ReQuest: A Bitcoin-native Bounty Mechanism for Coveted Data

someguy
someguy@durabit.org
www.request.market

4de67a207019fd4d855ef0a188b4519c7040281c9c121fc28574b0bd58bd3927
hash@request.market
www.request.market

Abstract. In order to fulfill further needs within the information goods market, ReQuest proposes creating a bounty bulletin board in order to incentivize the permanent publication of desired files. A user could post a Bitcoin[1] bounty requesting certain information, such as an FGC-9 .stl file, a coveted scientific pdf, an academic text file, or even data cables. This initial bounty would cover the publishing fees, as well as a service payment for fulfilling the bounty. If this data is simultaneously coveted by others, users could add to the bounty to increase the bounty odds of being fulfilled. This scheme could also be used to acquire a previously unavailable piece of data to seed a Durabit bond[2].

1. Introduction

The core premise of ReQuest is to enable users to post bounties requesting specific information. The initial bounty serves a dual purpose: it covers the inscription fees while also providing a service payment to those who successfully fulfill the bounty. Moreover, the protocol allows for the augmentation of bounties through contributions from other users, thereby increasing the likelihood of a successful fulfillment. Once a potential fulfillment of a bounty is submitted , the users who posted the bounty have the ability to engage in a process of social validation to confirm the accuracy of the data.

Implementation of this concept requires a blend of social vetting for data accuracy, weighing contributions to the bounty, and specific contractual parameters measured in byte size. The process of fulfilling bounties involves eligible fulfillers submitting their inscription transaction hash, or a live magnet link, for consideration. In cases where the desired data is already available but not natively published on Bitcoin or Durabit, or previously known but not currently available, such as a renowned .STL file or a software client update, the protocol offers an alternative method to social consensus for fulfillment. This approach involves hashing the file and verifying the resulting SHA-256 output, providing a foolproof means of meeting the bounty's requirements.

2. Bitcoin-native Inscriptions

Inscriptions[3] represent a method of embedding arbitrary content within Bitcoin transactions, effectively creating bitcoin-native digital artifacts, often referred to as NFTs. Notably, the process of creating inscriptions does not necessitate the use of a sidechain or a separate token.

The content model for inscriptions adheres to the structure commonly used on the web. Each inscription comprises a content type, often referred to as a MIME type, and the content itself, which is represented as a byte string. This approach allows for the retrieval of inscription content from a web server and the creation of HTML inscriptions that can utilize and remix content from other inscriptions.
Importantly, all inscription content is stored on-chain within taproot script-path spend witnesses. Taproot scripts offer substantial flexibility in terms of content and benefit from the witness discount, resulting in cost-effective inscription content storage.

Due to the limitations of taproot script spends, inscriptions are created through a two-phase commit/reveal process. First, in the commit transaction, a taproot output that commits to a script containing the inscription content is generated. Subsequently, in the reveal transaction, the output created in the commit transaction is spent, thereby disclosing the inscription content on-chain.

The serialization of inscription content is achieved using data pushes within unexecuted conditionals, referred to as "envelopes." Envelopes are constructed using an “OP_FALSE OP_IF … OP_ENDIF” structure, enveloping any number of data pushes. Notably, envelopes function as effectively inert operations, preserving the underlying script's semantics, and can be seamlessly integrated with other locking scripts.

3. OpenTimestamps

OpenTimestamps[4] is a timestamping infrastructure created by developer Peter Todd in order to address the shortcomings of previous timestamping schemes where individual files had their hashes timestamped directly on-chain individually.

The OpenTimestamps client simplifies the process of timestamp creation and verification. The system utilizes notaries and time attestations, leveraging the Bitcoin blockchain to validate timestamps. Timestamp proofs are constructed by a series of commitment operations applied to the message being timestamped. These operations form a tree-like structure, ensuring that any changes to the message would invalidate the timestamp. OpenTimestamps significantly improves scalability by allowing multiple files to be timestamped simultaneously in a single Bitcoin transaction. This is achieved through the creation of a merkle tree containing the digests of the files, with the root of the tree being timestamped.

To expedite the timestamping process, OpenTimestamps introduces public calendar servers. These servers aggregate individual file hashes to create a single hash commitment that is later timestamped on Bitcoin. This approach accelerates the timestamping process — temporarily unbeholden to Bitcoin’s confirmation time — while still relying on the decentralized Bitcoin blockchain for verification “settlement”.

