: The rest of the paper is organized as follows. In Section II, we review related work on trading data using Bitcoin. In Section III, we introduce our security and privacy model, the main operations allowed in our scheme and the basic tools we will be using throughout this work. The details of $\mathcal{DTPS}$ can be found in Section IV. Its security and efficiency aspects are analyzed in Section V. Further extensions of $\mathcal{DTPS}$ are discussed in Section VI. Finally, Section VII concludes the paper.

II. Related Work

There is an extensive literature for the problem of fair exchange in which two parties want to swap digital items such that neither of them can cheat the other. However, fairness cannot be achieved without the presence of a trusted third party (TTP) as was shown in [7]. Hence protocols rely on the use of a honest TTP which can help the two parties exchange the items fairly.

A TTP can be online or offline. An online TTP always mediates between the two parties to ensure the fairness of the exchange [8]. A more interesting case is when the TTP is offline in which case the TTP intervenes only if there is a dispute. Protocols of this kind are called *optimistic* fair exchange protocols [9]. In the common case where Alice and Bob are honest and behave correctly, the TTP does not have to be involved.

In this work, the Bitcoin blockchain will acquire the role of the "trusted party" in a fair exchange protocol. A few prior works already adopted this view. For example, in [10], Bitcoin is used to implement a penalty mechanism to enforce the proper behavior of the participants. Liu et al. [11] leverage the functionality of the Bitcoin blockchain to instantiate known protocols for the fair exchange of a cryptocurrency payment for a receipt. However, the work in [11] resides on the use of Ethereum smart contracts which assume a richer functionality from the underlying blockchain network. In particular, since users need to pay fees to execute a smart contract, storing and computing complex instructions typically results in high costs. Another fair exchange protocol based on Ethereum is FairSwap [12] which emphasizes on reducing the cost for running the smart contract on the blockchain.

Our protocol uses the Bitcoin protocol without assuming much from its scripting language: all we need is the ability to create a bitcoin payment transaction that specifies a value h for which anyone with an appropriate pre-image x such that $Hash(x) = h$ can claim the money. As this is the only part of the protocol that has to be executed in the blockchain, this greatly simplifies adoption.

Similarly to our proposal, Zero Knowledge Contingent Payments [13] use Bitcoin in combination with zero knowledge proofs to sell a secret, for example a solution to a sudoku puzzle. In this case the secret is a key used to encrypt the advertized solution. The key is released if and only if a payment is received thus guaranteeing strong fairness of the exchange. This work was later extended in [14] to allow a seller to receive payments after proving that a certain service has been rendered (as opposed to selling a secret in [13]).

In a setting similar to ours, Wang *et al*. [15] use bitcoins to reward users participating in crowd-sensing, however the protocol relies on miners verifying the quality of sensed data. Hence a malicious miner may forward the data to the server thus undermining the fairness of the process. CrowdBC [16] is another protocol tailored for crowd-sensing applications. However, miners again play a central role in collecting and evaluating user submitted data, hence they may collude with the provider and leak user sensitive information as well as the data themselves. Lu *et al*. [17] use smart contracts to handle both worker submissions and payments by the collector, however as all information goes through the smart contract this essentially turns the system into a centralized one. Finally, the work of Duan *et al*. [18] is also based on smart contracts bearing monetary rewards. Unfortunately, the threat model is very limited as the authors assume that data consumers and the service provider do not collude with each other since otherwise data confidentiality and user privacy is lost.

Our work avoids the use of smart contracts which essentially turn the system into a centralized one as all data have to go through the smart contract. Furthermore, smart contract transactions induce direct monetary costs to the collector, thus affecting the practicality of large scale systems. As we don't rely on miners to handle data submissions, the possibility of collusion is eliminated. Finally, most of the protocol steps take place offchain, thus contributing to the efficiency of our protocol.

A summary of the properties achieved by $\mathcal{DPTS}$ and a comparison with the most relevant works is shown in Table I. For token-based approaches that do not utilize the blockchain network the reader is referred to [19].

TABLE I
COMPARISON WITH EXISTING PROTOCOLS

Properties	[15]	[16]	[17]	[18]	This work
Smart contract based	Yes	Yes	Yes	Yes	No
Integrity	✓	✓	✓	✓	✓
Confidentiality	✓	✓	✓	✓	✓
Privacy/Anonymity	-	◇[4]	✓	-	✓
Fairness	-[1,2]	-[1,2,4]	-[1,4]	-	✓

✓: Provided ◇: Partially provided -: Not provided

1. Possible collusion between malicious miners and $\mathcal{UP}$.
2. Miners evaluate solution beforehand.
3. User identities are handled by smart contract.
4. Solution encrypted with $\mathcal{UP}$'s public key.

III. System model and Assumptions

In this work, we consider the setting in which owners of smart meters are willing to report detailed electricity measurements to a utility provider and then get rewarded for the data they provide.

A. System model

We consider a network where meters $\mathcal{M}$ associated with different households and owned by users $\mathcal{U}$ are connected to a Utility Provider $\mathcal{UP}$. Within this setting, we propose a protocol to report detailed measurements that ensures two main security properties: (i) *privacy*, guaranteeing that neither the provider nor other users of the system can learn anything about the user/meter identity during the data reporting and rewarding phases of the protocol. (ii) *fairness*, making sure that no party can cheat the other. Thus a provider cannot obtain the data for free or, more generally, in a price smaller than the data's worth. Similarly, a user cannot obtain a reward and refuse to release the data or, more generally, release data whose value is less than the promised one by the provider.

Our goal is to keep meters simple and to rely on cryptographic operations outside the tamper-evident part of the meter. For this reason, all operations are offloaded to the user who can afford larger connectivity and can deliver information about energy usage without imposing any computational overhead on the meters.

Figure 1 illustrates the main use case we would like to support. As detailed meter data can be used to extract appliance usage and track user activities, we differentiate between two types of data for smart metering. The first includes customer data that can be tied to a particular household and are sent *directly* by the meter to the utility provider. This is low-frequency data (e.g. every few days) that can typically be used for billing and management purposes. The second category includes detailed usage data that can be used to enhance power network management, facilitate demand response, improve energy generation and distribution, and so on. These data go beyond what is allowed by regulatory frameworks to collect, they are high-frequency data (e.g. every few seconds or minutes) and are highly sensitive as they impose a threat to customer privacy.

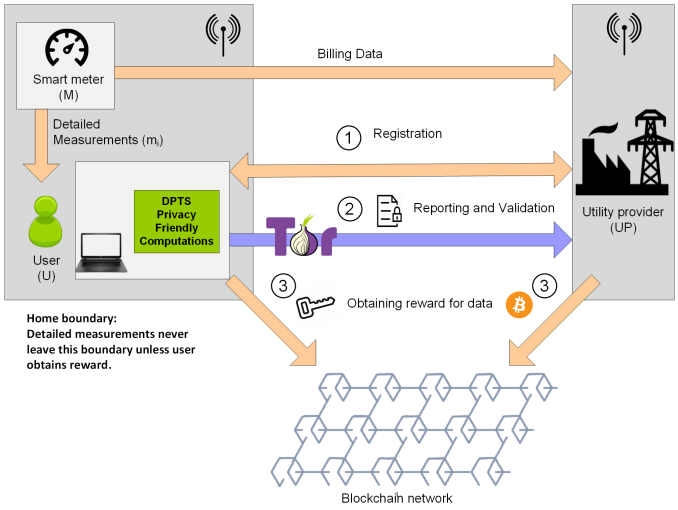

Fig. 1. System Model - Data submission and rewarding phases.

The user is in possession of these detailed measurements and may choose to release them for a sufficient monetary reward. The use of such incentives can help utility providers attract a larger number of participants, thus increasing both the size and the quality of collected data. Hence, these data never leave the home boundary unless the user obtains an appropriate reward for them.

Privacy-friendly calculations are performed on the user side along with proofs of correct valuation of the data. These proofs are then relayed back to the provider for verification and testing. If the provider is satisfied, the data are exchanged in a fair manner using the blockchain network. This mode of operation fits other domains where a provider is interested in obtaining user data such as, for example, *participatory or crowd-sensing*. Hence, with minimal modifications, our framework can also be used in incentivizing users to sense and collect data using their smart phones.

Figure 1 summarizes the core operations expected in our model. Before a user engages in data exchange, she must register first (Step 1). This is a one-time step needed for accountability purposes, i.e. to ensure that only authorized meter owners can participate in the system. However, this step does not have a negative impact on user privacy as we make sure that users remain anonymous in all subsequent phases of the protocol. In the main operational phase, the user submits encrypted data (Step 2) and collects bitcoin payments (Step 3). However to ensure that no party can cheat the other (the provider refusing to pay the user after getting the data, or the user refusing to release the data after getting paid), Step 5 is using the Blockchain network to ensure fairness. The key used to encrypt the data will be released *if and only if* an appropriate payment is received.

However, since the provider does not have access to the data beforehand, he also needs to be sure that electricity consumption data $cons$ worth some value val according to some utility function $Util()$ that has been established by $\mathcal{UP}$. The utility function $Util : cons \rightarrow value$ is a public function that takes in consumption data $cons$ and outputs a value to

be paid to the user for the data provided. The total value is computed by adding the individual values corresponding to all consumption data provided by $\mathcal{M}$, i.e. if n is the number of measurements, $value_{total} = \sum_{i=1}^{n} value_i$. To solve the fairness problem mentioned before, the user does not immediately reveal the measurements but commits to them and to the value to be rewarded. The $\mathcal{UP}$, using the commitments, evaluates itself the total value to be paid and if everything checks out, the data is exchanged *simultaneously* with the computed reward.

Here we consider Utility functions that are fairly typical, however we place an emphasis on efficiency. We first exemplify our construction with a Linear Utility that sets a value for each unit of data provided. In this case, the utility is given by the expression $val = a_1 \cdot cons + a_0$, specifying the value gained for each unit of consumption. Linear utilities can also be generalized to families of linear functions indicating a different utility to be applied depending on time of date, volume of consumption, etc. These factors determine the constants a_0, a_1 to be applied, however it will always be clear from the context (see also [20] for similar assumptions on pricing functions) which function should be applied. These functions are known in advance and are signed by the utility provider. In the sequel, we will also extend our results by considering more complex utility functions.

B. Definition of Data Transfer and Payment Scheme

The main operations of $\mathcal{DTPS}$ are listed below. These enable meters (through their owners) to register, evaluate utilities, submit electricity data, receive payments, and so on. For simplicity the terms 'owners of meters', 'users' and 'meters' will be used interchangeably when no conflict arises from the context.

- $\mathsf{Setup}(1^\kappa) \rightarrow \mathcal{CRS}$ is a probabilistic algorithm that on input a security parameter κ generates a public common reference string $\mathcal{CRS}$ to be used in the more advanced operations of the system.
- $\mathsf{UGen}(\mathcal{U}, \mathcal{CRS}) \rightarrow (pk_{\mathcal{U}}, sk_{\mathcal{U}})$ is the user's key generation algorithm which takes $\mathcal{CRS}$ as input and returns a public and private key pair $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$.
- $\mathsf{UPGen}(\mathcal{UP}, \mathcal{CRS}) \rightarrow (pk_{\mathcal{UP}}, sk_{\mathcal{UP}})$ is the utility provider's key generation algorithm which takes $\mathcal{CRS}$ as input and returns a public and private key pair $(pk_{\mathcal{UP}}, sk_{\mathcal{UP}})$.
- $\mathsf{Register}(\mathcal{U}, \mathcal{UP}) \rightarrow (pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}}, Cert_{\mathcal{U}})$ is a protocol executed between a user $\mathcal{U}$ and the provider $\mathcal{UP}$. The outcome of this protocol is a pair of *ephemeral* public/private keys which will be used by the owner $\mathcal{U}$ of the meter to submit meter data and obtain payment. Additionally, the algorithm outputs a certificate $Cert_{\mathcal{U}}$ that acts as an *authorization* credential allowing only authorized users to "sell" their data to the utility provider in a privacy-preserving manner.
- $\mathsf{EvalUtility}(\mathcal{U}, \{m_i\}) \rightarrow \langle \{c_i\}, val, r_{val}, \sigma \rangle$ is a protocol executed by an authorized user $\mathcal{U}$. The output is a set of commitments $\{c_i\}$ for the data $\{m_i\}$, the value val expected to be paid for the data after applying the utility function $Util()$, the sum of randomness r_{val} used in constructing the commitments, and a signature σ of the data signed with the ephemeral key $sk^e_{\mathcal{U}}$. The output values will be sent to the provider *offchain* through the use of an anonymizing network.
- $\mathsf{VfyUtility}(\{c\}_i, val, r_{val}, \sigma, pk^e_{\mathcal{U}}, Cert(h_e)) \rightarrow \{\bot, \top\}$ is a protocol executed by $\mathcal{UP}$ to check if the value val claimed by the user for the data committed in $\{c\}_i$ is correct or not. If it is correct, $Cert(h_e)$ is a valid certificate for the public key $pk^e_{\mathcal{U}}$, and σ is a valid signature then it returns $\top$. Otherwise, it returns $\bot$.
- $\mathsf{RedeemData}(\mathcal{U}, \mathcal{UP}) \rightarrow \{\bot, \top\}$ is a protocol executed between the user $\mathcal{U}$ and the provider $\mathcal{UP}$. At a high level, the owner encrypts the data m_i with an ephemeral key K and sends the encrypted data and the hash of the key to the provider using an anonymizing network. It also produces and sends a proof π that she knows the key K and that encrypted data match the output of operation $\mathsf{EvalUtility}$. The provider checks the proof and if the proof verifies, it posts a transaction to the Bitcoin network that pays val bitcoins to whoever presents a key K that matches the hash value sent before. This is the only place where an actual bitcoin transaction takes place. The algorithm returns $\top$ if all operations succeed, $\bot$ otherwise.

