Confirm Activity in the NuCypher Network

Ghada Almashaqbeh

NuCypher
ghada@nucypher.com

Abstract. In this document we introduce several aspects related to the confirm activity problem in the NuCypher network. These include the different definitions of this problem, the underlying threat model, the current adopted solution, and its relation to both service fees and inflation rewards. In particular, we consider two flavors of the confirm activity issue, namely, confirm availability and confirm service activity. The former confirms that Ursula is available all the time and replies to all requests coming from Bob, while the latter provides a proof that an Ursula has served a specific number of distinct re-encryption requests within a given a period. We present potential solutions for each of these flavors and discuss at what stage of the network operation they are likely to be used.

1 Introduction

The NuCypher network [1] is a decentralized key management system, encryption, and access control service. It uses proxy re-encryption, namely, the Umbral scheme [2], to delegate access to encrypted documents through a public network. This network is composed of a set of semi-trusted re-encryption servers, or Ursulas, that implement access control policies created by data owners, or Alices. Alice supplies each Ursula with re-encryption keys allowing Ursula to transform ciphertexts encrypted under Alice's public key into ciphertexts encrypted under the delegatee's, or Bob's, public key. This enables the latter to decrypt the ciphertext without revealing anything about the decryption keys or the raw data to the intermediate servers.

Joining the NuCypher network is governed by consensus rules defining the service setup and the monetary incentives paid to provide the service. In order to join the system, Ursula needs to stake an amount of NU tokens for a specific period which will be released when the pre-specified staking period is over. Alice chooses an Ursula to hold an access control policy with a probability proportional to the stake this Ursula pledged in the system. Therefore, the stake value influences the amount of service load an Ursula may receive. This stake is also used to punish Ursula financially, by revoking part of it, if she cheats and this cheating is detected.

Alice joins the network by creating a policy to be implemented by a set of Ursulas. This policy specifies the re-encryption key fragments each Ursula will need to answer requests coming from Bob(s), as well as the duration of the policy,

which is basically the timeframe during which Bob is authorized to access Alice's data. Furthermore, Alice will lock an amount of Ether into the policy to be used to pay Ursulas for the re-encryption service. As noted, Alice is not exposed to the NU token. All that she needs is an Ethereum wallet, awareness of the NuCypher network architecture to select Ursulas, and knowledge of the NuCypher rules of preparing access control policies.

Bob, on the other hand, does not deal with currency in the system. It interacts with the storage network to retrieve the encrypted data, and with Ursulas in the NuCypher network to request the re-encryption service.

1.1 Monetary Incentives in the NuCypher Network

Ursulas are rewarded for the re-encryption service by using two sources; the first is the service fees collected from the policy owner Alice, while the second is inflation rewards, i.e., newly minted NU tokens, coming from the NuCypher network. The latter will be provided during the early stage of the network operation to encourage adoption. It is expected that these rewards will disappear as the token cap is reached, similar to other cryptocurrencies in the space.

Currently, the inflation rewards for each round or period are computed for each Ursula based on the size of her stake given that she confirms to be online during this round. Confirming the status of being online is done by calling a simple function in one of the NuCypher contracts, which is like a signal that the calling Ursula is online.

As for fees, Ursula collects part of the Ether locked by Alice during the policy's duration in proportion to the number of periods up to that point in which Ursula has confirmed her online status.

1.2 Problems with Current Reward Computation Approach

The current approach of computing Ursula's inflation rewards and service fees suffers from two main problems:

- The rewards computation (both fees and inflation) is agnostic to the number of requests Ursula serves. Thus, an active Ursula who may serve a huge number of re-encryption requests, and one who does not receive any requests (or a malicious/lazy Ursula that does not answer Bob's requests), will both collect the same amount of rewards if they hold identical policies.
- Confirming being online is flawed in the sense that this function does not require any input or proofs from Ursula on providing the service. Ursula can stay offline and not respond to Bobs' requests, but come online merely to call the confirm activity function. This allows Ursula to collect both inflation rewards and fees without doing any work. Even worse, Ursula will be able to get her stake back once she unlocks its her stake (i.e., when policies being managed by Ursula expire).

