Silent Splitter: Privacy for Payment Splitting via New Protocols for Distributed Point Functions

Margaret Pierce
University of North Carolina at Chapel Hill
mapierce@cs.unc.edu

Saba Eskandarian
University of North Carolina at Chapel Hill
saba@cs.unc.edu

Abstract

In a world where financial transactions are primarily performed or recorded online, protecting sensitive transaction details has become crucial. Roommates sharing housing costs or friends splitting travelling expenses may use applications such as Splitwise to easily track debts and minimize the number of individual repayments. However, these apps reveal potentially sensitive financial transaction activity to their operators. In this paper, we present Silent Splitter, a privacy-preserving payment splitting system which enables users to securely set up groups, perform transactions within those groups, and "settle up" without revealing group membership or any sensitive transaction details (such as the users involved or amount of money exchanged) to the system itself. Silent Splitter operates in the two server setting and uses Distributed Point Functions (DPFs) to securely record transactions. Of independent interest, we also present new protocols for proving knowledge of properties of DPFs as part of our system.

Keywords

Distributed Point Functions, Payment Splitting

1 Introduction

Many everyday financial transactions, once protected by the anonymity of cash, are now performed under the watchful eye of credit card issuers, payment networks, and finance apps. Blurring the lines between financial services and social media, apps like Venmo and CashApp learn not only the locations and costs of purchases, but even the identities of people spending money together and the nature of social relationships between them.

While a number of privacy preserving technologies, especially in the blockchain space, seek to add privacy to the payment process, a combination of market pressures, regulatory requirements, and the need to prevent abuse mean that privacy preservation has not yet reached the realm of day-to-day digital financial transactions.

This paper takes a different approach. We seek to bolster the privacy-preserving properties of payment splitting tools, which mitigate the need for customers to directly interact with surveillant financial institutions. Payment splitting apps, like Splitwise and its competitors, maintain records of debts among groups of friends, eliminating the need for repayments to occur after each purchase. These apps internally record user debts without etching a permanent record with financial institutions. Thus, they reduce the number of user-to-user repayments, either by allowing debts to naturally balance out over time, or by allowing groups to settle accumulated debts at regular intervals. In this way, they reduce the frequency with which users must interact with financial institutions and thus reveal potentially sensitive information to them.

This paper presents a privacy-preserving version of payment splitting apps. That is, we create a system in which users can establish groups and make transactions without revealing group membership or transaction amounts to the app infrastructure. Since these transactions are simply records and not actual money exchanges, the internal transactions and social relationships of each group remain hidden from outside observers. To the best of our knowledge, the only prior work to consider privacy in payment splitting apps made strong non-collusion assumptions between clients and servers and did not hide the membership of groups, perhaps the most important sensitive information involved in such apps [43]. With the recent integration of payment splitting group features into the Venmo app [60], protecting this kind of group composition information has become all the more important.

Our system, Silent Splitter, combines techniques from single server and multi-server approaches. Users register their groups with a primary server who blindly keeps track of group membership and authenticates transactions using anonymous credentials [20, 21, 25, 26, 29], while the more frequent and computationally intensive transaction and balance retrieval functionalities are split between the primary server and a secondary server using function secret sharing techniques [11, 14, 15, 46]. This design results in strong performance for critical components of the protocol while conforming to real-world organizational trust relationships.

Given that Silent Splitter combines families of cryptographic techniques rarely used together, we introduce a number of techniques to bridge the gap between protocols that rely on encrypted or secret shared data. Along the way, we introduce new protocols for distributed point functions (DPFs) [15, 46] that are of independent interest. In particular, we show how a client can provably provide the Silent Splitter servers with commitments or encryptions to the non-zero index and value of a DPF. Our techniques can be viewed as generalizations and improvements over recent work on access control for function secret sharing schemes [50, 58].

An end-to-end implementation of our system, written in Rust, is available at https://github.com/mapierce23/psa. We evaluate our system's time and communication costs for group setup and balance retrieval, as well as the end-to-end latency and throughput for transaction requests. Silent Splitter assigns most of the heavy client-side computation to a one-time group setup protocol, meaning that client-side costs are very light for both transaction and retrieval requests. Meanwhile, transaction latency and throughput scale linearly with the number of users, and setup costs (both time and communication) on both sides scale linearly with the size of each group.

In summary, this paper makes the following contributions.

- New protocols for verifiable encryptions of secret values held in DPFs.

- The architecture and cryptographic techniques for Silent Splitter, a privacy-preserving payment splitting application that combines single server and multi-server privacy techniques, along with accompanying formal definitions and security arguments.
- A prototype implementation and evaluation of Silent Splitter that demonstrates the practicality and efficiency of our new techniques.

2 Background and Preliminaries

This section presents the cryptographic tools and notation that we will use throughout the paper. In addition to the tools described here, we will make use of several standard cryptographic primitives (MACs, PRFs [47], commitment schemes, zero knowledge proof systems [48]) and hardness assumptions (hardness of discrete logarithm, DDH [37]). Definitions and descriptions of these primitives can be found in standard texts (e.g., [13]).

Notation. Let $x \leftarrow F(y)$ denote the assignment of the output of $F(y)$ to x, and let $x \xleftarrow{R} S$ denote assignment to x of an element sampled uniformly random from set S. A function $\mathsf{negl}(x)$ is *negligible* if for all $c > 0$, there is a x_0 such that for all $x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We omit x if the parameter is implicit. Throughout the paper we also omit an implicit security parameter λ. Finally, we use $\perp$ to indicate an empty message or special character indicating failure. We use bracket notation to denote secret shares of values and vectors of values. That is, we use $[x]_1$ and $[x]_2$ to refer to values where $[x]_1 + [x]_2 = x$, and $[x] = ([x]_1, [x]_2)$.

We use the notation

$$\langle F_1(\mathsf{params}), F_2(\mathsf{params})\rangle \rightarrow \mathsf{output}$$

to denote an interactive protocol between parties running algorithms F_1 and F_2. If only one party has an output (and this is clear from context), the value output is the output of that party. If both parties have an output, then output is replaced by the tuple $(\mathsf{out}_1, \mathsf{out}_2)$.

Proofs of Knowledge. Proofs of knowledge allow a prover to convince a verifier in zero knowledge that it knows secret values $(x_1, ..., x_n)$ which cause a given statement ϕ on those values to be true [10, 34, 48]. We use the following notation and syntax to describe proofs of knowledge [22]. The variable pp represents public inputs to a statement that all parties involved in producing and verifying the proof will know.

- $\phi = \{(x_1, ..., x_n), pp : \text{statement about } pp, x_1, ..., x_n\}$
- $\mathsf{PoK.Prove}((x_1, ..., x_n), \phi) \rightarrow \pi$
- $\mathsf{PoK.Verify}(\phi, \pi) \rightarrow 1/0$

We will often keep the secret inputs to PoK.Prove implicit, explicitly writing out the statement ϕ and implying that the prover holds the relevant secret values.

Keyed-verification anonymous credentials (KVACs). A KVAC scheme is an anonymous credential scheme [20, 21, 29] where the issuer and verifier of credentials are the same party, and can therefore use a secret key sk in credential issuance/verification [25, 26].

KVAC schemes allow a credential issuer to produce credentials that include a number of values/attributes, some set S of which are known to the issuer, and some of which are issued blindly, meaning the issuer does not learn the attribute associated with the credential. When presenting a credential to the verifier, a credential holder can either show an attribute directly, or blindly prove a statement about the attributes held, proving the statement in zero-knowledge while keeping the attribute itself secret.

Following the notation of prior work, with slight modifications to match conventions used here, we will use a KVAC scheme with the following syntax.

- $\mathsf{CredKeyGen}(pp) \rightarrow \mathsf{sk}, iparams$
- $\langle \mathsf{BlindIssue}(\mathsf{sk}, S), \mathsf{BlindObtain}(iparams, (m_1, ..., m_n))\rangle \rightarrow cred$
- $\langle \mathsf{Show}(iparams, cred, (m_1, ..., m_n)), \mathsf{ShowVerify}(\mathsf{sk}, \phi)\rangle \rightarrow 1/0$

For brevity, we will generally leave the parameters *iparams* implicit in calls to this functionality.

The security properties that ensure credential can be issued and verified without the issuer/verifier seeing them are known as *blind issuance* and *anonymity*, respectively, and the property that a credential can only be proved if it has been issued is called *unforgeability*. An additional security requirement called *key-parameter consistency* requires that a malicious credential issuer cannot find different secret keys that correspond to the same public parameters [25].

Distributed point functions (DPFs). A point function $F_{\alpha,\beta}$ is one which evaluates to 0 everywhere in its domain, except for at input α, where $F_{\alpha,\beta}(\alpha) = \beta$. We refer to α as the *index* and β as the *value* of the function. A DPF is a technique for producing secret shares of evaluations of the function $F_{\alpha,\beta}$ that are more efficient than simply secret sharing the value of the function at each point in its domain. A DPF produces keys k_1 and k_2 that can be expanded to the evaluations of the shared function F, but which can be represented (in the 2 party setting) with size logarithmic in the domain of the function [14, 15, 46].

Since so many DPF applications require two servers to verify that they have received well-formed DPF keys k_1, k_2, and many such schemes exist in the literature [11, 12, 15, 32, 35, 44, 52, 59], we include a DPF verification protocol in our DPF syntax, shown below.

- $\mathsf{DPF.Share}(\alpha, \beta) \rightarrow k_1, k_2, \pi_1, \pi_2$
- $\langle \mathsf{DPF.S1Verify}(k_1, \pi_1), \mathsf{DPF.S2Verify}(k_2, \pi_2)\rangle \rightarrow 1/0$
- $\mathsf{DPF.Eval}(k, i) \rightarrow x$

The correctness property of the DPF requires that after using $\mathsf{DPF.Share}(\alpha, \beta)$ to produce keys k_1, k_2, it holds that

$$\mathsf{DPF.Eval}(k_1, i) + \mathsf{DPF.Eval}(k_2, i) = F_{\alpha,\beta}(i).$$

While there are a number of ways to formalize the confidentiality property required of DPFs, we adopt an indistinguishability-based definition [14]. A DPF is secure if an adversary who is given a strict subset of the DPF keys cannot distinguish which of two functions F_{α_0,β_0} and F_{α_1,β_1} has been shared. The DPF verification protocol requires a zero-knowledge property to ensure that it reveals nothing about the DPF key used to the other server, as well as a soundness

property that the parties will output 1 only if the function secret shared by k_1, k_2 is in fact a point function.

3 System Goals

This section gives a high level overview of the functionality and security goals of the Silent Splitter system, in addition to describing the overall architecture of the system.

3.1 Design Goals

A payment splitting application allows users to form groups and track debts among members. Within each group, the application keeps a balance for each group member representing how much that member owes other members or is owed by them. Members of a group may record real world transactions which result in another member becoming indebted to them by reducing the other group member's balance by some amount and increasing their own by the same amount. This process ensures that the sum total of balances within a group is always zero. Occasionally, group members will contact the application to "settle" their debts, asking the application to produce a list of all the group members' balances to facilitate offline repayment. Members of a group are expected to see intra-group balances of other group members, but only for the groups they have in common. A single user may be a member of multiple groups at once, but since the groups operate independently, a user will have a separate balance with each group.

A payment splitting application, as described here, must support three core operations:

- *Group registration*: A user can create a new group within the application, allocating a list of balances for group members, all initialized to zero.
- *Transaction processing*: Any member of a group can notify the application of a transaction involving another group member, thereby modifying the two balances to record how much is owed.
- *Balance settling*: A group member can obtain the list of balances within their groups, enabling them to coordinate settling debts between members.

We build the Silent Splitter system to support these operations, as any more advanced features offered by payment splitting apps can be built on top of this core functionality. For example, a transaction where one person pays for a group dinner with five others can be recorded as five separate transactions, one with each member of the group.

At first glance, it might appear that the functionality of Silent Splitter can easily be built on top of any metadata-hiding communication system [56], e.g., by having users send each other messages about how much they have spent without revealing to the platform who is messaging who. Note that in order to meet the strong security properties described below, the chosen metadata-hiding communication system needs to provide protection against traffic analysis attacks by a global passive network adversary, so low-latency solutions like Tor [38] would not suffice. While it is in principle possible to build payment splitting functionality on top of metadata-hiding messaging and have all the payment splitting logic be handled by the clients, this kind of approach would be less efficient and has proved unpopular in practice, even for simpler functionalities. For example, Signal previously used this kind of client-side management for maintaining private group membership, but in 2019 they transitioned to a centralized system where the Signal servers privately maintain group state [26, 54]. While the Signal app certainly worked fine before, the real-world intricacies of handling group consensus on the client side led to a worse user experience than what was provided by competitors with centralized group management. Thus we anticipate the same or greater benefits from centralization in the setting of payment splitting. That said, metadata-hiding messaging can accompany our payment-splitting functionality to provide human-readable transaction descriptions. See Section 6.3 for a more detailed comparison between our scheme and a client-only approach built on top of metadata-hiding messaging.