OpenTimestamps is designed to prove the existence of data at a particular point in time, offering the benefits of cryptographic timestamping, often referred to as "proofs-of-existence."

4. Nostr - Notes and Other Stuff Transmitted by Relays

When using Nostr[5], each individual operates a client, which can take various forms, such as native clients or web clients. To share content, users compose a post, apply their digital signature, and transmit it to multiple relays, which can be either hosted by third parties or self-hosted. To receive updates from other users, individuals engage multiple relays in inquiries about information related to these users. The key distinction from other networks is that anyone has the capacity to run a relay. These relays maintain simplicity and perform uncomplicated tasks, primarily involving the reception of posts from select users and their forwarding to others. Notably, the necessity for trust in relays is minimal, as the authentication of signatures occurs at the client level.

4.1 Nostr Clients and Relays:

Clients: Every user actively engages with Nostr through a client. These clients are individually managed and are responsible for constructing and signing user messages within the network.

Relays: Nostr's decentralized structure allows anyone to run a relay. These relays serve as intermediaries for data transmission between clients. Importantly, relays communicate directly with users but do not engage in relay-to-relay communication.

User identification within Nostr is achieved through the use of public keys. Each post made within the network is digitally signed to ensure authenticity and data integrity. Clients play a critical role in validating these digital signatures, guaranteeing the integrity and security of the information shared. Clients fetch data from relays of their choice and, if desired, publish data to other relays of their choosing. Upon initiating their client, users have the ability to query data from all known relays for updates from users they follow.

Posts on the Nostr network can contain diverse forms of structured data. While the network accommodates various data types, there is an ongoing effort to standardize the most commonly used formats. This standardization promotes seamless data handling across all clients and relays.

Nostr mitigates the impact of user bans and server closures through various mechanisms. A relay can block a user from publishing on that specific relay, but this has no effect on the user's ability to publish on other relays. As users are identified by their public key, their identities and follower base remain intact even in the event of a ban. To simplify the process of discovering new relays, Nostr clients have the capacity to automatically incorporate relay recommendations from users they follow. Users wishing to migrate to a different relay can publish a server recommendation to their previous relay, ensuring a smooth transition. In cases where users encounter bans from multiple relays, impeding the distribution of server recommendations, they can inform close associates through alternative means about the new relay they are using. These associates can then publish server recommendations for the new relay, ultimately enabling the user's follower base to reconnect with their posts.

Nostr promotes censorship resistance by allowing users to publish their updates on multiple relays. Furthermore, relays can charge fees for posting, reducing the risk of censorship. Users are given the flexibility to select alternative relays when facing restrictions on a specific relay, ensuring they can always find a willing host to serve their posts.

4.2 Nostr Event Chains

The nature of Nostr events being signed by the creator’s private key creates an interesting property that ReQuest takes advantage of. One of the internal fields of an event’s data structure is a “tag”. Tags are an array of key/value pairs, with the key denoting a tag type, and the value denoting the contents of the tag. One supported tag type is Nostr events, allowing an event to explicitly reference another event by its event ID.

This creates a dynamic similar to a blockchain, where an event can cryptographically commit to a prior event, which commits to a prior event, and so on. Such Nostr event chains give a cryptographically unalterable chain of messages that reference backwards to the ultimate originating event in the chain. This property will be of use in auditing user and oracle activity.

5. Civ.Kit

Civ.Kit[6] introduces a novel peer-to-peer electronic market system designed to facilitate unrestricted and censorship-resistant trading among participants within the global Bitcoin network. Their framework is anchored on the innovative Nostr protocol, serving as the foundation for a peer-to-peer order book, while utilizing the Bitcoin blockchain as the ultimate arbiter of truth for Civ.Kit’s Web-of-Stakes market ranking mechanism.

The implementation of this market system includes a secure architecture where trades are locked within Bitcoin contracts, thereby eliminating the need for reliance on trusted third parties for dispute resolution. Incentives for all nodes in the market are driven by privacy-preserving service credentials supported by Bitcoin-based payments. This comprehensive system is poised to enable global trade encompassing a wide spectrum of commodities, including fiat currencies, goods, and services, transcending geographical boundaries.