1.3 Problem Statement

The confirm activity problem is defined differently based on the stage of the NuCypher network. In early network operation, when inflation rewards are still distributed, we are concerned with the availability of Ursulas and being willing to serve all requests coming from Bob. In other words, regardless of the number of requests served, the rewards value will be computed based on the period during which Ursula is online and well-behaving. While in later stages, when inflation rewards disappear and only fees exist, confirm activity will be tied to providing the re-encryption service in the sense that the payment will be computed based on the amount of service provided. Thus, Ursula needs to confirm its activity by proving that she served a given number of distinct re-encryption requests during a given period. To make the distinction clear, we refer to the first as confirm availability, and for the second as confirm service activity.

In this document, we introduce several solutions for both forms of confirming activity. These solutions differ in the trust, efficiency, interactivity, and resource requirements, in addition to their security guarantees. We divide the document into two parts, the first is about confirm availability, while the second is about confirm service activity. We begin with outlining the adopted threat model for each problem definition, as well as the network model. Then, we present the proposed solutions, discuss the aforementioned trade-offs and requirements of each of them, with the goal of selecting one of these solutions to be adopted for each stage of the NuCypher network operation.

1.4 Threat and Network Models

This section describes a modification for the network model of NuCypher required by the proposed solutions, in addition to the threat model adopted in this work.

1.4.1 Network Model. In order for the proposed solutions to work, we need to ensure freshness, integrity, and authenticity of the requests (as well as service complaints as we will see later) issued by Bob. This can be achieved by requiring Bob to sign each re-encryption request it issues and to include a timestamp, or sequence number, in each request to ensure freshness. The keypair used for the signature should be separate from the keypair Bob uses for proxy encryption in the system.

Until now we assume that Alice pays for the service by using the Ether she locks in the policy contract. Other arrangement may emerge in the system like having Bob pay for each requests he issues. Such an arrangement and its implication on the confirm activity issue will be studied once it becomes part of the NuCypher network protocol.

1.4.2 Threat Model. In our threat model, we make the following set of assumptions:

- **Self-interested parties:** We do not place trust in any party and we assume that all participants are self-interested. This means that a party may decide to follow the protocol or deviate from it, either on its own or by colluding with other attackers, and such decision is solely based on what maximizes the financial profits of this party.
- **Attackers' collusion:** If it is profitable, an Ursula may, for example, spin out its own Alice and/or Bob or collude with Alice. We do not consider collusion between Bob and Ursula as a practical threat. This means that cases of Bob pretending that he got service from Ursula (while no service is delivered) is not an issue. We do not suspect this will be of any importance and there is no motivation to do it given that the work Ursula does for re-encryption is minimal.

  The above non-collusion assumption does not affect the solutions proposed to handle the confirm service activity discussed in Section 3. This means that these solutions provide higher security guarantees and address the issue of potential collusion between Bob and Ursula. It could be the case that relaxing the requirement of addressing this collusion case allows deploying more efficient solutions. In order to make an educated decision of the plausibility of this threat, we need more data about the system operation and the behavior of the participants once the inflation rewards disappears in the system. Therefore, we leave this issue until later when fees become the only source of rewards for Ursulas.
- **Honest Ursulas:** We also assume that when sampling a subset of Ursulas in the network, at least one of them is honest. This assumption can be achieved by having a global assumption regarding the lower bound of the number of honest Ursulas with respect to the total number of Ursulas in the system. (Or it can be achieved by deploying a special entity like an external verifier, that changes on a periodic basis, or one by the NuCypher company, that is trusted to faithfully participate as a member of each samples set. This verifier does not provide any other services in the system.)

