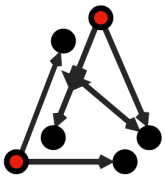

Least Authority
Freedom Matters

S4 Subscription Payments via Zcash Shielded Transactions and Tor

P4: Private Periodic Payment Protocol

Published Specification Version: 23 January 2019

Authors & Reviewers:

Jean-Paul Calderone, Least Authority
Ramakrishnan Muthukrishnan, Least Authority
Paige Peterson, Zcash
Liz Steininger, Least Authority
Chris Wood, Least Authority

Table of Contents

1 Executive Summary

The Private Periodic Payment Protocol (P4) aims to define the way in which subscription services can be funded using end-to-end private cryptocurrency payments. This protocol introduces periodicity to cryptocurrency payments through an ongoing relationship between the merchant and the customer without unintentionally disclosing personally identifiable information (PII). Least Authority is creating this protocol to integrate a truly end-to-end private subscription data storage solution, built using Tahoe-LAFS (for data storage) and Zcash (for payments).

At Least Authority, we believe people should be able to use Internet services while retaining control over their own data. We are building an ethical, usable, and lasting data storage solution with end-to-end cryptography and user-friendly interfaces. We hope to give more humans a real alternative for control of their own data. By allowing users a more private option for payments, Least Authority can further its mission of giving people the freedom to control their own data. P4 is a technical specification describing how users will be enabled to privately pay for S4, Least Authority's hosted Tahoe-LAFS service, or the "Simple Secure Storage Service."

By utilizing a cryptocurrency like Zcash with privacy features, Least Authority can facilitate an online exchange of value while gathering minimal personal data. The novel characteristic of the described system is the ability for users to supply periodic payments (i.e., on a "subscription" basis) while revealing no further identifying information about themselves than they would do in a one-time payment method.

The initial version of P4 is considered a "MVP" (Minimum Viable Product) and is intended to be iteratively improved in the future. So although this specification is for a fully functioning implementation, there are also details of future areas for improvement that should be a part of the consideration.

Since online subscription services are an increasingly common approach to business, Least Authority recognizes that the work done with cryptocurrency periodicity in P4 will be useful to the greater community. Additionally, by incorporating privacy-by-design in creating the protocol, Least Authority hopes to help cryptocurrency usage better adhere to the principle of privacy as a human right was well as to regulations such as the EU's General Data Protection Regulation (GDPR). Explanations of privacy-focused considerations are incorporated throughout the document to provide insight about these design decisions.

By sharing P4 openly, Least Authority hopes that other subscription services will implement a version of the protocol to advance the adoption of cryptocurrency payments in real world retail use cases that also incorporate privacy by design.

2 Project Details

2.1 Background

Currently, Least Authority offers S4: Simple Secure Storage Service as a hosted and managed version of Tahoe-LAFS. One of the primary features of Tahoe-LAFS is that it greatly limits what a service provider (such as Least Authority), knows about the data being stored via the service. The data is client-side encrypted and accessed by capabilities, not personal accounts. This means no personal information is stored in Tahoe-LAFS itself.

However, in offering a service such as S4, Least Authority needs to accept payments. This requires using existing online payment-processing tools to manage the subscriptions to the S4 service. Currently, Least Authority uses Stripe for payment processing and Chargebee to allow customers to manage their subscription payments.

In addition to the above payment method, Least Authority would like to offer a method that processes payments similar to how data is stored—the keys to the transactional data are stored on the user's device and not shared with third-party services. This also helps to minimize the collection of Personally Identifiable Information (PII)—information which can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context.

2.2 Specification Goals

Although Least Authority utilizes industry standard products for online service subscription payments, these products require Least Authority to collect personal information only for the payment processing activities and share this personal information with third-party companies for the convenience of online payment processing. By removing customers' personal information from the payment process, Least Authority can reduce the liability of keeping those data sufficiently secured. Least Authority strives to both respect individuals' right to privacy while seeking to limit the liabilities of obtaining customer data.

2.3 Requirements

The transfer of a value-carrying instrument is one obvious requirement for a payment system. An equally important requirement is for a particular instrument to unambiguously correlate with a particular subscription. This correlation must not extend beyond the subscription itself. For example, it must not be tied to financial information which is itself tied to an identity.

3 Scope of Effort

3.1 Payments: Zcash Shielded Transactions

Zcash is a privacy-protecting digital currency that can shield transaction data from the public while allowing users to selectively disclose that data with third-parties. Zcash supports two kinds of addresses: shielded (also referred to as z-addresses and start with a “z”) and transparent (also referred to as t-addresses and start with a “t”).

A Z-to-Z transaction (or shielded transaction) is recorded on the public blockchain such that it is known to have occured, but does not reveal specific transaction details like addresses, value transferred, and associated memo. The owner of an address may choose to disclose the z-address and transaction details to trusted third parties for certain purposes—such as audits and compliance requirements—through the use of view keys and payment disclosure.[1]

3.1.1 Zero Knowledge Proofs

Zero knowledge proofs are a scientific breakthrough in the field of cryptography. They allow you to prove knowledge about hidden information without revealing that information. The property of allowing both verifiability and privacy of data makes for a strong use case in all kinds of transactions, and Zcash integrates this concept into a blockchain for protecting transaction data.