Security from splitting trust. To realize payment splitting functionality in a privacy-preserving way, the Silent Splitter scheme is run by two servers, S_1 and S_2. We refer to S_1 as the *primary* server, as it manages aspects such as user registration and authentication, and we rely on S_2 only for operations that require splitting trust. In this sense, Silent Splitter is a hybrid between single-server and split trust models. Our aim with this approach is to build a system that is more compatible with existing single-server payment splitting services, making it easier deploy the system and take advantage of existing mechanisms, e.g., for user authentication.

While a two-server split trust setup may be difficult to achieve in many scenarios, recent large-scale deployments of split-trust systems for private browser telemetry in Mozilla Firefox [3, 41] and evaluation of the Apple/Google Covid-19 exposure notification system [1, 2] indicate that this approach holds promise.

3.2 Security Goals

In order to provide a useful privacy preserving service, Silent Splitter must ensure that an adversary cannot learn the details of users' groups, transactions, or balances. This protection must hold both against other users and against the servers running the system. Moreover, the platform must ensure that a malicious user cannot corrupt the balances of their group, e.g., by making balances within the group sum to a non-zero amount, which would render the system inoperable.

At a high level, our attacker model considers malicious server operators who wish to learn things about users' groups or transactions as well as malicious users who wish to violate the reliable bookkeeping of the system. This section discusses each necessary security property in detail, and Section 3.3 mentions potential security considerations that are out of scope for this work. Appendix A states formal security definitions that correspond to our goals.

Confidentiality. We wish to protect users' and groups' confidentiality against an adversary who controls one of the Silent Splitter servers and arbitrarily many users. Specifically, neither server should learn the membership of any group beyond the identity of whichever user initially registered the group. Moreover, neither server should learn which transactions take place within which group, the identities of the users involved in a transaction, or the amount of money transferred in a transaction, unless they control

a user who belongs to that group (as group members are expected to see transactions and balances within their own groups).

When users settle, neither server should learn the retrieved balances. Although a Silent Splitter deployment may wish to integrate with some third-party service to facilitate repayment of debts, this is not part of the core protocol, and we design the protocol to allow for maximum privacy, e.g., a group of users who only repay each other offline in cash. Since our settling operation expects users to see the balances of all group members to facilitate repayment, a malicious user may see the balances of other users in any group to which it belongs. This is expected behavior and is therefore not considered a security violation.

In Appendix A, we present a definition for *transaction confidentiality*, which captures the notion that users' transactions should reveal nothing about what other group memberships they have (if any) or what monetary amounts or users the transactions involve. The security experiment defined there allows an adversary to create groups containing a mix of honest and malicious (adversary-controlled) users and to control one of the two servers. The adversary can then make transactions or settle while playing the role of malicious users, or compel an honest user to make transactions of its choosing or settle. At the heart of the experiment is the adversary's attempt to distinguish between two transactions. Specifically, the adversary creates two transactions and compels an honest user to make exactly one of them (with the user's choice between the two based on a challenger's secret input). It then must guess which transaction occurred. This experiment is subject to the requirement that this "challenge transaction" is made in a group of all honest users, so the adversary can't simply settle to see what transaction happened. Since the honest user can be a part of multiple groups, this experiment requires that an adversary cannot tell which groups, which target user, or what amounts are involved in any transaction.

Transaction confidentiality protects the confidentiality of user transactions, even when the adversary knows a priori, and can even dictate, the groups in which a user is a member. Separately, we require the notion of *group confidentiality*: that an adversary cannot learn which groups a user joins. We do not introduce a separate definition for group confidentiality, as the group confidentiality of our construction follows immediately from the anonymous credential scheme we employ.

Integrity. We need to ensure that Silent Splitter's strong confidentiality properties do not impede the system's ability to provide the intended payment splitting service. Thus we must prevent malicious clients from producing malformed transactions that corrupt balances or masquerading as other users in order to make transactions. Silent Splitter's integrity definition requires that every user's transactions are "well-formed," meaning that transactions run by a user U in group G only affect the balance of the user U in group G and another balance within the same group G.

We present an integrity definition that captures these requirements in Appendix A. As in the confidentiality experiment, the integrity security experiment allows an adversary to create groups of its choosing, comprising honest and malicious users. In this experiment, the challenger plays the role of both servers, and the adversary can interact with the servers as a malicious user. It can also compel honest users to make transactions of its choosing or run the settling protocol. For each group that contains only honest users, the experiment keeps track of the expected balances for each user based only on transactions compelled by the adversary. When an honest user settles, we say that the adversary wins the security game if one of the two following conditions are met.

- The sum of all the balances in the group is nonzero.
- The group members are all honest users, but at least one user's balance disagrees with the experiment's expected balance for that user.

If the adversary can cause one of these two conditions to be true, it has successfully induced a transaction that it should not have been able to create (i.e., in a group where it does not control any members) or in disrupting the necessary relationship among balances within a group.

Note that the check that balances in a group sums to zero suffices for groups containing a mix of honest and malicious users because a malicious user can introduce transactions that arbitrarily shift balances within one of its groups, so long as the overall sum of balances still comes out to zero. This means that, as is the case for non-private payment splitting apps, users should only join groups with others whom they trust not to abuse the privileges afforded by the platform. That said, our definition does suffice to rule out classes of attacks that a malicious user could attempt at the expense of honest users in its groups, e.g., siphoning money out of one group to increase its balance in another.

It is important to note that Silent Splitter, while providing strong integrity protections against malicious users, trusts servers to be available. This means that we do not protect data integrity against a server who decides to ignore requests or corrupt data. However, a malicious server should not be able to use such corruption or other malicious misbehavior to compromise confidentiality. This is the same security property provided by a number of private communication systems that use a similar split trust model [5, 31–33, 42, 44, 52, 59, 61].

3.3 Limitations and Non-Goals

A crucial limitation of Silent Splitter, shared even with non-private payment splitting systems, is that the system cannot prevent users from lying about real-world transactions and submitting fraudulent charges. It can only make sure that the charges users make are structurally well-formed. As is the case in any other payment app, if a user in a group records a fake transaction, other users can decline it, e.g., by charging back the same amount (this can be made to look like rejecting a charge on the user interface).

Since Silent Splitter relies on the notion that each group of users trusts each other enough to spend money on each other's behalf, our group setup process assumes that one user can privately register a group on behalf of everyone and communicate this fact to others in the group. While we do not explore more elaborate group management operations, the fact that the group registration component of Silent Splitter operates in a single-server setting means that we can benefit from other work in group management in this setting [26]. Note that since some user needs to contact the servers to initialize a new group, the servers will learn who creates a new group. The servers do not learn who else is in that group or when/whether any users make transactions in that group.

Finally, since Silent Splitter users need to connect to the servers to make transactions, servers can learn who makes each transaction, even if they don't learn who the transaction charges, how much is spent in the transaction, or what groups are involved. To prevent servers from learning anything about which users are actually making transactions, users can regularly produce cover traffic in the form of *dummy transactions* that involve \$0 charges. This prevents servers from learning which users are actually making transactions and which are simply producing cover traffic. We discuss cover traffic in detail in Appendix B.

4 New Protocols for DPFs

One of the cornerstones of our protocol is a technique through which DPF key holders may privately compute commitments to, and prove knowledge of, the information specified by a DPF. These techniques are also of independent interest and can be thought of as extensions and generalizations of recent work on access control for distributed point functions [50, 58].

Proving knowledge of the DPF value. We begin by showing a protocol by which a client can send two servers S_1, S_2 shares (k_1, k_2) of a point function $F_{\alpha,\beta} : X \to Y$ in addition to verifiably sending server S_1 a Pedersen commitment $c = g^\beta h^r$ to the value β. Here g and h are two generators of a cyclic group G of prime order where DDH is hard, and where the discrete log between g and h is unknown. Importantly, while evaluating a DPF over the domain X naturally requires $O(|X|)$ DPF evaluations (or a single full domain evaluation of size $O(|X|)$), our protocol will require only $O(1)$ additional operations in the group G.

The protocol consists of the client sending the servers shares $[r]_1, [r]_2$ of the commitment randomness r along with the DPF shares. The servers verify the DPF and then evaluate it over the domain X, summing the results of all the evaluations. That is, the servers each compute the value

$$[\beta]_b \leftarrow \Sigma_{i=1}^{n} \mathsf{DPF.Eval}(k_b, i), b \in \{1, 2\}.$$

This results in each server holding a share $[\beta]_1, [\beta]_2$ of β, since β is the only non-zero value in the distributed point function $F_{\alpha,\beta}$.

The servers then compute $c_i \leftarrow g^{[\beta]_i} h^{[r]_i}, i \in \{1, 2\}$, and server S_2 sends c_2 to S_1. Server S_1 computes

$$c \leftarrow c_1 c_2 = g^{[\beta]_1} h^{[r]_1} g^{[\beta]_2} h^{[r]_2} = g^\beta h^r.$$

Finally, the client also sends S_1 a non-interactive proof of knowledge π attesting to the statement $\{(r, \beta), c : c = g^\beta h^r\}$, which server S_1 verifies. This proof can be sent along with the initial secret shares and commitments, so the client only needs to send a single message to the servers during the whole protocol.

Proving knowledge of the DPF index. We now extend our protocol to also allow the client to prove knowledge of the DPF index α. Intuitively, the idea is to have servers compute commitments to the product $\gamma = \alpha\beta$ in addition to β, and then the client sends separate commitments to α, β, proving they correspond to the values in the originally computed commitment.

To facilitate this proof, the client sends two sets of random shares: $[r_\beta]_1, [r_\beta]_2$, to be used to commit to β as before, and new random shares $[r_\gamma]_1, [r_\gamma]_2$. The servers modify their DPF evaluation process to additionally multiply each DPF evaluation by the input index to the function. That is, the servers now compute the following values.

$$[\beta]_b \leftarrow \Sigma_{i=1}^{n} \mathsf{DPF.Eval}(k_b, i), b \in \{1, 2\}$$
$$[\gamma]_b \leftarrow \Sigma_{i=1}^{n} i \cdot \mathsf{DPF.Eval}(k_b, i), b \in \{1, 2\}$$

Since the point function $F_{\alpha,\beta}$ has its only non-zero point at index α with value β, these values will be shares of β and $\gamma = \alpha\beta$. With these shares in hand, the servers compute the following commitments.

$$c_{\beta i} \leftarrow g^{[\beta]_i} h^{[r_\beta]_i}, i \in \{1, 2\}$$

$$c_{\gamma i} \leftarrow g^{[\gamma]_i} h^{[r_\gamma]_i}, i \in \{1, 2\}$$

Server S_2 then sends $c_{\beta 2}, c_{\gamma 2}$ to S_1, who computes

$$c_\beta \leftarrow c_{\beta 1} c_{\beta 2}, \quad c_\gamma \leftarrow c_{\gamma 1} c_{\gamma 2}.$$

Finally, the client needs to send a commitment $c_\alpha \leftarrow g^\alpha h^{r_\alpha}$ to server S_1, along with a proof that the values α, β, γ committed to in $c_\alpha, c_\beta, c_\gamma$ satisfy the relationship $\gamma = \alpha\beta$. To facilitate this, we have the client additionally send server S_1 the value $c_{r\alpha} = g^{r_\alpha}$, and the server S_2 sends S_1 the values $g^{[r_\beta]_2}, g^{[r_\gamma]_2}$, which S_1 multiplies with $g^{[r_\beta]_1}, g^{[r_\gamma]_1}$ to produce $c_{r\beta} = g^{r_\beta}, c_{r\gamma} = g^{r_\gamma}$.

Now the server can use a generic proof of an encrypted DH-triple (Example 20.4 in v0.6 of the Boneh-Shoup textbook [13], made non-interactive via Fiat-Shamir in the random oracle model [8, 45]) to prove the statement

$$\{(\alpha, \beta, r_\alpha, r_\beta, r_\gamma), c_\alpha, c_\beta, c_\gamma, c_{r\alpha}, c_{r\beta}, c_{r\gamma} : c_\beta = g^\beta h^{r_\beta}, c_\alpha = g^\alpha h^{r_\alpha}, c_\gamma = g^{\alpha\beta} h^{r_\gamma}, c_{r\alpha} = g^{r_\alpha}, c_{r\beta} = g^{r_\beta}, c_{r\gamma} = g^{r_\gamma}\}.$$

The protocol ends with S_1 verifying the proof. As was the case for proving knowledge of just β, the client can send all its contributions to the protocol in a single non-interactive message.

