Sipster: Settling IOU Privately and Quickly with Smart Meters

Sherman S. M. Chow*
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Ming Li
The University of Texas at Arlington
USA

Yongjun Zhao†
Nanyang Technological University
Singapore

Wenqiang Jin‡
Hunan University
Changsha, China

ABSTRACT

Cyber-physical systems revolutionize how we interact with physical systems. Smart grid is a prominent example. With new features such as fine-grained billing, user privacy is at a greater risk than before. For instance, a utility company (UC) can infer users' (fine-grained) usage patterns from their payment. The literature only focuses on hiding individual meter readings in bill calculation. It is unclear how to preserve amount privacy when the UC needs to assert that each user has settled the amount as calculated in the bill.

We advocate a new paradigm of cash payment settlement enabling payment privacy. Users pay their bills in unit amount so that they can hide in the crowd. Meanwhile, UC can obtain payments earlier in the pay-as-you-go model, leading to a win-win situation. A highlight of our proposed system, Sipster, is that the *receipts* for the payments can be *combined* into a $O(1)$-size receipt certifying the smart meter's certification. Without such aggregation, techniques such as zero-knowledge proof would fail since it typically cannot hide the size of the witness. Seemingly helpful tools, *e.g.*, aggregate signatures or fully homomorphic signatures, also fail.

The novelty of Sipster lies in fulfilling our five goals simultaneously: 1) privacy-preserving: the UC cannot infer a user's payment amount; 2) prover-efficient: *no* zero-knowledge proof is ever needed; 3) verifier-efficient: it takes $O(1)$ time to verify a combined receipt; 4) double-claiming-free: users cannot present the same receipt twice; and 5) minimalistic smart meter: it has the capability to report signed readings (needed even in a non-private setting).

CCS CONCEPTS

• **Security and privacy → Cryptography**; **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**.

KEYWORDS

privacy-enhancing technologies, payment, trusted smart meters

ACM Reference Format:
Sherman S. M. Chow, Ming Li, Yongjun Zhao, and Wenqiang Jin. 2021. Sipster: Settling IOU Privately and Quickly with Smart Meters. In *ACSAC '21, December 6–10, 2021, Austin, Texas, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3485832.3488029

1 INTRODUCTION

Pay-As-You-Go has recently emerged as a desired way to settle utility bills in some cyber-physical systems (CPS), such as smart grids. Many companies, such as AT&T [6], Siemens [52], and SmartGridCIS [53], have developed their pay-as-you-go payment programs for residential customers. It is forecasted that more than 2.6 million households in the United States will adopt this bill settlement model by the end of 2021 [47]. Meanwhile, this model is quite common in some other countries such as South Africa [32, 44]. Different from the traditional post-pay model, where customers use utilities for 30 days and then pay for a monthly bill, customers in the pay-as-you-go model usually pre-pay a certain amount in advance [7, 32, 44, 52, 53], which is then deducted based on the fine-grained usages, say, per day or even per hour, calculated according to the meter readings and real-time tariffs. The *utility company* (UC) is motivated to implement this model to mitigate late customer payments [53] and shorten investment cycles by charging in real-time. Customers can also become more aware of their utility consumption, partially due to *psychological factors*, and adjust their usages accordingly to avoid unusually high bills that were only *notified at the end of a long billing cycle*[1]. It thus helps to smoothen peaks in load demand, leading to more efficient utilization (of electricity generation capacities) and a higher level of resilience (*e.g.*, against unforeseen grid disruptions). Ideally and ultimately, it aids in global electricity usage reduction and saves our planet.

1.1 A Neglected Weakest Link in Privacy

The utility of fine-grained meter readings, apart from benefiting all stakeholders, raises privacy and safety concerns as well. Through meter readings, robbers can identify empty households or even if a robbery alarm has been set [50], or insurers can identify electricity usage patterns with fire risks (and charge customers higher premiums). Existing research studied how to preserve privacy in bill

*A preliminary version of this work has been presented in a keynote talk [23].
†Corresponding author (theory), email: yongjun.zhao@ntu.edu.sg. The work was started when the author was a Ph.D. student at The Chinese University of Hong Kong.
‡Corresponding author (artifact), email: wqjin@hnu.edu.cn. The work was done when the author was a Ph.D. student at The University of Texas at Arlington, United States.

[1]There is recurring news on complaints of unusually high monthly electric bills, *e.g.*, https://foxsanantonio.com/news/local/san-antonio-residents-complain-about-unusually-high-electric-bills-09-14-2019.

calculation under the post-pay model, *i.e.*, aggregating unit usages of a customer within a period or aggregating the usages across a group of customers, while hiding the usage of any individual. Section 6 discusses some representative works on these topics. Unfortunately, privacy issues of the bill settlement stage have been neglected. Obviously, hiding the *payment amount* is equally critical for its obvious correlation with the *utility usage* or the *bill amount.*

Preserving user privacy is not only for pleasing the customers or the financial benefits of pay-as-you-go. Privacy compliance with the fine-grained usage information, and its financial consequence if it fails[2], strongly motivate the UC to explore advanced solutions.

1.2 A New Problem Formulation

We advocate the following bill settlement paradigm. For each unit of utility consumption, the smart meter issues a "small bill" to the customer, who then pays anonymously to the UC from time to time in unit amount and is given a *receipt* in return. By a certain deadline, they *combine* all the receipts to convince the UC that what they have paid settled what they have used so far according to the smart meter. The benefits are twofold. The UC can get payments early as in the pay-as-you-go model. Meanwhile, customers can hide their payments in the crowd of all payments. We divide bill settlement into two processes, bill payment and bill verification. Cryptographic e-cash (*e.g.*, [15]), ensuring unlinkability of e-coins from the same user, can serve as a solution to the former. However, linkability is desirable for bill verification to show the UC that the bill has been settled. **Realizing bill verification in a privacy-preserving manner is an unaddressed problem very different from private payment**. The UC needs a special mechanism to verify that the hidden amount is correct, and a user has paid enough.

1.3 Our Contributions and Design Constraints

We propose Sipster, a cash payment settlement scheme for any smart-meter-enabled systems expecting fine-grained bill settlement with privacy. Sipster aims to satisfy five goals simultaneously.

- **Privacy.** The UC cannot infer the exact payment amount a user makes for the bill. Moreover, individual payments made by the same user are unlinkable by the UC.
- **Prover-efficient.** The residential user should only perform arguably the only necessary operation, which is to prove their bill is settled to the UC, ideally, by just a few group operations and one exponentiation. Despite being a privacy-preserving solution, our user should be free from performing any explicit zero-knowledge proof. Moreover, the user should not be required to maintain any state information.
- **Verifier-efficient.** The UC takes constant time (in the number of receipts/amount) to verify a combined receipt.
- **Double-claiming-free.** Users cannot present the same receipt twice that can pass the payment verification.
- **Deployable.** 1) Sipster should be modular and work seamlessly with *any* e-cash system deployed currently [10, 43] and in the future. 2) Smart meters remain "minimalistic." They need not possess *general computing functionality* beyond what is assumed in the non-private setting, *e.g.*, signing for ensuring the unforgeability of the billing amounts.

[2]It may be due to financial motivation, *e.g.*, Equifax was fined $700 mil. for data breach.

Having some form of a trusted computing base (TCB) is common and inherent in smart meters [33, 37, 48]. Specifically, we assume the TCB of the meters can perform primitive operations for digital signatures, including pseudorandom number generation and basic modular operations. In particular, the TCB only outputs group elements and signatures without taking any input (except in a limited form such as taking a signal to trigger the troubleshooting mode for fault tolerance), let alone processing any cryptographic objects such as zero-knowledge proofs. This captures the constraint of meters in other kinds of CPSes beyond power systems, greatly limits our design space, and rules out many potential techniques.

If we assume the TCB performs *general computing functionality*, the "prover-efficient" property is made less interesting because one could move all user computation into the TCB. In other words, our design challenge is to "extend the trust" from the minimalistic outputs of the TCB to an untrusted computation environment, which typically resorts to the "traditional" cryptographic data processing.

Note that typical hardware-aided cryptographic solutions and Sipster leverage the TCB differently for different goals. A general goal in the literature, especially those recent ones that appeared after the recent developments in commodity trusted execution environments, is to outsource computation over confidential data to some untrusted environment. Specific technical challenges thus often involve processing data within the limited memory space of the TCB (*e.g.*, [55]), interoperability with the different computing architecture of the untrusted environment, say, GPU (*e.g.*, [42]), or verifiability of computation result from the untrusted environment (*e.g.*, [54]). Sipster aims to use the TCB to "bootstrap" the security of a malleable-signature-like functionality (see Section 3.2), in contrast to the trendy processing encrypted data theme.

Roughly, Sipster can be seen as authenticating encrypted data while hiding the data size. Our problem boils down to hiding the size of the witnesses (*i.e.*, the number of individual signatures to be combined) or the domain of the function modeling the controlled malleability; both are not well explored in theoretical cryptography.

To the best of our knowledge, this is the first study on enabling customers to prove they have paid a bill without revealing the amount of usage, which appears to be inherent in fine-grained incremental billing. Specifically, we will discuss in Section 3 that:

- purely cryptographic solutions, even assuming heavyweight primitives (*e.g.*, multi-key fully-homomorphic signatures [36]), face several obstacles, which are unclear how to tackle;
- "trusted-computer-like" solution exists, but its deployment cost will be high; nevertheless, it can serve as a baseline to indirectly illustrate the novelty of our approach.

As such, we empirically evaluate a prototype implementation of Sipster without any baseline comparison. The smart meter is implemented on an ARM development board, whereas the other entities are implemented on regular PCs. Our experiments show that Sipster can be efficiently performed on resource-restricted systems. For example, it issues a 1-unit bill in 34.272ms and settles the bill in 5.665ms on average. Verifying a 100-unit bill takes only 9.212ms.

We discuss in the context of smart grids for concreteness. Sipster generally fits with the IOU ("I owe you") model with the amount the user owes certified by a trusted environment. Section 7 explains how to apply Sipster in privacy-preserving mobile payment.

2 SYSTEM MODEL AND USE CASES

2.1 Usage Model

Our CPS payment settlement system, Sipster, involves three entities.

Residential Users (RUs): All RUs first register with the UC to get a meter. To receive uninterrupted electricity provisions, an RU has to settle the bill with the UC by the end of a billing period[3].

We suppose there is a payment system that RU can first deposit money (say, at the bank) to receive (e-)coins. For every e-coin spent, the RU receives a receipt from the UC. All the collected receipts will be used to prove the correctness of this RU's bill payment during the verification process.

Smart Meters (SMs): SMs are deployed at the RU side to measure the electricity consumption. For each unit of expense, SM outputs an authenticated token to the RU. With the knowledge of real-time tariffs and consumption measurements, the SM computes the bill at the end of each billing period and sends it to its RU.

Utility Company (UC): Once received from the RU any proof of payment output by the underlying payment system, the UC returns the RU a receipt, collected and used later for payment verification. By the end of a billing period, the UC interacts with each RU to carry out the verification process. In this way, the UC knows which RU has (not) settled the bills.

Since there are multiple payments within one billing period, the time for payment and the time for bill verification are different. The billing period represents the period that the SM calculates and certifies a final bill for an RU. Within a single billing period, the UC can receive payment from the anonymous RU from time to time. The UC only needs to verify the bill at the end of a billing period.

2.2 Some Deployment Issues

Sipster needs to work with a base (e-)cash system. An electronic payment system facilitates automatic payment via the RU's device/software even when privacy/verifiability concerns are absent.

Moreover, Sipster works with any payment system ranging from cryptographic e-cash[4] to even paying physically at a booth. For example, the former can be instantiated by (efficient) compact e-cash [15] (with a compact wallet) or simple blind-signature-based approaches. For the latter, we assume RU can get some form of signatures corresponding to the payment, similar to the current practice. (Here, similar to what we expect from the SMs to be detailed below, we assume the booth will issue signature without embedding tracking information.) *Other applications* may use the same e-cash system at the same time, and it could be run by any external party. During a billing period, the RU pays using this payment system and retains the proof of payment to be presented to the UC via our protocol. Apart from this only linkage, we stress that this payment process is independent of Sipster.

Computing devices (*e.g.*, desktops) of RUs can also save the trouble of settling the bill in person[5]. They are responsible for receiving any outbound traffic of the meters, which helps in preventing the meters from directly interacting with the UC.

We suppose all meters are identical. Mass-manufacture of such meters is cost-effective. Finally, incremental deployment is possible after the system is deployed. Users can always choose not to use anonymous e-cash if they do not care about their privacy. Users may also choose to pay in one shot, which means they choose to forgo their privacy. Our paradigm is thus "backward compatible" as both kinds of users (concerning privacy or not) can co-exist.

2.3 Security Requirements and Threat Model

2.3.1 *Soundness.*

Soundness depends on two security properties, namely, unforgeability and double-spending prevention.

- **Unforgeability.** 1) The RU cannot produce a valid receipt unless he pays a valid e-coin to the UC. 2) The RU cannot forge a valid aggregated receipt as the aggregation result over K individual receipts, with only $K' < K$ valid receipts.
- **Double-spending prevention.** Any RU cannot "redeem" the same receipt twice.

For soundness, the SMs, which possess the secret (fine-grained usage) we aim to protect, are tamper-resistant devices (as in [4, 48]) that can perform lightweight cryptographic computations such as issuing digital signatures. It is a practical and necessary assumption in reality; otherwise, any RU can change the reading and pay less.