3.1.2 zk-SNARKs in Zcash

Zcash shielded transactions use a particular type of zero-knowledge proof called zk-SNARKs (or “zero-knowledge Succinct Non-interactive ARguments of Knowledge”). Within a shielded transaction, there exists a string of data that the sender of a transaction provides–the “zero-knowledge proof”–which cryptographically proves the properties of said hidden transaction data to transaction verifying network nodes. This process includes proving that the sender couldn’t have generated that particular string unless they had ownership over the spending key and unless the input and output values are equal. The proof also guarantees creation of a unique nullifier which is used to mark tokens as spent when they are, in fact, spent. These assurances allow for verification that the transaction is valid, while preserving privacy of the transaction details.

3.1.3 Payments with Shielded Addresses

The transfer of funds from the user to another user or merchant using Zcash shielded addresses is not believed to reveal any PII directly, unless the user purposefully includes PII in the encrypted memo field. However, network analysis may reveal indirect information about the customer’s IP address (see "Zcash Over Tor" § 3.1.4). The use of shielded addresses is of particular importance for this design because it is known that when even one address in a transaction is transparent, transaction metadata may reveal patterns leading to eventual PII exposure.

Wallet support for shielded addresses is currently limited. See this table on the official Zcash website for some available options.

[1] https://z.cash/technology/

3.1.4 Zcash Over Tor

Zcash is a global network using IP addresses over Transmission Control Protocol (TCP) for maintaining connections between nodes, and does not obfuscate users' IP addresses. As a result, it is possible for correlations to be made using IP addresses between transactions sent *from* shielded addresses and other network traffic including transactions sent from transparent addresses.

Advanced users may opt to connect through Tor to obfuscate their node's IP address, however, further exploration is needed on a vulnerability combining Bitcoin's Denial of Service mitigations (inherited into Zcash) and anonymous communication networks like Tor.[2]

3.2 Data Storage Service: Simple Secure Storage Service (S4)

Simple Secure Storage Service (S4) is Least Authority's hosted and managed data storage service offered for individual use and built with Tahoe-LAFS. Data can be stored with S4 utilizing the command line interface (CLI) or a desktop client, Gridsync. Both utilize Tahoe-LAFS client-side encryption before storing the data online on designated servers.

3.2.1 Tahoe-LAFS

Tahoe-LAFS (Least Authority File Store) is a free and open, secure, decentralized, fault-tolerant, distributed data store and distributed file system. It can be used as an online backup system, or serve as a file or web host. Tahoe-LAFS encrypts data on the client side. Tahoe-LAFS doesn't operate on Access Control Lists (ACL, or user accounts) but instead works using the object-capability model (OCAP, or key-based access). The key is stored on the device and access to the data is given by sharing the capability, which allows the user complete control over who can see the content.

Tahoe-LAFS itself is highly oriented towards privacy preservation. Administrators of the system, like Least Authority with S4, can not see the contents of the encrypted data because the keys are not stored with the company, but with the user. However, Least Authority, as the administrator of the system, can see uploads and downloads of ciphertext to the grid, which is a collection of storage nodes.[3]

Further investigation of Tahoe-LAFS for other possible PII leakage should be undertaken, as it has not undergone audits focusing on these privacy-preserving behaviors. However, Tahoe-LAFS is the service being offered and distinct from the payment protocol being proposed. If Tahoe-LAFS has privacy-compromising issues, they will not void the privacy-preserving properties of the rest of the system.

3.2.2 Gridsync

Gridsync is a cross-platform desktop application that provides a graphical user interface (GUI) for end users of Tahoe-LAFS, making it easier to understand and use. Through several iterations of user testing in partnership with Simply Secure and Internews, Gridsync has been greatly improved on the client side for

[2] https://z.cash/support/security/privacy-security-recommendations#network
[3] https://tahoe-lafs.readthedocs.io/en/tahoe-lafs-1.12.1/running.html?highlight=grid#introduction

non-technical users and across various operating systems. For P4, Gridsync will be extended to offer invoice management features.

3.2.3 Magic Wormhole

Magic Wormhole allows Least Authority to send configuration information from our Simple Secure Storage Service (S4) to the Gridsync application with the invite code and supports sharing files and folders in Gridsync. Gridsync uses the magic-wormhole library to provide human-pronounceable "invite codes" for joining storage grids and sharing folders with other users.

3.2.4 Grid Configuration

Tahoe-LAFS offers flexible cloud architecture that can provide for the multiple needs of diverse organizations. For instance, it can be used in a RAID-like fashion using multiple disks to make a single large RAIN pool of reliable data storage. This makes it possible to distribute the risk to data between particular groups of people working on them, instead of keeping all information in a single, centralized place (which can have catastrophic consequences in case of a data compromise).

S4 is the Least Authority implementation of Tahoe-LAFS offered as a hosted and managed service for data storage. The current S4 service has a single grid per user. For P4, Tahoe-LAFS will be configured as one grid for all users who pay subscriptions with Zcash shielded transactions. So, having all users share a single deployment of storage nodes, this grid configuration makes it harder to identify whose encrypted shares belong to a particular payment account. As a result, all Tahoe-LAFS interactions with a particular subscription will not be easily identified.