The Nostr protocol serves as a fundamental component in Civ.Kit’s orderbook design, offering significant advantages for building a censorship-resistant and fault-tolerant trading platform. The protocol is characterized by two key strengths:

Multicast Broadcast Mechanism: Nostr provides a multicast broadcast mechanism, allowing trade events to be disseminated with a "best-efforts" reliability to a group of interested clients. This feature ensures the timely and efficient communication of trade events within the network.

Client-Managed Credentials: Nostr clients play a crucial role in managing credentials, facilitating cost-effective migration between different market boards. This attribute enhances the flexibility and accessibility of the trading ecosystem.

In Civ.Kit, four distinct entities are involved: the bulletin board (i.e., a Nostr server linked to a Lightning gateway), the maker, the taker, and the onion routing hop. The protocol unfolds in three essential phases:

Dissemination: Trade orders are routed from the maker to the bulletin board through Lightning onion messaging.
Publication: Trade orders are published from the bulletin board to its subscribed clients.
Settlement: Trade orders are concluded between the maker and the taker over the Lightning Network[7].

6. Bounty Design

The primary issue of funding a bounty for a specific file is the potential for someone to announce a bounty, and then after receiving the desired file, refuse to pay out the bounty. This presents a problem in terms of aligning incentives to facilitate the desired outcome of locating a copy of a desired file. If a person in possession of the file the bounty issuer is seeking has no assurances that the bounty will actually be paid out once the file is provided, they have no incentive to do so. If the bounty issuer is capable of withholding payment after receiving the file, they have every economic reason to do so. These incentives are completely misaligned.

To solve this misalignment, it is necessary to introduce third party oracles, or escrow operators, to facilitate the vetting and fulfillment of the bounty. In combination with a reputation system, oracles can provide a strong guarantee of the bounty fulfillment if the requested file is provided, correcting the incentive misalignment.

6.1 Bounty Script and Transaction Structure

The bounty structure must allow for the introduction of control by parties outside of the funder to create assurances for any claimant that the bounty will be paid out successfully once the proper file is produced. At the same time, the bounty structure should not permanently relinquish control of the bounty funds to these other parties, requiring the funder to gain their permission to reclaim their funds if the bounty is never met.

Delegation of control in this conditional manner is relatively straightforward to implement using pre-signed transactions and a multisignature Bitcoin script. All that is required is a simple timelock refund mechanism for the funder, and a script structure guaranteeing their inability to unilaterally spend funds before that mechanism becomes available to use, i.e. an n-of-n multisig.