The confidentiality of this scheme follows from the confidentiality of the DPFs, from DDH (since our (c_{rx}, c_x) tuples can be thought of as El-Gamal encryptions of g^x under public key h), and from the zero-knowledge property of the proofs used. The soundness of the scheme follows from the soundness of DPF verification, from the binding property of Pedersen commitments, and from the extractability of the zero-knowledge proof of knowledge scheme used. Proofs of the confidentiality and soundness of these subprotocols are contained within our main security proofs in Appendix D.

5 The Silent Splitter Payment Splitting System

This section describes the Silent Splitter payment splitting system in detail. The Silent Splitter protocol consists of three phases – setup, transactions, and settling – corresponding to the main operations supported by a payment splitting service. Here we describe both the core cryptographic protocols and the higher-level application design considerations involved in using the protocols.

In Silent Splitter, the servers S_1 and S_2 will hold secret shares of a database DB of account balances, stored as elements of $\mathbb{F}_q$. We will refer to the entries in these databases as *addresses*. Each user of the system is uniquely identified by a user id uid that is known to the servers and used for login/authentication to the system (the details of which are not considered here). Users will have a separate address for each group to which they belong, and addresses are

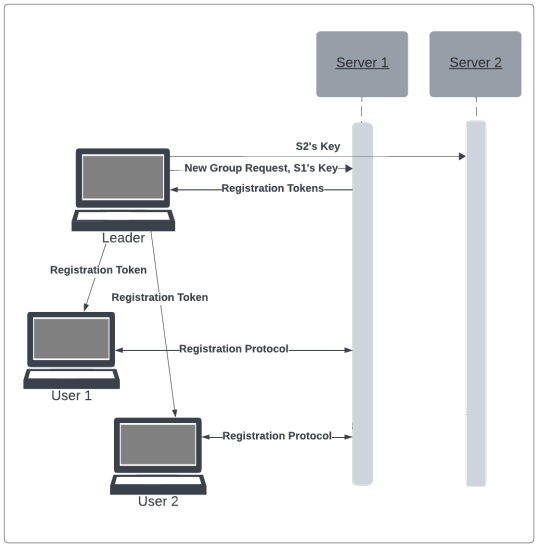

Figure 1: Diagram of the setup protocol showing communication flows.

allocated in contiguous blocks of M addresses per group, where M is a system-wide fixed maximum group size. We use N to denote the total number of groups registered in the system. The protocol makes use of a group G of prime order q with generators $g, h \in G$.

5.1 Group Setup

The setup phase allows users to create new groups. At the end of the setup phase, each member of the new group will hold a *group token* gt that can be used to create transactions and settle balances. Users receive a separate group token for each group to which they belong, and each user's group token is distinct from other members' tokens.

In our current prototype, group membership is static, and changes can only be made by creating a new group. We defer the question of applying privacy-preserving group management operations to future work but note that systems like Signal use the same credential system as us and are able to support such operations [26].

The setup phase itself consists of two parts. In the initialization protocol, a group leader U_1 interacts with the servers to create a new group. The servers necessarily learn who the group leader is but learns nothing else about the group composition. The leader U_1 contacts the other members $U_2, ..., U_M$ and gives each its *registration token* $\mathsf{rt}_i, i \in \{1, ..., M\}$. Each user then joins the group by interacting with S_1 in the registration protocol to obtain its group token gt_i. The full setup process is outlined in Figure 1.

Note that the few steps where U_1 interacts with other group members must be done without revealing communication metadata to the servers, lest this reveal group composition. This means the communication must happen out of band or via some metadata-hiding communication infrastructure. Moreover, for the overall system to enjoy the full privacy benefits of Silent Splitter, the mechanism used for this communication must provide the same level of metadata-hiding security as Silent Splitter itself. This is only a requirement for group setup, as after the setup phase, there is no longer a need for users to communicate cryptographic key material to each other.

Initialization protocol. Server S_1 allocates the next M available addresses for the account balances of the newly formed group. The leader U_1 selects two PRF keys ek_1 and ek_2 for a PRF $F : \mathcal{K} \times \mathbb{F}_q \to \mathbb{F}_q^M$, to be used during the settling phase. Key ek_1 is sent to S_1, and ek_2 is sent to S_2. Server S_2 is not involved in the group setup process after this step. If there is suitable public key infrastructure available, this process could be modified so that ek_2 is encrypted under S_2's encryption key, signed by the group leader, and sent to S_1 to be passed on to S_2, instead of directly sending to each server separately.

After allocating these M addresses, U_1 and S_1 produce the registration tokens rt_i for $i \in \{1, ..., M\}$. While this step logically comes after allocating addresses for the group, it can be done in parallel, meaning that the whole group initialization process requires only one round trip between the group leader and S_1.

The registration token rt_i is an anonymous credential issued by S_1 on the address aid_i (aid stands for "account id") and the corresponding user id uid_i to which that address is assigned. We employ the keyed-verification anonymous credential protocol by Chase et al. [24] to blindly issue the credential. This protocol allows U_1 to hide some or all of the attributes on which the credential is issued; in this case, U_1 sends aid_i in the clear and hides uid_i. Specifically, S_1 and U_1 participate in the protocol

$$\langle \mathsf{BlindIssue}(\mathsf{sk}_{S_1}, \mathsf{aid}_i), \mathsf{BlindObtain}(\mathsf{aid}_i, \mathsf{uid}_i) \rangle \to \mathsf{rt}_i.$$

After the initialization protocol, U_1 sends the other group members their registration tokens, as well as the keys ek_1 and ek_2. If the group contains fewer than M users, the additional registration tokens can be distributed among group members such that some members have multiple, but users do not need more than one for transactions.

Registration protocol. In the registration protocol, each user U_i exchanges a registration token rt_i for a group token gt_i to be used during the transaction phase. In principle, the setup phase could be eliminated and the registration protocol repeated as part of each transaction, but separating setup into a one-time step significantly reduces the computational cost of the transaction protocol.

To complete registration, a user U_i sends its user id uid_i and a commitment $\mathsf{com}_{\mathsf{aid}} \leftarrow g^{\mathsf{aid}_i} h^{r_{\mathsf{aid}}}$ to its assigned address aid_i to S_1. Using these values and the credential rt_i, received during initialization, the user completes the Show protocol described in [24],

$$\langle \mathsf{Show}(\mathsf{rt}_i, (\mathsf{aid}_i, \mathsf{uid}_i)), \mathsf{ShowVerify}(\mathsf{sk}_{S_1}, \phi) \rangle,$$

for the statement ϕ defined as

$$\{(\mathsf{aid}_i, r_{\mathsf{aid}}, \mathsf{uid}_i), \mathsf{com}_{\mathsf{aid}} : \mathsf{com}_{\mathsf{aid}} = g^{\mathsf{aid}_i} h^{r_{\mathsf{aid}}}, \mathsf{id}(U_i) = \mathsf{uid}_i\},$$

Here, we abuse our notation and use $\mathsf{id}(\cdot)$ to denote checking the user id of the user performing the protocol with S_1. This is a shorthand for standard user authentication that must occur as part of the protocol, not a statement to be proved in zero-knowledge, and uid_i is not kept secret in this step.

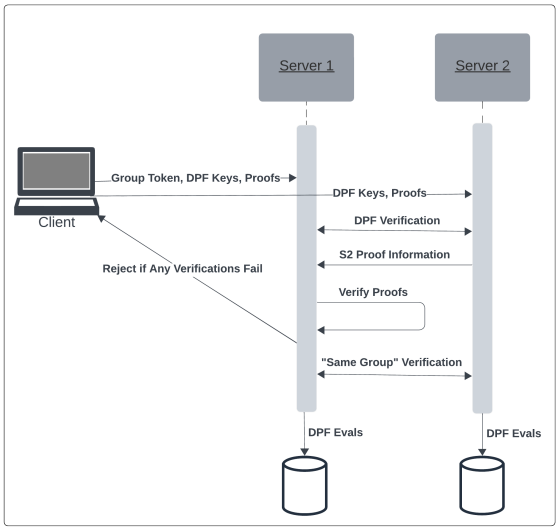

Figure 2: Diagram of the transaction protocol showing communication flows and the high-level steps of server-side transaction processing.

After verifying the credential from U_i, S_1 produces a MAC tag σ_i on $(\mathsf{uid}_i, \mathsf{com}_{\mathsf{aid}_i})$. U_i outputs the group token $\mathsf{gt}_i \leftarrow (\mathsf{uid}_i, \mathsf{com}_{\mathsf{aid}_i}, \sigma_i)$.

Observe that the registration protocol reveals nothing to the server that links the user U_i to the group created by U_1. The server S_1 learns that the user with uid_i has joined a group, but it learns nothing about which group has been joined. Moreover, everything sent from U_i to S_1 except uid_i is randomized, so U_i can repeatedly run the protocol with the same inputs. This allows users to run redundant instances of the registration protocol to provide cover traffic for other users who are joining new groups.

5.2 Transactions

In a given transaction, some user (the initiator) wants to request $\$x$ from another user (the target) in its group. Denote the initiator's address in the group by i, and the target's address in the group by j.

The basic idea. To make a transaction, the initiator produces two sets of DPF keys for the functions $F_{i,x}$ and $F_{j,x}$, where the domain of the DPF is the list of $M \cdot N$ registered addresses, and sends the resulting values to the servers. The servers can verify the DPFs and then evaluate them for each registered address, add the result into the balance stored at that address for the first function, and subtract the result for the second function. This set of operations results in adding shares of 0 to every address, except x is added to address i and subtracted from address j. Since the servers hold secret shares of the values involved, they cannot tell which addresses $i, j \in \{0, ..., MN-1\}$ are affected.

Protecting against malicious clients. This simple idea satisfies our confidentiality requirements, but it does nothing to prevent misbehavior by one or more malicious clients. To add integrity protections, we make use of our group tokens and proofs about DPFs from Section 4.

$\underline{\mathsf{ClientTx}(i, j, x, \mathsf{gt}, (r_{\mathsf{aid}_1}, ..., r_{\mathsf{aid}_n})) :}$

$(\mathsf{gt}_1, ...\mathsf{gt}_n) \leftarrow \mathsf{gt}$

for $\ell \in \{1, ..., n\}$:

$\quad (\mathsf{uid}_\ell, \mathsf{com}_{\mathsf{aid}_\ell}, \sigma_\ell) \leftarrow \mathsf{gt}_\ell$

$k_1, k_2, \pi_1, \pi_2 \leftarrow \mathsf{DPF.Share}(i, x)$

$k'_1, k'_2, \pi'_1, \pi'_2 \leftarrow \mathsf{DPF.Share}(j, x)$

$r_\alpha, [r_\beta]_1, [r_\beta]_2, [r_\gamma]_1, [r_\gamma]_2 \xleftarrow{R} \mathbb{F}_q^5$

$r_\beta \leftarrow [r_\beta]_1 + [r_\beta]_2$

$r_\gamma \leftarrow [r_\gamma]_1 + [r_\gamma]_2$

$c_\alpha \leftarrow g^i h^{r_\alpha};\ c_{r\alpha} \leftarrow g^{r_\alpha}$

$c_\beta \leftarrow g^x h^{r_\beta};\ c_{r\beta} \leftarrow g^{r_\beta}$

$\gamma \leftarrow i \cdot x$

$c_\gamma \leftarrow g^\gamma h^{r_\gamma};\ c_{r\gamma} \leftarrow g^{r_\gamma}$

//Prove knowledge of i, x and ownership of address i

$\phi \leftarrow \{(i, x, r_\alpha, r_\beta, r_\gamma, r_{\mathsf{aid}_1}, ..., r_{\mathsf{aid}_n}),$

$\quad c_\alpha, c_\beta, c_\gamma, c_{r\alpha}, c_{r\beta}, c_{r\gamma}, \mathsf{com}_{\mathsf{aid}_1}, ..., \mathsf{com}_{\mathsf{aid}_n}$:

$\quad c_\beta = g^x h^{r_\beta}, c_\alpha = g^i h^{r_\alpha}, c_\gamma = g^{ix} h^{r_\gamma},$

$\quad c_{r\alpha} = g^{r_\alpha}, c_{r\beta} = g^{r_\beta}, c_{r\gamma} = g^{r_\gamma},$

$\quad (\mathsf{com}_{\mathsf{aid}_1} = g^i h^{r_{\mathsf{aid}_1}}$

$\quad\quad \mathsf{OR}\ \mathsf{com}_{\mathsf{aid}_2} = g^i h^{r_{\mathsf{aid}_2}}$

$\quad\quad ...$

$\quad\quad \mathsf{OR}\ \mathsf{com}_{\mathsf{aid}_n} = g^i h^{r_{\mathsf{aid}_n}})\}$

$\pi \leftarrow \mathsf{PoK.Prove}((\alpha, \beta, r_\alpha, r_\beta, r_\gamma, r_{\mathsf{aid}_1}, ..., r_{\mathsf{aid}_n}), \phi)$

$\mathsf{S1out} \leftarrow (k_1, k'_1, \pi_1, \pi'_1, [r_\beta]_1, [r_\gamma]_1, \mathsf{gt}, c_\alpha, c_{r\alpha}, \pi)$

$\mathsf{S2out} \leftarrow (k_2, k'_2, \pi_2, \pi'_2, [r_\beta]_2, [r_\gamma]_2)$

output $(\mathsf{S1out}, \mathsf{S2out})$

Figure 3: Client steps to send a transaction.