2.3.2 *The Parameter K.*

We first discuss a terminology issue. In a nutshell, Sipster aims to hide the parameter K throughout the bill settlement process from the UC. As a crucial parameter of interest to be hidden by our new paradigm, it manifests in different "forms" throughout different phases of the system at different levels. From the perspective of the SM, viewing it issuing an individual bill from time to time, K refers to the "*bill size*." This individual bill servers as a cryptographic object to be processed by the UC; we thus call each of these outputs from the SM a "*token*," and hence K also reflects *the number of tokens*. Finally, from the accounting perspective, it is the *number of increments in a single bill*.

2.3.3 *Privacy.*

Privacy intuitively covers every action the RU performs regarding the payment, from paying in unit amounts to proving the paid amount is sufficient. Moreover, the final bill settlement proof (for sufficiency) cannot be linked to any prior payment.

- **Payment/Receipt Amount Privacy.** The UC cannot infer how many e-coins it receives from an RU.
- **Unlinkability.** The UC cannot link any spending of e-coin to a particular RU.

Even with privacy protection, the UC can still verify the correctness of each RU's bill payment without knowing the exact amount.

Game-based definitions of these requirements are formalized in Appendix C. Note that we do not explicitly consider an outsider an adversary since it has less power than and is covered by the UC.

For privacy, we make two necessary trust assumptions –
1) RUs' computing devices (*e.g.*, desktops) make the payments over an anonymous communication network (*e.g.*, Tor), which is necessary for any privacy-preserving protocols, including e-cash.
2) SMs are uniform across an anonymity set (*e.g.*, borough, zip code, street). To show our core technical contribution, we describe in terms of a standard signature scheme for the unforgeable reports. One may also use privacy-oriented approaches such as group signatures or their extensions (*e.g.*, [1]).

[3]There can be a grace period depending on UC's policy and RU's security deposit.
[4]We omit the standard and relatively lengthy definition of cryptographic e-cash and its privacy-preserving/anonymity properties, e.g., see [15, 25].
[5]Sipster does not require this device to be permanently online due to its high efficiency, even for a high-usage RU who needs to pay a lot and hence pays often.

Correspondingly, our formal definition considers there are at least two SMs do not collude with the UC, and we aim to protect the utility usage of the corresponding RUs. Similar (non-)colluding assumption is widely accepted in the literature [3, 27, 30, 35, 37, 48].

2.3.4 Additional Concerns. Firstly, if desired, one can always add a (local) differentially-private mechanism to further obfuscate the bill amount, similar to the existing works for bill calculation [3, 27].

Kleptographic attack (*e.g.*, the SMs embed the customer ID or the bill amount within the randomness sent to the UC) is beyond our scope. Specific cryptographic mitigations might be applicable, such as subversion-resistant signatures/commitments [5, 8, 26] or cryptographic reverse firewalls [29, 39]. Integrating them while retaining the security and privacy of Sipster is left as future work. A physical countermeasure would be having a law enforcement agent examine the meter and impose a penalty on such a malicious UC. A malicious meter reporting high consumption to make the RU pay more is also beyond our scope. Nevertheless, Section 4.4 discusses fault tolerance against non-malicious malfunction.

3 CHALLENGES AND OUR BASELINE SYSTEM

3.1 Notations

We write $x \leftarrow X$ for sampling uniformly at random from a set X. We write $\{x_i\}_n$ as a shorthand for $X = \{x_1, \ldots, x_n\}$ of n elements, and $[n]$ denotes the set $\{1, \ldots, n\}$. Unless otherwise stated, algorithms are all probabilistic polynomial-time (PPT). The output of algorithm $\mathcal{A}$ on input x is denoted by $y \leftarrow \mathcal{A}(x)$. λ is the security parameter, and $\mathsf{negl}(\lambda)$ denotes a negligible function in λ.

Pairing. For presentation brevity, our paper is written assuming a symmetric pairing group. Let $\mathbb{G}_1$ (source group) and $\mathbb{G}_T$ (target group) be cyclic multiplicative groups of order q as a λ-bit prime. A pairing, or a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, has the following two properties. (1) Bilinearity: $\forall x, y \in \mathbb{G}_1$, and $\forall a, b \in \mathbb{Z}_p^*$, $e(x^a, y^b) = e(x, y)^{ab}$; (2) Non-degeneracy: $e(g, g) \neq 1_{\mathbb{G}_T}$ for any generator g of $\mathbb{G}_1$, and $1_{\mathbb{G}_T}$ is the identity element in $\mathbb{G}_T$.

Apart from brevity, we chose to write our paper in symmetric pairings for two reasons. Looking ahead, the pairing-related components of Sipster do not require the decisional Diffie-Hellman (DDH) assumption to hold for the base groups, which asymmetric pairings could offer. Describing our scheme using asymmetric pairing might make a false impression that we also rely on DDH assumptions. In other words, Sipster can be instantiated with a wider variety of curves. Meanwhile, some might have an imprecise idea that a paper written in symmetric pairing groups must be insecure, if not incorrect. Even in the face of recent attacks against symmetric pairing groups, related intractability assumptions remain unbroken with appropriate security parameters, albeit affecting efficiency. Finally, while not every symmetric-pairing-based scheme can be ported to one using asymmetric pairing easily, many do [2, 21].

Signature Schemes. A signature scheme consists of three algorithms. The KGen algorithm takes as input the security parameter λ and outputs a pair of verification and signing keys (vk, sk). The Sign algorithm takes as input the signing key sk, a message $m \in \mathcal{M}$, and outputs a signature $\sigma \in \Sigma$. The Verify algorithm takes as input the verification key vk and a pair $(m, \sigma) \in \mathcal{M} \times \Sigma$ and outputs a bit indicating whether σ is valid signature for message m under vk.

3.2 Technical Overview and Specific Challenges

Despite the relatively extensive literature oncryptographic e-cash and related privacy-enhancing cryptographic techniques, the problem of privacy-preserving bill settlement is not straightforward.

3.2.1 Our Design Blueprint. The design blueprint is as follows. For each unit of power consumption, the SM outputs a random number R_i to the RU. Suppose at the end of the billing period t; the SM has output $R_1, R_2, \ldots, R_K$. For convenience, we presume each unit of power consumption corresponds to a unit payment. The RU should pay K payment units (*e.g.*, dollar) for the K power units consumed. The SM aggregates the set $\{R_i\}_{i=1}^K$ into a single one as $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$ with a certain function $F(\cdot)$, and signs the tuple $(R^{\mathsf{B}}_{\mathsf{ID},t}, \mathsf{ID}, t)$ that serves as the *final bill* for period t.

Before seeing the final bill, the RU pays the UC a unit of (anonymous) e-cash from time to time and asks the UC to sign one R_j it got from the SM. This signature acts as *a receipt of a payment unit.*

This design assumes the SM only outputs instead of taking any input. It thus excludes, particularly, an alternative design that only requires the SM to output one element R_1 – To ensure unlinkability, the function F now needs to repeatedly rerandomize some randomized version of R_1, which requires either the SM needs to take in some externally processed inputs (*e.g.*, signed by the UC) or the RU needs to maintain some state, both of which Sipster aims to avoid.

What remains is to instantiate our design with concrete details. We start with assuming a "powerful" smart meter as a baseline to illustrate what could not be done if we restrict to a "minimalistic" smart meter otherwise. We then discuss at the other end of the spectrum and formulate a "powerful" signature scheme for solving our problems. The goal is to illustrate that a *purely* cryptographic solution without any trust assumptions seems to be far-fetched. These two approaches form a concrete basis to argue for the technical challenges we encounter and resolve when designing Sipster.

3.2.2 Baseline Powerful-Smart-Meter Approach. Suppose that the SM is a *powerful* and *fully trusted* processor, a "simple" solution exists – The SM records all the random numbers it has issued. In the bill verification step, the RU presents all the signatures ever received from the UC to the trusted SM, who then checks if all the locally stored random numbers have a corresponding signature. If so, the SM simply certifies that the user has settled the bill.

This solution seems to work, except that the bill verification at the SM is $O(K)$ but not $O(1)$. Also, recall that each verification is on a random number. For verifying $O(K)$ signatures, the SM can deterministically re-generate them on the spot from a secret seed or store them in static storage. Either option implies a *linear-size (in K)* tamper-proof and rewritable (volatile) storage space. Linear-size storage seems trivial for typical computing devices but not resource-constraint smart meters. $O(1)$ storage would be desirable.

The core issue is that having the SM *actively participating* in bill verification incur various substantial implications in the hardware requirements. In particular, it requires the SM to have *an interface for feeding in signatures* instead of a typically passive device merely outputting readings. Having an *input interface* is risky. Various forms of attack exploiting *malicious inputs* have been shown in various kinds of systems. It simply broadens the attack surface that may compromise the integrity of the meters.

For economical mass production, each SM should be as minimal as possible. Practical deployment needs to minimize the trust assumptions and the functional requirements, as well as reduce the computation/memory cost of bill verification within the meter.

3.2.3 Unavailability of Cryptographic Tools. To free the SM from taking processed inputs from the UC and further processing them, our hope turns to the other major component, the signature scheme used by our UC for issuing receipts of a unit payment. For our design to work, we hope it possesses the following ***malleability*** – given signatures $\{\sigma_i\}_{i=1}^{K}$ on $\{R_i\}_{i=1}^{K}$, respectively, the RU can compute a signature on $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$. Suppose both the signature size and output length of $F(\cdot)$ are constant (independent of K). Further, suppose the malleability of the signature scheme is *restricted*: without signatures of all R_i, the RU cannot compute a signature on $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$. Then, by showing the UC this special signature together with a *normal* signature from the SM on the tuple $(R^{\mathsf{B}}_{\mathsf{ID},t}, \mathsf{ID}, t)$, the RU shows that the bill for the time period t has been settled without revealing K.

Assuming such a signature scheme exists, this design seems secure at first glance and has several nice features. Firstly, the workload of SM is low. The SM does not actively involve in bill verification. It only issues random numbers and *standard signatures* on them. Secondly, bill verification time is constant. The UC only needs to verify two constant-size signatures. Thirdly, the solution does not involve any zero-knowledge proof. Lastly, this design works for any CPS as long as the smart meter is tamper resistant. It is also independent of the (e-cash) payment method the RU used.

Indeed, our final solution follows the above design footprint. However, there are non-trivial challenges (or open problems) and a few caveats that we need to tackle to make it work.

To start with, the UC should not be allowed to see $R_1, \ldots, R_K$ in clear; otherwise, it can compute the aggregate function $F(\cdot)$ on different subsets of $\{R_i\}$ it received from different RUs in a "mix-and-match" manner to see which subset leads to a particular $R^{\mathsf{B}}_{\mathsf{ID},t}$. If there is a match, the UC can at least learn the payment amount K of the user ID. This attack remains valid even if $F(\cdot)$ is one-way.

An immediate solution to make the UC being blind to R_i's is to let the UC sign on each R_i using *blind signatures*. Unfortunately, now the RU can cheat: instead of asking the UC to sign R_i, the RU asks for signing $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$ directly. Then, this cheating user only needs to pay one unit of e-cash.

Another technical problem is that we are not aware of any efficient signature schemes that possess the "malleability" we need. Aggregate signatures [13] certify a number of messages while the signature size is constant. But the verification (done by the UC in our context) takes as input all the aggregated messages, *i.e.*, $R_1, R_2, \ldots, R_K$, which breaks privacy, let alone we still need to equip them with blind signing functionality. A related attempt aims for feautes similar to aggregate blind signature functionality [49]. Interestingly, it also relies on some trusted parameter generation. We defer a more in-depth discussion and comparison to Section 6.4.

Other "multi-party" blind signature variants are not likely to be relevant. For example, in threshold (partially) blind signature schemes (*e.g.*, [24]), *a single message* is signed by *multiple signers* in a way that the partial signatures can be "aggregated" into a single one, which does not match our usage since the receipts (of unit payments) in our setting should be signatures from the same authority (UC) on different messages.

Homomorphic signatures seem to be a better fit. However, most of them (*e.g.* [17]) allow *any linear combination* of signatures. A cheating user can then multiply an arbitrary constant to the message-signature pair to derive a new message-signature pair, and claim to have paid more. Lattice-based (and less practical) homomorphic signatures (*e.g.* [18]) may allow certifying the aggregation function (which rules out arbitrary operations such as linear combinations), yet, the privacy guarantee of the function is often *all or nothing*, *i.e.*, either the function is revealed in plaintext, or "complete context hiding" such that the function is hidden. In our context, it means either the number of inputs of the function, which corresponds to the number of signatures (as receipts), is revealed; or the function is arbitrary such that an adversarial user can claim more than what has been actually paid. Furthermore, we still need to add the blind-signing functionality while disallowing user cheating attacks.

4 SIPSTER: OUR PROPOSED SYSTEM

4.1 Improving the Baseline Solution

Insights. We adopt a hybrid approach that leverages a minimum degree of trust in smart meters. We observe that for any smart metering system to work in practice, certain trust assumptions in the smart meter are unavoidable. For example, the smart meter should be tamper-proof; otherwise, users can simply modify the hardware to pay less. For another example, the authenticity of the meter readings should be guaranteed in some way (*e.g.*, the smart meter signs them); otherwise, no one would trust the readings.

Even with these two minimal trust assumptions, it still seems miles away from the malleability we desire. We derive a few insightful ways to "exploit" what can be offered by a limited set of supported cryptographic operations. Firstly, the group element output by the meter can be treated as one obtained from a common reference string with an unknown discrete logarithm. In this way, the group elements have some implicit cryptographic structure that aids us in attaining cryptographic security outside of the trusted computing base. Furthermore, we assign them a semantic meaning, denoting a unit amount. Meanwhile, we restrict the meaning of their operations. For example, for group elements R_1 and R_2, their product denotes an amount of two units, but their semantic meanings is not overloaded further than that, *e.g.*, $R_1^{1/2}$ will not be encoding half a unit. All these insights collectively lead to Sipster. To the best of our knowledge, we are not aware of any related privacy-preserving payment systems sharing these insights.