Our data transfer and payment scheme $\mathcal{DTPS}$ is the collection of the functions defined above and is denoted as $\mathcal{DTPS} = \{\mathsf{Setup}, \mathsf{UGen}, \mathsf{UPGen}, \mathsf{Register}, \mathsf{EvalUtility}, \mathsf{VfyUtility}, \mathsf{RedeemData}\}$.

C. Security model and assumptions

We propose a protocol to ensure meter privacy when meters report measurements to the $\mathcal{UP}$. We incorporate the use of bitcoin payments by the $\mathcal{UP}$ as the means to reward those users who share details about their energy consumption. This process should not leak any information about the identity of the meters/users and should not be used in any way by the $\mathcal{UP}$ to track users or profile them. Hence our protocol should guarantee the following properties.

- *Legitimacy*. Bitcoins can be collected only by owners of legitimate meters. Only authorized users should be able to submit detailed energy consumption data, however in a privacy-preserving manner. The utility provider should be able to verify the origin of this data as coming from a legitimate smart meter/owner. Thus payments may only be created in the name of and used by legitimate users.
- *Balance*. The value of the data to be redeemed must be balanced with the payment amount. This property is composed of two sub-properties, *user balance* and *provider balance*. Provider balance ensures that a malicious user cannot claim more money than the value of the data whereas user balance ensures that a malicious provider cannot pay less than the data's worth. This property captures the fairness aspect of the protocol and builds upon the atomicity (all-or-nothing) character of bitcoin transactions (Section III-D).

- *Unlinkability and context privacy*. An adversary cannot tell whether two data transfer and payment operations come from the same user even if the adversary obtains the views of these operations. Thus, payments should be unlinkable to each other and user transactions (apart from initial registration) should not reveal any information about the user or the meter.
- *No reliance on Trusted Third Parties*. Reliance on third parties requires trust in their honesty and competence. All our protocols involve only the users and the utility provider thus eliminating the possibility of collusion attacks. The payment mechanism takes advantage of the blockchain ledger as a *distributed* trusted entity that allows the two entities to perform data exchange and rewarding in a fair way.

While the data transfer and payment mechanism may be privacy preserving, other sources of information can be used to break user privacy. One such side channel is the IP address of a user; if this is visible in any phase of the protocol, it can be linked to the ID of the user during data redemption, hence, anonymity is lost. Thus, we require that any message transmitted to the $\mathcal{UP}$ is sent through an anonymous connection.

1) Provider Security Definitions: In the provider security experiments, we formalize an adversary $\mathcal{A}$ who plays the role of the user but may behave dishonestly and not follow the corresponding protocols. $\mathcal{A}$ may concurrently interact with an honest provider $\mathcal{UP}$ an arbitrary number of times. In order to formalize this adversarial setting, we define a number of oracles the adversary may query:

- Reg($\mathcal{U}$) lets $\mathcal{A}$ initiate the Register protocol with an honest $\mathcal{U}P$ provided that there is no pending or successful Reg call for $\mathcal{U}$ yet. Following a successful query, $\mathcal{A}$ will hold a certified ephemeral key pair $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}}, Cert_{\mathcal{U}})$.
- Redeem($\mathcal{U}$) lets $\mathcal{A}$ initiate the RedeemData protocol with an honest $\mathcal{U}P$.

Now we consider adversarial goals against the two properties, namely legitimacy and provider balance. A third property, *double-rewarding*, where a user tries to redeem the same data twice is not considered part of the threat model of $\mathcal{DPTS}$ since $\mathcal{DPTS}$ emphasizes more on the actual transfer and rewarding parts. However in the Discussion section we also explain how this attack can be prevented by assuming slightly more enhanced functionalities from smart meters.

The property of Legitimacy (Le), defined in Definition 1, is used to model adversaries who may succeed in holding a valid ephemeral key which is not signed by $\mathcal{U}P$, or in making a successful interaction of redeeming some data which are not bound with any $\mathcal{U}$ that was an input to a successful Reg call.

Definition 1: (Legitimacy) The data transfer and payment scheme holds the property of Legitimacy if for any PPT adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathsf{Le}}_{\mathcal{A}}(k)$ from Figure 2 the advantage of $\mathcal{A}$ defined below is negligible in k.

$$\mathsf{Adv}^{\mathsf{Le}}_{\mathcal{A}}(k) := \Pr[\mathsf{Exp}^{\mathsf{Le}}_{\mathcal{A}}(k) = 1]$$

The Provider Balance property (PBa), defined in Definition 2, is used to model adversaries who want to gain more than they actually deserve for the data they submitted to the provider. This property ensures that the amount redeemed by a user never exceeds the value of the data as captured by function $Util()$.

Experiment $\mathsf{Exp}^{\mathsf{Le}}_{\mathcal{A}}(k)$:
$\mathcal{CRS} \leftarrow \mathsf{Setup}(1^k)$
$(pk_{\mathcal{U}}, sk_{\mathcal{U}}) \leftarrow \mathsf{UGen}(\mathcal{CRS})$
$(pk_{\mathcal{UP}}, sk_{\mathcal{UP}}) \leftarrow \mathsf{UPGen}(\mathcal{CRS})$
$(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}}, Cert_{\mathcal{U}}) \leftarrow \mathsf{Register}(\mathcal{U}, \mathcal{UP})$
$b \leftarrow \mathcal{A}^{\mathsf{Reg},\,\mathsf{Redeem}}(pk^e_{\mathcal{U}}, pk_{\mathcal{UP}})$
The experiment returns 1 iff
1) $\mathcal{A}$ holds a valid (i.e. certified) key pair $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$ that is not an output from any Reg query; or
2) $\mathcal{A}$ makes a successful call to Redeem query such that the honest $\mathcal{U}P$ is convinced that the call involves a valid public-key $pk^e_{\mathcal{U}}$ for which there has not been a successful execution of Reg up to this call.

Fig. 2. Legitimacy experiment.

Experiment $\mathsf{Exp}^{\mathsf{PBa}}_{\mathcal{A}}(k)$:
$\mathcal{CRS} \leftarrow \mathsf{Setup}(1^k)$
$(pk_{\mathcal{UP}}, sk_{\mathcal{UP}}) \leftarrow \mathsf{UPGen}(\mathcal{CRS})$
$b \leftarrow \mathcal{A}^{\mathsf{Reg},\,\mathsf{Redeem}}(pk_{\mathcal{UP}})$
The experiment returns 1 iff
1) $\mathcal{A}$ managed to extract a valid authorization credential $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$ which can spend it by signing a new bitcoin transaction using $sk^e_{\mathcal{U}}$; or
2) $\mathcal{A}$ claims a payment that does not equal the value of previously submitted data with $pk^e_{\mathcal{U}}$.

Fig. 3. Provider balance experiment.

Definition 2: (Provider Balance) The data transfer and payment scheme holds the provider balance property if for any PPT adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathsf{PBa}}_{\mathcal{A}}(k)$ from Figure 3 the advantage of $\mathcal{A}$ defined below is negligible in k:

$$\mathsf{Adv}^{\mathsf{PBa}}_{\mathcal{A}}(k) := \Pr[\mathsf{Exp}^{\mathsf{PBa}}_{\mathcal{A}}(k) = 1]$$

2) User Security Definitions: In the user security definitions, we consider an adversarial utility provider $\mathcal{A}$ against two properties, namely user balance and user privacy. The adversary $\mathcal{A}$ will make use of the following oracle queries:

- Sys($\mathcal{U}P$) lets $\mathcal{A}$ initiate the system setup process and outputs the system parameters CRS and a public-private key pair for $\mathcal{U}P$, who is controlled by $\mathcal{A}$.
- RegU($\mathcal{U}$) lets $\mathcal{A}$ create a new user $\mathcal{U}$ with a Register protocol running between $\mathcal{A}$ and $\mathcal{U}$, and after a successful query $\mathcal{A}$ will obtain $\mathcal{U}$'s public key $pk_{\mathcal{U}}$ along with the transaction record from these two protocols.
- CorU($\mathcal{U}$) lets $\mathcal{A}$ interfere (corrupt) an honest user $\mathcal{U}$ and obtain $\mathcal{U}$'s secret key $sk_{\mathcal{U}}$.
- RedeemP($\mathcal{U}$) lets $\mathcal{A}$ initiate the EvalUtility and RedeemData protocols with an honest $\mathcal{U}$ and input $\{m_i\}$ of $\mathcal{A}$'s choice.

- RedeemP*($\mathcal{U}$) lets $\mathcal{A}$ initiate the EvalUtility and RedeemData protocols with an honest uncorrupted $\mathcal{U}$ where the messages are randomly selected between either $\{m_i^0\}$ or $\{m_i^1\}$, both chosen by $\mathcal{A}$.
- Challenge($\mathcal{U}_0, \mathcal{U}_1$) lets $\mathcal{A}$ initiate a RedeemData protocol by suggesting two honest users $\mathcal{U}_0$ and $\mathcal{U}_1$ that have not been corrupted by $\mathcal{A}$. The protocol is run between $\mathcal{U}_b$ for $b = \{0, 1\}$ and $\mathcal{A}$, where $\mathcal{A}$ acts as the provider $\mathcal{U}P$ making RegU and RedeemP queries for these two users.

We first consider the user balance property, used to model adversarial providers that desire to pay less than the value of the honest user's submitted data. This is formalised using an indistinguishability-based game where the adversarial provider has access to oracles that register and redeem payments with users. When the querying phase ends, the adversary will choose two challenge message sets $\{m_i^0\}$ and $\{m_i^1\}$ and the user will randomly choose to redeem one of them. Note that challenge messages must have equal total utility otherwise the adversary could trivially distinguish which one was chosen using VfyUtility. The adversary wins the game if it can extract the signing key of the user, submit a payment that does not match the utility of the submitted messages, or is able to distinguish between the two sets of messages, which would imply that the adversary has learned some information about the messages even before payment (e.g. the first few bits of the messages).

Definition 3: (User Balance) The data transfer and payment scheme holds the user balance property if for any PPT adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UBa}}(k)$ from Figure 4 the advantage of $\mathcal{A}$ is defined by:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UBa}}(k) := \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UBa}}(k) = 1] = 1/2 + \epsilon$$

where ϵ is negligible in k.

The second property related to user security that we consider is user privacy, in which the adversarial provider seeks to identify a user trying to submit data and receive payments. With the exception of the registration phase, the data transfer and payment scheme should not leak any user-sensitive information. In short, users should have the privacy guarantee that protocol interactions are unlinkable to each other and cannot be used for tracking an honest user.

The property of User Privacy (Priv), defined in Definition 4, is formalised by the indistinguishability game shown in Figure 5. Initially, the adversary asks an arbitrary number of users to register and optionally redeem some data with bitcoin payments. Once this learning phase is over, $\mathcal{A}$ initiates a Challenge phase with two users $\mathcal{U}_0$ and $\mathcal{U}_1$ at $\mathcal{A}$'s choice, in which a user $\mathcal{U}_b$ from these two is selected according to a random bit b unknown to $\mathcal{A}$ and asked to redeem data $\{m_i\}$. Then, $\mathcal{A}$ outputs the b' value. The scheme will be privacy-preserving if the adversary is unable to identify the bit b (i.e., $b' = b$) with probability better than random guessing. Intuitively, the adversary should not be able to link users to the payments they received under the condition that all other actions (registration, redeem, etc.) are controlled by $\mathcal{A}$.

Experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UBa}}(k)$:

$\mathcal{CRS} \leftarrow \mathsf{Setup}(1^k)$
$b \xleftarrow{\$} \{0,1\}$
$(pk_{\mathcal{U}P}, sk_{\mathcal{U}\mathcal{P}}) \leftarrow \mathcal{A}^{\mathsf{Sys}}(1^k)$
$(\mathcal{U}^*, \{m_i^0\}, \{m_i^1\}) \leftarrow \mathcal{A}^{\mathsf{RegU,\ CorU,\ RedeemP}}(pk_{\mathcal{U}P})$
such that $\sum_i Util(m_i^0) = \sum_i Util(m_i^1)$
$\mathsf{transRecord}(\mathcal{U}^*) \leftarrow \mathcal{A}^{\mathsf{RedeemP}}(\mathcal{U}^*, \{m_i^0\}, \{m_i^1\})$

The experiment returns 1 iff

1) $\mathcal{A}$ managed to extract a valid credential $(pk_{\mathcal{U}}^e, sk_{\mathcal{U}}^e)$ which can then use to refund back its payment by signing a new bitcoin transaction using $sk_{\mathcal{U}}^e$; or
2) $\mathcal{A}$ submits a payment that does not equal the value of the data submitted by $\mathcal{U}^*$; or
3) $\mathcal{A}$ outputs $b' = b$.

Fig. 4. User balance experiment.

Experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(k)$:

$\mathcal{CRS} \leftarrow \mathsf{Setup}(1^k)$
$b \leftarrow \mathcal{A}^{\mathsf{Sys,\ RegU,\ CorU,\ RedeemP,\ Challenge}}(1^k)$

The experiment returns 1 iff $\mathcal{A}$ passes the following phases:

- *Setup phase*: $(pk_{\mathcal{U}P}, sk_{\mathcal{U}\mathcal{P}}) \leftarrow \mathcal{A}^{\mathsf{Sys}}(1^k)$
- *Learning phase*: $\mathsf{transRecord} \leftarrow \mathcal{A}^{\mathsf{RegU,\ CorU,\ RedeemP}}(pk_{\mathcal{U}P})$
- *Challenge phase*: $\mathsf{transRecord}(\mathcal{U}_b) \leftarrow \mathcal{A}^{\mathsf{Challenge}}(\mathcal{U}_0, \mathcal{U}_1)$

Finally, $\mathcal{A}$ outputs $\mathcal{U}_{b'}$ that is equal to $\mathcal{U}_b$.

Fig. 5. Privacy experiment.

Definition 4: (Privacy) The data transfer and payment scheme holds the property of meter privacy if for any PPT adversary $\mathcal{A}$ in the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(k)$ from Figure 5 the advantage of $\mathcal{A}$ is defined by

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Priv}}(k) := Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(k) = 1] = 1/2 + \epsilon$$

where ϵ is negligible in k.

D. Tools

Throughout this work, we say that a function $\epsilon(.)$ is negligible (denoted $negl$), if $\epsilon(\kappa) < \kappa^{-c}$ for all $c > 0$ and sufficiently large κ.

1) Digital signatures: A digital signature scheme consists of a tuple of probabilistic polynomial time (PPT) algorithms (Keygen, Sig, SigVerify) defined as follows:

- $\mathsf{Keygen}(1^\kappa)$: a PPT algorithm that, on input the security parameter κ, outputs a key pair (pk, sk) where sk is the signing key and pk is the signature verification key.
- $\mathsf{Sig}(sk, m)$: a PPT algorithm that, on input a signing key sk and a message m, returns a signature σ. We write $\mathsf{Sig}_{\mathcal{U}}(m)$ to denote the signing of m using the secret key $sk_{\mathcal{U}}$ of $\mathcal{U}$.
- $\mathsf{SigVerify}(pk, m, \sigma)$: an algorithm that, on input a verification key pk, message m and a signature σ, outputs either True or False. We write $\mathsf{SigVerify}_{\mathcal{U}}(m, \sigma)$ to

denote the verification of (m, σ) using the public key $pk_{\mathcal{U}}$ of $\mathcal{U}$.

The correctness and security properties are informally summarized below:

- *Completeness*. With overwhelming probability, for any message m and signature $\sigma \leftarrow \mathsf{Sig}(sk, m)$, it holds that $\mathsf{SigVerify}(pk, m, \sigma) = \text{True}$, where $(pk, sk) \leftarrow \mathsf{Keygen}(1^\kappa)$.
- *Unforgeability*. Let A be a polynomial-time attacker that adaptively chooses $t = poly(\kappa)$ distinct messages $m_1, ..., m_t$ and obtains signatures $\sigma_i = \mathsf{Sig}(sk, m_i)$ for all $i \in [t]$. Then with only negligible probability can A forge a new message-signature (m^*, σ^*) such that $m^* \neq m_i$ and $\mathsf{SigVerify}(pk, m^*, \sigma^*) = \text{True}$.

2) Blind signatures: A blind signature scheme [21] is a digital signature scheme where the Sig algorithm is replaced with the following:

- $\mathsf{BSig}(pk, sk, m)$: an interactive protocol between a PPT signer $S(sk)$ with a signing key sk and a PPT verifier $V(pk, m)$ with a verification key vk and a message m. The output is a signature σ. For simplicity of notation, when it is clear from context, pk is omitted from the set of inputs.

Furthermore, blind signatures should also possess the following security property (in addition to unforgeability):

- *Blindness*. Let A be a polynomial-time adversarial signer A that interacts with an honest verifier in BSig. Furthermore, let $\sigma_b \leftarrow \mathsf{BSig}(sk, m_b)$ and $\sigma_{1-b} \leftarrow \mathsf{BSig}(sk, m_{1-b})$ be the outputs of two different interactions of A with the honest verifier where $b \xleftarrow{\$} \{0, 1\}$ and (m_0, m_1) are chosen by A. Then $A(\sigma_0, \sigma_1)$ can guess b with probability at most $1/2 + negl(\kappa)$.

In this work, we will implement the standard RSA-based blind signature [21] within our protocol, which is described in more detail in Section IV-A and was proven to possess both blindness and unforgeability in the random oracle model.

3) Pedersen commitments: A commitment scheme is a pair of algorithms $(\mathsf{Commit}, \mathsf{Open})$ typically executed between a committer and a receiver. In the commitment phase, the committer uses algorithm Commit, which takes as input a message m and auxiliary information (an unpredictable random number) r_m, to produce a commitment $c_m = \mathsf{Commit}(m, r_m)$. In the opening phase, it sends (m, r_m) to the receiver which checks whether the opening algorithm $\mathsf{Open}(c_m, m, r_m)$ returns $Accept$. A commitment scheme is secure if it is binding and hiding. The "hiding" property ensures that the receiver has no information about m before the opening phase, while the "binding" property ensures that, once committed, a malicious committer cannot find values $m' \neq m$ and r' such that $\mathsf{Open}(c_m, m', r') = Accept$.

We will be using the Pedersen commitment scheme [22]. To commit to a message m using randomizer r_m, we compute $c_m = \mathsf{Commit}(m, r_m) = g^m h^{r_m}$ where g, h are generators of G. To open the commitment, we reveal (m, r_m) and check whether $g^m h^{r_m} \stackrel{?}{=} c_m$. A useful property of this commitment scheme is that it is *homomorphic*: for all (m, r) and (m', r') we have $\mathsf{Commit}(m, r) \times \mathsf{Commit}(m', r') = \mathsf{Commit}(m + m', r + r')$.

4) QAPs and zkSNARKs: We will base our constructions on a class of zero-knowledge Succinct Non-interactive ARguments of Knowledge (*zkSNARKs*) that was introduced in [23]. Such arguments can be used to prove NP statements about Quadratics Arithmetic Programs (QAPs) without revealing anything about the corresponding witnesses. After taking a QAP Q as input, a trusted party conducts a one-time setup that results in two public keys: an evaluation key EK_Q and an verification key EV_Q. The evaluation key allows an untrusted prover to produce a proof π regarding the validity of the QAP NP statement. The non-interactive proof is a zero knowledge proof of knowledge, thus anyone can use the verification key to verify the proof π. In our setting, the use of *zkSNARKs* will be used to guarantee that data to be released by the user have certain properties.

A *zkSNARK* for a QAP Q is a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{KeyGen}(Q, 1^\kappa) \rightarrow (EK_Q, VK_Q)$. On input a security parameter 1^κ and a QAP Q, this function produces a public evaluation key EK_Q and a public verification key VK_Q.
- $\mathsf{Prove}(EK_Q, x, w) \rightarrow \pi_Q$. On input a public evaluation key EK_Q, a $x \in L_Q$, where L_Q is the NP decision language defined by the QAP, and a corresponding witness w, this function produces a proof π_Q that w is a valid witness for x.
- $\mathsf{Verify}(VK_Q, x, \pi_Q) \rightarrow \{\bot, \top\}$. On input a public verification key VK_Q, x and a proof π_Q, this function outputs $\top$ if it is convinced that $x \in L_Q$ and $\bot$ otherwise.

The properties expected by *zkSNARKs* are informally summarized below:

- *Completeness*. Given $(x, w) \in R_Q$, where R_Q is the NP relation for the language L_Q, the prover P can produce a proof π such that the verifier V accepts (x, π) with probability 1.
- *Soundness*. No polynomial-time adversary can generate a proof π for $x \in L_Q$ that fools the verifier V to accept (x, π).
- *Zero-knowledge*. There exists a (randomized) polynomial simulator S, such that for any $x \in L_Q$, $S(x)$ generates a proof that is computationally indistinguishable from a honestly generated one.

We say a *zkSNARK* is secure if all the above properties hold.

On trusted setup and CRS: While standard *zkSNARKs* as defined here require a trusted party to generate the common reference string (CRS) for the generation and the verification of the proofs, a malicious party can provide a CRS that allows it to break the ZK property and learn information about the user's data. This attack can be prevented by extending *SNARKs* to be "subversion-resilient" [24], [25], [26], which would ensure that the ZK property is preserved even under maliciously chosen CRS. Note, however, that only standard knowledge soundness is guaranteed (i.e. not security against malicious provers) but this is sufficient to ensure soundness

and ZK security if the verifier is responsible for generating and distributing the CRS. Other alternatives to this approach include building NIZKs in the random oracle model (using the Fiat-Shamir transform [27]), and constructing the CRS using a multiparty computation (MPC) protocol initiated between the prover and verifier [28] which preserves soundness if at least one party is honest.

5) Bitcoin transactions: We assume limited familiarity with Bitcoin and blockchains, for more information the reader is referred to [29]. A blockchain is a linked-list data structure in which data is organized as blocks, and blocks are connected together through hash pointers (pointer to the hash value of the previous block) to form a chain. Maintaining a hash pointer instead of a simple pointer turns the blockchain into an append-only data structure. The process of extending the blockchain is called *mining*. Miners compete against each other to extend the blockchain with new blocks containing valid Bitcoin transactions.

The Bitcoin system consists of *addresses* and *transactions* that send/receive money to these addresses. An address is simply the hash of a public key. To transfer bitcoins from one address to another, a transaction must be created with one or more input addresses from which the money will be taken and one or more output addresses to which the money will be sent. Each input must contain a proof of fulfillment of the established conditions of the output it tries to redeem from. Both conditions and proofs are coded using Script, a simple stack-based language with no loops. In order to validate a transaction, the full script is executed by concatenating both locking and unlocking conditions.

The most common transaction within Bitcoin is the standard *Pay-To-Public-Key-Hash* where a digital signature is needed to redeem the transaction; the output specifies an address and in order to spend this output, one must sign with the associated private key. The script of this transaction is shown below:

`ScriptPubKey:`	`OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`
`ScriptSig:`	`<sig><pubKey>`

A more advanced transaction that was not originally included in the Bitcoin scripting language is *Pay-to-Script-Hash* (P2SH) which is shown below:

`ScriptPubKey:`	`OP_HASH160 <redeemScriptHash> OP_EQUAL`
`ScriptSig:`	`<sig><pubKey><redeemScript>`

To redeem an output sent to a P2SH address, one must specify a script that *hashes* to this address, and then meet the conditions specified in the script. We will be using this transaction to trade meter data with bitcoins of appropriate value. At a high level, the owner of the meter will sent *offchain* the encrypted data along with a hash of the key used to encrypt the data. The provider will include the hash of the key in a P2HS transaction which will be traded with bitcoins when the user releases the key used to encrypt the data. However, before posting the transaction, the provider must be convinced about the validity of the data. The details of the validation procedure and the message exchanges which occur offchain is the main part of this work and will be described in the next section.