At a high level, our approach will be to have the client produce a commitment c_i to i and to also send a list of its group tokens. It is important that the client send tokens for all its groups so that servers cannot link transactions to the use of a particular token. The client will then provide a proof that c_i is a commitment to the index of the DPF sent for $F_{i,x}$, and that i appears as the address aid in one of the provided group tokens. The servers verify the MACs on all provided group tokens as well as the client's proofs to ensure that the client is authorized to transfer funds to address i and is a member of the corresponding group.

Next, we need to verify that the addresses i and j are in the same group. Observe that if i and j are in the same group, they must be in the same consecutive block of M entries in DB. Thus we can verify that i and j are in the same group by having the servers sum their shares of each block of M consecutive evaluations of their shares of $F_{i,x}$ and $F_{j,x}$ and then take the difference between these vectors of sums.

$\mathsf{ServerTx}_b(\mathsf{DB}_b, k_b, k'_b, \pi_b, \pi'_b, [r_\beta]_b, [r_\gamma]_b, \mathbf{r},$
$(k_{MAC}, \mathrm{gt}, c_\alpha, c_{r\alpha}, \pi)/\perp)$:

//Verify and evaluate DPFs

$\mathsf{Ver} \leftarrow \langle \mathsf{DPF.S1Verify}(k_1, \pi_1, \mathsf{DPF.S2Verify}(k_2, \pi_2)) \rangle$

$\mathsf{Ver}' \leftarrow \langle \mathsf{DPF.S1Verify}(k'_1, \pi'_1) \mathsf{DPF.S2Verify}(k'_2, \pi'_2)) \rangle$

if $\mathsf{Ver} = 0$ or $\mathsf{Ver}' = 0$: output $\perp$

for $\ell \in \{0, ..., |\mathsf{DB}_b| - 1\}$:

  $f_{i\ell} \leftarrow \mathsf{DPF.Eval}(k_b, \ell)$

  $f_{j\ell} \leftarrow \mathsf{DPF.Eval}(k'_b, \ell)$

//Check client's proofs and tokens

$[\beta]_b \leftarrow \Sigma_{\ell=1}^{|\mathsf{DB}_b|} f_{i\ell}$

$[\gamma]_b \leftarrow \Sigma_{\ell=1}^{|\mathsf{DB}_b|} \ell \cdot f_{i\ell}$

$c_{\beta b} \leftarrow g^{[\beta]_b} h^{[r_\beta]_b}$; $c_{r\beta b} \leftarrow g^{[r_\beta]_b}$

$c_{\gamma b} \leftarrow g^{[\gamma]_b} h^{[r_\gamma]_b}$; $c_{r\gamma b} \leftarrow g^{[r_\gamma]_b}$

//Server S_1 does most of the proof verification work

if $b = 2$: send $c_{\beta 2}, c_{\gamma 2}, c_{r\beta 2}, c_{r\gamma 2}$ to S_1

if $b = 1$:

  receive $c_{\beta 2}, c_{\gamma 2}, c_{r\beta 2}, c_{r\gamma 2}$

  $c_\beta \leftarrow c_{\beta 1} c_{\beta 2}$; $c_{r\beta} \leftarrow c_{r\beta 1} c_{r\beta 2}$

  $c_\gamma \leftarrow c_{\gamma 1} c_{\gamma 2}$; $c_{r\gamma} \leftarrow c_{r\gamma 1} c_{r\gamma 2}$

  $(\mathrm{gt}_1, ..., \mathrm{gt}_n) \leftarrow \mathrm{gt}$

  for $\ell \in \{1, ..., n\}$:

    $(\mathsf{uid}_\ell, \mathsf{com}_{\mathsf{aid}\ell}, \sigma_\ell) \leftarrow \mathrm{gt}_\ell$

    if $\mathsf{MAC.Verify}(k_{\mathsf{MAC}}, (\mathsf{uid}_\ell, \mathsf{com}_{\mathsf{aid}\ell}), \sigma_\ell) \neq 1$:

      output $\perp$

  if $\mathsf{PoK.Verify}(\phi, \pi) \neq 1$: output $\perp$

//Check that both DPFs affect same group

for $\ell \in \{0, ..., N-1\}$:

  $[u_\ell]_b \leftarrow \Sigma_{m=\ell M}^{\ell M + M - 1} f_{im}$

  $[v_\ell]_b \leftarrow \Sigma_{m=\ell M}^{\ell M + M - 1} f_{jm}$

  $[w_\ell]_b \leftarrow ([u_\ell]_b - [v_\ell]_b) \cdot -1^b$

$h_b \leftarrow \Sigma_{i=1}^{N} r_i [w_i]_b$

exchange h_b

if $h_1 + h_2 = 0$:

  //apply changes to database

  for $\ell \in \{0, ..., |\mathsf{DB}_b| - 1\}$:

    $\mathsf{DB}_b[\ell] \leftarrow \mathsf{DB}_b[\ell] + f_{i\ell} + f_{j\ell}$

else : output $\perp$

Figure 4: Server-side transaction processing protocol for server $b \in \{1, 2\}$. Variables M and N refer to the maximum group size and the number of registered groups, respectively, so $|DB_b| = MN$. The statement ϕ is the same as the one shown in Figure 3.

Let $\mathbf{u}$ and $\mathbf{v}$ be the length-N vectors generated by summing consecutive blocks of M addresses in the shares produced by the servers' DPF keys. Concretely, the servers hold shares $[\mathbf{u}]_1$, $[\mathbf{v}]_1$ and $[\mathbf{u}]_2$, $[\mathbf{v}]_2$, respectively, of the following vectors.

$$\mathbf{u} = \left(\Sigma_{\ell=0}^{M-1} F_{i,x}(\ell),\ \Sigma_{\ell=M}^{2M-1} F_{i,x}(\ell),\ ...,\ \Sigma_{\ell=(N-1)M}^{NM-1} F_{i,x}(\ell)\right)$$

$$\mathbf{v} = \left(\Sigma_{\ell=0}^{M-1} F_{j,x}(\ell),\ \Sigma_{\ell=M}^{2M-1} F_{j,x}(\ell),\ ...,\ \Sigma_{\ell=(N-1)M}^{NM-1} F_{j,x}(\ell)\right).$$

If addresses i and j are indeed in the same group of M contiguous addresses, then there is only one non-zero index in each of these vectors, and it is at the same position in both vectors. Thus their difference is a vector $\mathbf{w}$ of all zeros: $\mathbf{w} = \mathbf{u} - \mathbf{v} = \mathbf{0}$. The servers can check this property to ensure that i and j are indeed in the same group.

Instead of having the servers reveal their shares to each other directly, we have the servers take a random linear combination of the entries in their shares of $\mathbf{w}$. Using a random vector $\mathbf{r}$ generated from a pre-shared seed, the servers compute the dot product $\langle \mathbf{r}, [\mathbf{w}]_b \rangle = \Sigma_{i=1}^{N} r_i [w_i]_b$. This reduces communication costs compared to sending shares of the entire vector $\mathbf{w}$.

We present an outline of the transaction protocol in Figure 2 and the full client and server transaction processes in Figures 3 and 4, respectively.

Human-readable transaction records. We have described how the Silent Splitter servers store and update user balances, but payment apps usually also allow users to add human-readable messages recording, e.g., the purpose of a given transaction. We describe two ways a Silent Splitter deployment can add support for this feature.

(1) **In-person exchange of records.** Since many transactions within a group occur when the relevant group members are physically present in the same location, the users' devices can communicate this additional information to each other out-of-band, e.g., via messages passed to each other over Bluetooth, or over a third-party messaging app. This approach allows users to keep each other's devices informed of transactions opportunistically while ensuring that the Silent Splitter servers always keep an authoritative and consistent record of balances.

(2) **In-app activity log.** We augment the group setup phase to include selection of a group secret s. Users upload a message encrypted under s with each transaction and receive a message index i in return from the servers. These messages identify the relevant group id and include human-readable descriptive text. The app makes available a log of all recently received transactions. Users can receive message indices for relevant messages out-of-band from other members of their groups, or they can scan the transaction log to find relevant messages that their secret s can decrypt correctly. This approach is more costly in terms of computation and storage, but it serves as a backup method for exchanging transaction messages when in-person exchange is not possible.

We note that this aspect of private payment splitting closely resembles general-purpose anonymous group messaging. Whereas the core payment splitting protocol requires additional structure to make security guarantees about the semantic content of messages sent to the group, i.e., that a transaction is well-formed, there is

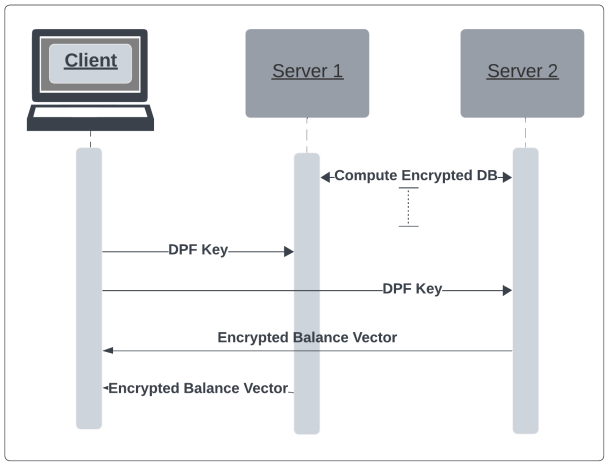

Figure 5: Diagram of settling protocol showing communication flows between parties. The servers can compute the encrypted DB once and reuse it for multiple settling operations.

no comparable requirement of a text description of a transaction's real-world purpose.

5.3 Retrieving and Settling Balances

Settling balances in Silent Splitter requires a user to download the list of account balances for users in their group. Once a user has the balances for their group, they can compute how much to pay which users to settle balances, e.g., in cash or via a third-party payment service. We discuss various possibilities for real-world repayment of balances in Appendix C.

Since the servers cannot learn which group's balances a user is requesting, users cannot just ask for a given group's balances. Instead, we use a PIR approach. Unfortunately, we cannot directly use PIR on the secret-shared database of account balances, as PIR requires the database to be replicated across two servers, and simply merging the shares before running PIR would reveal account balances. Our solution is to have the servers use the keys ek_1 and ek_2, generated during the setup of each group, to mask balances before merging them. Since each server holds only one of the keys, neither will see the account balances when merged, but group members can remove the masks and read the balances. This mechanism doubles as an access control protection, preventing users from reading others' balances. A similar idea has been used for access control in the private messaging space [44].

To enable settling, the servers convert their shares DB_1 and DB_2 of DB into a masked database DB' by sharing a secret random seed $r \xleftarrow{R} \mathbb{Z}_q$ and using it as an input to the PRF F, keyed separately for each group. That is, the servers separately encrypt each group's balances in CTR-mode before merging their shares. More formally for $b \in \{1, 2\}$, and using bracket notation to denote access to addresses, the server b computes, for each $i \in \{0, ..., |\mathsf{DB}| - 1\}$,

$$\mathsf{DB}'_b[i] \leftarrow \mathsf{DB}_b[i] + F(\mathsf{ek}_{b,j}, r + (i \bmod M)),$$

where $\mathsf{ek}_{b,j}$ is the key ek_b for the $j = \lfloor i/M \rfloor$th group.