3.2.5 Tahoe-LAFS Over Tor

Tahoe-LAFS has native Tor and I2P support and Gridsync 0.4 introduces support for Tor. By requiring users to make use of these features, the most obvious PII leak (a user's IP address) is avoided and thus provides network location privacy. Additionally, the network addresses supplied in the Tahoe-LAFS configuration will be Tor ".onion" addresses which can ***only*** be used over Tor, further limiting the chances of an accidental privacy violation.

Because S4 customers who choose to pay with Zcash shielded transactions will be required to use Tor, Least Authority will be able to tell which S4 instance they are utilizing. Additionally, Least Authority could correlate interactions within the S4 instance to be one user. However, Least Authority will not know who the user is because of the Zcash shielded transactions and will not be able to link the encrypted shares themselves to any one user of Tahoe-LAFS.

4 Functional Design

4.1 Subscription Workflow

Tor Network	Customer Client-side Applications			Merchant Server	Payment Network
	Tor Browser	**Gridsync**	**Zcash Wallet**	**Least Authority's S4**	**Zcash**
Step 1: Subscription Initiation	Visit webpage and complete subscription request				
Step 2: Subscription Creation and Invite Code				Create Zcash address and initial database entry	
				Generate a magic-wormhole invite code	
Step 3: Client-Side Subscription Configuration		Connection via magic-wormhole to receive initial S4 configuration and invoice			
		Update configurations and interpret invoice structure and check signature			
Step 4: Payment and Server Update			Make payment to designated receiving address		
					Process payment and write to the blockchain
				Confirm payment has been received and publish next invoice	
Step 5: Recurring Invoice and Payment Process		Check for the next invoice, interpret invoice structure and check signature			
		Display updated invoice information			

4.2 Subscription Initiation (Step 1)

The Least Authority website and links to web assets are accessible as either Tor addresses (`something.onion`) or on the clear Internet (via https).

4.2.1 Tor-based Signup Webpage

When a user decides to use Zcash shielded transactions as a payment mechanism, they will be required to visit a Tor network address to complete the process. The signup server will operate on a Tor network address, avoiding the creation of a metadata link from the subscription to an actual customer-associated IP address. Other incidental information that could be collected is the browser's window size and screen resolution. Such information may not immediately appear to be PII, but it is metadata which can be used to link different activities to an individual together and therefore may potentially de-anonymize certain activities by linking them to other non-anonymous activities.[4] However, Least Authority will not collect this information.

In the case of the Tor hosted website including one or more https links, the use of Tor would provide basic client IP obfuscation from traffic observation leaving the Tor network. However, it may be possible to perform time-based correlation of information received from a user while they are browsing the Least Authority website and correlated links via their IPv4/IPv6 addresses with requests received at the Tor address. The risks of potential correlations could be mitigated by having more users on the site and more users creating subscriptions. Such risks could also be mitigated by switching users to the Tor address as early as possible in their visit to the site or making sure there are no https links on the Least Authority Tor instance.

4.3 Subscription Creation and Invite Code (Step 2)

Selection of an actionable button, such as “Create a Subscription”, will return a temporary, one-time-use invite code (generated using the magic-wormhole library) to the user for redemption.

4.3.1 Server Setup

A subscription is created when a user interacts with the Least Authority S4 signup server by completing the Tor-based webform as described above.

4.3.1.1 Zcash Address Creation

A Zcash shielded address unique to each subscription will be created.

4.3.1.2 Initial Database Entry

The Zcash address and subscription identifier will be inserted into a database that is private to Least Authority. The new subscription will be immediately usable and marked as active, although only for a few hours if it remains unpaid.

---

[4] See: https://www.researchgate.net/publication/260115377_User_Tracking_on_the_Web_via_Cross-Browser_Fingerprinting

4.3.2 Invite Code Generation

Using Magic Wormhole, the Invite code will be generated and displayed for the user after signing up for the S4 service.

4.4 Client-Side Subscription Configuration (Step 3)

4.4.1 Gridsync Connection

Entering the invite code into the Gridsync client will establish a secure connection to Least Authority's servers through which the credentials necessary to connect to and use the S4 service will be delivered. Users will be required to complete the magic-wormhole invitation redemption using Tor. The magic-wormhole interaction intentionally collects no additional information from the user. The user is directed to use Tor so that the incidental information that could be collected does not include the user's real IP address. It is not believed that magic-wormhole will incidentally reveal any further information.

4.4.2 Receiving Initial Configuration and Invoice

The user will then receive the first invoice (see "Invoice Structure" § 4.3.3) through magic-wormhole that instructs how to make the initial subscription payment and includes a link to the location of the next invoice. Within a few hours after creation, the initial payment must be received for the subscription to remain active (see "Checking Payment Status" § 4.3.4).

Least Authority will also sign the initial configuration and invoice message with its public key in the magic-wormhole to be stored in Gridsync for checking that future invoices are authentic (see "Signature Verification" § 4.3.4.2).