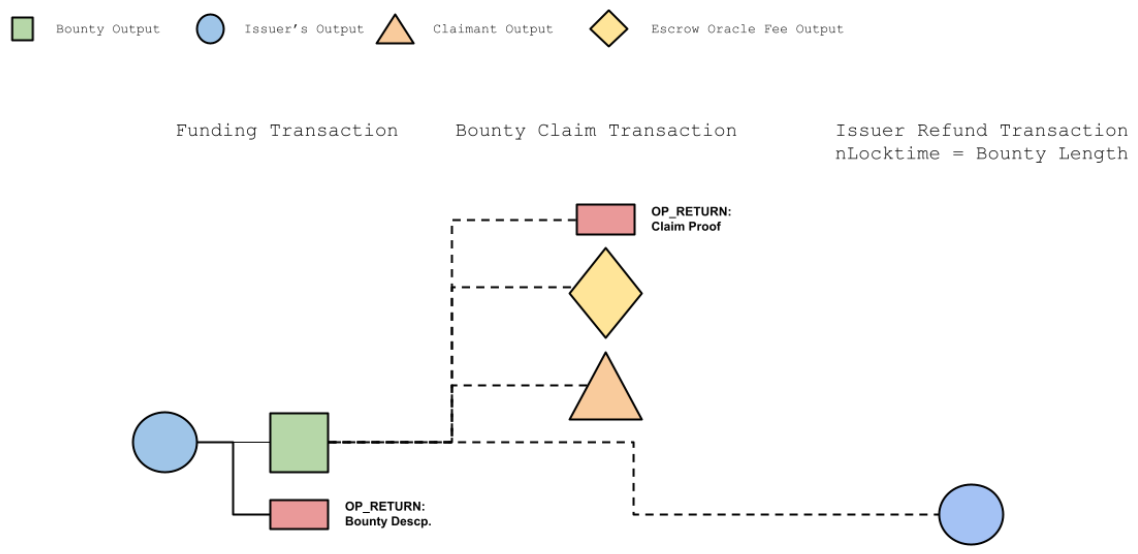

Figure 1. The bounty structure is a very simple scheme. After coordinating the signing of a refund transaction timelocked to the duration of the bounty, the issuer shall fund a multisig output containing the bounty escrow logic (described below). From this point if a claimant appears providing a file to claim the bounty, the escrow logic will be executed. If no claimant comes forward before the end of the bounty duration, the funder will reclaim the funds with the refund transaction.

Figure 1 is the basic transaction flow in the optimistic case for the lifetime of a bounty. After creating the escrow keyset composed of the funder and escrow oracles, and coordinating the signing of a refund transaction and dispute transactions with their own matching refund transaction (see Figure 2), the creator of the bounty funds an output encumbered to that keyset and contract logic with the appropriate amount of satoshis. This funding transaction includes an OP_RETURN output with the event ID of a Nostr event describing the bounty. The specifics of the event structure are described below.

During the course of the bounty being active, the funder and escrow oracles together can sign a transaction paying out to a successful claimant of the bounty. This transaction will include fee payments to the oracles for successfully arbitrating the bounty, the payout to the claimant, and an OP_RETURN output containing the Nostr event ID approving the claim against the bounty. If during the lifetime of the bounty no claimant is successful in meeting the terms of the bounty, the funder can reclaim the bounty funds by submitting their refund transaction to the blockchain after the timelock expiry.

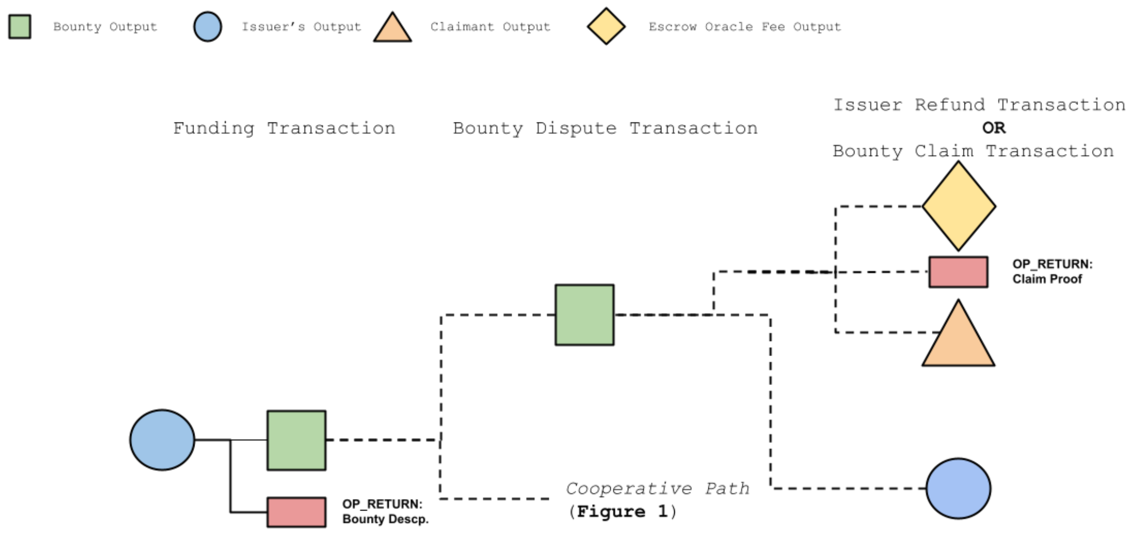

Figure 2. In the case of a dispute between the oracles and funder as to whether the bounty has been fulfilled after a claim attempt, a dispute resolution path exists. This takes the form of a pre-signed transaction that moves the escrow output from an address where the funder is a signatory to one where they are not. This bounty dispute transaction creates the output encumbered by the escrow logic.