Finally, to ensure that the bitcoins of the provider are not locked forever if the user does not release the unlocking key, the provider can make the transaction a *time-locked* one by using the `CheckLockTimeVerify` or `CheckSequenceVerify` opcodes. In a time-locked transaction, outputs require a certain time in the future to be reached in order to be redeemed. Using the previous two opcodes, spending time can be absolute (i.e. a specific future date) or relative to the time the transaction was posted in the blockchain, respectively. An example of relative time lock (10 days) is shown below:

`ScriptPubKey:`	`<10d> OP_CHECKLOCKTIMEVERIFY OP_DROP`
`ScriptSig:`	`<sig>`

In the sections that follow we describe our protocols for reporting and rewarding in the smart grid through the use of bitcoin transactions. Our goal would be to ensure the security and privacy of these transactions against both external (any entity eavesdropping on data communications) and internal attackers ($\mathcal{UP}$ and perhaps other smart meters) that may have access to more detailed transaction data.

IV. Detailed description of $\mathcal{DPTS}$

Consider a user $\mathcal{U}$ who is the owner of a meter $\mathcal{M}$. The user has detailed electricity data m worthy of value v that is of interest to a provider $\mathcal{UP}$. $\mathcal{U}$ would like to exchange the data in a fair way with bitcoins of value v. For this reason the blockchain can be used as a trusted entity in the place of a judge in a fair exchange protocol.

The protocol consists of two phases. In the *offchain* one, the user produces a proof of validity that meter data has value v based on some pre-agreed utility function $Util()$. Since the data m need to be kept private at this point, they are encrypted under a key K known only to the user. Hence the user must convince $\mathcal{UP}$ that the encrypted data is of value v. Since the latter is basically an NP statement, it can be proven in zero knowledge. Hence the most important issue is how to efficiently implement this ZK proof in a non-interactive manner. Once the proof π is constructed, the user sends to the $\mathcal{UP}$ the proof π, the encrypted data and a hash h_K of the encryption key K.

When the $\mathcal{UP}$ verifies the proof, the *onchain* phase begins; $\mathcal{UP}$ posts a transaction to the blockchain that says

> "Transfer v bitcoins to the user who presents a pre-image of h_K within time T and signs the transaction with private key sk_U."

Then the user posts a transaction that says

> "Here is K that satisfies $H(K) = h_K$. Transfer the v bitcoins to my address."

Notice that none of the parties can cheat the other. The $\mathcal{UP}$ is sure that nobody can claim the money unless they present a key K that matches the key used to encrypt the data in the proof π. If this does not happen within time T, the $\mathcal{UP}$ can get back the v bitcoins (recall time-locked transactions). Without

this refund condition, in the event that $\mathcal{U}$ decides to abort the protocol, $\mathcal{UP}$'s money would be locked forever.

Similarly, the user can claim the money by signing the transaction that publishes K with the private key $sk_{\mathcal{U}}$, which in turn will allow $\mathcal{UP}$ to recover the encrypted meter data m. Note that $\mathcal{UP}$'s transaction requires that $\mathcal{U}$ provides both the pre-image K as well as a signature. This is to prevent an attack in which $\mathcal{U}$ broadcasts K to the network and a malicious miner (or any other party) sees K and uses it to claim $\mathcal{UP}$'s funds. To prevent this attack, the transaction requires $\mathcal{U}$'s signature, which nobody else can produce. However to ensure unlinkability and prevent the $\mathcal{UP}$ associating the signing key with multiple submissions of data from $\mathcal{U}$, this key has to be an *anonymous and ephemeral* (yet *authorized*) key as will be explained in Section IV-A.

The above approach makes the combined transaction "atomic" in the sense that payment and delivery of data take place at the same time. Thus, if any of the entities leaves the protocol and does not complete its part, no losses incur. More specifically, this atomic exchange ensures that *fairness* holds (see [13], [14] for similar assumptions). Hence we will build upon this to ensure the stronger guarantees offered by our Balance property (Definition 2). Finally, only one blockchain transaction is needed to exchange the data with bitcoins while the majority of the protocol takes place *offchain*. This greatly contributes to the efficiency of the whole process. In the remaining of this paper, we mostly focus on the offchain part of the protocol.

A. Initialization and Registration

In what follows, we assume that the owner $\mathcal{U}$ of a smart meter $\mathcal{M}$ has a public-private keypair $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$. The public key is registered with the provider along with substantial evidence (e.g. electricity bill, civil ID) that $\mathcal{U}$ is a valid owner. Thus when the provider sees a message signed with $sk_{\mathcal{U}}$, he knows it is coming from a legitimate user. This long term key-pair will be used to establish *ephemeral*, one-time keys $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$ to ensure the unlinkability between reported data and rewards.

Essentially, the hash of $pk^e_{\mathcal{U}}$ will correspond to a bitcoin address which will be used with the $\mathcal{UP}$ to reward the meter with bitcoins corresponding to the data provided. This one-time key can also be thought as an *authorization credential* that allows only authorized meters to subsequently submit detailed consumption reports in a privacy-respecting manner, without the need for additional, explicit authentication. This key will be *updated* with every new data reported, thus also providing for unlinkability between rewards.

A description of the initialization phase is shown in Algorithm 1. For simplicity we assume the provider possesses an RSA key pair, although any *blind* signature scheme can be used instead. So, let $N_{\mathcal{UP}}$ be an RSA modulus and $(e_{\mathcal{UP}}, d_{\mathcal{UP}})$ be the public/private key pair of the provider. To obtain a blind signature from $\mathcal{UP}$, the meter first asks the utility provider to sign a blinded version of a new bitcoin address h_e (Steps 1–3). The utility provider first checks the authenticity of $\mathcal{U}$'s signature and verifies it is coming from a *legitimate* user (Step 4–7). This is possible since $\mathcal{UP}$ is aware of the public keys of all valid owners. Then $\mathcal{UP}$ signs the blinded version of h_e and gives it back to the user which removes the blinding factor r, thus obtaining the provider's signature on h_e (Step 8). At this point the user can submit consumption data later on.

Algorithm 1 *Register*: Executed between meter $\mathcal{U}$ and $\mathcal{UP}$ to generate ephemeral keys certified by $\mathcal{UP}$

Output: The hash of ephemeral public key $pk^e_{\mathcal{U}}$ signed by the $\mathcal{UP}$ and known only to $\mathcal{U}$.

1: $\mathcal{U}$ generates a fresh key pair $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$ and sets $h_e = \mathsf{Hash}(pk^e_{\mathcal{U}})$ as the bitcoin address corresponding to $pk^e_{\mathcal{U}}$.
2: $\mathcal{U}$ blinds h_e by computing $h^* = r^{e_{\mathcal{UP}}} h_e \mod N_{\mathcal{UP}}$, where r is a fresh random number.
3: $\mathcal{U} \rightarrow \mathcal{UP} : h^*$ and $\mathsf{Sig}_{\mathcal{U}}(h^*)$.
4: **if** $\mathsf{SigVerify}_{\mathcal{UP}}(h^*, \mathsf{Sig}_{\mathcal{U}}(h^*)) = False$ **then**
5: $\quad \mathcal{UP}$ aborts ▷ Bad Signature
6: **else**
7: $\quad \mathcal{UP} \rightarrow \mathcal{U} : \sigma^* = \mathsf{Sig}_{\mathcal{UP}}(h^*) = (h^*)^{d_{\mathcal{UP}}} \mod N_{\mathcal{UP}}$
8: $\mathcal{U}$ computes $\sigma = r^{-1}\sigma^* \mod N_{\mathcal{UP}}$ and recovers $\mathcal{UP}$'s signature on h_e.

In the end of this protocol, the user is equipped with an authorized ephemeral public key $pk^e_{\mathcal{U}}$ and a certificate $Cert(h_e)$ bearing the signature of $\mathcal{UP}$ on the hash of $pk^e_{\mathcal{U}}$. Although at the moment of signing the utility provider is not aware of the value $pk^e_{\mathcal{U}}$ (or its hash h_e), when presented with such a bitcoin address during the data reporting phase it will be convinced it is coming from an authorized user. Hence this method provides for a way to certify ephemeral bitcoin public keys/addresses.

Notice that this protocol needs to be executed when the meter has to register for the first time or when the meter needs to obtain a new ephemeral key. However, subsequent keys can easily be obtained by piggypagging this process with the data reporting phase as explained in Section IV-D.

B. Commitment of data and Utility evaluation

The goal of this phase is for the user to convince the utility provider that has measurements to be reported, however without revealing these first. On the other hand, $\mathcal{UP}$ must be sure that this data worths some value val under the utility function $Util()$. The use of commitments will help us realize these two requirements. Then these consumption data need to be exchanged with bitcoins of value equal to val in fair and trustworthy manner.

Let $m_1, \ldots, m_n$ be the measurements to be reported to the $\mathcal{UP}$. For each measurement, the user evaluates $Util()$ on input m_i to get a value $v_i = a_1 \cdot m_i + a_0$, along with commitments c_i to the measurements (here we exemplify the construction using a *linear* utility function, however other more complicated functions can be supported as well). The user adds these values to obtain the total value $v = \sum_i v_i$. This is detailed in Algorithm 2. The output of this algorithm is a message $\langle \{c_i\}, val, r_{val} \rangle$, signed with the user's ephemeral key $sk^e_{\mathcal{U}}$. The commitments c_i will be used by the utility provider

(or any other third party) to verify the value of the data as explained below.

Algorithm 2 *Evaluate utility and commit to measurements*: Executed by $\mathcal{U}$

Input: $Util()$ function and measurements m_i to be reported to the $\mathcal{UP}$. Generators g, h used in commitment scheme. Ephemeral signing key $sk^e_{\mathcal{U}}$.
Output: Signed message consisting of total value and commitments to data.

1: **for all** measurements m_i **do**
2: $v_i \leftarrow a_1 \cdot m_i + a_0$ ▷ Value of data m_i
3: $r_i \leftarrow_R \{0,1\}^{l_G}$
4: $c_i \leftarrow \mathsf{Commit}(m_i, r_i) = g^{m_i} h^{r_i}$
5: **end for**
6: $val \leftarrow \sum_i v_i$ ▷ Total value for all data
7: $r_{val} \leftarrow a_1 \cdot \sum_i r_i$ ▷ Auxiliary information to be used in verifying commitment to total value
8: $\sigma \leftarrow \mathsf{Sig}_{sk^e_{\mathcal{U}}}(\{c_i\}, val, r_{val})$
9: **Return** $\langle \{c_i\}, val, r_{val}, \sigma \rangle$

The signed message $\langle \{c_i\}, val, r_{val} \rangle$ will be sent to $\mathcal{UP}$ *offchain* (i.e. not using the blockchain network), so that $\mathcal{UP}$ knows that a user has to report data worthy of value val. Now $\mathcal{UP}$ has to be convinced about the validity of the commitments and the value of the data. This is achieved using $\mathsf{VfyUtility}(\{c_i\}, val, r_{val}, \sigma)$, which is outlined in Algorithm 3.

Algorithm 3 *Verify utility*: Executed by $\mathcal{UP}$

Input: $Util()$ function, $\langle \{c_i\}, val, r_{val} \rangle$, signature σ and certified public key $\langle pk^e_{\mathcal{U}}, Cert(h_e) \rangle$.
Output: 'Accept' or 'Not Accept' depending on the validity of data.

1: **if** $\mathsf{SigVerify}(H(pk^e_{\mathcal{U}}), Cert(h_e)) = False$ **then**
2: $\mathcal{UP}$ aborts ▷ Invalid $Cert(h_e)$. Key signature not verified by $\mathcal{UP}$
3: **if** $\mathsf{SigVerify}(\langle \{c_i\}, val, r_{val} \rangle, \sigma) = False$ **then**
4: $\mathcal{UP}$ aborts ▷ Bad user signature
5: **Return** $\mathsf{Open}(\Pi_i (c_i^{a_1} \times g^{a_0}), val, r_{val})$

If Algorithm 3 returns 'Accept', $\mathcal{UP}$ is convinced about the value of the data reported by the user. To see why, let $c_{v_i} = c_i^{a_1} \times g^{a_0} = g^{a_1 \cdot m_i + a_0} h^{a_1 \cdot r_{m_i}}$ and $c_{val} = \Pi_i c_{v_i}$. By substitution,

$$c_{val} = g^{\sum_i (a_1 \cdot m_i + a_0)} h^{a_1 \sum_i r_i} = g^{val} h^{r_{val}}, \qquad (1)$$

where $val = \sum_i (a_1 \cdot m_i + a_0)$ and $r_{val} = a_1 \cdot \sum_i r_i$. Hence c_{val} is a valid commitment on the pair (val, r_{val}) and Open in Line 5 will return 'Accept'.