Minimizing Fancy Cryptography. Now we are ready to describe our two modifications to our baseline design, relying on the SM to perform random group element generation, simple modular multiplication and exponentiation, and signing on the group element. Firstly, the SM still generates a random (group) element R_i for each unit of power consumption, but instead of sending R_i to the RU directly, it masks the element with a random blinding factor r_i. The SM sends the blinded element $\tilde{R_i}$ with a digital signature on it. The RU forwards the blinded element and its signature to the UC, who returns a "malleable signature" on the blinded element if and only if the SM's signature is valid. Thus, only the SM can unblind the group element, and the RU does not have the freedom to ask the UC to sign on an arbitrary element, hence preventing cheating attacks.

The rest essentially follows the original design blueprint. We will build a scheme such that the RU can compute a signature on $\tilde{R}^{\mathsf{B}}_{\mathsf{ID},t} = F(\tilde{R_1}, \tilde{R_2}, \ldots, \tilde{R_K})$, where all inputs to $F(\cdot)$ are blinded elements. The SM incrementally aggregates all the blinding factors as well as the random elements as its *constant-size* local state. Such information is given to the RU at the end of the billing period to unblind the signature on $\tilde{R}^{\mathsf{B}}_{\mathsf{ID},t}$ into a signature on $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$.

Here, we make other observations that lead to our second improvement. Note that the "cheating capability" of RU is very limited: it only sees signatures on $\tilde{R_1}, \tilde{R_2}, \ldots$ chosen uniformly at random by the SM. To argue security of the above revised solution, we only need to show that RU cannot compute a signature on $R^{\mathsf{B}}_{\mathsf{ID},t} = F(R_1, R_2, \ldots, R_K)$ when it missed one or more signatures on $\tilde{R_i}$. As a result, a full-fledged malleable signature scheme seems to be an overkill. Our second modification is to replace the malleable signature with a tailor-made and surprisingly simple scheme.

Finally, the SM just acts "spontaneously" according to the change in meter reading, instead of being a reactive device as needed in the powerful smart meter approach we just outlined.

4.2 High-Level Ideas of Sipster

Within each billing period, the SM records the real-time consumption of the RU to calculate the fine-grained charge on-the-fly. For each unit to be charged, the SM prepares a *fresh* random group element R and a random exponent r. The SM signs (*e.g.*, using ECDSA) $\tilde{R} = g^r R$, which generates $\sigma_{\tilde{R}}$, and gives the token $\mathsf{tk} = (\tilde{R}, \sigma_{\tilde{R}})$ to the RU. Let $\{R_i\}$ and $\{r_i\}$ be the set of random group elements and exponents used in this period so far. The SM locally maintains the aggregated blinding factor $r_\tau = \sum r_i$ and the aggregated random group element $R_\tau = \prod R_i$, which will be reset in the next period.

The RU settles the bill in units as follows. After paying the UC a unit of e-cash, the RU presents a token $\mathsf{tk}_i = (\tilde{R_i}, \sigma_{\tilde{R_i}})$ and requests a "signature"[6] on $\tilde{R_i}$. The "signature" that the RU receives from the UC serves as a receipt for the payment.

At the end of the period t, the SM generates the bill $\mathsf{bill}_{\mathsf{ID},t} = (R_\tau, \mathsf{ID}, t)$ for the RU with identity ID and signs it using a standard (*e.g.*, ECDSA) signature scheme. The signature $\sigma^{\mathsf{B}}_{\mathsf{ID},t}$ and the aggregated blinding factor r_τ are sent to the RU. To show that the bill for the period t has indeed been settled, the RU aggregates all the "signatures" received from the UC and uses r_τ from the SM to unblind the aggregated "signature." The RU then presents $R_\tau, \sigma^{\mathsf{B}}_{\mathsf{ID},t}$ and the unblinded aggregated "signature" above to the UC.

Mindful readers may notice that the "signature" generated by the UC bears the properties of blind signature [22] and aggregate signature [13]. However, it is not a standalone full-fledged "aggregate blind signature scheme." Instead, we leverage the minimum trust in the SM to tailor-make an efficient signature-like scheme whose security guarantee is enough for our specific purpose. Furthermore, such a trust assumption allows us to prove the security of our incredibly simple scheme in the *standard model*. In contrast, many efficient signature schemes are only proven secure in the *random oracle model*. Hence we can instantiate this part of the system efficiently with smaller parameters with the same security level.

[6]Technically, this is not a standard EUF-CMA-secure signature scheme (Definition 1, Appendix B) due to the trusted nature of the SM, but it still suffices for our purpose.

Algorithm 1 TokenGen (with internal state τ) by the SM

```
Require: $(\mathsf{sk}_{\mathsf{SM}}, \tau)$
Ensure: $(\mathsf{tk}, \tau)$
1: procedure
2:     Parse $\tau$ as $(r_\tau, R_\tau)$
3:     $r \leftarrow \mathbb{Z}_p^*$, $R \leftarrow \mathbb{G}_1$                ▷ Choose fresh randomness
4:     $\tilde{R} := g^r R$, $\sigma_{\tilde{R}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{SM}}, \tilde{R})$, $\mathsf{tk} := (\tilde{R}, \sigma_{\tilde{R}})$
5:     $r'_\tau := r_\tau + r$; $R'_\tau := R_\tau \cdot R$
6:     $\tau := (r'_\tau, R'_\tau)$                ▷ Update the internal state $\tau$
7:     return $(\mathsf{tk}, \tau)$
8: end procedure
```

4.3 Formal Descriptions

We naturally divide our Sipster construction into four conceptual phases: setup, bill issuing, bill settlement, and bill verification. Setup (resp. bill verification) happens strictly before (resp. after) the billing period. However, note that the bill issuing and settlement phases can happen during the billing period at (essentially) the same time.

Setup Phase:

Smart meters SM: The manufacturer runs the KGen algorithm of a digital signature scheme $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ to get a verification/signing key pair $(\mathsf{vk}_{\mathsf{SM}}, \mathsf{sk}_{\mathsf{SM}})$ for the SM. The signing key $\mathsf{sk}_{\mathsf{SM}}$ is hard-coded in the SM, and the verification key $\mathsf{vk}_{\mathsf{SM}}$ is made public. (As in Section 2.3, one could use a group signature scheme.)

The utility company UC: It chooses groups $\mathbb{G}_1$ and $\mathbb{G}_T$ of order p (a λ-bit prime), which admit a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, chooses a random secret exponent $\alpha \leftarrow \mathbb{Z}_p^*$, and a generator $g \in \mathbb{G}_1$. It then publishes public parameters $\mathsf{PP}_{\mathsf{UC}} = (\mathbb{G}_1, \mathbb{G}_T, g, g^\alpha)$, and locally stores the secret value $\mathsf{sk}_{\mathsf{UC}} = \alpha$. In practice, the UC can use a different (vk, sk) pair for different periods at ease.

We follow the original work of Boneh *et al.* [14] in using symmetric pairings, which is known to work for asymmetric pairings as well [21]. Notably, Sipster does not require a full-domain hash.

Residential users RU: each is assigned a unique identity ID.

Bill Issuing Phase (SM $\rightarrow$ RU**):**

At the beginning of the period t, the SM initializes its internal state $\tau = (r_\tau, R_\tau) = (0, 1_{\mathbb{G}_1})$. Within a billing period t, the SM records the real-time consumption of the RU and calculates the fine-grained expense. For each unit of expense, the SM runs the stateful algorithm TokenGen (Algorithm 1) to get the token $\mathsf{tk} = (\sigma_{\tilde{R}}, \tilde{R})$, and forwards it to the RU. The SM's internal state τ is also updated.

Bill Settlement Phase (RU $\leftrightarrow$ UC**):**

For each token tk received in the bill issuing phase, the RU can settle the bill for this particular token immediately with the following steps. Therefore, this phase can overlap with the bill issuing phase significantly. For each tk, the RU engages in the following protocol:

RU $\rightarrow$ UC: RU presents a token tk to UC and pays a unit of e-cash.

Algorithm 2 ReceiptGen by the UC

```
Require: (tk = ($\tilde{R}, \sigma_{\tilde{R}}$), vk$_{SM}$, sk$_{UC}$ = $\alpha$)
Ensure: rcpt$_i$
1: procedure
2:     Parse tk as ($\tilde{R}, \sigma_{\tilde{R}}$)
3:     if Verify(vk$_{SM}$, ($\tilde{R}, \sigma_{\tilde{R}}$)) = 0 then
4:         return $\perp$
5:     else
6:         return rcpt$_i$ = $\tilde{R}^{\alpha}$.
7:     end if
8: end procedure
```

UC → RU:

(1) The UC outputs $\mathsf{rcpt}_i \leftarrow \mathsf{ReceiptGen}(\mathsf{tk}, \mathsf{vk}_{\mathsf{SM}}, \mathsf{sk}_{\mathsf{UC}})$ (Algorithm 2) after payment confirmation.
(2) If $\mathsf{rcpt}_i \neq \perp$, the UC sends it to the RU; otherwise, it aborts.

The RU checks $e(\mathsf{rcpt}_i, g) \overset{?}{=} e(\tilde{R}, g^{\alpha})$. If not, the RU aborts.[7]

Bill Verification Phase (SM → RU, RU → UC):
Suppose at the end of the billing period t, the RU with identity ID is charged by K units of e-cash. Then, the SM must have issued K tokens $\{\mathsf{tk}_i\}_K = \{\tilde{R}_i, \sigma_{\tilde{R}_i}\}_K$ to the RU. If the RU had paid K units of e-cash to the UC, he would have received K receipts $\{\mathsf{rcpt}_i\}_{i\in K}$.

This is the only phase in which the RU reveals the identity ID to the utility company, perform bookkeeping locally (and reminding the other customers who did not settle the bill).

SM → RU: The SM executes BillGen (Algorithm 3) to generate the bill for the RU whose identity is ID. The SM forwards the output of BillGen, *i.e.*, bill = $(\tau, m_{\mathsf{ID},t}, \sigma^{\mathsf{B}}_{\mathsf{ID},t})$, to the RU. Here, τ is the internal state of the SM, which consists of the aggregated random exponent and aggregated random element (See Algorithm 1). $\sigma^{\mathsf{B}}_{\mathsf{ID},t}$ is a signature on the billing information $m_{\mathsf{ID},t}$.

RU → UC: Suppose the RU has settled the bill bill = $(\tau,\ m_{\mathsf{ID},t},\ \sigma^{\mathsf{B}}_{\mathsf{ID},t})$ from the SM, *i.e.*, received K receipts $\{\mathsf{rcpt}_i\}_{i\in K}$ from the UC. The RU executes CombineReceipt(bill, $\{\mathsf{rcpt}_i\}_{i\in K}$, PP$_{\mathsf{UC}}$) (Algorithm 4) to obtain $\sigma^{\mathsf{R}}_{\mathsf{ID},t}$. Then, the RU presents the proof $\pi_{\mathsf{ID},t} = (m_{\mathsf{ID},t}, \sigma^{\mathsf{B}}_{\mathsf{ID},t}, \sigma^{\mathsf{R}}_{\mathsf{ID},t})$ to the UC, who verifies it by checking whether $\mathsf{Verify}(\mathsf{vk}_{\mathsf{SM}}, (m_{\mathsf{ID},t}, \sigma^{\mathsf{B}}_{\mathsf{ID},t})) = 1$ and $e(\sigma^{\mathsf{R}}_{\mathsf{ID},t}, g) = e(R_{\mathsf{ID},t}, g^{\alpha})$, where $R_{\mathsf{ID},t}$ is a part of $m_{\mathsf{ID},t}$. If both hold, the UC is convinced that the RU with identity ID has settled the bill for the period t.

We remark that while the SM uses the same signing key sk$_{\mathsf{SM}}$ to create both $\sigma^{\mathsf{B}}_{\mathsf{ID},t} = \mathsf{Sign}(\mathsf{sk}_{\mathsf{SM}}, m_{\mathsf{ID},t})$ here and $\sigma_{\tilde{R}} := \mathsf{Sign}(\mathsf{sk}_{\mathsf{SM}}, \tilde{R})$ in TokenGen, their respective message spaces are different, and the UC will distinguish two different kinds of signatures.

If everything was executed honestly and with the needed verifications, the RU will convince the UC in the bill verification phase, which follows from the correctness of the signature scheme $\mathcal{SS}$ and the equality below:

[7]A malicious UC can always refuse to return a receipt after receiving a coin. This can be generally resolved by requiring a fair exchange protocol. The RU may report any misbehavior to some third-party arbitrator.