In the event of a disagreement between the bounty funder and the escrow oracles as to whether or not the bounty was successfully fulfilled, the escrow oracles can submit to chain a pre-signed transaction removing the bounty funder from the multisig controlling the bounty UTXO. While negotiating the bounty funding, the funder and escrow oracles negotiate a series of dispute transactions in addition to the funder's refund transaction. These pre-signed transactions use the nLocktime field of the transaction to encode which escrow parties initiated arbitration of the dispute. Each individual escrow agent will have their own unique nLocktime value for their version of the dispute transaction. During the signing of these dispute initial transactions, the respective "owner" of each one shall be the last participant in the signing process. This will guarantee that each oracle only has a fully signed copy of their own dispute transaction, ensuring that the use of it will be correctly attributed to the appropriate party. The bounty funder will have a timelocked refund transaction corresponding to each dispute transaction with the same locktime as the primary refund transaction.

In the event that the oracles agree unanimously that a bounty has been successfully fulfilled, they will submit a payout transaction to the claimant, collect their fees, and include the Nostr event ID showing the acceptance of the claim — just like in the optimistic case. In the event the oracles do not agree amongst themselves that the bounty has been fulfilled, the funder will claim the bounty funds with the refund transaction after the timelock expiry.

6.2 Bounty Nostr Messages

The core bounty script structure and protocol design depends implicitly on the use of Nostr messages to record the history of social interactions in regards to the bounty itself, from negotiating the creation of the bounty, the process of arbitration, and the ultimate settlement. Selectively referencing key Nostr events in the chain of these interactions in the bounty transactions, in combination with the cryptographic guarantees Nostr events inherently provide (described in section 4.2), allow an irreversible connection between the off-chain Nostr events and the on-chain activity correlated to the bounty. This is a necessary precondition for thoroughly auditing the history of different escrow oracles.

The standardized message contents for the relevant Nostr events involved in negotiating, fulfilling, and disputing a bounty are defined:

Bounty Offer: For the funder of a bounty to advertise a bounty they wish to fund. This event ID is included in the bounty funding transaction's OP_RETURN output.

- Description: A natural language description of the file being sought after.
- Hash: If known, the exact file or magnet link hash.
- File Type: If known, the exact file type. If not known, the types of files that could conceivably be the desired file.
- UTXO: the transaction ID and output index of the funds to be contributed to the bounty.
- Feerate: Maximum fee the bounty funder will provide to each escrow oracle.
- UTXO Signature: a signature from the bounty funding UTXO key on the serialized fields of the rest of the Nostr event.

Bounty Offer Accept: For the escrow oracles to signal acceptance of a Bounty Offer.

- Escrow Key: The public key to be used in composing the escrow multisig.
- Fee: The fee the escrow oracle will charge.
- Past Bounties: The event IDs of past bounties the escrow oracle has participated in.
- Bond UTXO: The on-chain UTXO collateralizing the escrow oracles Nostr identity.
- Bond Signature: A signature from the bond UTXO key over the serialized fields of the rest of the Nostr event.

Bounty Finalize Proposal: For the bounty funder to finalize the bounty funding.

- Funding PSBT: An unsigned PSBT of the funding transaction.
- Refund PSBT: An unsigned PSBT of the refund transaction.
- Dispute PSBTs: Unsigned PSBTs of the dispute transactions.
- Dispute Refund_PSBTs: Unsigned PSBTs of the refund transactions spending from the dispute transactions, with additional metadata assigning nLocktime values to each escrow oracle.
- Keys: A list of key:value pairs consisting of each escrow agent's Nostr key and provided Bitcoin key, in addition to the funder's Nostr and selected Bitcoin key for the escrow multisig.
- Response Order: The order in which each escrow oracle should reply to the prior event with their own Bounty Accept or Reject event.
- Response Timeout: The length of time after which the funder will abort the bounty finalization.

Bounty Finalize Response: For the escrow oracles to respond to the Bounty Finalize Proposal.

- Funding PSBT: A partially signed PSBT of the funding transaction.
- Refund PSBT: A partially signed PSBT of the refund transaction.
- Dispute PSBTs: Partially signed PSBTs of the dispute transactions.
- Dispute Refund_PSBTs: Partially signed PSBTs of the refund transactions spending from the dispute transactions, with additional metadata assigning nLocktime values to each escrow oracle.

Bounty Finalize Abort: For the escrow oracles to abort transaction finalization and exit the bounty.