Other than the above, we have usual assumptions like dealing with computationally bounded adversaries that cannot break secure cryptographic primitives with non-negligible probability. (We may need to work in the random oracle model, but this depends on the security requirements of the proposed solutions.)

2 Confirm Availability

In this section, we present a solution for the issue of confirming availability of Ursulas. As mentioned before, in the early stage of the network launch we only require Ursulas to be online and responsive to any reencryption requests issued by Bob.

We begin with the design details of our solution, after which we discuss its resilience to any potential security attacks and its efficiency in terms of implementation requirements and compatibility with the current status of the NuCypher network.

2.1 Optimistic Challenge-based Approach

The proposed solution is centered around two important concepts: requiring minimal changes to the current network implementation, and minimizing the additional overhead.

Discovering working Ursulas. Recall that in the NuCypher network, Bob receives a map from Alice containing the set of Ursulas that has the key fragments needed to implement the access control policy. When issuing a re-encryption request, Bob uses this map (in association with the network discovery protocol) to reach each of these Ursulas and obtain the service.

In particular, in the NuCypher network there is a separation between the role of a staker and a working Ursula. All stakers' Ethereum addresses, along with their stake values, are recorded in one of the NuCypher network smart contracts, namely, the StakeEscrow contract. Each of these stakers will be tied, or bonded, to a working Ursula to provide the re-encryption service by having its address listed next to the staker's address.

Note that the StakeEscrow contract does not contain any information about how to reach a specific working Ursula. Thus, the map that Bob receives includes only the Ethereum addresses of the working Ursulas. To allow the parties to communicate with each other, each participant, i.e., Alice, Bob, Ursula, employs a discovery protocol (more like a gossiping protocol) to discover the IP addresses and ports of the working Ursulas in the network.

The discovery process also includes retrieving the keypair a working Ursula uses for the re-encryption service purposes. We call this keypair the stamp of a working Ursula, and we require an Ursula to use its stamp when signing all messages related to the proposed confirm availability protocol.

Optimistic Ursula Challenging. Once Bob discovers the working Ursulas listed in the policy map, he can connect with each of them and start issuing service requests. As noted, this communication is one-to-one meaning that no other Ursulas (or any NuCypher network participant) mediates the communication between Bob and each working Ursula. This in turn means that no one can attest to whether Ursula has responded if Bob complains later that he did not hear back.

The main idea is to add a mediator, or potentially a witness, in the service process (if needed) to monitor a specific working Ursula and challenge its responsiveness. Hence, it is an optimistic protocol invoked only when Bob complains about not receiving the service. In detail, for each round (where a round is the time needed to mine a block on the blockchain) a set of Ursulas is selected at random. We call these gateway Ursulas. Bob contacts a working Ursula asking for the re-encryption service as usual. If this Ursula does not respond, Bob complains to a gateway that this Ursula did not respond. At this point, the gateway will act as an intermediary and forwards Bob's request to this working Ursula on behalf of Bob and waits for the answer. If the working Ursula does not answer, this will trigger other gateways to perform the same process, i.e., forward the request on behalf of Bob and wait for an answer. If the working Ursula is still unresponsive, these gateways will collectively sign a witness, basically a statement saying that the working Ursula was contacted by these gateways and she did not respond, against this Ursula and publish it (see Section 3.1.2 for an overview of collective signing). Once this witness is verified by the NuCypher network, part of the working Ursula's stake will be slashed as a punishment.

Gateway Ursulas Selection. Let's assume that for each challenge phase a set of n gateways Ursulas is selected. A witness will be accepted by the network if it is signed by at least t gateways among this set (this threshold is a relaxation of requiring all gateways to sign since it could be the case that not all of them are active).

A simple idea to select the gateways is to use a high entropy source of randomness to obtain a random beacon for each round. Then feed this beacon, concatenated with the hash of Bob's request, to an iterative hashing process. After each iteration, map the hash to a working Ursula index as listed in the StakeEscrow contract. That is, the working Ursula bonded with the first staker listed in the contract has index 0, the next has index 1, and so on. Then, rehash the previous hash and map the output to an index. If any collision happens, i.e., a previous index is produced again, discard it and proceed to the next hash iteration. This process is repeated until a distinct set of n indices is computed.

