Confirm Activity in the NuCypher Network

Ghada Almashaqbeh

NuCypher
ghada@nucypher.com

Abstract. This document introduces 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 the fees and the inflation rewards. Then we present potential solutions for the two types 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 discuss the different solutions and outline 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 ciphertext encrypted under Alice's public key into ciphertext encrypted under the delegatee's, or Bob's, public key. This enables the latter to decrypt the data 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. As such, 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, when 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 or data 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 out there.

Currently, the inflation rewards are computed for each Ursula based on the size of their stake, and their confirmation that they are online, once per period. Confirming the status of being online is done by having each Ursula call a simple function in one of the NuCypher contracts to signal that she is online.

As for fees, Ursula collects 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 their 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 is agnostic to the number of requests Ursula serves. Thus an active Ursula who may serve a huge number of requests, and one who does not receive any requests (or a malicious Ursula that does not answer the requests she receives from Bob), will both collect the same 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, and simply come online merely to call the confirm activity function. This allows Ursula to collect both inflation rewards and fees without doing anything, and even worse, it will be able to get her stake back once it unlocks (policies being managed by Ursula will have expired by this point).

1.3 Proposed Solutions

We need a provably secure mechanism to be used by Ursula to report the number of distinct Bob requests it handled during a specific period, and use this number

to compute the fees to be collected from Alice. Thus, we call this confirm service activity, as opposed to confirm activity that refers to solely being available online.

Although we can use the number of served requests to compute the inflation rewards as well, this will encourage Ursula to collude with Bob to issue a large number of requests. If inflation rewards were to be distributed based on the earning of fees (which is desirable, as it would link node subsidization more closely with the network service), this might encourage Ursula to deploy its own Alice(s) and Bob(s) to increase the demand, and thus increase its cut of the inflation rewards.

In this document, we introduce several solutions for confirming service activity. These solutions differ in the trust, efficiency, interactivity, and resource requirements, in addition to their security guarantees. We present each of these solutions, where one of them is still a work in progress, discuss the aforementioned trade-offs and requirements of each of them, with the goal of selecting one of these solutions to be adopted in the NuCypher network.

1.4 Roadmap

The rest of this document is organized as follows. Section 3.1 outlines the network and threat models adopted by the proposed solutions. Section 3.2 provides an overview of the underlying cryptographic primitives and protocols, while the proposed solutions are presented in Section 3.3 with their security and efficiency aspects analyzed in Section 3.4. Lastly, directions of future work on the confirm activity issue are outlined in Section 3.5.

2 Confirm Availability

3 Confirm Service Activity

3.1 Threat and Network Models

This section describes a modification for the network model of NuCypher required by the proposed confirm service activity solutions, in addition to the threat model we adopt in this document.

3.1.1 Network Model In order for the proposed solutions to work, we need to ensure freshness, integrity, and authenticity of the requests issued by Bob. This can be achieved by requiring Bob to sign each request it issues and to include a timestamp, or sequence number, in each request to ensure freshness.

[david] NuCrypher currently supports this by allowing Bob to include a blockhash, although it's not being used for the moment.

The following approach can be used here; given that Umbral already requires Bob to have a key pair, this pair can be used for signing and verifying the signed requests. This also means that Ursula should be aware that the secret key can be used for signing the requests and the public key should be known to Ursula to verify the signatures.

[david] Reusing the same keypair for encryption and signing may introduce some key management problems. Is it possible to find a solution that supports separate keypairs?

3.1.2 Threat Model We do not place trust in any party and we assume that all participants are self-interested. This means that a party may follow the protocol or deviate from it, either on its own or by colluding with other attackers, and decisions are solely based on what maximizes the financial profits of this party. As such, Ursula may collude with Bob or Alice (by spinning its own Alice and/or Bob for example) to achieve her financial goals.

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 in the confirm service activity protocol. 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 proposed solutions.)

3.2 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.

3.2.1 Proof Carrying Data (PCD) The paradigm of PCD [3] 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.

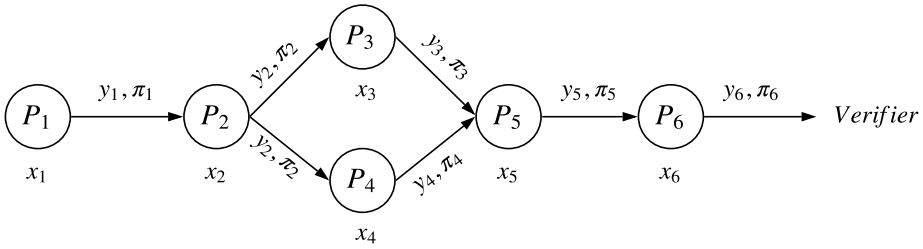

Fig. 1: An example of a PCD application. A distributed computation involving parties P_i, for $i \in \{1, \ldots, 6\}$, each of which has a local input x_i and produces an output y_i with a proof π_i.

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.

3.2.2 Collective Signing (CoSi) CoSi [4] 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 [5–7], 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 [5] 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 [4] 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.

3.3 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.

3.3.1 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.

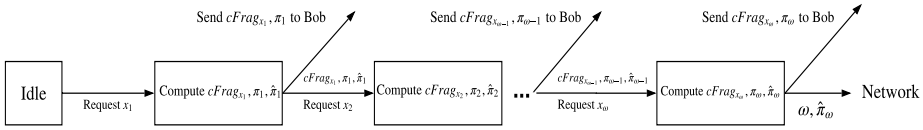

Fig. 2: PCD-based solution diagram. Ursula starts a round in the idle state. When the first request arrives, Ursula responds to the request and starts composing the proofs as new requests arrive. At the end of the round, Ursula announces the number of served requests and a single proof attesting to its 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 3.5 for more information.

[david] I think it is possible to aggregate current Umbral proofs. I started to explore this option in the past but there was no use for it at that moment.

3.3.2 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 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.)
   [derek] Signature only validates request, but for distinct requests, presumably blockhash/sequence number needs to also be checked?
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 3.2.2. 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).

3.3.3 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.

3.4 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 after discussing with the team to avoid biasing opinions about the proposed schemes.

3.5 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. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.
4. 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.
5. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
6. 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.
7. 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.