- Abort Message: A standard message from an escrow oracle that wants to exit the bounty setup.
- Final Flag: Indicates whether the exit from the bounty is permanent or due to incorrect setup parameters.
- Error: Human readable explanation of what parameter in the Bounty Finalize Proposal is incorrect.

Bounty Finalization Announcement: For the bounty funder to communicate the finalized bounty funding transaction and other bounty criteria.

- Description: A natural language description of the file being sought after.
- Hash: If known, the exact file or magnet link hash.
- File Type: If known, the exact file type. If not known, the types of files that could conceivably be the desired file.
- Funding TXID: The transaction idea funding the bounty contract on-chain.
- Decryption Keys: If the bounty coordination and finalization was conducted using private events, the keys necessary to decrypt the events coordinating the creation of the bounty.

Bounty Claim Attempt: For claimants attempting to fulfill the bounty.

- Magnet Link: If fulfilling the bounty through a Durabit bonded torrent, the magnet link for the torrent.
- Inscription TXID(s): If fulfilling the bounty through an inscription, the TXID(s) constituting the file inscription.
- Claim Address: An on-chain address, or network endpoint ecash can be delivered to (see Section 9).
- Natural Language Payload: If there is any discrepancy between the requested file and provided file — i.e. a different file format, quality, etc. — a rationale for why this still meets the criteria of the bounty.

Bounty Claim Accept: For the escrow oracles and funder to accept a bounty claim. This event ID is included in the bounty claim transaction's OP_RETURN output.

- Accept Message: Signatures from escrow oracles and funders Nostr keys acknowledging bounty fulfillment.

Bounty Claim Reject: For the escrow oracles and funder to reject a bounty claim.

- Reject message: Signatures from the escrow oracles and funders Nostr keys rejecting bounty fulfillment.
- Reason: A human readable reason for the rejection.

Bounty Claim Dispute: For the escrow oracles or funder to dispute a bounty rejection.

- Dispute message: Signatures from the escrow oracles or funders Nostr keys who dispute a bounty claim rejection.
- Reason: A human readable reason for disputing the rejection.

- Dispute TXID: If the dispute is taken to the point of removing the funder from the contract with a dispute transaction, the relevant dispute transaction TXID.

Bounty Refund Notification: For the funder to notify of a refund submission after the bounty expires. This event ID is included in the refund

- Refund Message: A standardized message signaling a refund has occurred.
- Refund TXID: The TXID of the used refund transaction.

Arbitration Message: For all parties involved in any communication relating to dispute resolution and arbitration.

- Tags: Tags pointing to the standard Bounty message(s) or prior Arbitration messages being replied to.
- Content: Human readable messages.

Oracle Announcement: For oracles to advertise their reputational identity and bounty history.

- Bond UTXO: The on-chain UTXO collateralizing the escrow oracles Nostr identity.
- Bond Signature: A signature from the bond UTXO key over the serialized fields of the rest of the Nostr event.
- Bounty Tags: Event IDs of the final Nostr event in all prior bounty event chains this oracle has participated in.

The order in which these events are defined above, with the exception of the Oracle Announcement message (which is defined to allow bounty funders to proactively seek out and select oracles rather than respond reactively to oracles seeking bounties), are the order in which the protocol will be executed. Each event will include tags for the event(s) it is responding to, in the end forming a directed acyclic graph (DAG) structure guaranteeing the final event in the bounty message history will trace back through the entire course of the DAG due to the inherent nature of Nostr's event reference model.