4.4.3 Invoice Structure

The invoice is made up of a Zcash receiving address (z-addr) suffixed with a query string[5] which accepts the following keys:

`c`	(required)	The **c**urrency/cybercoin name (e.g., "ZEC").
`a`	(required)	The **a**mount due (e.g., "0.1").
`d`	(required)	The **d**ate due of the subscription period, as ISO 8601 (e.g., "2018-09-04T12:32:42").
`e`	(required)	The **e**xtension date of the subscription, as ISO 8601 (e.g., "2018-10-04T12:32:42").
`l`	(required)	The **l**abel for the service (e.g., "Least Authority S4"). (urlsafe-base64)
`u`	(required)	The **U**RL to for the next invoice (e.g., "http://example.onion/86u4Cx1a")[6]. (urlsafe-base64)

[5] Inspired by BIP21

[6] Onion addresses are self-authenticating so HTTP URIs are even safer than non-onion HTTPS URIs.

`m`	(optional)	A **m**essage to the user. Application-specific. Used, e.g., to deliver storage fURLs (urlsafe-base64)[7]
`r`	(optional)	C**r**edit currently applied to the account as a result of a partial payment or an overpayment of a previous invoice. A payment of **a** - **r** is required for this invoice to be paid in full.
`v`	(optional)	A **v**ersion identifier. Assumed to be "1" if omitted.
`p`	(required)	The Ed25519 **p**ubkey with which the next invoice will be signed. (urlsafe-base64)
`s`	(required, last)	The Ed25519 **s**ignature of the current invoice string. (urlsafe-base64)

4.4.3.1 Example Invoice String

The following string is an example of what the invoice would look like:

```
zs1z7rejlpsa98s2rrrfkwmaxu53e4ue0ulcrw0h4x5g8jl04tak0d3mm47vdtahatqrlkngh9sly?c
=ZEC&a=0.1&d=2018-09-04T12:32:42&e=2018-10-04T12:32:42&l=TGVhc3QgQXV0aG9yaXR5IF
M0&u=aHR0cDovL2V4YW1wbGUub25pb24vODZ1NEN4MWE=&p=9eS4JZwugQmTcAFXDTQ5VKUzkKP01rK
zfB_Soxi4dhU=&v=1&s=GZWwXcxlh6kVUBoT67fudhpnJi1JJT7rZvQ0RCZXMA5LePUdXcq3lmeq_xz
OPdM2nua3kuPT9xPifiSKoc0=
```

In this example, a user/customer is being asked to pay 0.1 ZEC by September 4, 2018 for the service "Least Authority S4". If the payment is made by the given deadline, the subscription will be extended until October 4, 2018.

4.4.3.2 Client-Side Invoice Validation

Upon the receipt of a new invoice, a client-side "P4"-compatible application determines the validity of the invoice. In the specific case of Gridsync, the application will verify that the invoice contains the requisite payment-related information (i.e., any and all of the required key/value pairs specified above) in the expected encodings/format and that its corresponding signature is valid (i.e., the message was cryptographically signed by the expected key as specified by the previous invoice). Should any of these checks fail, the user will be notified immediately and, depending on the nature of the failure and where possible, an appropriate course of action will be suggested.

4.5 Initial Payment and Server Update (Step 4)

4.5.2.1 Sending Payment

The customer will make their initial payment according to the invoice details using their Zcash wallet. While it is possible to pay from a transparent address, it is highly recommended to use a shielded address for all subscription payments (see "Payments with Shielded Addresses" § 3.1.3 regarding transaction linkability considerations for transparent addresses).

[7] This is not to be confused with the embedded Zcash encrypted memo field which is not used in this system.

The subscription cost will be denominated in Zcash while maintaining parity to the USD/EUR at the time of invoice creation. Therefore, the cost in Zcash is likely to change with each invoice due to ongoing price volatility.

4.5.2.2 Receiving Payment

As part of the S4 infrastructure, Least Authority will operate a Zcash full node using the official zcashd reference client. This node will be used to scan the Zcash blockchain for payments to the addresses associated with invoices. When such a payment is found, the corresponding subscription is looked up in the address→subscription database. Since the z-addr is an identifier for the subscription, the z-addr can change from invoice to invoice to facilitate provider-side key rotation. The amount of the transaction is then credited to that subscription. If the amount equals or exceeds the balance due, then the subscription is transitioned to the *good standing* state. It will be the responsibility of users to make these payments on a timely basis.

4.6 Recurring Invoice and Payment Process (Step 5)

4.5.1 Checking Payment Status

Upon receiving a message containing an invoice string ($Invoice_1$), the customer's client is expected to automatically store both the next-invoice URL (`u`) and the next-invoice pubkey (`p`) contained therein. A payment is considered received when it appears on the Zcash blockchain. And this can be considered valid after 24 blocks have been written on top of it (which takes approximately one hour).

It is expected that the client will display the invoice details and overall subscription status to the user.