$\underline{\mathsf{ServerSettleSetup}_b(\mathsf{DB}_b, \mathsf{ek}_{b1}, ..., \mathsf{ek}_{bN}, r)}$:
for $i \in \{0, ..., |\mathsf{DB}| - 1\}$:
  $j \leftarrow \lfloor i/M \rfloor$
  $\mathsf{DB}'_b[i] \leftarrow \mathsf{DB}_b[i] + F(\mathsf{ek}_{b,j}, r + (i \bmod M))$
exchange DB'_b
$\mathsf{DB}' \leftarrow \mathsf{DB}'_1 + \mathsf{DB}'_2$
output DB'

$\underline{\mathsf{ClientSettleRequest}(\mathsf{aid})}$:
$j \leftarrow \lfloor \mathsf{aid}/M \rfloor$
$f_1, f_2 \leftarrow \mathsf{DPF.Share}(j, 1)$ //Domain is $0, ..., N - 1$
output (f_1, f_2)

$\underline{\mathsf{ServerSettle}_b(\mathsf{DB}', f_b, r)}$:
for $\ell \in \{0, ..., M - 1\}$:
  $\mathsf{resp}[\ell] \leftarrow \Sigma_{i=0}^{\mathsf{N}-1}\big(\mathsf{DPF.Eval}(f_b, i) \cdot \mathsf{DB}'[iM + \ell]\big)$
output (resp, r)

$\underline{\mathsf{ClientSettle}(\mathsf{i}, \mathsf{ek}_1, \mathsf{ek}_2, \mathsf{resp}_1, \mathsf{resp}_1, \mathsf{r})}$:
for $i \in \{0, ..., M - 1\}$:
  $\mathsf{resp}[i] \leftarrow \mathsf{resp}_1[i] + \mathsf{resp}_2[i]$
  $\mathsf{bals}[i] \leftarrow \mathsf{resp}[i] - F(\mathsf{ek}_1, r + i) - F(\mathsf{ek}_2, r + i)$
output bals

Figure 6: Settling protocol for client and server $b \in \{1, 2\}$. Variables M and N refer to the maximum group size and number of regisered groups, respectively. We omit the DPF verification proofs from the DPF output because they are not used in this protocol. The SettleSetup **input r can be selected by the servers at random or, if only one client will settle, by that client.**

The servers then merge $\mathsf{DB}' \leftarrow \mathsf{DB}'_1 + \mathsf{DB}'_2$. This operation could be computed repeatedly each time a user requests to settle, but it can also be precomputed on a regular basis to allow multiple users to retrieve data from the same instance of DB'. If the database DB' is generated for a single client to settle, the value of $r \xleftarrow{R} \mathbb{Z}_q$ could equivalently be sampled by the client instead of the servers.

When a user wishes to retrieve balances for their group, they first compute the group index $j \leftarrow \lfloor \mathsf{aid}/M \rfloor$ for the value of aid from their registration token. The user then produces a DPF keys f_1, f_2 for a point function defined over the domain $\{0, ..., \mathsf{N} - 1\}$ that evaluates to 1 at index j. It sends this DPF to the servers, who (for $b \in \{1, 2\}$) compute and return the masked balance vectors

$$\mathsf{resp}[\ell] \leftarrow \Sigma_{i=0}^{\mathsf{N}-1}\big(\mathsf{DPF.Eval}(f_b, i) \cdot \mathsf{DB}'[iM + \ell]\big)$$

for $\ell \in \{0, ..., M - 1\}$. This is the vector of masked balances for all M addresses in the group. The result of this operation is equivalent to the client simultaneously making separate DPF PIR queries for each of the M balances in their group, but this approach improves

performance by reducing the DPF domain by a factor of M. The servers also send the user the choice of r that was used to form DB'.

Note that since all entries in DB' are masked with secret keys, the servers do not need to verify that the DPF keys f_0, f_1 are well-formed, making the generation and evaluation of the DPF in the settling phase slightly more efficient.

Having retrieved its DPF PIR response $\mathsf{resp} \in \mathbb{F}_q^M$ and r, the receiving user computes the balance vector bals as

$$\mathsf{bals}[i] \leftarrow \mathsf{resp}[i] - F(\mathsf{ek}_1, r+i) - F(\mathsf{ek}_2, r+i)$$

for $i \in \{0, ..., M-1\}$.

All the steps of the settling protocol are summarized in Figure 6 with the communication patterns outlined in Figure 5.

5.4 Security

Intuitively, the group and transaction confidentiality of our scheme follow from the fact that the view of an adversary at every stage of the protocol cannot be distinguished from random. Group registration uses a KVAC scheme with a zero-knowledge property, transactions are made using DPFs to privately write into an address, supported by a zero-knowledge proof over a number of group elements, which act as an El-Gamal encryption of the client's secrets. The settling protocol masks the databases held by the servers with the output of a PRF on a key unknown to the servers and uses a DPF to privately retrieve the contents of a user's group. The only information visible to the servers in the course of the protocol is the list of group tokens of a user making a transaction, which allows identifying the user making a transaction, but does not reveal anything about the user's groups or transactions. We discuss how to use cover traffic to minimize this leakage in Appendix B.

Likewise, integrity largely follows from the various integrity properties of the tools used to build the protocol. An additional argument is needed, however, to show that the statements we prove via DPF verification and zero-knowledge proofs of knowledge suffice to show that an adversary cannot disrupt groups' balances.

We provide proof sketches for the following theorems in Appendix D.

Theorem 5.1 (Transaction Confidentiality)**.** *Assuming the security of the underlying KVAC scheme, the zero-knowledge property of the proof system, the confidentiality of the DPF scheme, the security of the PRF F, and the hardness of DDH in G, our scheme satisfies the transaction confidentiality definition for payment splitting schemes (Definition A.1).*

Theorem 5.2 (Integrity)**.** *Assuming the unforgeability of the underlying KVAC scheme, the extractability of the zero-knowledge proof system, the existential unforgeability of the MAC scheme, the soundness of the DPF verification proofs, and the hardness of discrete log in G, our scheme satisfies the integrity definition for payment splitting schemes (Definition A.2).*

6 Implementation and Evaluation

This section describes our implementation of Silent Splitter, inclusive of the setup, transaction, and settling protocols, as well as our evaluation of the system's performance and a comparison to a naïve approach where payment splitting groups are maintained by clients on top of a metadata-hiding messaging system.

Table 1: Group setup computation time. Initialization is a one-time process between a single user and server to create a group, followed by each group member registering to join the group.

Group Size	Initialization	Registration
10	0.44s	0.08s
15	0.55s	0.08s
20	0.62s	0.08s
35	0.89s	0.08s
50	1.24s	0.08s
100	2.16s	0.08s

6.1 Implementation

We implemented our system in 4000 lines of Rust, publicly available at https://github.com/mapierce23/psa. We utilized the Rust implementation of verifiable, incremental DPFs created for [11], extending their sketching protocol to provide support for DPF values other than 0 or 1. For efficiency, we also added an optimized full-domain DPF evaluation function. For the keyed-verification anonymous credentials in the registration protocol, we utilized cmz, Ian Goldberg's implementation of [25], publicly available at https://git-crysp.uwaterloo.ca/iang/cmz.git. All group operations are performed in the Ristretto group, built on the elliptic curve Curve25519. The DPFs are evaluated over the field F_q, where q is the order of the Ristretto group. In the settling phase, the database is encrypted using AES in counter mode, with Rust's aes128 crate. We also utilized Henry de Valence's zkp create for ease in producing and verifying our system's non-interactive zero knowledge proofs.

6.2 Evaluation

We ran our implementation on Google Cloud compute instances, using two instances with 32 vCPUs, 16 cores, and 128GB memory each. These instances simulated the two servers; S1 was located in the western US and S2 was located in the central US, with the round-trip time between them 38.6 ms. An HP Laptop (Intel® Core™ i7-10510U CPU @ 1.80GHz, 2304 Mhz, 4 Cores, 8 Logical Processors) running Windows 10.0.19045 with 16GB memory played the role of the clients. Round-trip time between the client and servers was found to be 74.4 ms ($S1$) and 38.4 ms ($S2$), while round-trip time between the two servers was 38.67 ms. Bandwidth between the servers was 602 Mb/s and from client to servers was 230 Mb/s ($S1$) and 296 Mb/s ($S2$). We measured client and server communication costs, defined as the bytes sent by each party during the protocol, the latency and throughput of transactions, and the costs of group setup and settling.

6.2.1 Setup The setup protocol consists of two parts: group initialization, where a leader interacts with S_1 to obtain registration tokens, and user registration, where a user interacts with S_1 to exchange its registration token for a group token. We tested group sizes from $M = 10$ to $M = 100$. Group setup times, measuring total computation across all parties involved, are listed in Table 1. Note that even for large groups of size 100, this one-time creation of registration tokens takes only 2.16 seconds. We found that individual user registration takes 0.08 seconds independent of all parameters.

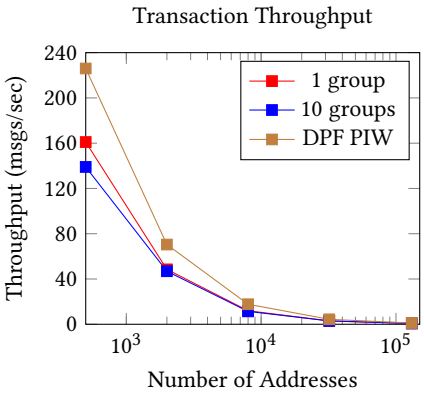

Figure 7: Transaction Throughput, compared to the DPF PIW Baseline, when clients each belong to 1 or 10 groups. As the number of addresses increase, performance is limited primarily by the servers' AES throughput, since DPF evaluations consist primarily of repeated AES evaluations.

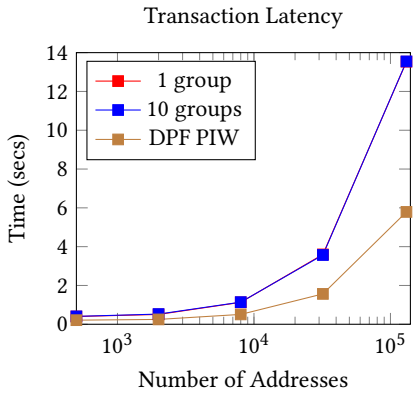

Figure 8: Transaction Latency when clients each belong to 1 or 10 groups, as compared to the DPF PIW Baseline. Increasing the number of groups has a barely perceptible impact on latency, as the group size-independent DPF proof costs dominate the protocol's computational costs.

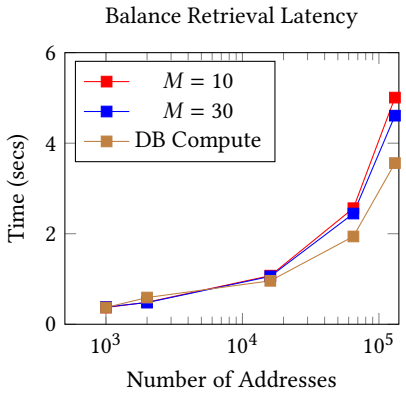

Figure 9: The DB Compute line shows the time required for the servers to produce the encrypted database used for settling, which can be reused for many users' queries and constitutes most of the cost. As before, number of groups has minimal impact. Variable M is the number of users per group.

Table 2: Setup communication costs for clients [C] and servers [S]. Initialization costs increase linearly with group size, while registration remains constant.

Group Size	Init. [C]	Init. [S]	Registration [C + S]
10	2698B	4120B	265B
15	4023B	6080B	265B
20	5348B	8040B	265B
35	9323B	13920B	265B
50	13298B	19800B	265B
100	26548B	39400B	265B

Table 3: Transaction communication (KB sent) for clients in 1 or 3 groups. Each group token adds 136B to client cost, and costs increase logarithmically in database size.

DB Size	Client [1 Gp]	Client [3 Gps]	Servers
1000	2.83KB	3.10KB	1.17KB
2000	2.87KB	3.14KB	1.17KB
4000	2.91KB	3.18KB	1.17KB
8000	2.94KB	3.21KB	1.17KB
16000	2.98KB	3.25KB	1.17KB
32000	3.02KB	3.29KB	1.17KB

These costs are due to the proofs that need to be created and verified when using anonymous credentials.

We also recorded communication costs for clients and servers in the setup phase, shown in Table 2. Group initialization requires 265B sent per group member on the client side and 392B per group member on the server side in addition to the small fixed cost of sending the PRF keys and server public key. This is again due to the proofs required by anonymous credentials. Thus, communication costs for setup scale linearly with the size of M, requiring tens of kilobytes of communication both ways when $M = 100$. For user registration, communication costs are modest at 265B on both the client-side and the server-side. These protocols are the most expensive part of the system and are only run during the setup phase.

6.2.2 Transactions We anticipate that transactions will account for the vast majority of requests in Silent Splitter. Unlike the setup costs, transaction costs are independent of the group size; instead, they scale with the database size, i.e., the total number of registered addresses.

Client-side costs. Our results show that client-side computation and communication costs are essentially constant, irrespective of the database size. Computation costs are modest: generating the DPF keys and preparing the proof took approximately 14ms. Communication costs from client to server are on the order of 2-3KB, depending on the server in question (recall that the client's group tokens and the zero-knowledge proof are sent only to Server 1). We present the specific communication costs in Table 3, where we list the Server 1 costs because those are larger.