Algorithm 3 BillGen (with internal state τ) by SM

```
Require: (sk$_{SM}$, ID, $t$, $\tau$)
Ensure: bill
1: procedure
2:     Parse $\tau$ as ($r_{ID,t}, R_{ID,t}$)
3:     $m_{ID,t}$ := ($R_{ID,t}$, ID, $t$)        ▷ Prepare the bill information
4:     $\sigma^{B}_{ID,t}$ := Sign(sk$_{SM}$, $m_{ID,t}$)
5:     return bill := ($\tau, m_{ID,t}, \sigma^{B}_{ID,t}$)
6: end procedure
```

Algorithm 4 CombineReceipt by RU

```
Require: (bill, {rcpt$_i$}$_{i\in K}$, PP$_{UC}$)
Ensure: $\sigma^{R}_{ID,t}$
1: procedure
2:     Parse bill as ($\tau, m_{ID,t}, \sigma^{B}_{ID,t}$)
3:     Extract $r_{ID,t}$, $g^{\alpha}$ from $\tau$ and PP$_{UC}$, respectively
4:     $\sigma^{R}_{ID,t}$ := ($\prod_{i=1}^{K}$ rcpt$_i$)/($g^{\alpha}$)$^{r_{ID,t}}$
5:     return $\sigma^{R}_{ID,t}$
6: end procedure
```

$$
\begin{aligned}
e(\sigma^{\mathsf{R}}_{\mathsf{ID},t}, g) &= e((\prod_{i=1}^{K} \mathsf{rcpt}_i)/(g^{\alpha})^{r_{\mathsf{ID},t}}, g) \\
&= e(\prod_{i=1}^{K} (g^{r_i} R_i)^{\alpha} / g^{\alpha r_{\mathsf{ID},t}}, g) \\
&= e(g^{\sum_{i=1}^{K} r_i} \prod_{i=1}^{K} (R_i)/g^{r_{\mathsf{ID},t}}, g^{\alpha}) \\
&= e(\prod_{i=1}^{K} R_i, g^{\alpha}) && \triangleright \text{ since } r_{\mathsf{ID},t} = \sum_{i=1}^{K} r_i \\
&= e(R_{\mathsf{ID},t}, g^{\alpha}). && \triangleright \text{ since } R_{\mathsf{ID},t} = \prod_{i=1}^{K} R_i
\end{aligned}
$$

4.4 Flexible Payment and Fault Tolerance

Each random element R_i corresponds to a unit payment. We can easily extend it such that the RU can choose to pay more for each time. Suppose ten dollars is also a valid denomination. The RU can choose to hide among the set of users who pay ten dollars at a time.

Sipster can handle non-malicious malfunction of either the RU or the SM with some small twists. In the above description, if one payment/receipt for a particular R_i term is missed, all the other receipts will become useless in the final aggregated receipt verification. To add fault tolerance, instead of choosing (r_i, R_i) uniformly at random (or forcing each SM to store every single pair (r_i, R_i)), the SM computes $(r_i, R_i) = \mathsf{PRF}(k_{\mathsf{PRF}}, (\mathsf{ID}||i))$, where k_{PRF} is a hard-coded key for the pseudorandom function $\mathsf{PRF} : \{0,1\}^{|k_{\mathsf{PRF}}|} \times \{0,1\}^* \rightarrow \mathbb{Z}_p^* \times \mathbb{G}_1$. The RU can request the SM for the missing i-th token and re-pay the UC. This resilience feature requires the SM to have a simple interface for keying in a number. The internal state of the SM can also store the total number of receipts issued so far.

If the SM stops functioning in the middle of a billing period, we can make further modifications so that the RU does not need to pay the entire bill again. Namely, the internal state of the SM also stores an "interim" signature σ_{interim} on $m_{\text{interim}} = (R_\tau, \text{ID}||\text{"interim"}, t)$. When the need arises, the RU requests the smart meter company staff to fix the meter physically and retrieve this special signature and the internal state r_τ before resetting the SM to the initial state. The RU completes the proof almost as usual, but now the ID is appended with a tag "interim" in clear. The UC is then assured that the RU had paid the latest amount up to the breaking point.

In the very rare case, the ("trusted") SM completely fails, and even σ_{interim} is lost too, any financial loss can still be resolved by pre-defined agreements or different dispute resolution procedures independent/transparent to Sipster. For example, if the meter is deemed faulty by the staff, the smart-meter company is responsible; otherwise, the RU should compensate for the loss and possibly the checking and fixing fees. In practice, there are usually collateral requirements on the RU (*e.g.*, originally for late payment), and the UC probably has some waiving/capping policy too.

4.5 Security Analysis

Section 2.3 identified three goals for Sipster: privacy, unforgeability, and double-spending prevention. Accordingly, we formulate two security notions for our Sipster systems: privacy (Definition 5) and soundness (Definition 7, covering both unforgeability and double-spending prevention), via cryptographic games between a challenger and an adversary, who has oracle access to various algorithms. We describe the intuition behind the formalism below and leave the formal details in Appendix C.

Intuitively, with our specific design, and the typical blind signing trick in the underlying signature scheme, the UC only sees and signs (pseudo)random and unlinkable tokens and verifies a constant-size proof of correct-bill-settlement, all these can be simulated without any secret knowledge nor any identifying information of any payer.

The intuition for soundness is more difficult to explain in purely high-level terms as it largely depends on the cryptographic construction. Intuitively, while the application settings are different, the core arguments for unforgeability are similar to the implicit underlying aggregate signature [13]. Likewise, for double-spending prevention, it roots from the unforgeability of the user identity component and the "restricted signing functionality" that only works on the randomness chosen by the trusted meter, but not adversarially-chosen values (as they cannot be fed to the meter).

Remarks on Usages of Pairings. The use of asymmetric primitives such as pairing enables public verifiability, particularly over the tokens generated by the UC and the proof of bill-settlement generated by the RU. This comes in handy in dispute resolution.

If asymmetric pairing group $(\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle, \mathbb{G}_T)$ is preferred, Sipster can be rewritten in the asymmetric setting easily. Specifically, g_1^α can be released for $\mathsf{CombineReceipt}$ (Algorithm 4) and g_2^α can be released for UC verification. The security proofs will remain more or less the same, except Theorem 2 will rely on the co-gap-DH assumption [14].

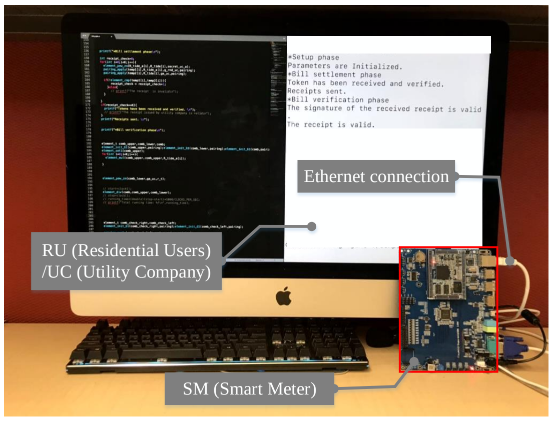
```
*Setup phase
Parameters are Initialized.
*Bill settlement phase
Token has been received and verified.
Receipts sent.
*Bill verification phase
The signature of the received receipt is valid
.
The receipt is valid.
```

Figure 1: An Illustration of the Experimental Setup

5 PERFORMANCE EVALUATION

We evaluate the computation and communication performances of the proposed Sipster system to show its practicability. Fig. 1 shows the experimental setups of Sipster: The SM is built on a testbed with a 1.5GHz ARM Cortex-A9 processor and 1GB RAM, reflecting the moderate computation power of a typical smart meter.[8] Both the RU and UC are implemented on PCs with a 3.4GHz Intel-i5 CPU and 32GB RAM. The communication among them is established via local Ethernet with 89.6Mbps bandwidth.

We instantiate the signature used by the SM by ECDSA [41] from OpenSSL [56]. Its elliptic curve is over a prime field of $n = 256$ bits. As a prototype, we use the Pairing Based Cryptography Library[9] with a "Type A" elliptic curve generated for 256-bit group order and 512-bit base field. One could easily port it over other curves, such as "Type-F" curves. We follow a basic implementation without any optimization, *e.g.*, pre-computation of some pairing values, or speeding up exponentiation for those sharing the same base, which will further decrease the timing figures. Table 1 summarizes the time needed for all basic operations. We report the performance in three dimensions: (aggregated) computation time (Section 5.1), service latency (Section 5.2), and communication cost (Section 5.3). We omit the underlying (e-)cash system since it is not part of our design, and any such system can do. All experimental results represent an average of 10000 trials.

5.1 Computation Performance

Sipster has four phases: setup, bill issuing, bill settlement, and bill verification. Setup is carried offline only once at the system

[8] Cortex A-series has been adopted, *e.g.*, STMicroelectronics offers energy SM products with Cortex-A9 as the control unit (https://www.st.com/content/ccc/resource/sales_and_marketing/promotional_material/brochure/eb/29/0b/3e/a3/7a/4b/7d/brmeter.pdf/files/brmeter.pdf/jcr:content/translations/en.brmeter.pdf). It was first released in 2013. That means the great performance (in the order of milliseconds) of Sipster is demonstrated using an old platform. We also remark that smart meters in deployment do not need the generic interfaces and peripheral devices offered as a development board, meaning the cost of practical deployment per smart meter would be much less than the consumer market price for the board before economies of scale kick in.
[9] http://crypto.stanford.edu/pbc

Table 1: Computation Time per Operation

Operation	Pairing	Addition	Exponentiation	Multiplication	Division	ECDSA Sign	ECDSA Verify
Shorthand	pairing	add	exp	mul	div	sign	verify
SM Comp. Time (ms)	17.567 $\pm 2.601 \times 10^{-4}$	0.997×10^{-4} $\pm 1.181 \times 10^{-6}$	12.898 $\pm 5.06 \times 10^{-4}$	3.366×10^{-2} $\pm 3.164 \times 10^{-6}$	3.047×10^{-2} $\pm 1.532 \times 10^{-6}$	1.866 $\pm 1.992 \times 10^{-4}$	2.148 $\pm 6.48 \times 10^{-4}$
RU/UC Comp. Time (ms)	1.675 $\pm 1.365 \times 10^{-4}$	0.162×10^{-6} $\pm 1.081 \times 10^{-6}$	1.196 $\pm 4.05 \times 10^{-6}$	3.265×10^{-3} $\pm 0.982 \times 10^{-6}$	3.932×10^{-3} $\pm 1.135 \times 10^{-6}$	0.277 $\pm 1.805 \times 10^{-4}$	0.366 $\pm 3.92 \times 10^{-4}$

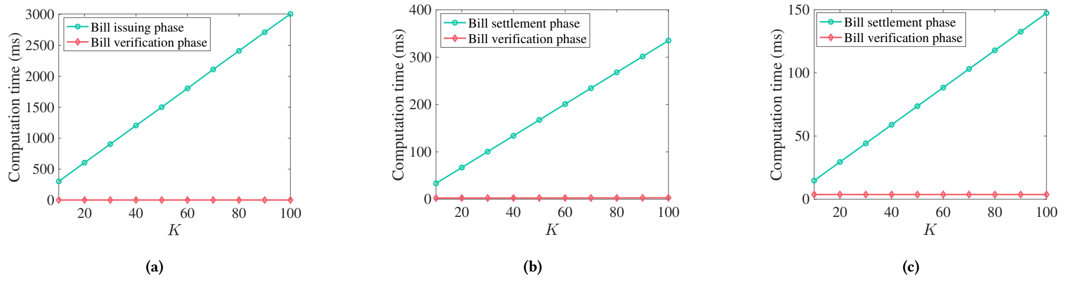

Figure 2: Computation Time at (a) SM, **(b)** RU, **and (c)** UC

initialization[10], so we focus on measuring the performances of the three online phases for different entities, namely, SM, RU, and UC.

As discussed in Section 2.3.2, the parameter K, denoting *the bill amount that the* RU *has to pay*, which manifests in different phases and affects the performance most. In the bill issuing phase, the computation at the SM includes generating $\tilde{R}_i$, updating internal state τ, and computing the signature $\sigma_{\tilde{R}_i}$. Thus, its computation complexity contains $K \times$ exp, $2K \times$ mul, $K \times$ add, and $K \times$ sign. Moreover, the SM also takes charge of generating billing information for RU in the bill verification phase. Its computation complexity is dominated by the generation of signature $\sigma^{\mathsf{B}}_{\mathsf{ID},t}$, which is $1 \times$ sign.

Fig. 2(a) depicts its computation time under different K's. Apparently, the computation time increases linearly as K grows. When $K = 100$, the SM's computation time reaches around 3s in total, which is the *total* amount of time required to generate all K tokens. Nevertheless, these tokens are processed only once in a while. The estimated total amount of time should be spread over for the unit of granularity for fine-grained bill settlement. So, it should not be interpreted as a 3s delay at the SM. See Section 5.2 for details.

For the RU, in the bill settlement phase, its computation includes $2K \times$ pairing. Its computation in the bill verification phase includes aggregating over rcpt_i to obtain the combined receipt $\sigma^{\mathsf{R}}_{\mathsf{ID},t}$, which consists of $(K-1)\times$mul, $1\times$div, and $1\times$exp. Fig. 2(b) shows how RU's computation time varies with bill amount K in each phase. An RU needs to compute the pairing results of its own records $\{(\tilde{R}_i, g^{\alpha})\}$ and the UC's receipts $\{(\mathsf{rcpt}_i, g)\}$ whose sizes are dependent on K. For example, when $K = 10$, the RU's computation takes 2.477ms, and it increases to 3.031ms as K grows to 60. Even though the computation time of the bill verification phase is linear in K, its slope is rather flat compared with the bill settlement phase.

[10]The setup phase is quite minimal. There is no user setup. The UC setups a bilinear group context and a key pair. The SM should also have equipped with a key pair.