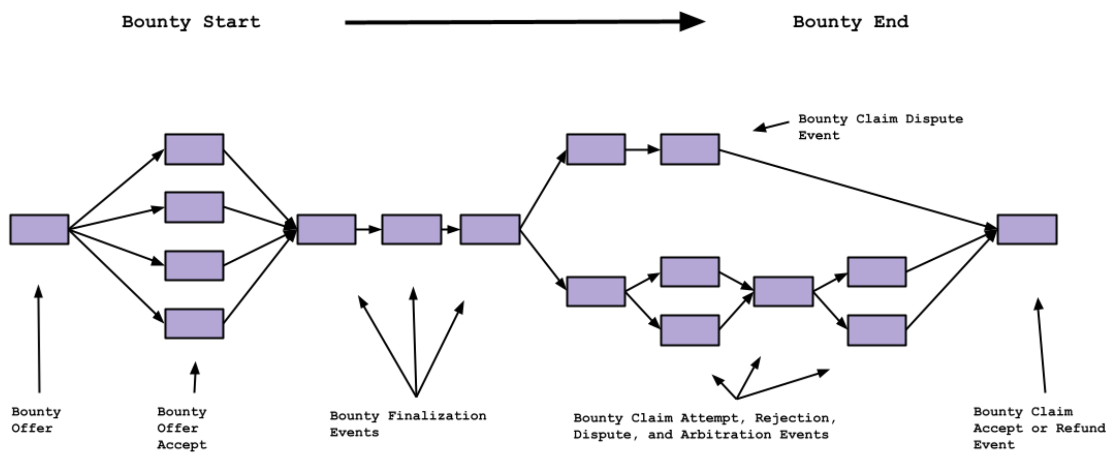
Bounty Start
Bounty End
Bounty Claim Dispute Event
Bounty Offer
Bounty Offer Accept
Bounty Finalization Events
Bounty Claim Attempt, Rejection, Dispute, and Arbitration Events
Bounty Claim Accept or Refund Event

Figure 3. A simple directional flow diagram illustrating the message flow for coordinating the fulfillment of the bounty using the Nostr message scheme defined in Section 6.2.

This structure facilitates the social verification and vetting of oracles' reputations in facilitating prior bounties they were involved in.

6.3 Reputation Rating

The inherent way that Nostr events cryptographically point to each other forms the foundation of what can facilitate a web-of-trust reputational system. The lack of a centralized relay operator the protocol is dependent on, and the fact that all messages are cryptographically signed by a key corresponding to an identity, means that it would be exceedingly difficult (if not impossible) to ensure that a signed message once broadcast was removed from relays and made unavailable. For example, it would be nearly impossible to hide evidence of malicious behavior in past bounties once the Nostr events indicating them were signed and broadcast. It would also be impossible, without compromising the security of another participant's private key, to fraudulently sign an event creating the appearance of willing cooperation and not malicious actions. Finally, it is not possible even in the event of compromising another participant's private key to prevent them from signing and broadcasting a contradictory event indicating key compromise occurred. To add an additional layer of integrity to auditing, OpenTimestamps can be used to record roughly when events in a bounty event chain were created.

All of these factors together create a basis for social vetting the trustworthiness of any given oracle based on their past bounty history. Discoverability of interaction history even in an environment where people have an incentive to obscure or hide it, integrity guarantees of the provenance and authenticity of messages, and the ability to verify any participant referencing a past message is inherently aware of all prior messages in the bounty chain.

Some aspects of this could be automated based on simple heuristics, such as unanimous agreement on every stage of the bounty, i.e. the establishment of the bounty, the response and handling of any claims against it, and the settling of the bounty. Any instance of a Nostr event chain lacking any dispute from establishment to fulfillment or refund of the bounty could be parsed by a user client and automatically flagged as honest behavior. This would leave just disputed bounties or edge cases to manually review.

The last broad scope problem left would be how to distinguish between event chains involving legitimate actors and event chains that are fraudulent and sybiled. Nostr's use as a social media platform offers one route to mitigating this problem by seeking for overlaps in their pre-existing social graph on Nostr and use of the ReQuest protocol. This could provide an initial weighting metric to detect sybil attacks. Nostr users manually reviewing bounty oracle histories and replying with their own Nostr events to rate bounty histories as fraudulent or legitimate could be used to start boot strapping a reliable reputation mechanism.

This could form the basis of a minimum viable trust and reputation system, heavily relying on manual human review to assess the trustworthiness of individual oracles. In the long term, as Civ.Kit begins bootstrapping their platform, this could be used as the basis of the reputational system for ReQuest rather than attempting to build a completely independent system.

7. Bounty Incentive Mechanism

This type of bounty system was executed quite well on www.what.cd. Part of the specification of the bounty was based on the audio fidelity, which is slightly easier to actualize when bitrates and bitdepth are determined by the file.