4.5.1.1 Payment Status Codes

Querying the next-invoice URL will be sufficient to inform the customer of their subscription "standing": if the next-invoice URL returns an HTTP error code of **402** ("payment required") and an invoice in the body, this signals that the customer's current invoice has not yet been paid (if an incomplete payment has been made the included invoice indicates this with an updated C**r**edit field); once full payment has been received, the next-invoice URL shall return an HTTP status code of **200** ("OK") with the next invoice string ($Invoice_2$) in the body of the response. When the transaction is seen by Least Authority in the mempool but before the transaction has been confirmed by the 24 blocks, the HTTP status code will be set to **202** ("Accepted") with the number of current and required block-confirmations in the body (e.g., "17/24"). This would allow the customer's client/wallet to reflect the state that the transaction has been "seen" by the merchant but is not yet considered fully valid/settled (i.e., a "pending" state).

4.5.1.2 Signature Verification

Invoices will be digitally signed using the same key provided with the initial configuration and invoice message. Signatures will be created by the merchant server using a public-key signature system to provide the same properties as a MAC: message integrity and authentication. It proves the message content hasn't changed because a change would result in the signature becoming incorrect. In addition, it provides authentication because only the holder of the private key can construct the correct signature.

Before paying $Invoice_2$, customers' clients can—and should be expected to—always verify that the signature in $Invoice_2$ was created using the public key included in $Invoice_1$ (and, likewise, that $Invoice_3$ was

signed by the public key included in $Invoice_2$, and so on), thereby preserving a “chain of trust” between the customer and merchant throughout the duration of the subscription. The only exception to this should be in the first, original invoice which, in the case of S4, is delivered to the user’s client over a magic-wormhole message (which is presumed that the user trusts that the same message also contains the credentials necessary to use the service).

4.5.2 Invoice Tracking

Like the previous invoice, the current invoice will contain a link to the next invoice that will not be published until payment for the current invoice is received. This loop will continue for subsequent invoices, until an invoice is not paid on time.

Gridsync—or other future “P4”-compatible agents—shall keep track of invoices on the client side. In the specific case of Gridsync, the application will periodically poll the current next-invoice URL (`u`) as described above and maintain a database of past and current invoices both a) locally on the user’s device and b) remotely in the grid using Gridsync’s built-in recovery system[8]). This database will naturally be used to notify users of upcoming payment due dates and to provide access to other information pertaining to the current state of the user’s subscription (such as whether and when a previous payment was received, the effective term-length of the current subscription period, and so on).

4.6 Subscription End

4.6.1 Revocation with Distinct Onion Addresses

This is a scheme for revoking access from users who have allowed their subscriptions to lapse (i.e., who are no longer paying for service). In this scheme, when a customer creates a new subscription they receive a response containing one or more storage server fURLs which are not shared by other subscriptions. The scheme provides the ability to immediately cancel a subscription. It preserves the subscriber’s privacy but correlates all of the subscriber’s actions. This scheme is *incompatible* with free trials because a trial subscription may access data stored by a previous, now cancelled subscription (or even a previous trial subscription) so long as the client retains the file capabilities, and therefore no payment is *ever* required as a condition of service.

In this scheme, a commitment is made to enforcing ".onion"-only addresses in storage fURLs. Least Authority would need to maintain a database mapping z-addrs to subscription identifiers in order to to process payments. This database is used to store a signup-time Tor-generated ".onion" address[9] (unique to each subscription). That subscription’s ".onion" address is given as the only connection hint of the fURLs given to the subscriber. A single collection of storage servers are maintained for all subscribers, and all “.onion” addresses are distributed and routed to these servers. So long as that customer remains in "good standing", their onion service is kept up and running. If they stop paying, only their onion service is closed, denying them access to the grid (until payment is restored) without affecting the connections of other customers. This process avoids the need to significantly reconfigure parts of the Tahoe-LAFS grid *or* the Tahoe-LAFS client.

[8] https://github.com/gridsync/gridsync/blob/master/docs/recovery-keys.md
[9] https://txtorcon.readthedocs.io/en/latest/txtorcon-onion.html#ephemeralonionservice

5 S4 Implementation

5.1 Development Plan

All development of the implementation is captured in a Least Authority public repository on GitHub: https://github.com/LeastAuthority/S4-2.0. Check the repository online for the latest status of the implementation work for Least Authority's S4 utilizing P4.

Appendix: Potential Future Work

The following is some potential future work that could be implemented in addition to or in replacement of certain parts of the current specification. Nonetheless, this includes the current assessment of this potential work, at this point. This will be the foundation for future analysis and implementation work. However, these suggestions need further assessment for both technical value and resulting privacy implications.

It is hoped that by sharing these details, useful insight for alternative implementations is provided. This is not a comprehensive list, and it is likely that new ideas for future work will result from community feedback.

A.1 Improvements to the Current Implementation

A.1.1 Single Tor Onion Address

S4 customers would have considerably stronger privacy guarantees if they all used the same onion address. The distinct onion addresses could be used to correlate user-activities; since service providers create/maintain those onion addresses, they ultimately decide what internal IP/port they point to ,and thus could, for example, route any or each onion address through a unique forwarding proxy which logs requests between the Tor network and our storage nodes.

Privacy could, however, be improved by using one Tor onion address for all users. For example with S4, Least Authority could make all subscriptions due on the same day of the month and then send out the same onion address to all paying customers. Although this removes distinguishing features of customer accounts, it also removes the flexibility for customers to select subscription activation- and due-dates.