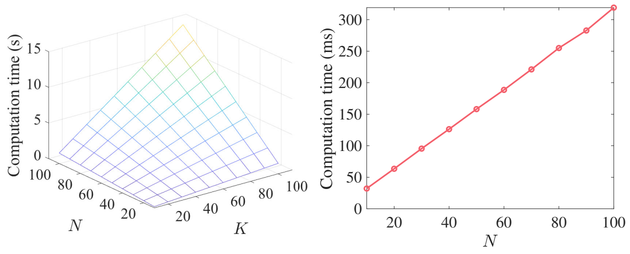

Figure 3: (a) Computation Time at the UC **in Bill Settlement, (b) Computation time at the** UC **in Bill Verification**

For the UC, its computation in the bill settlement phase includes verifying K received tokens $\mathsf{tk}_i = (\tilde{R}_i, \sigma_{\tilde{R}_i})$ and computing K receipts: $\mathsf{rcpt}_i = \tilde{R}_i^{\alpha}$, which correspond to $K \times$ verify and $K \times$ exp operations. In the bill verification phase, its computation complexity is $1\times$verify and $2\times$pairing. Fig. 2(c) depicts the relation between bill amount K and the UC's computation time in different phases. We observe that in the bill settlement phase, the computation time increases linearly as K grows, while the bill verification indicates its computation time, *i.e.*, 3.726ms, is independent of K. This confirms the expectation that the number of operations of the UC's bill verification phase is fixed and independent from the bill amount K.

We then consider another parameter N, denoting *the number of* RU*s handled by the* UC. Fig. 3(a) depicts the UC's computation time in the bill settlement phase with different N's, assuming that all RUs share the same K. We notice that the computation time is a linear function of both K and N. Specifically, when $K = 100$ and $N = 100$, the total computation time at the UC for generating receipts is around 13.5s, which is still a relatively short duration compared with the billing granularity, say one hour. We believe that the computation performance at the UC can be further improved

Table 2: Computation Performance

	Computation Complexity
SM	$K \times$ exp, $2K \times$ mul, $K \times$ add, $K \times$ sign, $1 \times$ sign
RU	$2K \times$ pairing, $(K-1) \times$ mul, $1 \times$ div, $1 \times$ exp
UC	$K \times$ verify, $K \times$ exp, $1 \times$ verify, $2 \times$ pairing

Table 3: Service Latency of Sipster

	Issuing	Settlement	Verification
Latency (ms)	34.272 ± 2.807	5.665 ± 0.819	9.212 ± 0.826

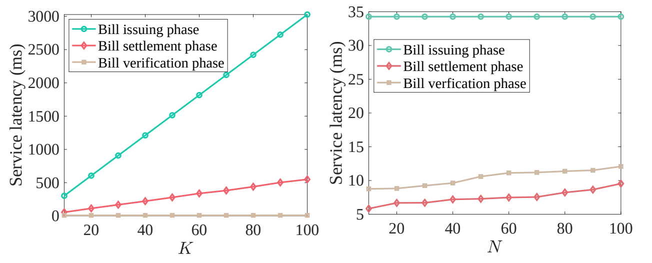

Figure 4: (a) Service Latency under Different K (b) or N

by adopting powerful server clusters or cloud computing services. Recall that UC is implemented on a PC with moderate computation capability in our experiments. Fig. 3(b) demonstrates the relation between computation time and N in the bill verification phase. It increases linearly as N grows.

To sum up, for a fixed bill amount K, the SM has the longest computation time while the RU has the shortest. This is because the SM typically is a low-cost micro-computation unit with much lower computation capacity. However, as observed in Fig. 5(a), most bills are handled within a very short time; the longest service latency is less than 3s, which is negligible to RUs. Naturally, the UC experiences the longest computation time when the number of RUs is relatively large. In real-world implementations, it can be greatly shortened by employing computation unit clustering. Table 2 summarizes the computation complexity.

5.2 Service Latency

We also evaluate the service latency of Sipster in terms of the overall delay in each of the three phases: bill issuing, bill settlement, and bill verification. We focus on computation but not communication since our protocols are 2-message protocols.

Table 3 shows the service latency by considering one RU. Particularly, for the bill issuing and the bill settlement phases, we examine the total computation time for handling one unit of expense, *i.e.*, $K = 1$. For the bill verification phase, K is set to 100, indicating that the RU consumes total utility at expense K for one billing period. We set $K = 1$ for the first two phases because the SM generates a token and forwards it to the RU once each unit of expense is incurred, regardless of how much expense the RU causes in total in one billing period. The service latency of bill issuing (34.272ms) is the largest among the three phases. Its most time-consuming operation is for the RU generating the tokens (tk, τ).

Fig. 4(a) depicts the average service latency with respect to K when $N = 1$. The bill issuing is executed by the SM. Its service

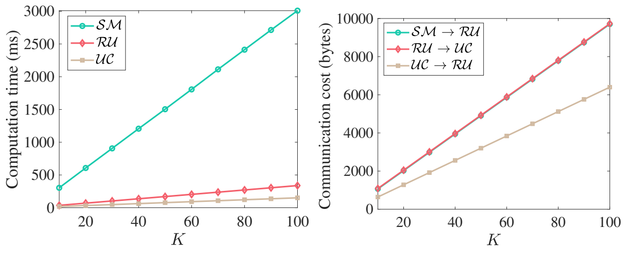

Figure 5: (a) Computation Time, (b) Communication Cost

latency is proportional to K for the fine-grained billing. Note that these bills are not necessarily generated all at once. They can be from multiple cycles. In this case, the service latency for bill amount K reflects the cumulative latency over multiple cycles.

The service latency for bill settlement also increases since the RU performs more pairing operations to verify the receipts generated by Algorithm 2. For bill verification, its latency slightly increases with K. For example, the latency is 8.701 ± 0.352ms when $K = 20$, and it is 9.212 ± 0.826ms when $K = 100$. This slight increase is due to more receipts being combined (generation of $\sigma^{\mathrm{R}}_{\mathrm{ID},t}$ in Algorithm 4). We remark that the receipt combination takes place in the RU side and does not require computation from the UC.

Fig. 4(b) shows the impact of N, the number of RUs. In the code, we create a separate thread for handling operations from each RU at the UC side. Thus, the UC processes RUs' requests in parallel. The bill issuing still takes a longer time for the signing process in a resource-constrained smart meter. Specifically, it is 34.272 ± 2.807ms when $N = 100$, while the latencies for bill settlement and verification are 9.541 ± 0.359ms and 12.08 ± 1.187ms, respectively. The latencies are acceptable even under a short billing period, say one hour. Besides, for bill verification, its latency can be further reduced by adopting a more powerful cluster at the UC.

5.3 Communication Performance

We then evaluate the transmitted payload data size between different entities. In our system, communication only occurs between the RU and the SM, and between an RU and the UC.

In the bill issuing phase, the SM transmits K tuples of $(\tilde{R}_i, \sigma_{\tilde{R}_i})$, totaling $K|n| + K|\mathbb{G}_1|$ bits, to the RU, where $|\mathbb{G}_1|$ denotes the size of an element in $\mathbb{G}_1$. For $|\mathbb{G}_1| = 512$ bits and $|n| = 256$ bits, the cost is $96K$ bytes. In the bill settlement phase, the RU forwards K tuples of $(\tilde{R}_i, \sigma_{\tilde{R}_i})$ to the UC, resulting in $96K$ bytes transmission. The UC replies with K receipts $\tilde{R}_i^{\alpha}$, which costs $K|\mathbb{G}_1|$ and translates to $64 \cdot K$ bytes. In the bill verification phase, the SM sends the RU $(\tau, \sigma^{\mathrm{B}}_{\mathrm{ID},t})$ with the communication cost of $|p| + |\mathbb{G}_1| + |n| = 104$ bytes as $|p| = 160$ bits. Then, the RU forwards $(R_{\mathrm{ID},t}, \sigma^{\mathrm{B}}_{\mathrm{ID},t}, \sigma^{\mathrm{R}}_{\mathrm{ID},t})$ to the UC, resulting in $|\mathbb{G}_1| + 2|n| = 128$ bytes of transmission. In total, the SM transmits $96K + 104$ bytes to the UC. The data transmitted from the RU to the UC is $96K + 128$ bytes, and the UC replies $64K$ bytes back.

Table 4: Communication Cost Performance

	Communication Cost (bytes)
SM $\rightarrow$ RU	$96 \cdot K + 104$
RU $\rightarrow$ UC	$96 \cdot K + 128$
UC $\rightarrow$ RU	$64 \cdot K$

Table 4 and Fig. 5 summarize the communication cost between different entities when K varies. Overall communication costs between each pair of entities are all in the order of 10^3 bytes, even when $K = 100$.

6 RELATED WORK

6.1 Cryptocurrency-based Solutions

Our approach of paying unit coins may have some resemblance to the Zerocoin [38] approach, where a user also first pays for units of coins (by publishing commitments of the coins on a public bulletin board) and proves later that the bill has been settled by proving the knowledge of the openings (blinding factors) to K of them. Technically, it does not solve our problem immediately. First, it requires continual updates on both the accumulator [16] of the coins and hence the witness for proving the knowledge of a coin stored in the commitment. A more serious problem is that the RU needs to prove the knowledge of K *distinct* openings. Its complexity is $O(K)$, and it is *unclear how to hide* the value K for this proof.[11]

Cryptocurrencies that support confidential transactions (*e.g.*, Monero[12] and Zerocash [10]) and specifically the privacy of the transactional amount seem to be the right tool for our problem. However, note that the amount privacy is protected from outsiders who oversee and maintain the underlying blockchain/consensus system, but not the payee. More seriously, the core problem, namely, privacy-preserving proof that a user has settled a bill, and having the payee return the acknowledgment, remains unsolved. One could imagine a complex zero-knowledge proof that somehow tightly couples the encryption of the total amount to encryption of the corresponding customer identity, and some secure-two-party computation that returns a signature on the identity. Even assuming it can be efficiently done, an integral part of it contains a witness-indistinguishable proof among all pending and certified amounts (otherwise, one can easily prove an imaginary bill of 0 amount has been settled), which is essentially a paraphrase of our problem.

6.2 Privacy-Preserving Billing in Smart Grids

The general goal for privacy-preserving billing aims to protect the privacy of fine-grained utility usage in the *bill generation* while remaining verifiable by the UC. To protect individual readings, Jawurek *et al.* [33] hide them by commitments and use their homomorphic property for computing a linear function, summing up the multiplication of the readings and the cost from tariffs/price schema for different periods. Rial and Danezis [48] further apply zero-knowledge proof to verify the correctness of the bill in each billing period without revealing it. Realizing that even the aggregated bill may leak individual values, Danezis *et al.* [27] use a differentially-private mechanism to add noise to the aggregated bill from the previous approach [48]. The noise serves as a configurable trade-off between payment accuracy and privacy level. Since the payment amount differs from the original one, they further design an oblivious payment protocol that allows the users to get rebates (in the amount of their noise) in future payments. Finally, Lin *et al.* [37] proposed privacy-preserving bill generation and load monitoring by using distributed pseudorandom functions [40], which have been utilized in distributed cryptosystems [20] and many other privacy-preserving aggregation schemes ([28, 51], also see Appendix A). Nevertheless, all of them did not consider hiding the *total amount* beyond adding noise, which is not an effective protection mechanism without causing too much error.

6.3 Privacy-Preserving Pricing in E-Tolling

Electronic toll collection system is another kind of CPS that makes use of trusted hardware on the car as a "meter" to record how much toll a driver should pay after passing through a certain place. There are quite a few privacy-preserving electronic toll pricing systems. VPriv [46] uses secure two-party computation (2PC) to ensure tolling privacy. To ensure users cannot cheat on the total toll price, the scheme asks them to upload their historical driving information, including license plate, location, and time. Meanwhile, a trusted authority needs to randomly record users' driving information as well and then challenge them with these records during payment. Apparently, it compromises the user privacy at the "checking spots." PrETP [9] is proposed with similar goals. Rather than using 2PC, the toll charger first collects homomorphic commitments from users. Then the toll service provider asks them to open the commitments of certain location-time tuples corresponding to its random spot-checks. Each user only reveals the payment amount and location-time tuples in the physical vicinity of random spot-checks.

Moreover, they prevent users from manipulating the toll price by spot checks, which reveal the price, inevitably violating privacy.

6.4 On "Aggregate Blind Signatures"

Aggregate signature [13] certifies a number of messages, and the number of messages reveals the payment amount. A recent aggregate signature scheme considers blind signing functionality [45]. It requires the signatures to be aggregated come from *different* signers, which does not match our need since there is only one company that issues receipts as signatures. In other words, aggregating signatures from a *single* signer remains a technical challenge.

The notion of aggregate blind signature, intuitively, is vulnerable to the following issue. When the messages can be hidden, a malicious user may *hide multiple messages to be aggregated* in a single blind-signing request. Any scheme should remain secure against such kinds of attacks. The closest related work we can find is a privacy-preserving payment system for public transport [49] with a refund feature. Similar to our approach, the above potential security issue is resolved by assigning an authenticated random group element to each of the signature requesters. Its core cryptographic mechanism appears to be a "history-free sequential aggregate blind signature" signing protocol over a restricted message set. A signer

[11] Alternatively, if one coin is proven each time, an additional two-party protocol is required to let the UC know that a particular customer has paid a coin. For example, the RU needs to present another signature from the UC and requests to update such a signature on a decreasing outstanding balance, both in zero-knowledge. This approach still takes $O(K)$ time and a high number of communication rounds.

[12] https://www.getmonero.org, http://ia.cr/2015/1098

can derive the next aggregate using only the previous aggregate (but not the history of previous messages and the corresponding public keys) apart from (the obvious inputs of) the current message to be signed and the signature key. In their application, all signatures to be aggregated come from the same signer, who cannot learn previous aggregates and the corresponding messages. Moreover, the aggregate signature verification is over an aggregate of the messages instead of all the messages, conveniently storing the amount to be refunded.