C. Getting paid for the data

In this last phase, the Redeem protocol is executed between the user and the $\mathcal{UP}$. First, the data m_i are encrypted with a fresh, one-time symmetric key K chosen by the user to produce a ciphertext $C = E_K(\{m_i\})$. Then C is encrypted with the provider's public key to produce $Enc_{\mathcal{UP}}(C)$. This double encryption is needed to prevent others from accessing the data in C, once the key K is released to the blockchain. Then both $Enc_{\mathcal{UP}}(C)$ and the hash h_K of K are sent to the $\mathcal{UP}$ through an anonymizing network along with a proof π that proves knowledge of K and that data m_i match the data in the commitments c_i (output of operation $\mathsf{EvalUtility}$).

So, given $\mathcal{UP}$'s knowledge of $\langle C, h_K, \{c_i\} \rangle$, the user must prove in ZK that it knows $(m_1, \ldots, m_n, r_1, \ldots, r_n, K)$ such that:

1) $H(K) =^? h_K$ and K was used for the encryption of the data m_i
2) $c_i =^? g^{m_i} h^{r_i}$

More formally, if we let $U := U_{a_1,a_0}(m) = a_1 m + a_0$ be the linear utility function, we define the NP language $\mathcal{L}_U$ for the *zkSNARK*-proof system as a set of the following NP statements:

$$\mathcal{L}_U = \left\{ \langle C, h_K, \{c_i\} \rangle \;\middle|\; \begin{array}{l} \exists\, \{m_i\}_{i=1}^n, \{r_i\}_{i=1}^n, K : \\ h_K = H(K), \\ C = E_K(\{m_i\}) \\ \forall i \in [n], c_i = g^{m_i} h^{r_i} \end{array} \right\}$$

If the above *zkSNARK*-proof π is correct, $\mathcal{UP}$ is convinced about the value of the committed data. It can now post a transaction to the Bitcoin network that pays *val* bitcoins to whoever presents a key K that matches the hash value sent before. This is the only place where an actual bitcoin transaction takes place.

This last part ensures the fair transfer of the data m_i (actually the key K that decrypts the ciphertext C) with bitcoins of value val. This is straightforward and can be implemented using hash-locked transactions and time commitments (Section III-D5), once the *zkSNARK* is implemented successfully. The implementation of π is described in Section V-B. Operation Redeem is outlined in Algorithm 4.

Algorithm 4 *Redeem*: Executed between $\mathcal{U}$ and $\mathcal{UP}$

$\mathcal{U}$
1: $C = E_K(\{m_i\})$, where $E()$ is a secure block cipher and K a fresh symmetric encryption key.
2: Create proof π.
3: $\mathcal{U} \rightarrow \mathcal{UP}$: $Enc_{\mathcal{UP}}(C), H(K), \pi$
$\mathcal{UP}$:
4: **if** $\perp \leftarrow \mathsf{VfyUtility}(\{c_i\}, val, r_{val}, \sigma, \langle pk^e_{\mathcal{U}}, Cert(h_e) \rangle)$ **then**
5: $\mathcal{UP}$ aborts ▷ Utility value does not verify
6: **if** $\perp \leftarrow \mathsf{Verify}(\mathcal{CRS}, C, h_K, \{c_i\}, \pi)$ **then**
7: $\mathcal{UP}$ aborts ▷ Bad *zkSNARK* proof
8: **else**
9: $\mathcal{UP}$ posts time-locked transaction $T_{\mathcal{UP}}$ to transfer val bitcoins to user who presents K
$\mathcal{U}$:
10: **if** $T_{\mathcal{UP}}$ not posted or posted with incorrect val **then**
11: $\mathcal{U}$ aborts
12: **else**
13: $\mathcal{U}$ posts transaction $T_{\mathcal{U}}$ to release K and get credit of val bitcoins

A summary of the information exchanged between the user and the $\mathcal{UP}$ is shown in Figure 6. As it can be seen in the figure, only one message needs to be sent from the user to the $\mathcal{UP}$ contributing to the efficieny of the protocol.

D. Obtaining a new ephemeral key

We mentioned in the registration phase that the ephemeral key is blindly signed by the $\mathcal{UP}$. Hence the first key of the user $\mathcal{U}$ is authenticated but the provider cannot link it to the user's ID. As it is important for different data submissions/payments to remain unlinkable, once this key has been used to receive a payment, it should not be used again.

One way to do this is to have the provider blindly sign more than one key during registration. Thus, instead of $\mathcal{U}$ sending just one blind hash value (Line 3 of Algorithm 1), the user may send a collection $\{h_1^*, h_2^*, \ldots\}$ of them that can individually be signed by the $\mathcal{UP}$. After unblinding those, $\mathcal{U}$ will have a collection of keys to be used in subsequent interactions with the $\mathcal{UP}$.

If the provider is not willing to sign a collection of such keys, a new ephemeral key can be *piggybacked* on Message 1 of Figure 6 as follows. The user picks a new ephemeral key $pk^{e'}$, computes its hash $h_{e'}$ and sends a blinded version $r'^{e_{\mathcal{P}}} h_{e'}$, signed with the *previous* ephemeral key, along with the rest of the components in the message (recall signature σ). The provider knows this is coming from an authenticated user but cannot tell which one due to the security of the blind signature. Once the provider signs $r'^{e_{\mathcal{P}}} h_{e'}$ with its RSA key, the user can remove the blinding factor and obtain a new ephemeral key to be used for the next submission of data. This way, a series of signed keys can be generated on the fly which are all unlinkable to each other.

E. Other utility functions

The case for linear utilities can be extended to piecewise linear functions defined over various intervals. For example, if the data m falls in the range $[l_i, r_i)$, the utility might given by some linear function $a_1^i(m) + a_0^i$ defined in this specific interval. Further generalizations are possible by considering polynomial utilities that can be used to express arbitrary functions as general splines over intervals, at the expense of a more complex commitment scheme and ZK proof.

In the following we will demonstrate how *arbitrary* computable utilities can be incorporated in the scheme. Since homomorphism in the commitment cannot be exploited any more, Pedersen commitments will be replaced with simpler hash commitments. Additionally, VfyUtility (Algorithm 3) degenerates to just checking the signatures on the data and the ephemeral key (lines 1-4). Verification of the commitment will have to be pushed *inside* the ZK proof. The updated version of EvalUtility is highlighted in Algorithm 5.

Operation Redeem stays the same. First the data m_i are encrypted with an one-time key K to produce a ciphertext $C = E(\{m_i\})$. Then the hash of the key and $Enc_{\mathcal{UP}}(C)$ are sent offchain to the $\mathcal{UP}$ along with a new proof π showing that data match the data in the commitment com and that the utility function was correctly applied. The new *zkSNARK* proof

Algorithm 5 *Evaluate utility and commit to measurements - updated*: Executed by $\mathcal{U}$

Input: $Util()$ function and measurements m_i to be reported to the $\mathcal{UP}$. Secure hash function H. Certified ephemeral signing key $sk_{\mathcal{U}}^e$.
Output: Signed message consisting of total value and commitment to data.

1: **for all** measurements m_i **do**
2: $v_i \leftarrow Util(m_i)$ ▷ Value of data m_i
3: $r \leftarrow_R \{0,1\}^{l_G}$
4: $com \leftarrow H(m_1, \ldots, m_n, r)$ ▷ Commitment to data
5: $val \leftarrow \sum_i v_i$ ▷ Total value for all data
6: $\sigma \leftarrow \mathsf{Sig}_{sk_{\mathcal{U}}^e}(com, val)$
7: **Return** $\langle com, val, \sigma \rangle$

is using information $\langle C, h_K, com, val \rangle$ to show that the user knows $m_1, \ldots, m_n, K, r$ such that:

1) $H(K) =^? h_K$ and K was used for the encryption of the data m_i
2) $com =^? H(m_1, \ldots, m_n, r)$
3) $val =^? \sum_i Util(m_i)$

More formally, if we let U_N be an arbitrary utility function, we define the NP language $\mathcal{L}_{U_N}$ for the *zkSNARK*-proof system as a set of the following NP statements:

$$\mathcal{L}_{U_N} = \left\{ \langle C, h_K, com, val \rangle \;\middle|\; \begin{array}{l} \exists\, \{m_i\}_{i=1}^n, r, K : \\ h_K = H(K) \\ C = E_K(\{m_i\}) \\ com = H(m_1, ..., m_n, r) \\ val = \sum_i U_N(m_i) \end{array} \right\}$$

V. Security Analysis and Performance

In this section we analyze the security, privacy and efficiency aspects of $\mathcal{DPTS}$.

A. System security

In this section, we prove that the $\mathcal{DPTS}$ scheme satisfies the three properties of legitimacy, balance, and privacy as per the definitions of system security introduced in Section III-C.

a) Legitimacy: We start by providing an intuition behind why the legitimacy property holds. Recall that by Definition 1, the adversary $\mathcal{A}$ can win the legitimacy game in two cases:

1) $\mathcal{A}$ holds a certified ephemeral key that is not an output from any Reg query which models the Register protocol. In this protocol, the provider $\mathcal{UP}$ authenticates the user $\mathcal{U}$ and then signs the blinded ephemeral key. $\mathcal{A}$ holding a valid ephemeral key without calling the Reg query means that $\mathcal{A}$ can either demonstrate to $\mathcal{UP}$ that $\mathcal{A}$ is the owner of a signed blinded ephemeral key i.e., knowing the secret key of $\mathcal{U}$, or simply forge a certificate without the involvement of $\mathcal{UP}$. Both conditions contradict the assumption that the signature schemes used in our proposed scheme are unforgeable.
2) $\mathcal{A}$ makes a successful call to Redeem query such that the honest $\mathcal{UP}$ is convinced that the call involves a valid ephemeral key for which there has not been a successful

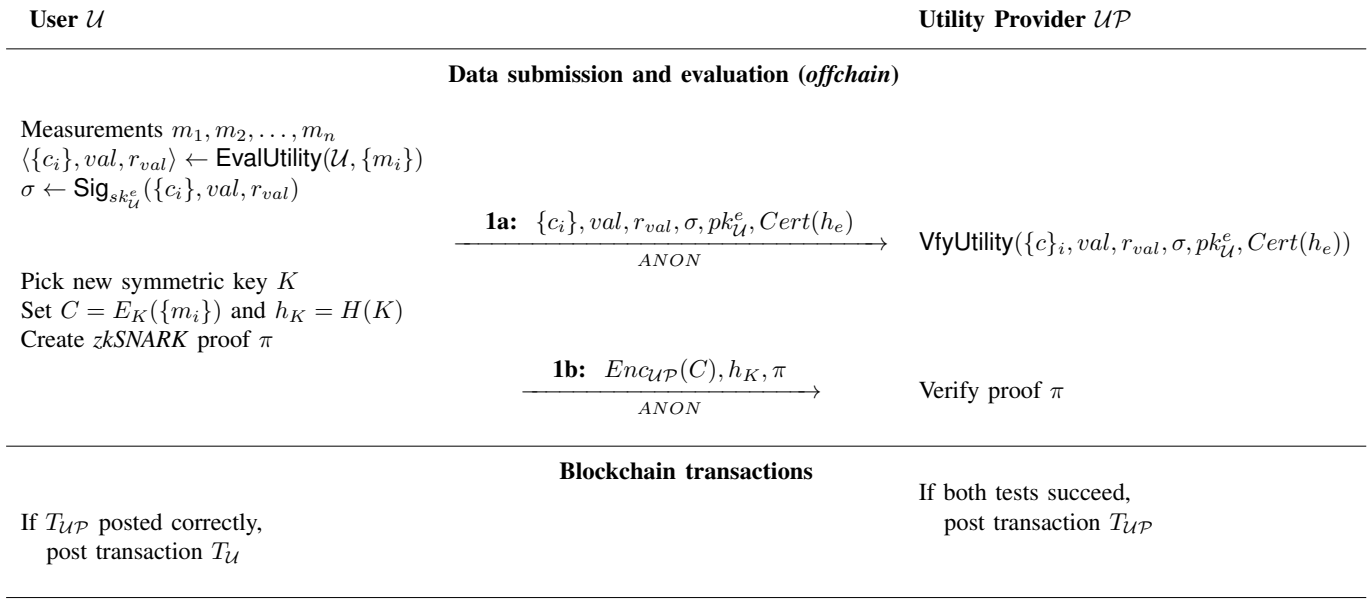

Fig. 6. Data submission, evaluation and blockchain transactions. Steps 1a and 1b are shown separately for presentation clarity, but they can be merged into one message.

execution of Reg up to this call. The Redeem query models the RedeemData protocol which makes use of the VfyUtility algorithm. Can $\mathcal{A}$ pass the Redeem call without knowing the private ephemeral credential? The answer is no since otherwise a contradiction occurs due to the fact that the signing algorithms are secure.