A.1.2 Display Invoice Information on the Website

However, the invoice/payment-request information could be shown to the user on the website. Doing so would likely help to instill user confidence (since the user could, for example, confirm that the zaddr shown in their Gridsync window matches the zaddr shown on the website) but could lead to other issues (since the user could, for example, pay from the website while an attacker trashes the wormhole mailbox, thereby preventing them from using the service even after they pay).

A.2 Zcash Transactions/EMF-based Subscription Workflow

In the future, Zcash's encrypted memo field (EMF) and transactions could be utilized for the subscription workflow as an alternative to using the Tor Onion addresses.

A.2.1 Signup via Zcash Transactions

In this scheme, Least Authority publishes a Zcash shielded address and a list of services and prices. When a user wants to sign up, they send an amount of ZEC appropriate for the selected service to the published shielded address and includes a return shielded address in the encrypted message field of the transaction. Least Authority then sends a (low-value) transaction to the return shielded address with an introducer fURL (or other necessary Tahoe-LAFS configuration information) in the encrypted message

field. This information is then used by the user to set up a Tahoe-LAFS client. Then, as long as sufficient ZEC is received with a matching return shielded address, the service remains accessible.

A.2.2 Subscription Redemption via Zcash Transactions

As an alternative to the redemption via invite code described above, it is possible to deliver the same credentials by using Zcash's "encrypted memo field" (EMF).he user may, for example, disclose their own z-addr inside the encrypted memo field of their initial transaction to which Least Authority could send an additional, small transaction containing the required credentials inside of the EMF. While this alternative is advantageous, insofar as it allows the possessor of the z-addr viewing key to retrieve the credentials in the future (e.g., in the event of connectivity issues when redeeming the invite code) it may be undesirable for other reasons (including leaving a "permanent" record of said credentials on the blockchain, the requirement of *two* transactions to complete the exchange, and so on). In addition, no tools or applications currently exist to watch for the second transaction and automatically load the credentials from the EMF into the user's client.

A.2.3 Checking Payment Status

One way to notify payment status would be to provide an option to deliver an additional copy of $Invoice_2$ via the EMF if the customer includes a "return" z-addr in the EMF of their transaction for $Invoice_1$. Use of the Zcash EMF requires the invoice string to be less than 512 bytes.

A concern would be revocation/forward-secrecy; with the current planned design of using rolling onion services, Least Authority can easily remove old invoices once the new ones have been paid. If, for example, a customer is on $Invoice_4$ and the URL for $Invoice_3$ is removed, an attacker that learns the URL for $Invoice_2$ they can't follow the chain of payments forward. Whereas with the EMF, the holder of the viewing key(s) can always view the contents of a corresponding invoice. This choice depends on what sort of durability vs. confidentiality trade-offs people are willing to make. From the perspective of confidentiality, a month is a long enough term to allow for time to get service restored in the event of downtime. In other words, an unreachable invoice URL probably won't be unreachable for very long from a monthly perspective.

This approach avoids requiring the user to send any information to Least Authority using a browser and therefore side-steps any privacy-violating actions that might result from use of a browser. The entire signup occurs in shielded Zcash transactions which should maximize privacy compared to other schemes. It does require the user to discover the shielded address used by Least Authority for signup but this can be done using read-only browser interactions, and because the address is fixed, it may be feasible to securely publish it in other ways that do not involve an interaction with Least Authority's web servers.

This approach requires us to know their z-addr (or t-addr) and is a "push" by us to remind them. The storage-provider cannot tell *when* the billing information has been "received"/read by the customer. This pattern can be "standardized" and generalized to work with and/or be adopted by other "subscription" services, benefiting the Zcash ecosystem more broadly.

A.3 Tahoe-LAFS-based Improvements

In the future, more features of Tahoe-LAFS could be utilized for the subscription workflow as an alternative to using the Tor Onion addresses.

A.3.1 Communicating Subscription Status

A mutable read cap could be sent to the Tahoe-LAFS client and it could periodically read this and find out subscription information. One downside is that this communication approach is specific to Tahoe-LAFS. As such, there could be issues in communicating if a subscription lapses and the customer could no longer access their capability string, but they need to access it to find out what they owe. However, an advantage is that this approach could also function as a more general communication channel that could be used for lots of other interesting user communications.

This communication could be done "In-band" via Tahoe-LAFS, whereby the storage-provider periodically writes/updates billing information to a machine-readable file—a "billing cap"—on the customer's grid that the customer's client regularly checks/parses. This is probably not too difficult to implement and allows some existent privacy/security properties of Tahoe-LAFS/Tor to be leveraged. Also, the same mechanism could be generalized for other purposes in the future (e.g., a grid-wide "news cap" informing customers of scheduled upgrades, potential downtime, etc.)

However, service providers like Least AUthority can potentially de-anonymize users since "billing caps" would presumably be maintained on a "per-customer" basis (and service providers could, for example, log/monitor when shares corresponding to that cap are requested, and thereby determine time zone or other behavioral patterns).