The signature requester needs to *maintain state information* for the *sequential* aggregation, which is slightly inconvenient in general. The most important distinction from our approach is that, an amount d is encoded as the power of a secret α as the exponent of the random group element. Correspondingly, to enable verification, the public key size of their scheme is as large as the magnitude of the aggregatable amount, *i.e.*, $O(K)$-size when the amount to pay is K unit dollars. With respect to the random group element, the "aggregate signature" is deterministic, meaning that a zero-knowledge proof is needed to hide the amount. In contrast, our approach is based on the collection of K signatures to aggregate instead of an explicit encoding of the value K, freeing us from the zero-knowledge proof. Our underlying primitive allows arbitrary (or *non-sequential*) aggregation and features a *constant-size* public-key independent of the maximum aggregatable amount.

Finally, we remark that it is of theoretical interest to propose an efficient aggregate blind signature scheme in the standard model without any trusted device.

7 APPLICATIONS IN MOBILE PAYMENT

To show the wide applicability of our Sipster system beyond the context of cyber-physical systems, we describe how it can be applied to mobile payment systems to enhance payment privacy. We note that some current mobile payment systems also rely on the TPM in the cellphone, but for payment security instead of payment *privacy*.

The system involves three types of parties: the bank, the merchants, and bank clients holding a (lightweight) trusted device manufactured by a third-party company that does not collude with the bank. The clients want to enjoy the convenience of mobile payment, yet do not want to reveal their spending pattern to the bank or major global-scale smartphone manufacturers.

We can apply Sipster to realize a privacy-preserving credit-card system [25]. Namely, the bank client can carry out transactions with different merchants. From time to time, the client pays the bank in unit amounts. These payments should collectively settle all the transactions made within a certain billing period.

To use Sipster, the bank acts as the UC, the trusted device corresponds to the SM, and bank clients correspond to RUs. When a bank client wants to pay the merchant K units of money, it enters the amount K to the device, which locally outputs K (blinded) random tokens as in TokenGen (*cf.* Algorithm 1), but this time it further outputs a (normal) signature on the amount K (and a random serial number). The merchant will redeem money from the bank by presenting this signature. For each token generated by the trusted device, the bank client pays back the bank some (real) money in some private manner, such that the bank will sign on the token after receiving the payment. The bank's signature serves as the receipt, in the same way as ReceiptGen (*cf.* Algorithm 2). At the end of the billing period, the trusted device outputs a bill via BillGen (*cf.* Algorithm 3), representing the total amount that the client should settle. Suppose the bank client has indeed settled the bill; it combines the receipts in the manner of CombineReceipt (*cf.* Algorithm 4) and proves to the bank analogously.

The security of the above mobile payment system comes directly from the security of our Sipster system. The bank learns nothing about the client's payment pattern and amount. Nevertheless, it is assured that it has received enough money from its client if the bill verification algorithm passes. The threat model of the above payment system is the same as our CPS example (Section 2.3), with the corresponding parties' identity substituted as above.

8 CONCLUSION

We propose Sipster, a novel anonymous payment settlement protocol for cyber-physical systems without generic (and hence heavier) techniques such as zero-knowledge proof or two-party computation protocols. It protects the privacy of customers against utility companies, while at the same time ensuring that the utility company gets correct payment according to fine-grained tariff policy.

We conduct extensive simulations to demonstrate that our system is efficient for practical deployment. In particular, the verification time at the utility company is only linearly dependent on the number of customers but independent of the individual amount.

Technically, we devise a variant of "aggregate blind signature" functionality[13] using smart meters equipped with a TCB. We believe such kinds of functionalities will find application elsewhere, most likely in the context of e-cash and anonymous credentials. We briefly mention how this can be used as a privacy-preserving mobile payment system in the credit card or IOU model, where the mobile device locally records the transaction amount of its owner for a given period, that the owner should eventually settle with the bank.

This work refutes the thought (of some non-cryptographers perhaps) that there be no payment privacy since the utility company must receive and see (sufficient) money. We demonstrate that a "win-win" situation is possible: the utility company can enjoy pay-as-you-go while users can hide unit payments "in-the-crowd."

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments and Wing-Cheong Lau for a quick conversation on mobile payment. We are also grateful to the interactive review process offered by the ACSAC Program Committee and Artifact Committee. Finally, we thank Giorgio Giacinto for his support in the publication stage.

Chow: This research is supported by the Research Grant Council, University Grants Committee, Hong Kong under the General Research Fund (CUHK 14210217, 14209918, 14210319, and 14210621).

Zhao: This research is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

[13]The blind signing functionality is deterministic and hence does not suffer from the recent polynomial-time algorithm [11] of solving the "random inhomogeneities in an overdetermined solvable system of linear equations" problem for large enough dimensions, which affects a large class of Schnorr-style (blind) signatures.

REFERENCES

[1] Masayuki Abe, Sherman S. M. Chow, Kristiyan Haralambiev, and Miyako Ohkubo. 2013. Double-Trapdoor Anonymous Tags for Traceable Signatures. *Int. J. Inf. Sec.* 12, 1 (2013), 19–31. Preliminary version appeared in ACNS 2011.

[2] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. 2016. Design in Type-I, Run in Type-III: Fast and Scalable Bilinear-Type Conversion Using Integer Programming. In *CRYPTO Part III*. 387–415.

[3] Gergely Ács and Claude Castelluccia. 2011. I Have a DREAM! (DiffeRentially privatE smArt Metering). In *Information Hiding*. 118–132.

[4] Ross Anderson and Shailendra Fuloria. 2010. On the Security Economics of Electricity Metering. In *Workshop on the Economics of Information Security (WEIS)*. 18 pages.

[5] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. 2020. Subversion-Resilient Signatures: Definitions, Constructions and Applications. *Theor. Comput. Sci.* 820 (2020), 91–122.

[6] AT&T. 2015. AT&T Innovation Leads to Cellular Communications Module Reference Design and Advanced Prepay Energy. https://about.att.com/story/att_expands_smart_grid_leadership.html.

[7] AT&T. 2016. PrePay Energy. http://www.business.att.com/content/productbrochures/smartgrid-prepay-energy-product-brief.pdf.

[8] Karim Baghery. 2020. Subversion-Resistant Commitment Schemes: Definitions and Constructions. In *Security and Trust Management (STM)*. 106–122.

[9] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. 2010. PrETP: Privacy-Preserving Electronic Toll Pricing. In *USENIX Security*. 63–78.

[10] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE S&P*. 459–474.

[11] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. 2021. On the (In)security of ROS. In *EUROCRYPT Part I*. 33–53.

[12] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography (PKC)*. 31–46.

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*. 416–432.

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT*. 514–532.

[15] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *EUROCRYPT*. 302–321.

[16] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO*. 61–76.

[17] Dario Catalano, Dario Fiore, and Luca Nizzardo. 2015. Programmable Hash Functions Go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. In *CRYPTO (Part II)*. 254–274.

[18] Dario Catalano, Dario Fiore, and Bogdan Warinschi. 2014. Homomorphic Signatures with Efficient Verification for Polynomial Functions. In *CRYPTO (Part I)*. 371–389.

[19] Jae Choon Cha and Jung Hee Cheon. 2003. An Identity-Based Signature from Gap Diffie-Hellman Groups. In *Public Key Cryptography (PKC)*. 18–30.

[20] Melissa Chase and Sherman S. M. Chow. 2009. Improving Privacy and Security in Multi-Authority Attribute-Based Encryption. In *ACM Conference on Computer and Communications Security (CCS)*. 121–130.

[21] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. 2010. Comparing Two Pairing-based Aggregate Signature Schemes. *Des. Codes Cryptogr.* 55, 2-3 (2010), 141–167.

[22] David Chaum. 1982. Blind Signatures for Untraceable Payments. In *CRYPTO*. 199–203.

[23] Sherman S. M. Chow. 2016. Functional Credentials for Internet of Things. In *Workshop on IoT Privacy, Trust, and Security, IoTPTS@AsiaCCS*. 1.

[24] Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. 2005. Two Improved Partially Blind Signature Schemes from Bilinear Pairings. In *Australasian Conference on Information Security and Privacy (ACISP)*. 316–328.

[25] Sherman S. M. Chow, Russell W. F. Lai, Xiuhua Wang, and Yongjun Zhao. 2016. Privacy Preserving Credit Systems. In *Network and System Security (NSS)*. 184–199.

[26] Sherman S. M. Chow, Alexander Russell, Qiang Tang, Moti Yung, Yongjun Zhao, and Hong-Sheng Zhou. 2019. Let a Non-barking Watchdog Bite: Cliptographic Signatures with an Offline Watchdog. In *Public Key Cryptography (PKC) Part I*. 221–251.

[27] George Danezis, Markulf Kohlweiss, and Alfredo Rial. 2011. Differentially Private Billing with Rebates. In *Information Hiding*. 148–162.

[28] Amit Datta, Marc Joye, and Nadia Fawaz. 2019. Private Data Aggregation over Selected Subsets of Users. In *Cryptology and Network Security (CANS)*. 375–391.

[29] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. 2016. Message Transmission with Reverse Firewalls - Secure Communication on Corrupted Machines. In *CRYPTO Part I*. 341–372.

[30] Zekeriya Erkin and Gene Tsudik. 2012. Private Computation of Spatial and Temporal Power Consumption with Smart Meters. In *Applied Cryptography and Network Security (ACNS)*. 561–577.

[31] Flavio D. Garcia and Bart Jacobs. 2010. Privacy-Friendly Energy-Metering via Homomorphic Encryption. In *Security and Trust Management (STM)*. 226–238.

[32] B. Kelsey Jack and Grant Smith. 2015. Pay as You Go: Prepaid Metering and Electricity Expenditures in South Africa. *The American Economic Review* 105, 5 (2015), 237–241.

[33] Marek Jawurek, Martin Johns, and Florian Kerschbaum. 2011. Plug-In Privacy for Smart Metering Billing. In *Privacy Enhancing Technologies Symposium (PETS)*. 192–210.

[34] Marek Jawurek, Florian Kerschbaum, and George Danezis. 2012. *Privacy Technologies for Smart Grids - A Survey of Options*. Technical Report MSR-TR-2012-119. https://www.microsoft.com/en-us/research/publication/privacy-technologies-for-smart-grids-a-survey-of-options

[35] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. 2011. Privacy-Friendly Aggregation for the Smart-Grid. In *Privacy Enhancing Technologies Symposium (PETS)*. 175–191.

[36] Russell W. F. Lai, Raymond K. H. Tai, Harry W. H. Wong, and Sherman S. M. Chow. 2018. Multi-key Homomorphic Signatures Unforgeable Under Insider Corruption. In *ASIACRYPT Part II*. 465–492.

[37] Hsiao-Ying Lin, Wen-Guey Tzeng, Shiuan-Tzuo Shen, and Bao-Shuh Paul Lin. 2012. A Practical Smart Metering System Supporting Privacy Preserving Billing and Load Monitoring. In *Applied Cryptography and Network Security (ACNS)*. 544–560.

[38] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *IEEE S&P*. 397–411.

[39] Ilya Mironov and Noah Stephens-Davidowitz. 2015. Cryptographic Reverse Firewalls. In *EUROCRYPT*. 657–686.

[40] Moni Naor, Benny Pinkas, and Omer Reingold. 1999. Distributed Pseudo-random Functions and KDCs. In *EUROCRYPT*. 327–346.

[41] National Institute of Standards and Technology. 2000. *FIPS PUB 186-2: Digital Signature Standard (DSS)*. NIST.

[42] Lucien K. L. Ng, Sherman S. M. Chow, Anna P. Y. Woo, Donald P. H. Wong, and Yongjun Zhao. 2021. Goten: GPU-Outsourcing Trusted Execution of Neural Network Training. In *AAAI*. 14876–14883.

[43] Shen Noether and Adam Mackenzie. 2016. Ring Confidential Transactions. *Ledger* 1 (2016), 1–18. https://ledgerjournal.org/ojs/index.php/ledger/article/view/34

[44] Faranaaz Parker. 2010. Pay-As-You-Go Healthcare Launches in South Africa. http://mg.co.za/article/2010-12-02-payasyougo-healthcare-launches-in-south-africa.

[45] David Pointcheval and Olivier Sanders. 2016. Short Randomizable Signatures. In *Cryptographers' Track at the RSA Conference (CT-RSA)*. 111–126.

[46] Raluca A. Popa, Hari Balakrishnan, and Andrew J. Blumberg. 2009. VPriv: Protecting Privacy in Location-Based Vehicular Services. In *USENIX Security Symposium*. 335–350.

[47] Quindi Research. 2017. Smart Prepaid Utilities in the United States: Forecasts & Analysis. https://www.exceleron.com/wp-content/uploads/2017/03/QRL-US-Smart-Prepaid-Utilities-Report-Summary-Exceleron-1.pdf.

[48] Alfredo Rial and George Danezis. 2011. Privacy-Preserving Smart Metering. In *Workshop on Privacy in the Electronic Society (WPES)*. 49–60.

[49] Andy Rupp, Foteini Baldimtsi, Gesine Hinterwälder, and Christof Paar. 2015. Cryptographic Theory Meets Practice: Efficient and Privacy-Preserving Payments for Public Transport. *ACM Trans. Inf. Syst. Secur.* 17, 3 (2015), 10:1–10:31.

[50] Antonio G. Ruzzelli, C. Nicolas, Anthony Schoofs, and Gregory M. P. O'Hare. 2010. Real-Time Recognition and Profiling of Appliances through a Single Electricity Sensor. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. 279–287.

[51] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS*. 17 pages.

[52] Siemens. 2019. Smart Metering Software Prepayment Metering Solutions. http://w3.siemens.com/smartgrid/global/en/products-systems-solutions/smart-metering/software-systems-solutions/pages/prepayment.aspx.

[53] SmartGridCIS. 2015. SmartGridCIS Fuels "Pay-as-you-Go" Program for Wake Forest Power. http://sgcisonline.com/wp-content/uploads/2015/07/Wake-Forest-Case-Study.pdf.