We now state the theorem that will formally prove the above intuition.

Theorem 1: (Legitimacy) If the signature scheme used to create the ephemeral keys is unforgeable, the proposed $\mathcal{DPTS}$ scheme satisfies the legitimacy property as defined in Definition 1.

Proof: Let $\mathcal{A}$ be an adversary in the legitimacy experiment $\mathsf{Exp}^{\mathsf{Le}}_{\mathcal{A}}$. Let $\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},1}$ be the event that Case 1 happens, that is $\mathcal{A}$ manages to obtain a valid ephemeral key pair that was not an output of any Reg query. Let $\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},2}$ be the event that Case 2 happens, that is $\mathcal{A}$ manages to successfully redeem data from $\mathcal{UP}$ through a Redeem call and using an ephemeral key pair (pk^e_*, sk^e_*) that was not an output of any previous Reg query. Then, for any security parameter κ, we have the following:

$$\mathsf{Adv}^{\mathsf{Le}}_{\mathcal{A}}(\kappa) \leq \Pr[\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},1} \vee \mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},2}]$$

We will now show that, if either these bad events happen, then we can break the unforgeability of the underlying signature scheme. Let $\mathcal{A}$ be an adversary that can make either of these bad events happen. Then there exists an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability of the underlying signature scheme. The adversary $\mathcal{B}$ works as follows: $\mathcal{B}$, as an adversary in the unforgeability game, first receives as input the verification keys $(pk^e_{\mathcal{U}}, pk_{\mathcal{UP}})$ from the challenger. It then runs $\mathcal{A}^{\mathsf{Reg},\mathsf{Redeem}}(pk^e_{\mathcal{U}}, pk_{\mathcal{UP}})$, acting as the provider for $\mathcal{A}$ while simulating Reg and Redeem queries. $\mathcal{B}$ simulates Reg queries by generating a new key pair (pk^e, sk^e) then calling the signing oracle (of the unforgeability game) to get $Cert(h_e) \leftarrow \mathsf{Sig}(sk_{\mathcal{UP}}, H(pk^e))$. $\mathcal{B}$ simulates Redeem queries by verifying utilities and their signatures/proofs. Note that this perfectly simulates the game for $\mathcal{A}$.

If $\mathcal{A}$ recovers $sk_{\mathcal{U}}$ or forges $Cert(h_e)$ for the key pair $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$ (i.e. $\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},1}$ occurs) then $\mathcal{B}$ could use $(H(pk^e_{\mathcal{U}}), Cert(h_e))$ as its forgery in the unforgeability game to win. Furthermore, if $\mathcal{A}$ is successfully able to pass verification when redeeming using forged credentials $(pk^e_*, sk^e_*, Cert(h_e))$ that was not an output of some Reg query (i.e. $\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},2}$ occurs) then $\mathcal{B}$ could use $(H(pk^e_*), Cert(h_e))$ as its forgery in the unforgeability game to win. Therefore:

$$\Pr[\mathcal{B}\ \mathsf{wins}] \geq \Pr[\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},1} \vee \mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},2}]$$

However, because the underlying signature scheme is secure, that there exist a negligible function ϵ such that:

$$\Pr[\mathcal{B}\ \mathsf{wins}] \leq \epsilon$$

Thus,

$$\mathsf{Adv}^{\mathsf{Le}}_{\mathcal{A}}(\kappa) \leq \Pr[\mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},1} \vee \mathsf{Bad}^{\mathsf{Le}}_{\mathcal{A},2}] \leq \epsilon$$

□

b) Provider Balance: We start by providing an intuition behind why the provider balance property holds. Note that, because the proposed scheme satisfies the legitimacy property, the adversary cannot claim an unused authorization credential $(pk^e_{\mathcal{U}}, sk^e_{\mathcal{U}})$. Thus, by Definition 2, an adversary can win the provider balance game in three cases.

1) The adversary observes the transaction posted with the ephemeral public key and extracts the corresponding private key $sk^e_{\mathcal{U}}$. Now she can redeem the payment by signing another bitcoin transaction. However, this is infeasible due to the security of the underlying signature scheme.
2) The adversary claims a value larger than the value of the committed data. Again this is not possible due to

the binding property of the commitment scheme. Once the data are committed, it is the provider who checks the validity of their value. Hence it is not possible to claim a different value than the one estimated through the application of the utility function. To see why, suppose that $x = (val, r_{val})$ and $x' = (val', r'_{val})$ where $val \neq val'$ but $g^{val}h^{r_{val}} = g^{val'}h^{r'_{val}}$. This would imply that $\sum_i(a_1 m_i + a_0) \neq \sum_i(a_1 m'_i + a_0)$ and $a_1 \sum_i r_i \neq a_1 \sum_i r'_i$, which indicates that there exists at least one commitment $c_j = g^{m_j}h^{r_j} = g^{m'_j}h^{r'_j}$ that can be opened up into two different messages $m_j \neq m'_j$, breaking the binding property.

3) The adversary during the Redeem process produces a proof of an invalid statement (i.e. encryptions of undervalued messages) to convince the provider that they are of larger value. However, this is not possible due to the soundness of the *zkSNARK*.

We now state the theorem that will formally prove the above intuition.

Theorem 2: (Provider Balance) If Theorem 1 holds, the signature scheme is secure, the commitment scheme is binding, and the *zkSNARK* scheme is sound, the proposed $\mathcal{DPTS}$ scheme satisfies the provider balance property as defined in Definition 2.

Proof: Let $\mathcal{A}$ be an adversary in the provider balance experiment $\mathsf{Exp}^{\mathsf{PBa}}_{\mathcal{A}}$. Let $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},\mathsf{i}}$ be the event that Case i happens, where $i \in \{1, 2, 3\}$. Then for any security parameter κ we have the following:

$$\mathsf{Adv}^{\mathsf{PBa}}_{\mathcal{A}}(\kappa) \leq \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},1}] + \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},2}] + \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},3}]$$

Let $\mathcal{A}$ be an adversary that triggers the event $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},1}$. Then there exists an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability of the underlying signature scheme. The adversary $\mathcal{B}$ works as follows: given a public key $pk^e_{\mathcal{U}}$ as input, $\mathcal{B}$ would obtain $(pk_{\mathcal{UP}}, sk_{\mathcal{UP}}) \leftarrow \mathsf{UPGen}(\mathcal{CRS})$ then run $\mathcal{A}^{\mathsf{Reg},\,\mathsf{Redeem}}(pk_{\mathcal{UP}})$ simulating $\mathcal{A}$'s queries as the provider. When $\mathcal{A}$ has created a new bitcoin transaction and forged a signature under $pk^e_{\mathcal{U}}$, $\mathcal{B}$ would submit this forgery to win the game. Therefore:

$$\Pr[\mathcal{B}\ \mathsf{wins}] \geq \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},1}]$$

However, since the signature scheme is secure, we have that there exists a negligible function ϵ such that:

$$\Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},1}] \leq \Pr[\mathcal{B}\ \mathsf{wins}] \leq \epsilon$$

Next, suppose that $\mathcal{A}$ triggers the event $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},2}$. Then there exists an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the computational binding property of the underlying Pedersen commitment scheme. The adversary $\mathcal{B}$ starts by obtaining $(pk_{\mathcal{UP}}, sk_{\mathcal{UP}}) \leftarrow \mathsf{UPGen}(\mathcal{CRS})$ then executing $\mathcal{A}^{\mathsf{Reg},\,\mathsf{Redeem}}(pk_{\mathcal{UP}})$ while simulating $\mathcal{A}$'s queries and acting as the provider. Because event $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},2}$ occurs, during the course of simulating $\mathcal{A}$'s Redeem queries, $\mathcal{A}$ will output commitments $\{c_i\}$ that can be opened into val or $val' < val$. Since it is unknown which of the $q = poly(\kappa)$ queries to Redeem might cause this event to happen, $\mathcal{B}$ would select one of the Redeem queries at random and submit the commitments to its challenger to win the game with non-negligible probability $1/q$. Therefore:

$$\Pr[\mathcal{B}\ \mathsf{wins}] \geq \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},2}]/q$$

However, since the commitment scheme is binding, we have that there exists a negligible function ϵ such that:

$$\Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},2}] \leq \Pr[\mathcal{B}\ \mathsf{wins}] \leq q.\epsilon$$

Lastly, suppose that $\mathcal{A}$ triggers the event $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},3}$. Then there exists an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the soundness of the underlying *zkSNARK*. The adversary $\mathcal{B}$ starts by obtaining $(pk_{\mathcal{UP}}, sk_{\mathcal{UP}}) \leftarrow \mathsf{UPGen}(\mathcal{CRS})$ then executing $\mathcal{A}^{\mathsf{Reg},\,\mathsf{Redeem}}(pk_{\mathcal{UP}})$ while simulating $\mathcal{A}$'s queries and acting as the provider. Because event $\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},3}$ occurs, during the course of simulating $\mathcal{A}$'s Redeem queries, $\mathcal{A}$ will output a proof π^* for a statement $x^* \notin \mathcal{L}_U$ such that the false proof passes verification. $\mathcal{B}$ can then submit (x^*, π^*) to its challenger to break the soundness. Therefore:

$$\Pr[\mathcal{B}\ \mathsf{wins}] \geq \Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},3}]$$

However, since the *zkSNARK* is sound, we have that there exists a negligible function ϵ such that:

$$\Pr[\mathsf{Bad}^{\mathsf{PBa}}_{\mathcal{A},3}] \leq \Pr[\mathcal{B}\ \mathsf{wins}] \leq \epsilon$$

Since each of the bad events have negligible probability of happening, we have that $\mathsf{Adv}^{\mathsf{PBa}}_{\mathcal{A}}(\kappa) \leq \epsilon'$ for some negligible function ϵ'. □

c) User Balance: We start by providing an intuition behind the security proof. By Definition 3, there are six ways that an adversarial provider may win this experiment during the Redeem phase.

1) The adversary attempts to post a transaction which does not match the value of the data. However, in that case the user aborts and does not release the encryption key K.
2) The adversary observes the transaction posted with the ephemeral public key and extracts the corresponding private key $sk^e_{\mathcal{U}}$. $\mathcal{A}$ can now refund back the payment by signing another bitcoin transaction. However, this is infeasible due to the security of the underlying signature scheme.
3) The adversary can extract information about the message from the commitments $\{c_i\}$. However, this would violate the hiding property of commitments.
4) The adversary can extract information about the message from the encrypted messages C. However, this would violate the semantic security of the private-key encryption scheme.
5) The adversary can invert the hash $H(K)$ to get the symmetric key and decrypt C. However, given that we will model H as a random oracle, this would violate the one-wayness of H.
6) The adversary can extract information about the message/key from the *zkSNARK* proof. However, this would violate the zero knowledge property of the proof.

We now state the theorem that will formally prove the above intuition.

Theorem 3: (User Balance) If Theorem 1 holds, the public-key encryption and signature schemes are secure, the commitment scheme is hiding, the *zkSNARK* scheme is zero-knowledge, then the proposed $\mathcal{DPTS}$ scheme satisfies the user balance property in the random oracle model as defined in Definition 3.