There would have to be a social vetting on data accuracy, with the decision making of oracles ultimately held accountable socially through their reputation. The more exact criteria are defined in the bounty, the less arbitrary decision making is left in the hands of the oracles given the assumption that obviously erroneous or malicious decisions would lead to a loss of good reputation. For example, a request for a specific song recorded live at a concert decades ago could leave a large degree of ambiguity for oracles to make a decision within. Contrast that with a magnet link that is publicly known but no longer being actively seeded. There is no degree of ambiguity.

8. Implementation

The ideal implementation would be a simple Nostr client with the additional support for the Nostr event formats described in Section 6 bundled with an integrated Bitcoin wallet. Nostr is the only communication medium necessary to coordinate the construction, funding, arbitration, and fulfillment of a ReQuest bounty.

As Civ.Kit matures and bootstraps, the natural long term goal would be to slowly integrate the aspects of Civ.Kit necessary to make use of its reputational system in lieu of continuing to build out an independent reputation and arbitration system.

As noted in Section 9, a Cashu[8] wallet integration in the long term is also a logical consideration to address concerns of high on-chain fees limiting the viability of small value bounties.

9. Future Extensions

A number of possible extensions that could be made to the protocol:

1. Integration of Cashu, specifically NUTS 10 spending conditions supporting a subset of Bitcoin script, to allow for low value bounties to make use of Cashu mints in the event of prohibitively high fees on the Bitcoin network.
2. Integration of the Lightning Network, with custom modifications to support sender generated invoice preimages and nested multisignature scripts to require a group of individuals to authorize channel updates for one side of the channel. This would allow a hodl invoice to be sent to a claimant with a preimage generated by the bounty funder. Upon satisfaction of fulfillment the funder of the bounty would release the preimage unlocking the payment to the claimant, or if not satisfied allow it to timeout. In the event of a dispute between the funder and oracles, the required threshold of oracles could simply allow the hodl invoice payment to expire and fulfill the bounty unilaterally without the funder. This would break many of the guarantees of public visibility or audibility of whether or not a payment was successfully made, such as definitively verifiable proof of payment. It would also break the guarantee of the bounty funder being able to claim a refund in the absence of unilateral agreement between the oracles on the fulfillment of the bounty.

10. Conclusion

The ReQuest platform presents a novel approach to address the evolving demands of the information goods market. The introduction of a data bounty bulletin board stands as a compelling solution to incentivize the permanent publication of desired files. This model not only covers publishing fees but also offers a service payment to those who successfully fulfill the bounty, thus aligning incentives for both users and fulfillers. These ReQuest bounties, if of a large file size and thus not suitable for direct inscription to the Bitcoin blockchain, could instead be upheld once fulfilled within a Durabit magnet link. The collaborative nature of these bounties, combined with the versatility of the data types they can cover, ensures that ReQuest's model can address a wide spectrum of information needs within the market.

The success of this concept hinges on a judicious fusion of social vetting, user-contributed bounties, and contractual specifications in byte size. By combining economic incentives, social validation, and a robust protocol for fulfillment, the platform offers a comprehensive solution for the secure and immutable storage of sought-after data within a torrent file and the Bitcoin blockchain. As the digital landscape continues to evolve, ReQuest's approach has the potential to reshape how users access and locate valuable information, and its adaptability positions it as a significant technological advancement in the peer-to-peer, decentralized information ecosystem.

References

1. Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," (October 31, 2008), https://bitcoin.org/bitcoin.pdf.
2. someguy and hash, “Durabit: A Bitcoin-native Incentive Mechanism for Data Distribution,” (October 31, 2023), www.durabit.org
3. Casey Rodarmor, "Ordinals" (February 2, 2022), https://docs.ordinals.com/inscriptions.html.
4. Peter Todd, "OpenTimestamps: Scalable, Trust-Minimized, Distributed Timestamping with Bitcoin," (September 15, 2016).
5. fiatjaf, “nostr: Notes and Other Stuff Transmitted by Relays,” https://fiatjaf.com/nostr.html (November 20, 2019)
6. Nicholas Gregory, Ray Youssef, and Antoine Riard, "Civ Kit: A Peer-to-Peer Electronic Market System," (2023), (https://github.com/civkit/paper/blob/main/civ_kit_paper.pdf).
7. Joseph Poon and Thaddeus Dryja, “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments,” February 2015
8. Calle, “Cashu: Chaumian ecash wallet and mint for Bitcoin,” (September 23, 2022), https://github.com/cashubtc/nuts