We also note that in our current implementation, the client generates and sends the Beaver multiplication triples needed for the DPF sketching protocol outlined in [11] (though our extension requires more triples). As mentioned in that paper, these triples can be compressed, although they do not include this optimization in their implementation, upon which ours is based. Such a compression would significantly lessen our communication costs, as the triples currently account for about 600B of our transaction requests.

Server-side costs. Server communication is constant: each server sends 1KB to the other while verifying the validity of a transaction. Meanwhile, server computation costs during a transaction are the most intensive part of the protocol. This is primarily due to the cost of evaluating and verifying the DPFs, but it also involves the verification time of the zero-knowledge proof. Our trials show that the proof verification time is independent of database size and is

only slightly impacted by the number of group tokens the client sends (since the zero-knowledge proof involves an OR-proof over the group tokens). When a client belongs to 3 groups, for example, proof verification takes 4.5 ms, versus 3.1 ms when it belongs to only one group.

Latency and Throughput. We report the latency and throughput of our end-to-end implementation of Silent Splitter in Figures 7 and 8. Since database size is the primary factor impacting computation time, we report these numbers for a wide range of database sizes, where the database size is the number of registered addresses in the system (number of groups times group size). As stated previously, the other factor impacting computation time is how many groups to which the client belongs, as this affects proof verification time. We report the latency and throughput for cases where users belong to 1 or to 10 groups.

To establish a baseline for comparison, we note that every transaction requires the servers to run the DPF Private Information Writing (PIW) protocol once for each key. This consists of a full-domain DPF evaluation, the production and verification of sketches, and the actual write to the database. Thus, at minimum, transactions in Silent Splitter are as expensive as the DPF PIW protocol. We tested a version of our system where transactions consisted only of the DPF PIW components and report the results in Figures 7 and 8. Our results show that Silent Splitter itself adds little overhead to this baseline demanded by the DPF PIW protocol. Moreover, increasing the number of groups to which users belong has an almost imperceptible impact on performance because each additional group adds one more MAC verification and one more commitment to the proof of knowledge.

6.2.3 Settling In the settling protocol, the client simply prepares and sends two DPF keys, which are lightweight and fast to compute. The heavyweight computation and communication belongs to the servers, which first encrypt each address of the database using AES in counter mode, then merge their encrypted database shares before completing the the rest of the protocol. As a result, server-side communication and computation costs both grow linearly with the size of the database. Specifically, balance retrieval latency is bounded by the speed with which the servers can perform AES evaluations for each address. We report this latency in Figure 9.

To optimize this step, the servers can store an encrypted version of the database which they re-compute periodically rather than on-demand after each request. Then, when a settling request is received, the servers could simply use the most recently encrypted version of the database rather than encrypting it from scratch. This would save time as well as minimize the communication costs of the servers having to send each other their encrypted vectors. To demonstrate the improvement this would incur, we have included the cost of computing only the encrypted database as a baseline in Figure 9.

6.3 Discussion: Client-Side vs Server-Side Payment Splitting Groups

Having described the performance properties of the Silent Splitter system, we now revisit a possible alternative approach described in Section 3.1. Instead of using Silent Splitter, clients could use group messages on top of a metadata-hiding communication system to broadcast transactions to group members, and each client could keep its own local record of what transactions have happened in the group. As mentioned earlier, this approach may run into issues resulting from a lack of a single, centralized, and authoritative source of the true set of balances within a group, but we will now explore the performance properties such a scheme may exhibit, as compared to Silent Splitter.

Since many metadata-hiding messaging schemes that provide the correct level of security for comparison to Silent Splitter make use of DPFs [32, 44, 52, 59], we can use the performance of our DPF implementation as a rough estimate of the performance of such a scheme, although in reality performance would be worse due to the additional overhead needed to support messaging. Next, we need clients to have a way to form groups on top of the metadata-hiding messaging scheme. Fortunately, standards like MLS [7], or metadata-hiding variants thereof [49], provide more than sufficient support for this. With these pieces in place, users can send MLS messages of the form, "Alice charges Bob $x" over the metadata-hiding system to represent transactions.

Such an approach supports balance retrieval "for free" because clients can just check their own local transaction records to recover balances within each group. Latency would remain comparable to the performance shown in Figure 8, modulo any additional overhead due to messaging on top of the DPF. On the other hand, transaction throughput would become significantly worse. Since each transaction needs to be sent to every member of the group, this approach incurs an $M\times$ overhead per transaction. Loosely speaking, this would result in an $M\times$ shift down of each point in the DPF line in the throughput graph in Figure 7. Given that Silent Splitter's throughput is so close to that of just running a DPF without considering any other messaging-related overhead, this client-side approach would exhibit considerably worse throughput, regardless of the chosen group size.

In light of this comparison, Silent Splitter's design can be thought of as an approach that collapses the metadata-hiding, group management, and transaction processing layers of the client-only approach in order to dramatically improve transaction throughput in the resulting system. While a naïve implementation of the payment splitting functionality would have an order of magnitude additional overhead compared to metadata-hiding messaging, Silent Splitter closely tracks the performance of metadata-hiding messaging alone, despite the additional constraints that the payment splitting problem places on message structure and content. An interesting direction for future work would be to consider how to build payment splitting functionality over other forms of metadata-hiding messaging with potentially weaker privacy guarantees. This could allow for different tradeoffs between performance and security than the one offered by Silent Splitter.

7 Additional Related Work

To our knowledge, the only prior work to directly consider the question of privacy-preserving payment splitting is a system of Eskandarian et al. [43]. That work is motivated by the same payment splitting problem addressed here, but it focused exclusively on hiding information about transactions within a group, revealing

all group membership information to the server and requiring that each transaction explicitly identifies the group to which it belongs.

Our problem is closely related to privacy-preserving digital currencies and general-purpose anonymous communication. The rest of this section briefly discusses these areas.

Private digital currency. An extensive body of work – beginning with Chaum's blind signature techniques [27, 28] – proposes various schemes for private digital currencies [6, 16–19, 23, 30, 53, 62]. More recently, Bitcoin [51] and other decentralized currencies, especially those targeting private transactions, starting with zerocash [9], have also become a focus for research on private transactions. While they sometimes use similar techniques to our work, these systems operate in a very different setting, and solutions for general-purpose private currencies, decentralized or not, cannot directly be applied as backends for payment splitting apps.

Anonymous communication. Payment splitting can be thought of as an augmented form of anonymous communication, and anonymous messaging in particular. The Tor system is the best known system in this area [38], but the DPF techniques used here have also formed the basis of a number of anonymous messaging schemes, e.g., Riposte [32], Express [44], Blinder [4], Sabre [59], and Spectrum [52]. These systems use DPFs for private writing, much like Silent Splitter, and each offers a unique mechanism to either verify DPFs directly or otherwise validate DPF writing transactions end-to-end. An important distinction between payment splitting and general-purpose anonymous messaging is that payment splitting places additional restrictions on the content and structure of messages compared to messaging. From this perspective, our scheme can be thought of as combining the techniques of DPF-based anonymous messaging with a generalization of recent work on access control for function secret sharing schemes [50, 58] and anonymous credential techniques used for private currencies or group management in messaging systems [25, 26].

8 Conclusion

We have presented Silent Splitter, a private payment splitting system that allows groups of users to keep track of what they owe each other without the application infrastructure learning anything about the groups they interact with, the people involved in their transactions, or the amounts of money they spend with their friends. Silent Splitter relies on split trust techniques and introduces minimal additional overhead compared to standard private writing with DPFs. This combination of functionality and performance is made possible by new techniques for proving knowledge of properties of DPFs, allowing users to efficiently prove that they are only making transactions in groups to which they belong, and only using public-key cryptography and zero-knowledge proofs on statements that do not scale with the number of users or groups in the overall system.

Acknowledgments

We would like to thank the anonymous reviewers and revision editor for their helpful comments and feedback to improve this paper.

This material is based upon work supported by the National Science Foundation under Grant No. 2234408. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] 2021. Analytics in Exposure Notifications Express: FAQ. https://github.com/google/exposure-notifications-android/blob/master/doc/enexpress-analytics-faq.md. Accessed 5/1/2023.
[2] 2021. Exposure Notification Privacy-preserving Analytics (ENPA) White Paper. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf. Accessed 5/1/2023.
[3] Josh Aas and Time Geoghegan. 2020. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/. https://www.abetterinternet.org/post/introducing-prio-services/
[4] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder: MPC Based Scalable and Robust Anonymous Committed Broadcast. In *ACM CCS*.
[5] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous Messaging via Secure Multiparty Computation. In *USENIX Security*.
[6] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. 2015. Anonymous Transferable E-Cash. In *PKC*.
[7] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. 2023. The Messaging Layer Security (MLS) Protocol. RFC 9420. https://doi.org/10.17487/RFC9420
[8] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS*.
[9] 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 Symposium on Security and Privacy*.
[10] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, Janos Simon (Ed.). ACM, 103–112.
[11] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 762–776.
[12] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2023. Arithmetic Sketching. Cryptology ePrint Archive, Paper 2023/1012. https://eprint.iacr.org/2023/1012 https://eprint.iacr.org/2023/1012.
[13] Dan Boneh and Victor Shoup. 2023. *A Graduate Course in Applied Cryptography (version 0.6)*. https://cryptobook.us.
[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *EUROCRYPT*.
[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *ACM CCS*.
[16] Stefan Brands. 1993. Untraceable Off-line Cash in Wallets with Observers (Extended Abstract). In *CRYPTO*.
[17] Jan Camenisch. 1998. *Group signature schemes and payment systems based on the discrete logarithm problem*. Ph. D. Dissertation. ETH Zurich, Zürich, Switzerland.
[18] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *EUROCRYPT*.
[19] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2006. Balancing Accountability and Privacy Using E-Cash (Extended Abstract). In *SCN*.
[20] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. 93–118.
[21] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*. 56–72.
[22] Jan Camenisch and Markus Stadler. 1997. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. 410–424.
[23] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. 1998. Easy Come - Easy Go Divisible Cash. In *EUROCRYPT*.
[24] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. 2020. Secret Shared Shuffle. In *ASIACRYPT*.
[25] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. 2014. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *Proceedings of the 2014 ACM*

SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 1205–1216.
[26] Melissa Chase, Trevor Perrin, and Greg Zaverucha. 2020. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1445–1459.
[27] David Chaum. 1982. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*. 199–203.
[28] David Chaum. 1983. Blind Signature System. In *CRYPTO*.
[29] David Chaum. 1985. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* 28, 10 (1985), 1030–1044.
[30] David Chaum, Amos Fiat, and Moni Naor. 1988. Untraceable Electronic Cash. In *CRYPTO*.
[31] Raymond Cheng, Will Scott, Bryan Parno, Irene Zhang, Arvind Krishnamurthy, and Thomas Anderson. 2016. *Talek: a Private Publish-Subscribe Protocol*. Technical Report UW-CSE-16-11-01. University of Washington Computer Science and Engineering, Seattle, Washington. http://www.cs.washington.edu/public_files/grad/tech_reports/UW-CSE-16-11-01.pdf
[32] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *IEEE Symposium on Security and Privacy*.
[33] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *USENIX Security*.
[34] Ronald Cramer. 1997. *Modular Design of Secure yet Practical Cryptographic Protocols*. Ph. D. Dissertation.
[35] Leo de Castro and Antigoni Polychroniadou. 2022. Lightweight, Maliciously Secure Verifiable Function Secret Sharing. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 150–179.
[36] Richard A. DeMillo and Richard J. Lipton. 1978. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.* 7, 4 (1978), 193–195.
[37] Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inf. Theory* 22, 6 (1976), 644–654.
[38] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*.
[39] Cynthia Dwork. 2006. Differential Privacy. In *ICALP*.
[40] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, 265–284.
[41] Steve Englehardt. 2019. Next steps in privacy-preserving Telemetry with Prio. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/
[42] Saba Eskandarian and Dan Boneh. 2022. Clarion: Anonymous Communication from Multiparty Shuffling Protocols. In *NDSS*.
[43] Saba Eskandarian, Mihai Christodorescu, and Payman Mohassel. 2020. Privacy-Preserving Payment Splitting. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 67–88.
[44] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. 2021. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. In *USENIX Security*.
[45] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*.
[46] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT*.
[47] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. On the Cryptographic Applications of Random Functions. In *CRYPTO*.
[48] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18, 1 (1989), 186–208.
[49] Keitaro Hashimoto, Shuichi Katsumata, and Thomas Prest. 2022. How to Hide MetaData in MLS-Like Secure Group Messaging: Simple, Modular, and Post-Quantum. In *ACM CCS*. ACM, 1399–1412.
[50] Keyu Ji, Bingsheng Zhang, and Kui Ren. 2023. Fine-grained Policy Constraints for Distributed Point Function. *IACR Cryptol. ePrint Arch.* (2023), 1672.
[51] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer Electronic Cash System.
[52] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. 2022. Spectrum: High-bandwidth Anonymous Broadcast. In *NSDI*.
[53] Tatsuaki Okamoto and Kazuo Ohta. 1989. Disposable Zero-Knowledge Authentications and Their Applications to Untraceable Electronic Cash. In *CRYPTO*.
[54] Jim O'Leary. 2019. Technology Preview: Signal Private Group System. https://signal.org/blog/signal-private-group-system/.
[55] Torben P. Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings (Lecture Notes in Computer Science, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, 129–140.
[56] Sajin Sasy and Ian Goldberg. 2024. SoK: Metadata-Protecting Communication Systems. *Proc. Priv. Enhancing Technol.* 2024, 1 (2024), 509–524.
[57] Jacob T. Schwartz. 1980. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM* 27, 4 (1980), 701–717.
[58] Sacha Servan-Schreiber, Simon Beyzerov, Eli Yablon, and Hyojae Park. 2023. Private Access Control for Function Secret Sharing. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 809–828.
[59] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 1953–1970.
[60] Venmo. 2023. Introducing Venmo Groups. https://newsroom.paypal-corp.com/2023-11-14-Introducing-Venmo-Groups.
[61] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *OSDI*.
[62] Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. 2018. PRCash: Centrally-Issued Digital Currency with Privacy and Regulation. *IACR Cryptology ePrint Archive* (2018).
[63] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings (Lecture Notes in Computer Science, Vol. 72)*, Edward W. Ng (Ed.). Springer, 216–226.