Proof: Let $\mathcal{A}$ be an adversarial provider in the user balance experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UBa}}$. First, note that Case 2 is identical to the first case of the Provider Balance proof, so we refer to the proof there for this case. To prove Cases 3-6, we define a sequence of computationally indistinguishable hybrid experiments and show that $\mathcal{A}$'s advantage in the game where $\{m_i^0\}$ is selected is close to the experiment where $\{m_i^1\}$ is selected. We define the hybrids as follows:

- Hyb_0 : This is experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UBa}}$ where $\{m_i^0\}$ was chosen as the challenge message.
- Hyb_1 : This is the same as Hyb_0 except that $\mathcal{CRS}$ is generated by the *zkSNARK* simulator S, and in the challenge phase the proof during the $\mathsf{RedeemP}^*$ call is simulated as $\pi \leftarrow S(x)$ where $x = (C, h_K, \{c_i\})$.
- Hyb_2 : This is the same as Hyb_1 except that, instead of hashing the key K, we set $h_K \xleftarrow{\$} R$ to be a uniformly random value.
- Hyb_3 : This is the same as Hyb_2 except that, instead of encrypting messages $\{m_i^0\}$, we would encrypt $\{m_i^1\}$. That is, $C = E_K(\{m_i^1\})$.
- Hyb_4 : This is the same as Hyb_3 except that, instead of committing to the messages $\{m_i^0\}$, we would commit to all $\{m_i^1\}$. That is, $c_i = \mathsf{Commit}(m_i^1, r_i)$ for all i.

Let $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_i}}$ denote the adversary's advantage in Hybrid i. Our goal is to show that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_0}}$ is close to $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_4}}$, which would imply that $\mathcal{A}$ would not be able to distinguish between whether the user submitted one message or the other.

First, we show that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_0}}$ - $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_1}}$ is negligible. Let $\mathcal{A}$ be an adversary for which the above is not true. Then we can construct an adversary $\mathcal{B}$ that, given $\mathcal{A}$, would break the underlying zero knowledge property of the *zkSNARK*. $\mathcal{B}$ works as follows: after receiving the $\mathcal{CRS}$ as input, $\mathcal{B}$ would run the adversarial provider $\mathcal{A}$ making sure to simulate all queries asked by $\mathcal{A}$. Note that $\mathcal{B}$, as the user, is able to generate on its own the secret keys for signatures and symmetric encryptions so it is able to answer all of $\mathcal{A}$'s queries. Once $\mathcal{A}$ issues the $\mathsf{RedeemP}^*$ query and sends over its challenge messages $\{m_i^0\}, \{m_i^1\}$, $\mathcal{B}$ would use $\{m_i^0\}$ to form the statement $(C, h_K, \{c_i\})$, then submit it to its challenger in the zero-knowledge game. Once it gets back the proof π, $\mathcal{B}$ forwards (C, h_K, π) to $\mathcal{A}$. Observe that, if the proof π is real then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_0, so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_0}} = \Pr_{\text{real}}[\mathcal{B} = 0]$ Otherwise, if the proof π is simulated then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_1 so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_1}} = \Pr_{\text{sim}}[\mathcal{B} = 0]$. Since the underlying proof is zero-knowledge, we have that $\Pr_{\text{real}}[\mathcal{B} = 0] - \Pr_{\text{sim}}[\mathcal{B} = 0] = negl(\kappa)$. Therefore:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_0}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_1}} \leq negl(\kappa)$$

Next, we show that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_1}}$ - $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_2}}$ is negligible. This is indeed the case when H is a random oracle since its distribution is statistically close to a truly random function. As a result, any computationally unbounded (poly-query) adversary can differentiate between the two worlds with negligible probability:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_2}} \leq negl(\kappa)$$

Next, we show that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_2}}$ - $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}}$ is negligible. Let $\mathcal{A}$ be an adversary for which the above is not true. Then we can construct an adversary $\mathcal{B}$ that, given $\mathcal{A}$, would break the underlying semantic security of the private-key encryption scheme. $\mathcal{B}$ works as follows: after receiving the $\mathcal{CRS}$ as input, $\mathcal{B}$ would run the adversarial provider $\mathcal{A}$ making sure to simulate all queries asked by $\mathcal{A}$. Note that $\mathcal{B}$, as the user, is able to generate its own the signing keys for signatures. However, whenever it needs to encrypt messages to send to $\mathcal{A}$, it would forward such messages to its challenger (which holds the symmetric key). Once $\mathcal{A}$ issues the $\mathsf{RedeemP}^*$ query and sends over its challenge messages $\{m_i^0\}, \{m_i^1\}$, $\mathcal{B}$ would forward these messages to its own challenger for encryption and get back C. It then creates the simulated proof $\pi \leftarrow S(x)$ for $x = (C, R, \{c_i\})$ and forwards (C, R, π) to $\mathcal{A}$. Observe that, if the encryption were of $\{m_i^0\}$, i.e. $C \leftarrow E_K(\{m_i^0\})$ then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_2, so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_2}} = \Pr_{m^0}[\mathcal{B} = 0]$ Otherwise, if the encryption were of $\{m_i^1\}$ then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_3 so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}} = \Pr_{m^1}[\mathcal{B} = 0]$. Since the encryption scheme is semantically secure, we have that $\Pr_{m^0}[\mathcal{B} = 0] - \Pr_{m^1}[\mathcal{B} = 0] = negl(\kappa)$. Therefore:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_2}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}} \leq negl(\kappa)$$

Lastly, we show that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}}$ - $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_4}}$ is negligible. Let $\mathcal{A}$ be an adversary for which the above is not true. Then we can construct an PPT adversary $\mathcal{B}$ that, given $\mathcal{A}$, would break the underlying hiding property of the commitment scheme. $\mathcal{B}$ works as follows: after receiving the $\mathcal{CRS}$ as input, $\mathcal{B}$ would run the adversarial provider $\mathcal{A}$ making sure to simulate all queries asked by $\mathcal{A}$. Note that $\mathcal{B}$, as the user, is able to generate its own the signing and symmetric encryption keys. However, if it needs to create commitments for $\mathcal{A}$ as part of answering $\mathsf{RedeemP}$ queries, it would need to forward such requests to its challenger. Once $\mathcal{A}$ issues the $\mathsf{RedeemP}^*$ query and sends over its challenge messages $\{m_i^0\}, \{m_i^1\}$, $\mathcal{B}$ would forwards these message pairs to its challenger for commitment to get back $\{c_i\}$. $\mathcal{B}$ then creates $C = E_K(\{m_i^1\})$ and generates the simulated proof $\pi \leftarrow S(x)$ for $x = (C, R, \{c_i\})$, which it then uses to forward (C, R, π) to $\mathcal{A}$. Observe that, if the commitments were of $\{m_i^0\}$, i.e. $c_i \leftarrow \mathsf{Commit}(m_i^0, r_i)$ then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_3, so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}} = \Pr_{m^0}[\mathcal{B} = 0]$ Otherwise, if the commitments were of $\{m_i^1\}$ then $\mathcal{B}$ simulates $\mathcal{A}$'s view in Hyb_4 so $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_4}} = \Pr_{m^1}[\mathcal{B} = 0]$. Since the commitment scheme is computationally hiding (in the homomorphic setting even if r_{val} is released), we have that $\Pr_{m^0}[\mathcal{B} = 0] - \Pr_{m^1}[\mathcal{B} = 0] = negl(\kappa)$. Therefore:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_3}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_4}} \leq negl(\kappa)$$

Combining the negligible probabilities of each pair of successive hybrids yields:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_4}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb_0}} \leq negl(\kappa)$$

Observe that Hyb_0 is just $\mathsf{Exp}^{\mathsf{UBa}}_{\mathcal{A}}$ where $b = 0$ and Hyb_4 is $\mathsf{Exp}^{\mathsf{UBa}}_{\mathcal{A}}$ where $b = 1$. Therefore we can rewrite this as:

$$|\Pr[\mathsf{Exp}^{\mathsf{UBa}}_{\mathcal{A}} = 1|b = 1] - \Pr[\mathsf{Exp}^{\mathsf{UBa}}_{\mathcal{A}} = 1|b = 0]| \leq negl(\kappa)$$

□

d) Privacy: By Definition 4, we observe that an adversarial provider can distinguish or link between different keys only if were able to recover the ephemeral public keys that it is signing and certifying during the Register phase. However, this would violate the blindness property of the RSA-based signature scheme. We prove this in more detail below.

Theorem 4: (Privacy) If the signature scheme is blind, the proposed $\mathcal{DPTS}$ scheme satisfies the privacy property as defined in Definition 4.

Proof: An adversary $\mathcal{A}$ that wins in the privacy game of Experiment 5 can be used to distinguish between two different blind signatures. In particular, consider an adversary $\mathcal{B}$ of the blind signature scheme that runs $\mathcal{A}$ and simulates its oracle queries' answers. During the learning phase, when $\mathcal{A}$ makes RegU and RedeemP queries, $\mathcal{B}$ can simulate $\mathcal{U}$ in its interaction with $\mathcal{A}$ while using the certified ephemeral public-private key pairs when evaluating utilities and signing transactions.

During the challenge phase, $\mathcal{A}$ will choose $\mathcal{U}_0$ and $\mathcal{U}_1$ and send their respective ephemeral public keys to $\mathcal{B}$. The adversary $\mathcal{B}$ now generates $h_0 = H(pk^e_{\mathcal{U}_0})$ and $h_1 = H(pk^e_{\mathcal{U}_1})$ for $\mathcal{U}_0$ and $\mathcal{U}_1$, respectively. $\mathcal{B}$ will submit h_0 and h_1 as its chosen messages in the blind signature game and receive signatures $\sigma_0 = \mathsf{BSig}(sk, h_b)$ and $\sigma_1 = \mathsf{BSig}(sk, h_{1-b})$ for some $b \xleftarrow{\$} \{0, 1\}$. $\mathcal{B}$ will forward σ_0 to $\mathcal{A}$ as part of the transRecord($\mathcal{U}_b$). After the RedeemP query, $\mathcal{A}$ outputs a bit b', which B can also use as its guess. If $b' = b$ (i.e. $\mathcal{A}$ wins its experiment) then the signed message is h_b, thus $\mathcal{B}$ can distinguish between two blind signatures. Conversely, if the blind signature scheme is secure then the privacy property of the $\mathcal{DPTS}$ scheme holds. □

B. Experimental Results

In this section, we describe in more detail the environment used to test our scheme and provide the experimental set up and results for different utilities. The benchmarks were evaluated on a virtual machine running Ubuntu 18.04.3 LTS x86_64 with a Linux 5.0.0-25-generic kernel. The processor used was a i7-8650U CPU @ 1.90GHz with access to 8 GB of RAM.

To implement the *zkSNARK* for the NP language described in Section IV, we used the xJsnark [30] framework to write our verification program then compile it into an arithmetic circuit. This circuit is constructed in such a way that is recognizable by libsnark [31], which is a C++ library that builds a preprocessing *zkSNARK* for our circuit by reducing it to a rank-1 constraint system (R1CS) and eventually to a QAP. All operations are performed over the bilinear BN128 curve.

1) Results with Linear Utility: Table II shows the timing and memory-related measurements when the utility used $U(m) = a_1 m + a_0$ is linear in the message and ZK-proofs for statements in $\mathcal{L}_U$ are generated (see Section IV-C). The messages used to test the system are 128-bit long, and the coefficients a_1 and a_0 were randomly chosen from the same group as the messages. The number of constraints used to represent the circuit is 47143.

TABLE II
THE PERFORMANCE MEASUREMENTS FOR *zkSNARK* KEY GENERATION, PROVING, AND VERIFICATION FOR A LINEAR UTILITY

	Time	Size
Key Generation	6.0636 s	PK: 11.328 MB, VK: 3.24 KB
Prover	1.109 s	Proof Size: 287 B
Verifier	3.5 ms	

2) Results with Polynomial Utility: Table III shows the timing and memory-related measurements when the utility used $U_N(m) = a_1 m^3 + a_0$ is a polynomial in the message and ZK-proofs for statements in $\mathcal{L}_{U_N}$ are generated (see Section IV-E). The messages used to test the system are 128-bit long, and the coefficients a_1 and a_0 were randomly chosen from the same group as the messages. The number of constraints used to represent the circuit is 79948.

TABLE III
THE PERFORMANCE MEASUREMENTS FOR *zkSNARK* KEY GENERATION, PROVING, AND VERIFICATION FOR POLYNOMIAL UTILITY

	Time	Size
Key Generation	8.092 s	PK: 19.07 MB, VK: 3.125 KB
Prover	1.716 s	Proof Size: 287 B
Verifier	3.6 ms	

3) Results with non-linear Utility: Table IV shows the timing and memory requirements when the utility $U_N(m) = \max(u_{min}, \min(a_1 m^2 + a_0 m, u_{max}))$ is a custom non-linear function on the data, combining a range with a polynomial for some constants $u_{min}, u_{max}, a_0, a_1$ (functions like this are typical of urban sensing applications). Based on this utility, ZK-proofs for statements in $\mathcal{L}_{U_N}$ were generated (see Section IV-E). The messages used to test the system are 128-bit long. The number of constraints used to represent the circuit is 81597.