Note that the selection process may contain Ursulas for which Bob does not have a contact information. Hence, Bob has to discover these Ursulas before being able to submit a complain. To reduce the delays, and as outlined earlier, Bob starts with complaining to one gateway Ursula and involves the rest if the working Ursula does not respond to the challenge from this gateway. Hence, Bob can start with a gateway that he already knows how to reach (if any) and in the meantime works on discovering the rest of the selected gateways (if he does not already have their contact information).

Furthermore, the above protocol assumes working in the random oracle model, meaning that hash functions are modeled as random oracles. Thus, the iterative hashing process selects at random the working Ursulas. Although, we believe this is sufficient for our purposes, this can be replaced with more sophisticated approaches to produce random strings (that are then mapped to indices), e.g.,

a verifiable random function [3]. We leave this as part of our future work in case we decide to pursue this path.

As for the high entropy source of randomness, there are several potential approaches here. We can rely on an external source, e.g., the NIST random beacon [4] or a combination of multiple sources, for that. Or we can work also in the random oracle model and assume that block hashes on the blockchain are drawn from a uniform distribution, and thus, use the block hash as a random beacon. In particular, for the current round, the hash of the block that was mined y rounds ago is used (y is confirmation interval in the underlying cryptocurrency system). Alternatively, and to avoid the random oracle assumption, we can use randomness extractors to produce a high entropy beacon from the block hashes which itself could be low entropy [5].

Witness Verification.

Quantifying the Financial Punishment or Slashing Value.

Operating Only During The Challenge Phase.

DoS Attacks.

Unresponsive Gateways.

2.2 Security Analysis

2.3 Performance Analysis

3 Confirm Service Activity

In this section, we present potential solutions for the confirm service activity problem. The goal is to come up with provably secure solutions that allows an Ursula to prove that it has served a given number of distinct re-encryption requests within a given period.

The rest of this section provides an overview of the underlying cryptographic primitives and protocols, after which it discusses the proposed solutions along with an analysis of their security and efficiency aspects. Lastly, the section concludes with directions of future work on the confirm service activity issue.

3.1 Preliminaries

This section provides an overview of two concepts that we use in the proposed solutions. These include the framework of proof carrying data (PCD) and the collective signing (CoSi) protocol.

3.1.1 Proof Carrying Data (PCD). The paradigm of PCD [6] allows proving to the correctness of a distributed computation involving untrusted parties. It produces a single proof for the output that attests not only the correctness of the final result, but also the correctness of the entire history of intermediate computations that produced this result. Correctness here is defined as a polynomially computable predicate that abstracts the properties, or invariants, that must be satisfied.

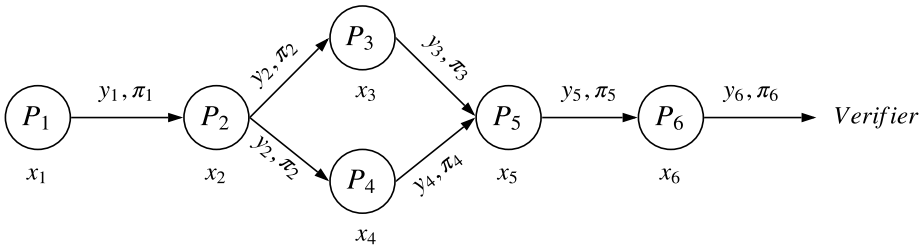

Fig. 1: An example of a PCD application. A distributed computation involving parties P_i, for $i \in \{1, \ldots, 6\}$, each of which has a local input x_i and produces an output y_i with a proof π_i.