A.3.2 Zero-Knowledge Proof-of-Standing

One disadvantage to the current specification is that there is no hiding amidst the mass of all S4 2.0 users nor even of all Tor users when accessing the service via Tor. This is roughly equivalent to the privacy compromise that would come with any identity-based subscription management scheme. An accounting scheme has been imagined and even partially implemented (though not accepted in the Tahoe-LAFS master branch) which operates exactly in this way and which would therefore noticeably improve on the privacy-protecting properties of a Tahoe-LAFS deployment which leveraged it on a single shared grid instead of using many small per-subscription grids.

To avoid this issue, a more sophisticated scheme allowing a Tahoe-LAFS client to prove it has the right to access the service would be required. An example would be the presentation of a zero-knowledge proof that the client knows *some* account identifier (such as a return shielded address) for a subscription in good standing.

This is a scheme for revoking access from users who have allowed their subscriptions to lapse (ie, no longer paying for service). The scheme provides cancellation with granularity of the a fixed grace period. It preserves the subscribers privacy and provides a configurable level of correlation between a particular subscriber’s actions.

In this scheme, a subscription identifier is generated at signup time and conveyed to the subscriber (presumably using the same mechanism as is used to convey the introducer fURL). Subsequently, when

making Tahoe-LAFS API calls, the client presents a zero-knowledge proof that it knows a subscription identifier for an account in good standing. To construct this proof:

1. The client asks the server for a MACed and timestamped set of hashes of subscription identifiers for accounts currently in good standing.

2. The client executes a SNARK which proves it knows a subscription identifier which hashes to a value included in the set.

The server will accept proofs performed against sets of up to a certain age (a grace period). A client presenting a proof older than this will be denied service and must repeat the proof construction. A client *may* repeat the proof construction more frequently than required. The more frequently the proof is reconstructed, the more its actions are de-correlated.

The server will maintain the set of subscriptions in good standing based on Zcash payments it observes, adding and removing elements over time as necessary.

This scheme requires Tahoe-LAFS protocol changes such that a proof is presented with each API call. It also requires the creation of a subscriptions-in-good-standing service to provide one of the inputs to the proof. The Tahoe-LAFS plugin-based Accounting API may be flexible enough to be used to do this.

This scheme also makes public information about the number of subscriptions in good standing (it may be possible to mitigate this to some degree by quantizing the size of this set by padding it with garbage values).

This scheme also requires the periodic transfer of a non-trivial amount of data. For example, 32 bytes per hash multiplied 1e6 subscriptions is around 30 MiB of data for each client to download once per **Grace Period**. It may be possible to mitigate this by partially restricting the size of the set transferred, creating a smarter proof construct with something like a merkle tree or private information retrieval scheme. For example, if a client knows its identifier hashes to a string beginning with ***A*** then it could request the subset of hashes which begin with ***A***. This has the downside of revealing some information to the server, but if there are a sufficiently large number of subscribers then this may be a negligible disclosure. The length of ***A*** could be dynamically adjusted over time to ensure sufficient privacy (a sufficiently large result set) balanced against reasonable download size.

This scheme has the same privacy-related shortcomings as Zcash shielded transactions in the face of few users. If there is only one user on the system, then by presenting this proof, the client reveals exactly their identity (in terms of subscription identifier) to the server, regardless of other factors.

A.3.3 Rolling fURLs for Access Revocation

This is a scheme for revoking access from users who have allowed their subscriptions to lapse (no longer paying for service). In this scheme, payment in full receives a response containing an introducer fURL which will be valid for the duration of the period covered by the payment. This scheme provides cancellation with granularity of the billing period. It preserves the subscribers privacy without correlating all of the subscriber's actions with each other. This scheme is ***incompatible*** with free trials.

- Let **BillingPeriod** be a time delta giving the billing period.

- Let **Interval**$_N$ represent the Nth billing interval.
- Let **Introducer**$_N$ represent the Nth introducer.
- Let **StorageServer**$_{NK}$ represent the Nth instantiation of the storage server having access to the underlying storage uniquely identified by K.
- Let **StorageServerSet**$_N$ represent all storage servers from the Nth instantiation.
- Let **Grace** be an integer giving the overdue payment grace period in multiples of **BillingPeriod**.

At the beginning of **Interval**$_N$, the service provider creates **Introducer**$_N$ and **StorageServerSet**$_N$. Subscribers who remit payment during **Interval**$_N$ are given the *fURL* of **Introducer**$_N$ in response.

At the end of **Interval**$_N$, the service provider destroys **Introducer**$_{N\text{-}Grace}$ and **StorageServerSet**$_{N\text{-}Grace}$.

The Tahoe-LAFS *node id* is set such that it the value is shared by StorageServer$_{NK}$ for all N. This will cause Tahoe-LAFS clients to treat these storage server instantiations as equivalent for the purposes of peer selection and share placement, which is important because no data is moving as part of this scheme.

Either Tahoe-LAFS, Gridsync, or both in combination with a Zcash wallet-type tool is updated to receive the service provider's payment response, extract the updated fURL, rewrite the configuration, and restart the client (runtime reconfiguration would be preferable).

In this way, continuous service is provided to subscribers who are less than **Grace** billing periods late on their payments while service is discontinued for subscribers who are **Grace** or more billing periods late on their payments.