TABLE IV
THE PERFORMANCE MEASUREMENTS FOR *zkSNARK* KEY GENERATION, PROVING, AND VERIFICATION FOR A CUSTOM NON-LINEAR UTILITY

	Time	Size
Key Generation	9.866 s	PK: 19.52 MB, VK: 3.085 KB
Prover	1.6729 s	Proof Size: 287 B
Verifier	3.5 ms	

Thus, in the case of more complex utilities, the provider basically has to validate only the proof π. As shown above, this takes a few milliseconds, allowing the provider to handle a large number of requests in urban sensing applications.

4) Communication overhead: Perhaps it is more instructive to consider the communication overhead of the protocol. Considering again the more general case of complex utilities, during the data submission phase the user sends one message consisting of

$$\langle val, com, \sigma, pk^e_{\mathcal{U}}, Cert(h_e), Enc_{\mathcal{UP}}(C), h_K, \pi \rangle.$$

The data utility val can be treated as a 64-bit number, while both com and h_K are 160-bit hash values. Bitcoin's cryptography is based on the secp256k1 elliptic curve which is used along with the ECDSA signature algorithm. Hence the public key $pk^e_{\mathcal{U}}$, which corresponds to the user's Bitcoin address, has length 33 bytes. Similarly, the signature σ is bounded by 73 bytes. On the other hand, $Cert(h_e)$ is the provider's RSA signature on the user's public key, so this will contribute another 128 bytes to the total. Hence most of the overhead comes from the proof π which is bounded by 288 bytes. Thus, without considering the size of the encrypted data C, the *fixed* overhead of the protocol amounts to 570 bytes.

The rest comes from $Enc_{\mathcal{UP}}(C)$, the encryption of C with the public key of the provider. In practice, instead of encrypting large amounts of data using public key cryptography, one uses a hybrid approach which combines the convenience of a public-key cryptosystem with the efficiency of a symmetric one. Thus, data are first encrypted with a symmetric key and then the key is encrypted with the public key. In our case, this adds an overhead of 128 bytes which is what we get if we encrypt an 128-bit symmetric key using the provider's RSA public key (these numbers can be further decreased by using elliptic curve variants).

The rest of the communication overhead comes from the size of the encrypted data C which is proportional to $size(m)$, the amount of the actual data when encrypted using a semantically secure symmetric cryptosystem. As the user engages in the protocol when she has to send some data m anyway, we conclude that the communication overhead is minimal, bounded by $size(m) + 598$ bytes.

The above analysis shows that our system does not require any heavy computational power on behalf of the provider while keeping user communication at a bare minimum. This demonstrates the practicality of $\mathcal{DPTS}$.

VI. Discussion

$\mathcal{DPTS}$ emphasizes on the rewarding part of data submitted to the provider, however as the provider cannot see the data in advance (this would break both privacy and fairness), a malicious user may attempt to redeem the same data twice. This is not exactly *double-spending* in the traditional sense but comes very close to it as the user may try to obtain another val bitcoins by submitting exactly the same data or a small variant of it (eg. a subset of it).

This problem can be solved by assuming that meters come equipped with a Trusted Platform Module (TPM) chip that can be used to (i) vouch for the correct operation of the meter, and (ii) anonymously sign a hash of the data during the initial phases of the protocol (for the use of TPM for secure billing see [32], [33]). To protect user privacy, the TPM supports unlinkable keys that can be used to maintain anonymity between different parties who require proof of identity. In particular, we will assume that the hash of the data to be submitted is digitally signed with the meter's Direct Anonymous Attestation (DAA)[1] signing key K_{DAA} as exemplified below.

Let τ be a reporting task created by the $\mathcal{UP}$ that is sent to all relevant households or home owners within a region. The task asks for electricity measurements for a certain frequency and period of time, and may look like the one shown below:

$$\tau = \langle \textit{TaskID } \#5324, \textit{ Frequency } 1 \text{ min}, \textit{ Start } \text{Nov. } 10, \textit{ End } \text{Nov. } 20 \rangle.$$

Meters that accept this task, collect the measurements $m_1, m_2, \ldots, m_n$ and produce a signature $\sigma' = Sig_{K_{DAA}}(\#5324, h')$ of the $TaskID$ and the hash h' of the measurements (an implicit assumption here is that meter software has not been compromised, so the measurements retrieved from storage match the interval requested in the task. Code attestation may help in this respect as well).

This signature along with the task ID and the hash value h' will be sent to the $\mathcal{UP}$ using Algorithm 2. The modified return statement (line 9) of this algorithm is shown below:

9: **Return** $(\{c_i\}, val, rval, \sigma, TaskID, h', \sigma')$

During VfyUtility (Algorithm 3), $\mathcal{UP}$ performs an additional test which consists of checking its database for an entry that matches the given $TaskID$ and hash value of measurements. If a match is found, $\mathcal{UP}$ aborts since the user attempts to get a reward for already submitted data. Furthermore, the *zkSNARK* proof sent must have an additional statement that shows that the hash of the data equals h'. Hence the proof must be augmented to include evidence that $h' =^? H(m_1, m_2, \ldots, m_n)$, which for the case of complex utilities this is already the case.

If the proof verifies during operation Redeem (Algorithm 4), $\mathcal{UP}$ posts the time-locked transaction to transfer the bitcoins. When the user releases the key K and $\mathcal{UP}$ obtains the data, $\mathcal{UP}$ stores in a database the task ID and the hash value of the measurements to prevent future double-spending attempts.

VII. Conclusions

In this work we developed $\mathcal{DPTS}$, a data payment and transfer scheme that can be used to trade household electricity data with bitcoins of appropriate value. While $\mathcal{DPTS}$ was described in the smart grid setting, it can also be applied in other application domains where monetary incentives are used to increase user participation. One such domain is participatory or mobile crowd-sensing.

[1]For completeness, we briefly summarize how DAA can be adopted in this setting; the interested reader may look at how the scheme has been used for remote, anonymous authentication of TPMs [34]. At the core of the scheme is the certification of the DAA key by an issuer, which however learns nothing about the key. Signing with such a key provides anonymity-preserving assurance that the smart meter has a valid DAA key. Yet, neither the verifier nor the issuer, even if they collude with each other, can tell which meter signed a message, only that the signature comes from a valid meter. Furthermore, DAA also allows a meter to generate in advance an arbitrarily large set of pseudonyms which can be used with every new submission of data.

Of particular importance are the fairness guarantees offered by $\mathcal{DPTS}$. Indeed our scheme ensures that no party can cheat the other as the data is released if and only if an appropriate bitcoin payment is made. Furthermore, no information is disclosed before or after the transfer of the data; payments are unlinkable to each other and transactions do not leak any information about the user or the meter. Thus the protocol is also privacy-preserving.

Finally, we have studied the efficiency properties of our proposal. The protocol incurs acceptable performance on the user side and minimal overhead on the provider side.

VIII. Acknowledgements

The authors would like to thank the reviewers for their comments that helped improve the paper considerably.

References

[1] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker. "Inferring personal information from demand-response systems." IEEE Security & Privacy, 8(1), 11–20, 2010.

[2] T. Dimitriou, G. Karame, "Enabling anonymous authorization and rewarding in the smart grid." In IEEE Transactions on Dependable and Secure Computing, 14(5), 565–572, 2015.

[3] T. Dimitriou and M. K. Awad,"Secure and scalable aggregation in the smart grid resilient against malicious entities." In Ad Hoc Networks, 50, 58–67, 2016.

[4] S. Sultan. "Privacy-preserving metering in smart grid for billing, operational metering, and incentive-based schemes: A survey." In Computers & Security, 84, 2019.

[5] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A.T. Campbell. "A survey of mobile phone sensing." IEEE Communications magazine 48, no. 9 (2010): 140-150.

[6] Ioannis Krontiris, Tassos Dimitriou. "A platform for privacy protection of data requesters and data providers in mobile sensing." In Computer Communications 65: 43-54, 2015.

[7] R. Cleve. "Limits on the security of coin ips when half the processors are faulty (extended abstract)." In the 18th STOC, 1986.

[8] Matthew K. Franklin and Michael K. Reiter. "Fair exchange with a semi-trusted third party (extended abstract)." In the 4th ACM Conference on Computer and Communications Security, CCS '97, pages 1–5, 1997.

[9] N. Asokan, V. Shoup, and M. Waidner. "Optimistic fair exchange of digital signatures." IEEE Journal on Selected Areas in Communications, 18(4):593-610, 2000.

[10] Iddo Bentov and Ranjit Kumaresan. "How to use bitcoin to design fair protocols." In CRYPTO 2014.

[11] J. Liu, W. Li, G. O. Karame, N. Asokan, "Toward fairness of cryptocurrency payments." IEEE Security & Privacy, 16(3), 81-89, 2018.

[12] S. Dziembowski, L. Eckey and S. Faust. "Fairswap: How to fairly exchange digital goods." In ACM CCS, 2018.

[13] G. Maxwell. "Zero knowledge contingent payment", 2015. https://en.bitcoin.it/ wiki/Zero_Knowledge_Contingent_Payment.

[14] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo. "Zero-knowledge contingent payments revisited: Attacks and payments for services." In ACM CCS, 2017.

[15] Jingzhong Wang, Mengru Li, Yunhua He, Hong Li, Ke Xiao, and Chao Wang. "A blockchain based privacy-preserving incentive mechanism in crowdsensing applications." In IEEE Access 6, 2018.

[16] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.N. Liu, Y. Xiang, and R. H. Deng. "CrowdBC: A blockchain-based decentralized framework for crowdsourcing." IEEE Transactions on Parallel and Distributed Systems 30, no. 6 (2018): 1251–1266.

[17] Yuan Lu, Qiang Tang, and Guiling Wang. "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain." In 38th International Conference on Distributed Computing Systems (ICDCS), pp. 853–865, 2018.

[18] Huayi Duan, Yifeng Zheng, Yuefeng Du, Anxin Zhou, Cong Wang, and Man Ho Au. "Aggregating Crowd Wisdom via Blockchain: A Private, Correct, and Robust Realization." In IEEE International Conference on Pervasive Computing and Communications (PerCom2019), 2019.

[19] Tassos Dimitriou, Thanassis Giannetsos, Liqun Chen. "REWARDS: Privacy-preserving rewarding and incentive schemes for the smart electricity grid and other loyalty systems." In Computer Communications 137: 1–14, 2019.

[20] A. Rial and G. Danezis, "Privacy-preserving smart metering,", in Proceedings of WPES, 2011.

[21] D. Chaum. "Blind Signatures for Untraceable Payments." In Advances in Cryptology: Proceedings of CRYPTO '82, pp. 199–203, 1982.

[22] T.P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." In: Feigenbaum J. (eds) Advances in Cryptology CRYPTO 91, pp. 120–140, 1992

[23] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic span programs and succinct NIZKs without PCPs." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 626-645. Springer, Berlin, Heidelberg, 2013.

[24] M. Bellare, G. Fuchsbauer, and A. Scafuro. "NIZKs with an untrusted CRS: security in the face of parameter subversion." In ASIACRYPT 2016.

[25] B. Abdolmaleki, K. Baghery, H. Lipmaa, M. Zajac. "A Subversion-Resistant SNARK". In Advances in Cryptology – ASIACRYPT 2017, pp. 3–33, 2017

[26] G. Fuchsbauer. "Subversion-Zero-Knowledge SNARKs". In Public-Key Cryptography PKC 2018, pp. 315–347, 2018

[27] A. Fiat, A. Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems." In Advances in Cryptology — CRYPTO' 86, pp. 186–194, 1986

[28] S. Bowe, A. Gabizon, M. Green, A. Zohar. "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK". In Financial Cryptography and Data Security 18, pp. 64–77, 2018

[29] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[30] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. "xJsnark: A Framework for Efficient Verifiable Computation." In 2018 IEEE Symposium on Security and Privacy (SP), pp. 944-961. IEEE, 2018.

[31] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza."Succinct non-interactive zero knowledge for a von neumann architecture." In the 23rd USENIX Conference on Security Symposium, SEC'14, 2014

[32] M. Jawurek, M. Johns, and F. Kerschbaum. "Plug-in privacy for smart metering billing." In Privacy Enhancing Technologies Symposium, 2011.

[33] S. Finster, and I. Baumgart. "Privacy-aware smart metering: A survey." IEEE Communications Surveys & Tutorials 16, no. 3, pp. 1732–1745, 2014

[34] E. Brickell, J. Camenisch, and L. Chen. "Direct anonymous attestation." In 11th ACM CCS, pp. 132–145, 2004.