A Security Definitions

This section formalizes security definitions for the confidentiality and integrity properties described in Section 3.

Definition A.1 (Transaction Confidentiality)**.** The transaction confidentiality experiment $\mathsf{TCONF}[\mathcal{A}, \lambda, M, n, b]$ is conducted between an adversary $\mathcal{A}$ who controls one server and a number of malicious users, and a challenger C who controls the other server and any honest users and receives secret input bit b. The experiment begins with the adversary selecting a server S_a, $a \in \{1, 2\}$, to control during the experiment, and outputting n distinct user identifiers $\mathsf{uid}_1, ..., \mathsf{uid}_n$ to identify honest users. We will refer to the set of honest user ids as Hon. During the experiment, the group size is set to M, and the adversary is given access to a number of oracles, specified below. At the end of the experiment, the adversary outputs a distinguishing bit b', which is passed on as the output of the experiment.

During the course of the experiment, the challenger maintains a table U that maps user ids to a list of corresponding addresses.

- $\mathsf{HonSetup}(\mathsf{uid}_{i_1}, ..., \mathsf{uid}_{i_M})$: The challenger plays the role of honest users $\mathsf{uid}_{i_1}, ..., \mathsf{uid}_{i_M}$ interacting with the servers to set up a group. The challenger adds the new aid value to $U[\mathsf{uid}_i]$ for each user and does the necessary bookkeeping to record tokens held by each user throughout the experiment. We refer to groups set up via this oracle as *honest* groups.
- $\mathsf{MalSetup}(\mathsf{uid}_{i_1}, ..., \mathsf{uid}_{i_{M'}})$: The adversary plays the role of a malicious user interacting with the servers to set up a group, of which M' members are honest. The challenger plays the role of the honest users in the setup process, and the experiment aborts if the honest users fail to correctly set up their group credentials. The challenger adds the new aid value to $U[\mathsf{uid}_i]$ for each user and does the necessary bookkeeping to record tokens held by each honest user throughout the experiment.
- $\mathsf{MalTx}()$: The adversary plays the role of a malicious user interacting with the servers to make a transaction.

- HonTx(uid, aid_i, aid_t, x) : If any of the following conditions are violated, the oracle aborts and outputs $\perp$.
  (1) uid $\in$ Hon
  (2) $\mathsf{aid}_i \in U[\mathsf{uid}]$
  (3) $\lfloor \mathsf{aid}_i/M \rfloor = \lfloor \mathsf{aid}_t/M \rfloor$
  Otherwise, the challenger plays the role of honest user uid making a transaction using aid_i of value x with target address aid_t in the same group.
- HonChalTx(uid, $(\mathsf{aid}_{i0}, \mathsf{aid}_{t0}, x_0), (\mathsf{aid}_{i1}, \mathsf{aid}_{t1}, x_1)$) : If any of the following conditions are violated, the oracle aborts and returns $\perp$.
  (1) uid $\in$ Hon
  (2) aid_{i0} and aid_{i1} both belong to honest groups.
  (3) $\mathsf{aid}_{i0}, \mathsf{aid}_{i1} \in U[\mathsf{uid}]$
  (4) $\lfloor \mathsf{aid}_{i0}/M \rfloor = \lfloor \mathsf{aid}_{t0}/M \rfloor$
  (5) $\lfloor \mathsf{aid}_{i1}/M \rfloor = \lfloor \mathsf{aid}_{t1}/M \rfloor$
  Otherwise, the challenger plays the role of honest user uid making a transaction using aid_{ib} of value x_b with target address aid_{tb}.
- HonSettle(aid) : The challenger plays the role of the honest user who controls aid to perform the settling protocol with the servers. If aid is not controlled by an honest user, this oracle aborts and returns $\perp$.
- MalSettle() : The adversary plays the role of a malicious user interacting with the servers to perform the settling protocol.

We say that a payment splitting scheme satisfies transaction confidentiality if for all efficient adversaries $\mathcal{A}$ and any $\lambda, M, n \in \mathbb{N}$, it holds that

$$\Big|\Pr\big[\mathsf{TCONF}[\mathcal{A}, \lambda, M, n, 0] = 1\big] - \Pr\big[\mathsf{TCONF}[\mathcal{A}, \lambda, M, n, 1] = 1\big]\Big| \leq \mathsf{negl}(\lambda).$$

Definition A.2 (Integrity)**.** The integrity experiment $\mathsf{INT}[\mathcal{A}, \lambda, M, n]$ is conducted between an adversary $\mathcal{A}$ who controls a number $m < n$ of malicious users, denoted by the set Mal, and a challenger C who controls the servers and the remaining $n - m$ honest users, denoted by the set Hon. The group size for the system is set to M.

During the course of the experiment, the challenger maintains a table T that records the transactions the adversary requests honest users to make in groups of all honest users. For honest users in any group, the challenger does the necessary bookkeeping to record which users control which addresses, including mapping each uid to the set of associated addresses aid and the corresponding secrets.

The experiment provides the adversary access to the following oracles.

- AddGroup($\mathsf{uid}_1, ..., \mathsf{uid}_M$): This oracle allows the adversary to create new groups. If all of $\mathsf{uid}_1, ..., \mathsf{uid}_M \in$ Hon, we refer to the newly created group as an *honest* group, and the challenger sets $T[\mathsf{aid}] \leftarrow 0$ for each aid allocated to the newly formed group. Otherwise, the challenger plays the role of any $\mathsf{uid}_i \in$ Hon and the adversary plays the role of any user $\mathsf{uid}_i \in$ Mal. If a group contains users from both Hon and Mal, we refer to it as a *mixed* group.
- MalTx(uid): this oracle allows the adversary to make a transaction with the servers while playing the role of malicious user uid $\in$ Mal. If uid $\notin$ Mal, the call to the oracle aborts and outputs $\perp$.
- HonTX($\mathsf{aid}_i, \mathsf{aid}_j, x$) : this oracle allows the adversary to compel an honest user who controls aid_i – call this user uid_i – to make a transaction. If $\mathsf{uid}_i \notin$ Hon or if aid_i and aid_j are not in the same group (i.e., $\lfloor \mathsf{aid}_i/M \rfloor \neq \lfloor \mathsf{aid}_j/M \rfloor$), the oracle aborts and outputs $\perp$. Otherwise, the challenger plays the role of honest user uid_i interacting with the servers to make a transaction of value x with aid_j. The challenger sends the transcript of the interaction to the adversary and sets

  $$T[\mathsf{aid}_i] \leftarrow T[\mathsf{aid}_i] + x,\ T[\mathsf{aid}_j] \leftarrow T[\mathsf{aid}_j] - x.$$

- Settle(aid): This oracle allows the adversary to run the settling operation with the servers. Let uid be the user id associated with aid. There are three cases.

  uid $\in$ Mal. In this case, the adversary plays the role of the user uid and interacts with the challenger, playing the role of the servers, to run the settling protocol.

  uid $\in$ Hon and aid belongs to a *mixed* group. In this case, the the challenger plays the role of of user uid and the servers interacting to perform the settling protocol, and the adversary is sent the transcript of communications between the parties. As a result of settling, the user uid recovers a vector of group balances Bals, $|\mathsf{Bals}| = M$. The experiment aborts and outputs 1 if $\Sigma_{i=1}^{M} \mathsf{Bals}[i] \neq 0$.

  uid $\in$ Hon and aid belongs to an *honest* group. In this case, the the challenger plays the role of of user uid and the servers interacting to perform the settling protocol, and the adversary is sent the transcript of communications between the parties. As a result of settling, the user uid recovers a vector of group balances Bals, $|\mathsf{Bals}| = M$. The experiment aborts and outputs 1 if any balance does not match the corresponding balance recorded by the challenger in T. That is, the experiment aborts if there exists some $i \in \{1, ..., M\}$ where

  $$\mathsf{Bals}[i] \neq T[\lfloor \mathsf{aid}/M \rfloor \cdot M + i].$$

If the experiment ends without aborting and outputting 1 in one of the calls to Settle, the experiment's output will be 0. We say that a payment splitting scheme has integrity if for all efficient adversaries $\mathcal{A}$ and parameters $\lambda, M, n \in \mathbb{N}$, it holds that

$$\Pr[\mathsf{INT}[\mathcal{A}, \lambda, M, n] = 1] \leq \mathsf{negl}(\lambda).$$

B Cover Traffic

While Silent Splitter's security properties require that a given transaction reveal nothing about a user's groups, transaction partners, or transaction amounts, repeated transactions between a given user and the Silent Splitter system can be linked, both at the application and network layers. That is, the servers learn who the user initiating a given transaction is, even if they don't learn anything about what the transaction contains (dollar amount or recipient user/group). Additional effort to directly obfuscate this information is of little value, as any application that runs the Silent Splitter protocol will likely require some kind of user login and authentication functionality anyway.

Unfortunately, the Silent Splitter servers could use this user identification metadata to observe patterns in when users make transactions, join groups, or settle balances to infer relationships among users, even if the transactions themselves hide this information. Silent Splitter requires using cover traffic to eliminate this form of leakage.

Silent Splitter supports cover traffic messages at each stage of the protocol. In order to allow even users who are not yet members of any group to participate in generating cover traffic, the Silent Splitter servers create an additional "dummy group" when the system is first set up, running the initialization protocol for this group themselves and distributing a registration token for this group to all clients. This is not necessary to support cover traffic, but it allows clients to start generating cover traffic independently of the time that they start joining real groups.

We now briefly discuss how to support cover traffic at each stage of the Silent Splitter protocol.

(1) **Group registration.** The Silent Splitter protocol does leak the identity of users who initialize new groups, but cover traffic can be used to hide timing information that could otherwise link other users to the group creator. Since the group registration protocol run by each group member simply involves presenting an anonymous credential, this step can be done repeatedly for the same group. Thus users can "re-join" their existing groups or the dummy group as a way to create cover for other users really joining new groups.
Note that this step does not involve the servers allocating new storage for cover transactions, as only group initialization triggers allocation of storage for a new group. Thus the Silent Splitter design allows for cover traffic in group setup without bloating the server-maintained databases beyond the single dummy group created at system setup.
(2) **Transactions.** Cover traffic transactions can be made within a group by simply setting the value x of a transaction to be 0. For users who are not in any group, they can join and make transactions in the system-generated dummy group.
(3) **Settling.** There is no in-protocol requirement for servers to authenticate or verify client requests to settle because access control for settling is determined by who holds the group's keys ek_1 and ek_2. Thus any client can make settling requests at any time, regardless of whether or not the system provides a dummy group. Moreover, since the settling protocol setup step, where the servers merge their database shares, is request-independent, this part of the protocol cost can be amortized over a number of settle requests generated by cover traffic.

After determining how to generate cover traffic for each step of the protocol, we need to also decide how much cover traffic to generate. At one extreme, Silent Splitter clients can operate on a lock-step schedule, where time is split up into a series of fixed "rounds" and each client makes one operation, either real or dummy, for each round. At another extreme, there can be no automatic cover traffic generation, and the app can rely on users to generate traffic naturally or by manual effort. Differential privacy techniques [39, 40] or other statistical approaches can also be used to determine how much cover traffic each client should generate. The Silent Splitter design is compatible with any of these options. We do not discuss each of these approaches further, as the issue of how much cover traffic to generate for payment splitting is identical to cover traffic questions in other anonymous communication settings.