To clarify how PDC works, consider a distributed computation task that involves the set of parties shown in Figure 1. Party P_1 starts the computation with its own input, produces an intermediate output with a proof saying that it performed the computation as defined by the protocol. It then sends both the output and the proof to the next party, which is P_2 in this case. Here, P_2 will have its own input x_2, as well as everything it received from P_1 (including the

proof) as input to the computation it will perform. Similarly, P_2 will produce another intermediate output and a proof. This proof not only attests to the correctness of the computation done by P_2, but also the history that lead to the result y_2. In other words, it implicitly includes the proof from P_1. The same process continues until the full computation is finished. Anyone can verify the correctness or compliance of the whole distributed protocol by only verifying the last proof π_6 produced by the exit party, which is P_6 in the figure.

As shown, PCDs enable untrusted parties to work with each other in a fully distributed fashion, without overwhelming the system with the storage and verification of individual proofs for each step of the performed protocol. Also, they attest to correctness or compliance to the prescribed protocol without re-executing any of the intermediate computations. Thus, they provide a promising paradigm for confirming service activity in a compact way.

3.1.2 Collective Signing (CoSi). CoSi [7] is a protocol that enables a set of distributed parties to cosign a statement together in a way that produces a single signature. As such, both the verification time and the space requirements are just like having a single signer. The difference is that this signature attests that all parties agree with the signed statement instead of only one.

CoSi builds upon Schnorr multisignatures [8–10], but combines them with communication trees to speed up the signing process in case of a large number of cosigners. In what follows, we only present the protocol with a plain communication architecture as it suffices for the confirm service activity solution introduced in this document.

Schnorr signatures work in a group $\mathbb{G}$ of a prime order q and generator g, such that the discrete log is believed to be hard in this group. Take n parties that want to sign a statement together. Each of these parties has a secret $sk_i \in \mathbb{Z}_q$ and a public key $pk_i \in \mathbb{G}$ such that $pk_i = g^{sk_i}$. To sign a message m, one of these parties, let's say P_1, coordinates the signing process as follows:

1. P_1 prepares a message m and sends it to the rest of the signers.
2. Each party P_i, possibly after verifying m, selects a secret $\tau_i \in \mathbb{Z}_q$ and compute $V_i = g^{\tau_i}$, then it sends V_i back to P_1.
3. P_1 aggregates all random values received from the signers, and its own, by computing $V = \Pi_{i=1}^{n} V_i$.
4. P_1 then computes $c = H(V||m)$, where H is an appropriate hash function. P_1 then sends c to the rest of the parties.
5. Each party computes a response $r_i = \tau_i - sk_i \cdot c$ and sends it to P_1.
6. Lastly, P_1 computes $r = \sum_{i=1}^{n} r_i$, and outputs the collective signature over m as (c, r).

Verifying the signature proceeds as in classical Schnorr signatures [8] with one difference. An aggregated public key pk is used in the verification process, which is computed as $pk = \Pi_{i=1}^{n} pk_i$.

The work in [7] tackles several issues related to the availability of the signing parties, and optimizing the communication between them. We believe that such

techniques can be used in the NuCypher network if we adopt the CoSi-based solution for the confirm service activity issue.

3.2 Potential Solutions

This section outlines several potential solutions for how to let Ursula prove that she served a given number of distinct requests in a publicly verifiable way. One of these solutions, the PCD-based one, is still a work in progress. The goal is to share the high-level idea of each of these solutions and to chose the best one, after we define the meaning of best, to be adopted by the NuCypher network.

3.2.1 PCD-based Scheme. The main idea here is to adapt the PCD framework so that Ursula can combine the correctness proofs she computes for Bobs' requests in a single proof attesting to the following fact: "Ursula has served ω distinct re-encryption requests correctly."