If (**Grace** - 1) * **BillingPeriod** is too small then the fURL transition is likely to cause a service interruption. Values should be chosen which allow any reasonable in-progress operations to complete (client quiescence).

During the overlap period, the "old" storage servers are put into read-only mode (by setting `[storage] read_only = true`; see the docs). This will cause clients to place shares only on the new servers.

A possible short-term solution might be to alter the storage server implementation so that it is possible to have it gracefully terminate. The listening port would be disabled so that no new connections would be accepted but it would continue to service all existing connections until they close normally, only exiting when the last such connection has closed.

A.4 Additional Subscription Features

Many subscription-based services offer other features that users have come to expect. Some of these have not been addressed by the current approach to P4.The following are some ideas on how to add these features in.

A.4.1 Providing Support

Currently, all support requests for S4 require a user to personally contact us and provide certain information to ZenDesk, the support management software. It would be ideal to explore integrating or offering as a side channel existing tools that allow for anonymous communications between the service

provider and the customer for the purpose of technical or billing support. An identifier would need to be used, but the communication channel could help to shield Least Authority from knowing further personal information.

A.4.2 Requesting and Issuing Refunds

In the EU, refunds are required to be offered.[10] Another necessary feature would be to explore how to support refunds without learning personal information from the account holder. This could be done via the same channel that offers the support (see above), or a special type of transaction with a reply address could be enabled to allow for a refund. It would be expected, however, that some number of customers will accidentally forget to include a return address or include the wrong return address. This would also require a client-side tool, and existing wallets don't currently support putting in a reply address.

A.4.3 Trial Periods

Currently, S4 offers a 30-day trial period for new subscription customers. At the end of the trial period, the subscription state transitions to a *payment needed* state. If a subscription remains in a "payment needed" state for too long, it is discontinued. It would be good to design a method for offering trial periods that doesn't require additional personal information to be shared. This could be done using some of the existing methods listed above for other communications and subscription cancellation management, such as rolling fURLS.

A.4.4 Price Adjustments and Predictability

Currently, users must check the invoice for the next payment amount and due to the volatility of cryptocurrency, changing the amount will likely be necessary. This is not in line with fixed-price subscription models, and it would be nice to add some predictability for the users. Further research should be done to look into other ways to adjust prices for the volatile cryptocurrency markets and manage fluctuating exchange-rates.

For example a) customers could be allowed to pay for multiple months of S4 at a time and/or far in advance (i.e., "top-up" style billing), or b) the "amount owing" could be re-calculated/collected at regular intervals in order to minimize risk for both parties. This issue is extra complicated for *subscriptions* (as opposed to single-payment purchases) since cryptocurrencies/Zcash prices can increase/decrease significantly both in between billing cycles and within a "grace period".

A.5 Better Wallets and Clients

It would be very helpful if wallets and clients were further developed to support subscription payment protocols. Management of subscriptions require that an external application periodically query the user's ”zcashd” and/or "viewing key". While this could be Gridsync, it could also be an entirely separate/minimal/lightweight "subscription wallet" made with the purpose to watch for "billing" messages to a z-addr/EMF and inform the end-user when a payment is due. This could prove valuable to the entire cryptocurrency ecosystem, not just the Zcash ecosystem.

Wallets can use the protocol to check account standing and handle the payments and to ensure a payment experience that does not excessively burden users when compared to conventional systems (i.e.

[10] https://europa.eu/youreurope/citizens/consumers/shopping/guarantees-returns/index_en.htm

credit cards).The development of tools to assist in making these payments (eg subscription-enabled wallets) should be supported.

A.5.1 Account State Management

The account-related state machine described in this paper is likely incomplete. However, improvements to that state machine should not negatively impact the novel characteristic of this payment system. The state machine takes the transaction amount as inputs only. It is expected that amounts will carry almost no identifying information because the required amounts are known in advance (namely, the price of the subscription). Certain time-based metadata about the payments **may** carry additional information. Strategies for reducing the possibility of information about a user identify being leaked this way should be discussed.

However, there are limitations to the usability of existing wallets and none of them can "speak"/parse P4 invoices, at this point in time. Also, there are no tools, yet, to parse out the EMF and alert the customer that a new payment is due, of course, but at least it would be available on the blockchain.

A.6 Incorporate Native Tokens

Another approach could be to incorporate a native token to assist with managing the cryptocurrency exchanges and volatility. On receipt of payment, a token of some sort could be issued and required for interaction with the storage server. One further advantage is that tokens can be invalidated or expired, if necessary, to require users to pay again. This approach may not require as many operational difficulties as rolling fURLS but it does require significant protocol changes to Tahoe-LAFS clients and servers. It would also require significant design efforts for the token itself, including the economics of storage, incentive design and mechanisms for interactions. There are many questions that would need to be answered, such as: Is it purely random and tracked in a database? Or a MAC'd expiration timestamp? Is it a pure bearer token, or does it have further restrictions? Can it be a privacy protecting token, or does it link all actions by its user? This approach of using tokens has been previously discussed in terms of a "grid access token" and in the context of supporting communications across grids—as opposed to within a grid, which is already supported by Tahoe-LAFS.