[54] Patrick P. Tsang, Sherman S. M. Chow, and Sean W. Smith. 2007. Batch Pairing Delegation. In *International Workshop on Security (IWSEC)*. 74–90.

[55] Jiafan Wang, Minxin Du, and Sherman S. M. Chow. 2020. Stargazing in the Dark: Secure Skyline Queries with SGX. In *Database Systems for Advanced Applications (DASFAA) Part III*. 322–338.

[56] Eric A. Young and Tim J. Hudson. 2011. OpenSSL: The Open Source Toolkit for SSL/TLS.

A PRIVACY-PRESERVING AGGREGATION

Bill generation is considered as a special case of aggregation [34]. Each user can mask the readings before submitting them to the company. The masks can be obtained through collaboration among all users [35], such that the aggregated masks can be canceled with each other, but it incurs heavy communication overhead. Acs and Castelluccia [3] employ a differential-privacy model in which the user adds carefully designed noise to the reading to hide the real value. The utility company can then obtain the sum of the noisy readings. This can be seen as sacrificing data accuracy for privacy.

Garcia and Jacobs [31] homomorphically encrypt secret shares of the measurements to each member in the aggregation group. With homomorphism, every user can obtain an aggregated share encrypted under the public key. The user then sends the decryption result of the aggregated shares to the utility company, who finally obtains the total consumption over all users without knowing individual measurements. Erkin and Tsudik [30] develop another aggregation scheme using encryption as homomorphic commitment. The scheme also first splits a value (specifically, the modulo) into random shares among the users. An individual user reading is committed with the exponent of the random term (which was the modulo originally) being the share. Like other schemes, when all commitments are combined, the random shares cancel each other.

To hide individual usage, all the above schemes aggregate individual consumption over either a long duration or a group of users. In short, privacy-preserving data aggregation is a kind of secure multi-party computation of secret values but is not related to any subsequent payment according to those values. On the other hand, our payment problem requires signatures certifying the secret amounts and aims to achieve seemingly contradictory requirements (*e.g.*, unlinkability between payments and receipts and preventing double claim of the same receipt).

B BASIC DEFINITIONS

Definition 1 (Existential Unforgeability). *Let $\mathcal{SS}$ = (KGen, Sign, Verify) be a signature scheme. We say that $\mathcal{SS}$ is (t, q, ϵ)-EUF-CMA-secure if for all PPT adversaries $\mathcal{A}$ running in time t:*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{vk},(m^*,\sigma^*)) = 1 \wedge m^* \notin Q : \\ (\mathsf{vk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda); (m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk})\end{array}\right] \leq \epsilon$$

where $Q = \{m_1, \ldots, m_q\}$ denotes the set of queries to the signing oracle. Whenever $q = \mathsf{poly}(\lambda)$ and $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, we simply say that $\mathcal{SS}$ is EUF-CMA-secure.

A group sampler is a PPT algorithm $\mathcal{IG}$ that on input the security parameter λ outputs a tuple $(\mathbb{G}, g, q) \leftarrow \mathcal{IG}(1^\lambda)$. It defines a poly($\lambda$)-bit representation for group elements in $\mathcal{G}$ of order q. For the sake of simplicity, we omit the indication of $\mathcal{IG}$ hereafter, but we note that in the following definitions, the probabilities are also taken over the randomness of $\mathcal{IG}$.

Definition 2 (Computational Diffie-Hellman Problem). *Given (g, g^x, g^y), compute g^{xy}, where $\{g, g^x, g^y\} \in \mathbb{G}$ and $(x, y) \leftarrow \mathbb{Z}_q^2$. Let the advantage of a PPT algorithm $\mathcal{A}$ in solving the computational Diffie-Hellman (CDH) problem be:*

$$\mathbf{Adv}_{\mathcal{A}}^{CDH} = \Pr[(x, y) \leftarrow \mathbb{Z}_q^2 : \mathcal{A}(g, g^x, g^y) = g^{xy}].$$

The CDH assumption requires that, for any PPT adversary $\mathcal{A}$, the advantage of solving the CDH problem $\mathbf{Adv}_{\mathcal{A}}^{CDH}$ is negligible.

Gap Diffie-Hellman (Gap-DH) groups (*e.g.*, [12, 19]) are those where the decisional Diffie-Hellman (DDH) problem can be solved in polynomial time, but no PPT algorithm can solve the CDH problem with non-negligible probability, *e.g.*, bilinear groups. Let $O_{DDH}(\cdot)$ denote an oracle machine that takes input $(g, g^x, g^y, g^z) \in \mathbb{G}^4$, outputs 1 if $z = xy$ and outputs 0 otherwise.

Definition 3 (Gap-Diffie-Hellman (Gap-DH) Problem). *Given (g, g^x, g^y) and access to a DDH oracle ($O_{DDH}(\cdot)$), compute g^{xy}, where $\{g, g^x, g^y\} \in \mathbb{G}$ and $(x, y) \leftarrow \mathbb{Z}_q^2$. Let the advantage of a PPT algorithm $\mathcal{A}$ in solving the gap-DH problem $\mathbf{Adv}_{\mathcal{A}}^{Gap\text{-}DH}$ be:*

$$\Pr[(x, y) \leftarrow \mathbb{Z}_q^2 : \mathcal{A}^{O_{DDH}(\cdot)}(g, g^x, g^y) = g^{xy}].$$

The gap-DH assumption requires that, for any PPT adversary $\mathcal{A}$, the advantage of solving the gap-DH problem $\mathbf{Adv}_{\mathcal{A}}^{Gap\text{-}DH}$ is negligible.

We assume that the gap-DH assumption holds in bilinear group $\mathbb{G}_1$. The use of pairing groups enables such a DDH oracle.

C FORMAL SECURITY ANALYSIS

C.1 Privacy

First, we claim that Sipster satisfies privacy, namely the UC cannot infer how many e-coins it receives from an RU, nor link any spending of e-coin to a particular RU.

To define privacy, we introduce a *user indistinguishability* game between an adversary $\mathcal{A}$, acting as the malicious UC, and a challenger C, acting as both the RU and SMs. This definition guarantees that a curious UC cannot differentiate users who made payments in Sipster, even when UC can choose these target users and adaptively induce them to perform payment-related operations of its choice.

Definition 4 (User Indistinguishability). *The user indistinguishability game between an adversary $\mathcal{A}$ and a challenger C proceeds in steps as follows:*

- *Setup Phase: C runs the key generation algorithm of a digital signature scheme to obtain a verification/signing key pair ($\mathsf{vk}_{\mathrm{SM}}, \mathsf{sk}_{\mathrm{SM}}$), and gives the verification key $\mathsf{vk}_{\mathrm{SM}}$ to $\mathcal{A}$. C plays the role of the* RU *and N smart meter instances $\{\mathrm{SM}_i\}_{i\in[N]}$ that share the same signing key $\mathsf{sk}_{\mathrm{SM}}$. The internal states $\{\tau_i\}_{i\in[N]}$ are initialized. C also initializes empty sets $\mathcal{ID}$ (for recording corrupted IDs) and $\{\mathcal{RC}_i\}_{i\in[N]}$ (for storing receipts). C chooses a random permutation $\pi : [N] \to [N]$. $\mathcal{A}$ chooses a billing period t. It runs the setup algorithm for the* UC *and forwards* $\mathsf{PP}_{\mathrm{UC}}$ *to C while keeping* $\mathsf{sk}_{\mathrm{UC}}$ *secret.*
- *Query Phase: $\mathcal{A}$ and C mimic the Sipster protocol:*
  1. *When $\mathcal{A}$ sends a* consume query *(Consume, i) to C, C runs* TokenGen *on input* $\mathsf{sk}_{\mathrm{SM}}$ *and state $\tau_{\pi(i)}$. The internal state $\tau_{\pi(i)}$ is updated. The output* $\mathsf{tk} = (\tilde{R}, \sigma_{\tilde{R}})$ *is forwarded to $\mathcal{A}$.*
  2. *$\mathcal{A}$ runs* $\mathsf{ReceiptGen}(\mathsf{tk}, \mathsf{vk}_{\mathrm{SM}}, \mathsf{sk}_{\mathrm{UC}})$ *to obtain* rcpt. *If* rcpt $\neq \perp$, *$\mathcal{A}$ sends* rcpt *to C.*
  3. *C checks if* rcpt *is valid w.r.t.* $\mathsf{PP}_{\mathrm{UC}}$. *If not, C aborts the game. Otherwise, C adds* rcpt *to the set $\mathcal{RC}_{\pi(i)}$.*

  The above queries can be issued polynomially many times.

- *Corruption Query: In the Query Phase, $\mathcal{A}$ can send a* corruption query *(Corrupt, i) to C at any time. C will return $\pi(i)$ and add i to $\mathcal{ID}$. This query can be issued polynomially many times.*
- *Challenge Phase: When $\mathcal{A}$ issues a* bill-generation query, *C checks for all $i \in [N] \setminus \mathcal{ID}$, whether $\mathcal{RC}_{\pi(i)}$ is empty. If there are less than two non-empty $\mathcal{RC}_{\pi(i)}$, it aborts; otherwise, for all non-empty $\mathcal{RC}_{\pi(i)}$, it runs* BillGen *on inputs* ($\mathsf{sk_{SM}}$, ID_i, t, $\tau_{\pi(i)}$) *to get* $\mathsf{bill}_i = (\tau_{\pi(i)}, m_{\mathsf{ID}_i,t}, \sigma^{\mathsf{B}}_{\mathsf{ID}_i,t})$. *Then, C runs* CombineReceipt *on inputs* (bill_i, $\mathcal{RC}_{\pi(i)}$, $\mathsf{PP_{UC}}$) *to obtain the results $\sigma^{\mathsf{R}}_{\mathsf{ID}_i,t}$. C sends all such* ($m_{\mathsf{ID}_i,t}$, $\sigma^{\mathsf{B}}_{\mathsf{ID}_i,t}$, $\sigma^{\mathsf{R}}_{\mathsf{ID}_i,t}$) *to $\mathcal{A}$.*
- *Guess Phase: $\mathcal{A}$ returns a pair (j, j_π) as its guess.*

Definition 5 (Privacy). *Sipster satisfies privacy, if the advantage of any PPT adversary defined by* $\mathsf{Adv}^{\mathbf{Priv}}_{\mathcal{A}} = |\Pr[j_\pi = \pi(j) | j \notin \mathcal{ID}] - \frac{1}{n}|$, *is negligible in λ, where n is the number of non-empty $\mathcal{RC}_i$'s, $i \in [N] \setminus \mathcal{ID}$.*

The privacy game mimics the interaction among the UC, RU, and SM. Definition 5 guarantees that neither the identity nor the payment amount is leaked to the adversary. In the game, $\mathcal{A}$ has full control over when to see receipt requests for $\mathsf{ID}_{\pi(j)}$ in the Bill Settlement Phase. $\mathcal{A}$ can also determine the payment amount for $\mathsf{ID}_{\pi(j)}$ by choosing how many Consume queries are issued for $\mathsf{ID}_{\pi(j)}$. The security notion says that such a powerful adversary still cannot guess $\pi(j)$ with a non-negligible advantage. It means that UC can infer neither the RU's identity nor the payment amount in the Bill Verification Phase. Effectively, RU_i remains hidden among the set of non-corrupted residential users that have made payments.

C.2 Soundness

We summarize the two properties of unforgeability and double-spending prevention as the soundness property. Similarly, we define this security notion via a game.

In the soundness game, adversary $\mathcal{A}$ plays the role of the RU while challenger C plays the roles of both the SM and UC.

Definition 6. *The soundness game between an adversary $\mathcal{A}$ and a challenger C proceeds in steps as follows:*

- *Setup Phase: Challenger C runs the setup algorithm for the* UC *and forwards* $\mathsf{PP_{UC}}$ *to $\mathcal{A}$. C runs the key generation algorithm to get a verification/signing key pair* ($\mathsf{vk_{SM}}$, $\mathsf{sk_{SM}}$), *and gives the verification key* $\mathsf{vk_{SM}}$ *to $\mathcal{A}$. C plays the role of the* UC *and one smart meter instance* SM *with the signing key* $\mathsf{sk_{SM}}$ *above. The internal state of the* SM *is then initialized. $\mathcal{A}$ chooses an identity* ID^* *and a billing period t^*, and informs C a payment amount K.*
- *Challenge Phase: $\mathcal{A}$ and C mimic the Sipster protocol:*
  1. *C runs the stateful algorithm* TokenGen *on input* ($\mathsf{sk_{SM}}$, τ) *for K times to obtain a set of K tokens $\{\mathsf{tk}_i\}_K$ (the internal state τ is also updated). All tokens are forwarded to $\mathcal{A}$.*
  2. *At the end of the billing period t^*, C executes* BillGen *on input* ($\mathsf{sk_{SM}}$, ID^*, t^*, τ) *to generate the bill for identity* ID^*, *and forwards the resulting* $\mathsf{bill} = (\tau, m_{\mathsf{ID}^*,t^*}, \sigma^{\mathsf{B}}_{\mathsf{ID}^*,t^*})$ *to $\mathcal{A}$.*
- *Query Phase: C initializes a counter $ctr = 0$. $\mathcal{A}$ sends receipt-generation queries:*
  1. *$\mathcal{A}$ sends tk' to C;*
  2. *If $ctr \geq K$, C ignores the query; otherwise, C computes* $\mathsf{rcpt} = \mathsf{ReceiptGen}(\mathsf{tk}', \mathsf{vk_{SM}}, \mathsf{sk_{UC}})$. *If* $\mathsf{rcpt} \neq \perp$, *increments the counter $ctr = ctr + 1$, and sends* rcpt *to $\mathcal{A}$; otherwise, ignores the query.*
- *Output Phase: $\mathcal{A}$ outputs π_{ID^*,t^*}. C accepts if π_{ID^*,t^*} is a valid proof for* ID^* *and period t^*.*

Definition 7 (Soundness). *Sipster satisfies soundness if the advantage of any PPT adversary in the above game, defined by* $\mathsf{Adv}^{\mathbf{Bill}}_{\mathcal{A}} = \Pr[C \text{ accepts}]$, *is negligible in the security parameter λ.*

C.3 Proof for Privacy

Theorem 1. *Sipster satisfies privacy and* $\mathsf{Adv}^{\mathbf{Priv}}_{\mathcal{A}} = 0$.