C Repaying Balances

Our settling protocol allows a user to see the list of balances for each user in their group. From this point, a number of different approaches can be taken to repay members in the real world.

(1) **No explicit repayment.** Perhaps the simplest way to handle settling balances is to not explicitly repay them at all. Users can occasionally check the balances in a group and adjust real-world behavior so that balances eventually even out. This approach, while slow, is easy to implement and requires no additional in-app functionality.
(2) **Out-of-band repayment.** Using the balance information available to each user in the group through the settling protocol, a client-side computation can compute a "repayment plan" that minimizes the number of real-world transactions needed in order bring balances for all group members (or some user-selected subset of the group) back to zero. When the plan is executed, these transactions can be entered into the system to record the updated state of user balances.
(3) **In-app repayment via third-party processor.** The most convenient option for users would be to integrate Silent Splitter with a third party payment processor who can settle transactions on users' behalf. Using in-app repayment necessarily comes with compromises compared to out-of-band repayment, e.g., repayment cannot happen in cash, but offers increased convenience. In this setting, computing a minimal repayment plan for the group before settling balances offers the additional benefit of hiding social graph information among members of the group in the aggregated transaction information of all its members. However, the application and the payment processor will necessarily be made aware of the final balances of users using this method of repayment.

D Deferred Proofs

Proof of Theorem 5.1 (Transaction Confidentiality).

Proof (sketch). The proof proceeds via a series of hybrids. We elide the details of reductions between hybrids and note places where a full proof would break a hybrid into a series of subhybrids. Without loss of generality, we present our proof as if the adversary chooses to corrupt server S_1. The case where the adversary corrupts S_2 is almost identical except that we no longer need to invoke the blind issuance property of the KVAC scheme because S_2 is not involved in that protocol step.

- Hyb_0: This is the transaction confidentiality game with $b = 0$, i.e., $\mathsf{TCONF}[\mathcal{A}, \lambda, M, n, 0]$.
- Hyb_1: This hybrid replaces all interactions with S_1 during group setup with simulated blind issuance and credential showing proofs. Formally, this step requires three subhybrids, where each is indistinguishable from the preceding one by the blind issuance, key-parameter consistency, and anonymity properties, respectively, of the KVAC scheme

used. Since this aspect of the protocol is not closely intertwined with the remaining parts, we skip the details of these hybrids.

- Hyb_2: This hybrid replaces the proof π sent by clients to S_1 during the transaction protocol with a simulated zero-knowledge proof. It is indistinguishable from the preceding hybrid by the zero-knowledge property of the proof system. Formalizing this hybrid requires one subhybrid to change the proof for each invocation of the HonChalTx oracle.
- Hyb_3: This hybrid replaces evaluations of the PRF $F(\mathsf{ek}_{2j}, \cdot)$ with evaluations of a random function $f(\cdot)$ for all honest groups $j \in \{1, ..., N\}$, rendering the contents of DB'_2 for those groups uniformly random. This hybrid is indistinguishable from the preceding one by the security of the PRF F and can be formalized by a series of subhybrids equal to the number of honest groups, where each subhybrid replaces invocations of F with one key ek_{2j} with a random function.
- Hyb_4: This hybrid replaces h^{r_γ} in the client's computation of c_γ during the transaction protocol with a uniformly random element of the group G. This results in c^γ being masked via multiplication by a random value. This hybrid is indistinguishable from the preceding one by the hardness of DDH in G because the values of h, $c_{r\gamma} = g^{r_\gamma}$, and h^{r_γ} form a DDH triple. A reduction would re-randomize the DDH challenge for each run of the protocol, resulting in independent-looking values for each call to the protocol.
- Hyb_5: This client replaces $h^{[r_\beta]_2}$ and $h^{[r_\gamma]_2}$ computed by S_2 during the transaction protocol with uniformly random elements of the group G. This results in $g^{[\beta]_2}$ and $g^{[\gamma]_2}$ being being masked via multiplication by a random value. This hybrid is indistinguishable from the preceding one by the hardness of DDH in G because the values $(h, g^{[r_\beta]_2}, h^{[r_\beta]_2})$ and $(h, g^{[r_\gamma]_2}, h^{[r_\gamma]_2})$ form DDH triples. A reduction would re-randomize the DDH challenge for each run of the protocol and for each of β and γ, resulting in independent-looking values for each call to the protocol.
- Hyb_6: This hybrid switches the experiment from $b = 0$ to $b = 1$. Since proofs are simulated, settling returns random values, and all the group elements sent between parties are independently random, the only change in the adversary's view is the change in the values of α, β used for the DPF. Thus this hybrid is indistinguishable from the preceding one by the confidentiality of the DPF.
- Hyb_7: This hybrid undoes the changes made in Hyb_5, replacing the random group elements from G with the values computed according to the protocol. It is indistinguishable from the preceding hybrid by the same argument relying on the hardness of DDH in G.
- Hyb_8: This hybrid undoes the changes made in Hyb_4, replacing the random group elements from G with the values computed according to the protocol. It is indistinguishable from the preceding hybrid by the same argument relying on the hardness of DDH in G.
- Hyb_9: This hybrid undoes the changes made in Hyb_3, replacing calls to the random function f with calls to the PRF $F(\mathsf{ek}_{2j}, \cdot)$. It is indistinguishable from the preceding hybrid by the security of the PRF F.
- Hyb_{10}: This hybrid undoes the changes made in Hyb_2, replacing the simulated proofs π sent to S_1 during the transaction protocol with real proofs. It is indistinguishable from the preceding hybrid by the zero-knowledge property of the proof system.
- Hyb_{11}: This hybrid undoes the changes made in Hyb_1, replacing the simulated credential issuance and presentation protocols with real ones. It is indistinguishable from the preceding hybrid by the security of the KVAC scheme.

Observe that hybrid Hyb_{11} is identical to the transaction confidentiality experiment with $b = 1$, i.e., $\mathsf{TCONF}[\mathcal{A}, \lambda, M, n, 1]$. Thus the theorem follows from the indistinguishability of the pairs of hybrids and the triangle inequality. □

Proof of Theorem 5.2 (Integrity).

Proof (sketch). The proof proceeds via a series of hybrids. We elide the details of reductions between hybrids.

- Hyb_0: this hybrid is the integrity experiment $\mathsf{INT}[\mathcal{A}, \lambda, M, n]$.
- Hyb_1: In this hybrid, the experiment runs the extractor for the proof system used in the KVAC scheme and aborts if the extractor fails. This event occurs with at most negligible probability, so the hybrid is indistinguishable from the preceding one. Formalizing this hybrid would require one subhybrid per interaction with a malicious user.
- Hyb_2: In this hybrid, the experiment aborts if the extracted aid from any user $\mathsf{uid} \in \mathsf{Mal}$ belongs to an honest group. This event occurs with negligible probability by the unforgeability of the KVAC scheme, so this hybrid is indistinguishable from the preceding one.
- Hyb_3: In this hybrid, the experiment runs the extractor for the zero knowledge proof system used to generate the proofs π during the transaction protocol and aborts if the extractor fails. The abort condition occurs with negligible probability, so this hybrid is indistinguishable from the preceding one. Formalizing this hybrid would require one subhybrid per transaction from a malicious user.
- Hyb_4: In this hybrid, the experiment keeps a table T_{MAC} of messages it has MACed. If ever a client sends a message, MAC pair $(\mathsf{uid}_\ell, \mathsf{com}_{\mathsf{aid}\ell})\sigma_\ell$ which verifies successfully but does not appear in T_{MAC}, the experiment aborts. This event occurs with negligible probability because of the existential unforgeability of the MAC scheme, so this hybrid is indistinguishable from the preceding one.
- Hyb_5: In this hybrid, the two servers merge their shares of the evaluations of each DPF after completing all their evaluations. If a DPF passed verification but has more than a single non-zero entry, the experiment aborts. This event occurs with negligible probability by the soundness of the DPF verification protocol, so this hybrid is indistinguishable from the preceding one.
- Hyb_6 : In this hybrid, the servers reconstruct the values of α, β, γ from their secret shares, and the experiment aborts if these values do not match the corresponding values (i, x, ix) extracted from the proof π. Observe that if these values

disagree, then they offer two different openings of the commitments $c_\alpha, c_\beta, c_\gamma$. The reduction from different openings of the commitments to discrete log is identical to that of the binding property of the Pedersen commitment scheme [55]. Thus this hybrid is indistinguishable from the preceding one by the hardness of discrete log in G because the abort criterion is triggered with negligible probability.

- Hyb_7 : In this hybrid, the experiment augments the table T_{MAC} to record the values $\mathsf{aid}_i, r_{\mathsf{aid}}$ extracted from the KVAC presentation protocol when issuing group tokens. Then, the experiment aborts if it ever extracts values of i, r_{aid_i} from the transaction proof π that do not match any of the values of $(\mathsf{aid}_i, r_{\mathsf{aid}})$ in T_{MAC} for the provided tags $\sigma_1, ...\sigma_\ell$. This hybrid is indistinguishable from the preceding one by the hardness of discrete log in G because this event means that i, r_{aid_i} is a valid second opening for one of the commitments $\mathsf{com}_{\mathsf{aid}_1}, ..., \mathsf{com}_{\mathsf{aid}_\ell}$.
- Hyb_8: In this hybrid, the two servers directly compare their shares of $\mathbf{w}$ after completing each successful transaction. If the transaction modified the database, but $\mathbf{w} \neq 0^N$, the experiment aborts. Since the elements of $\mathbf{w}$ have been multiplied by a uniformly random vector $\mathbf{r}$, this can be thought of as evaluating a multilinear polynomial with coefficients determined by $\mathbf{w}$ at a random point $\mathbf{r}$. By the Schwartz-Zippel-DeMillo-Lipton Lemma [36, 57, 63], if $\mathbf{w} \neq 0^N$, $\Sigma_{i=1}^N r_i[w_i]_b = 0$ with at most negligible probability (unconditionally). Thus this hybrid is statistically indistinguishable from the preceding one.

We now show that no adversary can win the integrity game in Hyb_8. To do this we will prove two claims:

(1) Each transaction affects exactly 1 group and does not change the sum of balances in that group.
(2) No malicious user can make a transaction that affects balances in an honest group.

Observe that since each group begins with balances that sum to zero (because all balances start at zero), claim 1 shows that this invariant is maintained after each transaction. This rules out the possibility of the adversary winning the experiment because settling some mixed or honest group results in $\Sigma_{i=1}^M \mathsf{Bals}[i] \neq 0$. Moreover, if no malicious user can make a transaction that affects balances in an honest group, then the balances recorded in the table T will exactly match the balances returned by settling, because T records the balances resulting from transactions made by honest users in an honest group. But this means that both cases where the experiment can output 1 have become unreachable, so the adversary can never win. We conclude the proof by proving the two claims.

Claim D.1. *Each transaction affects exactly 1 group and does not change the sum of balances in that group.*

Proof sketch. We know from DPF verification that the vectors $\mathbf{f_i}$ and $\mathbf{f_j}$ each have a single non-zero entry. Since $\mathbf{u}$ and $\mathbf{v}$ are formed by taking sums of consecutive blocks of M entries in these vectors (corresponding to the addresses in the same group), it must also be the case that $\mathbf{u}$ and $\mathbf{v}$ each have a single non-zero entry. We also know that $\mathbf{u} - \mathbf{v} = \mathbf{w} = 0^N$. Then it follows that $\mathbf{u}$ and $\mathbf{v}$ must have the same non-zero entry and the same value in that entry, otherwise we would have that $\mathbf{u} - \mathbf{v} \neq 0^N$, a contradiction. The claim follows because each entry in $\mathbf{u}, \mathbf{v}, \mathbf{w}$ corresponds to the values for addresses in one group. □

Claim D.2. *No malicious user can make a transaction that affects balances in an honest group.*

Proof sketch. Consider a transaction made by the adversary via the MalTx oracle. Since the value of β computed by the servers is a sum of all the entries in $\mathbf{f}_i$, it must be that β is the value of the one non-zero entry in $\mathbf{f}_i$. But by construction, then, γ is the product of β and the non-zero index α at which that value occurs. Also, we know that these values of α and β match the values of i and x extracted from the proof π, and that the value of i corresponds to the opening of one of the commitments $\mathsf{com}_{\mathsf{aid}_1}, ..., \mathsf{com}_{\mathsf{aid}_\ell}$ presented by the adversary when making the transaction. But since the experiment would have aborted if any honest group index were extracted from a malicious user's registration, it must be the case that i is not an address located in an honest group. But since, by Claim 1, a transaction only affects one group, and the DPF $\mathbf{f}_i$ does not affect the balance of an honest group, it must be that the transaction does not affect the balance of an honest group. □

□