Thus, in this setup, there is only one computation party, or prover, namely, Ursula. For each round, where a round could be the time needed to mine a block on the blockchain, Ursula starts with input x_1, which is a signed and fresh request from Bob. It answers this request with $cFrag_{x_1}$ and produces a correctness proof π_1 as defined in the Umbral scheme, in addition to another correctness proof $\hat{\pi}_1$ that will be used in the PCD composition. Then, when the next request x_2 arrives, which is the input for the next step in the computation, Ursula answers this request as before and produces $cFrag_{x_2}$ and a correctness proof π_2, then it uses $(x_2, cFrag_{x_2}, \pi_2)$ and the previous proof $\hat{\pi}_1$ to produce $\hat{\pi}_2$. $\hat{\pi}_2$ does not only attest to the correctness of $cFrag_{x_2}$, but also attests to the fact that Ursula has served two valid distinct requests until now. The same process is repeated until the end of the round to produce a single proof $\hat{\pi}_\omega$ along with the number of served requests ω. Figure 2 depicts this process pictorially.

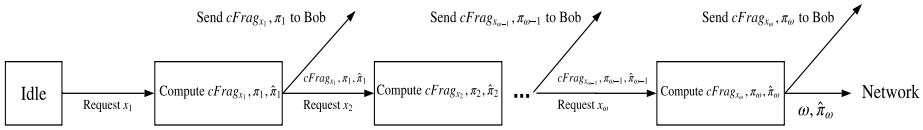

Fig. 2: PCD-based solution diagram. Ursula starts a round in the idle state. When the first request arrives, Ursula responds to the request and starts composing the proofs as new requests arrive. At the end of the round, Ursula announces the number of served requests and a single proof attesting to its claim to the network.

The final proof will be processed by the contract that governs Alice's policy. A valid proof allows Ursula to claim fees out of Alice's Ether escrow as a payment for ω re-encryption requests.

The above is a high level description of the idea. However, more time is need to understand PCD and SNARKs in order to come up with a concrete construction (if possible). Please see Section 3.4 for more information.

3.2.2 CoSi-based Scheme. This solution utilizes the idea of collective signing by electing a committee that will sign a report submitted by a specific Ursula after verifying that the report supports the claim that "Ursula has served ω distinct re-encryption requests correctly."

At a high level, the scheme works as follows. For each round, a working Ursula keeps a full record of all the requests that she served during the round. These include the signed requests received from Bob(s), and the produced $cFrag$ and the correctness proof π (as in the Umbral scheme) for each request. At the end of the round, a committee of t Ursulas is elected, where t is a small integer, e.g., $t = 5$. The working Ursula initiates the CoSi protocol to sign the message m stated above while replacing ω with the actual number of served requests in the round. This Ursula sends m along with the full report to each member in the committee. Each member Ursula (i.e., member in the elected committee) verifies m by checking the report as follows:

1. Check that each request is distinct by checking the sequence number of the request (or any value that is used for freshaness), and valid by verifying Bob's signature over the request. (Here the policy will contain Bob's public key to allow the committee use the right verification key.)
2. Verify the correctness of each $cFrag$ as in the Umbral scheme.
3. Check that the value of ω inside m agrees with the record.

If everything is fine, each member of the committee and the working Ursula finalize the CoSi signing protocol as described in Section 3.1.2. At the end, the working Ursula publishes the collective signature, which can be verified using the accumulated public keys of the cosigning Ursulas.

On the committee election. This can be done by using some deterministic computation over a block hash and mapping the output to Ursulas' public keys. So the selection is not determined by the working Ursula to prevent any potential collusion. The selection may also take into account the presence and size of each Ursula's stake, to avoid an attacker spinning up enormous numbers of Ursulas in order to increase the chance of them controlling the entire committee.

The above scheme works under the assumption that at least one Ursula in the elected committee is honest, which pertains to an assumption on the least number of honest Ursulas in the network. If the latter assumption is under threat, one mitigating approach is to increase t (the committee size).

Another approach is to have a changing set of external verifiers, like trusted partners, that all working Ursulas must use for the CoSi signing of their records.

A third approach is to keep the deterministic selection of the committee but with an external member, e.g., either a trusted partner verifier or a special verifier node deployed and maintained by the NuCypher company. Thus, achieving the assumption of at least one member of the elected committee is honest.