Proof. In the Query Phase, the response of a (Consume, i) query is of the form $(\tilde{R}, \sigma_{\tilde{R}}) = (g^r R, \mathsf{Sign}(\mathsf{sk_{SM}}, \sigma_{\tilde{R}}))$. Because all r's are chosen independently from a uniform distribution, what $\mathcal{A}$ sees in the whole Query Phase is a series of independently and identically distributed random group elements and signatures on them. Such distribution is independent of the index i.

Assume that challenger C does not abort. We argue that the view of $\mathcal{A}$ in the Challenge Phase is independent of the random permutation $\pi(\cdot)$. Let i be an arbitrary index such that $\mathcal{RC}_{\pi(i)}$ is non-empty. $\mathcal{A}$ sees $(m_{\mathsf{ID}_{\pi(t)},t} = (R_{\mathsf{ID}_{\pi(t)},t}, \mathsf{ID}_i, t), \sigma^{\mathsf{B}}_{\mathsf{ID}_{\pi(t)},t}, \sigma^{\mathsf{R}}_{\mathsf{ID}_{\pi(t)},t})$. In essence, $R_{\mathsf{ID}_{\pi(t)},t}$, $\sigma^{\mathsf{B}}_{\mathsf{ID}_{\pi(t)},t}$, $\sigma^{\mathsf{R}}_{\mathsf{ID}_{\pi(t)},t}$ substitute $R_{\mathsf{ID}_i,t}$, $\sigma^{\mathsf{B}}_{\mathsf{ID}_i,t}$, $\sigma^{\mathsf{R}}_{\mathsf{ID}_i,t}$ respectively. Therefore, the problem reduces to showing $(R_{\mathsf{ID}_{\pi(t)},t}, \sigma^{\mathsf{B}}_{\mathsf{ID}_{\pi(t)},t}, \sigma^{\mathsf{R}}_{\mathsf{ID}_{\pi(t)},t})$ is identically distributed as $(R_{\mathsf{ID}_i,t}, \sigma^{\mathsf{B}}_{\mathsf{ID}_i,t}, \sigma^{\mathsf{R}}_{\mathsf{ID}_i,t})$.

The last two elements $(\sigma^{\mathsf{B}}_{\mathsf{ID}_{\pi(t)},t}, \sigma^{\mathsf{R}}_{\mathsf{ID}_{\pi(t)},t})$ and $(\sigma^{\mathsf{B}}_{\mathsf{ID}_i,t}, \sigma^{\mathsf{R}}_{\mathsf{ID}_i,t})$ depend on the first one ($m_{\mathsf{ID}_{\pi(t)},t}$ and $m_{\mathsf{ID}_i,t}$): $\sigma^{\mathsf{B}}_{\mathsf{ID}_{\pi(t)},t}$ (resp. $\sigma^{\mathsf{B}}_{\mathsf{ID}_i,t}$) is a signature on $m_{\mathsf{ID}_{\pi(t)},t}$ (resp. $m_{\mathsf{ID}_i,t}$); $\sigma^{\mathsf{R}}_{\mathsf{ID}_{\pi(t)},t} = R^{\alpha}_{\mathsf{ID}_{\pi(t)},t}$ (resp. $\sigma^{\mathsf{R}}_{\mathsf{ID}_i,t} = R^{\alpha}_{\mathsf{ID}_i,t}$). So, the problem further reduces to showing $R_{\mathsf{ID}_{\pi(t)},t}$ and $R_{\mathsf{ID}_i,t}$ are identically distributed.

To see why they are identically distributed, note that $R_{\mathsf{ID}_{\pi(t)},t}$ (as a part of the internal state $\tau_{\pi(t)}$) is a random group element independent of $R_{\mathsf{ID}_i,t}$ as well as all r's in the Query Phase. (See Algorithm 1 and Algorithm 3.) A similar conclusion also holds for $R_{\mathsf{ID}_i,t}$. As a result, the distributions of $R_{\mathsf{ID}_{\pi(t)},t}$ and $R_{\mathsf{ID}_i,t}$ are identical even conditioned on what $\mathcal{A}$ saw in the Query Phase. □

C.4 Proof for Soundness

Theorem 2. *Sipster satisfies soundness if $\mathcal{SS}$ is an* EUF-CMA-*secure digital signature scheme and the gap-DH problem (Definition 3) is hard in $\mathbb{G}_1$.*

Proof. By contradiction, there exists an adversary $\mathcal{A}$ breaking the soundness game with a non-negligible probability. We show that one of the following must hold: (1) the underlying digital signature scheme $\mathcal{SS}$ is not EUF-CMA secure; (2) the gap-DH problem (Definition 3) is easy in $\mathbb{G}_1$.

Suppose the output of $\mathcal{A}$ is $\pi_{\mathsf{ID}^*,t^*} = (\tilde{m}_{\mathsf{ID}^*,t^*}, \tilde{\sigma}^{\mathsf{B}}_{\mathsf{ID}^*,t^*}, \tilde{\sigma}^{\mathsf{R}}_{\mathsf{ID}^*,t^*})$. We classify $\mathcal{A}$ into two types. A Type-I adversary $\mathcal{A}$ satisfies the following requirements: (1) $\tilde{m}_{\mathsf{ID}^*,t^*}$ in π_{ID^*,t^*} equals m_{ID^*,t^*} received from C in the Query Phase (in $\mathsf{bill} = (\tau, m_{\mathsf{ID}^*,t^*}, \sigma^{\mathsf{B}}_{\mathsf{ID}^*,t^*})$), and (2) for all the receipts rcpt_i's received in the Query Phase, their corresponding tokens tk_i's are all received from C in the Challenge

Phase. Any adversary who is not Type-I is a Type-II adversary. Clearly, any adversary must be either Type-I or Type-II.

Case One: $\mathcal{A}$ is Type-I. We build a reduction $\mathcal{B}_{Gap\text{-}DH}$ that solves the gap-DH problem with a non-negligible probability by using $\mathcal{A}$. Suppose $\mathcal{B}_{Gap\text{-}DH}$ receives the gap-DH challenge $(g, X, Y) = (g, g^x, g^y)$. $\mathcal{B}_{Gap\text{-}DH}$ plays the role of C in the soundness game. $\mathcal{B}_{Gap\text{-}DH}$ simulates the Setup Phase of the soundness game as follows: it simulates the output of the setup procedure for UC as $\mathsf{PP}_{\mathsf{UC}} = (\mathbb{G}_1, \mathbb{G}_T, g, X)$. It runs the KGen for $\mathcal{SS}$ to generate $(\mathsf{vk}_{\mathsf{SM}}, \mathsf{sk}_{\mathsf{SM}})$. After receiving a payment amount K from $\mathcal{A}$, $\mathcal{B}_{Gap\text{-}DH}$ selects a random index $i \in [1, K]$. $\mathcal{B}_{Gap\text{-}DH}$ selects $(K-1)$ random exponents $(r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_k) \leftarrow \mathbb{Z}_q^{K-1}$ and computes $R_1 = g^{r_1}, \ldots, R_{i-1} = g^{r_{i-1}}, R_i = Y, R_{i+1} = g^{r_{i+1}}, \ldots, R_K = g^{r_K}$. $\mathcal{B}_{Gap\text{-}DH}$ then simulates the Challenge Phase by computing, for all $j \in [1, K]$, $\mathsf{tk}_j = (R_j, \mathsf{Sign}(\mathsf{sk}_{\mathsf{SM}}, R_j))$. It also selects $r_{\mathsf{ID}^*, t^*} \leftarrow \mathbb{Z}_q$, computes $R_{\mathsf{ID}^*, t^*} = g^{r_1} \cdots g^{r_{i-1}} Y g^{r_{i+1}} \cdots g^{r_K} / g^{r_{\mathsf{ID}^*, t^*}}$. The output of BillGen is $\mathsf{bill} = (\tau, m_{\mathsf{ID}^*, t^*}, \sigma^{\mathsf{B}}_{\mathsf{ID}^*, t^*})$, where $\tau = (r_{\mathsf{ID}^*, t^*}, R_{\mathsf{ID}^*, t^*})$ and $m_{\mathsf{ID}^*, t^*} = (R_{\mathsf{ID}^*, t^*}, \mathsf{ID}^*, t^*)$. $\sigma^{\mathsf{B}}_{\mathsf{ID}, t}$ can be computed by using $\mathsf{sk}_{\mathsf{SM}}$. The simulations of the Setup and the Challenge Phases are perfect.

By our assumption that $\mathcal{A}$ is Type-I, we know that all the queries $\mathcal{A}$ made in the Query Phase come from $\{\mathsf{tk}_j\}_K$. Without loss of generality, we assume that $\mathcal{A}$ makes $(K-1)$ queries to $\mathcal{B}_{Gap\text{-}DH}$. If the query $\mathsf{tk}' = \mathsf{tk}_j$ for some $j \neq i$, $\mathcal{B}_{Gap\text{-}DH}$ responds with X^{r_j}. If $\mathsf{tk}' = \mathsf{tk}_i$, $\mathcal{B}_{Gap\text{-}DH}$ aborts and restarts the simulation from scratch. The probability that $\mathcal{B}_{Gap\text{-}DH}$ does not abort in the Query Phase is $1/K$, which is non-negligible. Conditioned on the event that $\mathcal{B}_{Gap\text{-}DH}$ does not abort, the simulation is also perfect in the Query Phase. Then, by our assumption, $\mathcal{A}$ outputs an accepting π_{ID^*, t^*} with a non-negligible probability. $\mathcal{B}_{Gap\text{-}DH}$ extracts $\sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*}$ from π_{ID^*, t^*} and outputs $\sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*} \cdot X^{r_{\mathsf{ID}^*, t^*}} / (X^{r_1} \cdots X^{r_{i-1}} X^{r_{i+1}} \cdots X^{r_K})$ as the solution for the gap-DH problem.

To see why $\sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*} \cdot X^{r_{\mathsf{ID}^*, t^*}} / (X^{r_1} \cdots X^{r_{i-1}} X^{r_{i+1}} \cdots X^{r_K})$ is the correct solution, recall that $\sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*}$ must satisfy the pairing equation:

$$\begin{aligned} e(\sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*}, g) &= e(R_{\mathsf{ID}^*, t^*}, X) \\ &= e(g^{r_1} \cdots g^{r_{i-1}} Y g^{r_{i+1}} \cdots g^{r_K} / g^{r_{\mathsf{ID}, t}}, X) \\ &= e(X^{r_1} \cdots X^{r_{i-1}} Z X^{r_{i+1}} \cdots X^{r_K} / X^{r_{\mathsf{ID}, t}}, g) \end{aligned}$$

where $Z = Y^x$ is the expected solution of the gap-DH problem. It follows that $Z = \sigma^{\mathsf{R}}_{\mathsf{ID}^*, t^*} \cdot X^{r_{\mathsf{ID}^*, t^*}} / (X^{r_1} X^{r_2} \cdots X^{r_{i-1}} X^{r_{i+1}} \cdots X^{r_K})$ by simple rearrangement of terms, which means $\mathcal{B}_{Gap\text{-}DH}$ solves the gap-DH problem with a non-negligible probability.

Case Two: $\mathcal{A}$ is Type-II. Given a Type-II adversary, it is easy to construct a reduction $\mathcal{B}_{\mathsf{EUF\text{-}CMA}}$ that breaks the EUF-CMA-security of the signature scheme $\mathcal{SS}$. $\mathcal{B}_{\mathsf{EUF\text{-}CMA}}$ simply uses the vk it received in its EUF-CMA-game as $\mathsf{vk}_{\mathsf{SM}}$ to simulate the soundness game for $\mathcal{A}$ without knowing $\mathsf{sk}_{\mathsf{SM}} = \mathsf{sk}$. The only algorithms that $\mathsf{sk}_{\mathsf{SM}}$ would be used are TokenGen and BillGen. For the former, $\mathcal{B}_{\mathsf{EUF\text{-}CMA}}$ queries the signing oracle for K random group elements to compute a set of K tokens $\{\mathsf{tk}_i\}_K$. For the latter, it queries the signing oracle on input m_{ID^*, t^*} to compute the output of BillGen. The message spaces for either kind of requests are different. By assumption, a Type-II adversary $\mathcal{A}$ outputs an accepting proof π_{ID^*, t^*} with a non-negligible probability. Thus $\mathcal{B}_{\mathsf{EUF\text{-}CMA}}$ can extract at least one forgery (a new token tk') in the Query Phase or from π_{ID^*, t^*} (a forgery for $\tilde{m}_{\mathsf{ID}^*, t^*} \neq m_{\mathsf{ID}^*, t^*}$).

Combining the two cases above, we conclude that if there exists $\mathcal{A}$ breaking the soundness game with a non-negligible probability, then either $\mathcal{SS}$ is not EUF-CMA-secure, or the gap-DH problem is easy in $\mathbb{G}_1$. □