On the publicity of the signed records. Having a valid collective signature from a committee (that has at least one honest member) suffices for the correctness of the signed statement. However, it could be better to keep the records that produced the message m available for a while so that any party can verify them (the verifying parties could maintain such a public record for a given period after which the logs can be discarded).

We can resort to this option just at the beginning to convince the participants of the trustworthiness of the committee or the validity of the assumption that at least one Ursula in the elected committee is honest.

On compensating the committee. This solution may raise the question of why would the elected committee (especially if it is composed of other Ursulas in the system) participate in the CoSi process, which also involves verifying the full request record presented by the working Ursula. This can be pictured as a collaborative work, the committee does the work so that others will do the same when the committee members play the role of working Ursulas.

Another option is to pay the committee for their work, either as part of the inflation rewards, or by having working Ursulas pay for it (however, this may complicate the system operation).

3.2.3 Commit/Challenge/Open-based Scheme. This solution utilizes the idea of commit/challenge/open protocols. In details, Ursula keeps track of the requests coming from Bob(s) during a time round and assign each of them a unique sequence number. That is, Ursula replies with $cFrag$ and the correctness proof along with a sequence number showing the order of the request within the batch of requests handled during a round. At the end of the round, Ursula constructs a Merkle tree of all requests, where the requests (and their replies) are the leaves of the tree ordered by the sequence numbers. Ursula signs the root of the tree, denoted as $root$ to produce a signature σ_{root}, then it publishes $(\omega, root, \sigma_{root})$ to the network (recall that ω is the number of served requests).

To prove correctness, Ursula will be challenged to open some leaves in the Merkle tree. This can be done by having the policy contract select at random (e.g. based on the block hash or any other mechanism) the leaf IDs to be opened. If Ursula fails to open them correctly within a predefined timeframe, she loses the whole fee she was supposed to collect for serving ω requests.

An alternative approach to the challenge/open scheme described above is to have Ursula publish the full Merkle tree on some known public space but not on the blockchain, and make the full record available online for a specific period to allow anyone to verify the work. If no one files a complaint about the tree (we still need to define correctness properties, like checking all Bob public keys are defined in the policy created by Alice, and that the response is valid by checking the proof produced by Umbral), Ursula collects the fees for the provided service. On the other hand, a valid complaint or proof-of-cheating costs Ursula the full allocation of fees, or potentially involves slashing her stake.

3.3 Discussion and Analysis

Here we will discuss the security, financial, and performance implications of the proposed solutions to choose the best. This will be delayed until we crystallize the details of each of the solutions.

3.4 Future Work

Most of the future work on the confirm service activity issue will be dedicated to construct a concrete PCD-based solution. This includes the following:

- Understand PCD and its applications in a greater depth. This also requires exploring NIZK in general and SNARKs in specific.
- Define an efficient output correctness predicate that characterizes the correctness of the statement that PCD will attest for. This could be hard for the confirm activity issue as the value of the output is dynamic based on the workload Ursula may receive.
- Define a way to compose the proofs efficiently and produce a single proof at the end. This will be tied to the formulated predicate.
- Argue about the correctness and security of the concrete scheme.

The above will take time, but the good news that this will also be helpful to the work on PPSC. So it is more like developing part of the background needed for PPSC.

References

1. Michael Egorov, MacLane Wilkison, and David Nuñez. Nucypher kms: decentralized key management system. *arXiv preprint arXiv:1707.06140*, 2017.
2. David Nuñez. Umbral: A threshold proxy re-encryption scheme. 2018. `https://github.com/nucypher/umbral-doc/blob/master/umbral-doc.pdf`.
3. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.
4. *NIST Random Beacon*. `https://beacon.nist.gov/home`.
5. Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.
6. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.
7. Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. Ieee, 2016.
8. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
9. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.
10. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 245–254, 2001.